

B.Sc

FIRST YEAR SEMESTER-I

BOTANY

Microbial Diversity, Algae and Fungi



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent....

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2022

CONTENTS

BLOCK / Unit	Title	Page
BLOCK-I	ORIGIN OF LIFE, MICROBIAL DIVERSITY	1
Unit-1 :	Origin of life	3
Unit-2 :	Geological Time Scale	10-14
Unit-3 :	Brief Account of Archaeobacteria, Actinomycetes, Chlamydiae and Mycoplasma	15-22
BLOCK-II	VIRUSES, BACTERIA & CYANOBACTERIA	25
Unit-4 :	Viruses	27-44
Unit-5 :	Bacteria	45-67
Unit-6 :	Cyanobacteria	68-79
BLOCK-III	ALGAE & LICHENS	81
Unit-7 :	General account of Algae	83-99
Unit-8 :	Life history of algae	100-120
Unit-9 :	Lichens	121-130
BLOCK-IV	FUNGI	131-132
Unit-10 :	General account of Fungi	133-155
Unit-11 :	Life history of Fungi	156-181
Unit-12 :	Crop diseases	182-216

B.Sc

FIRST YEAR

SEMESTER-II

BOTANY

**Bryophyta, Pteridophyta,
Gymnosperms and Paleobotany**



"We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...."

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2018

CONTENTS

BLOCK / Unit	Title	Page
BLOCK-I BRYOPHYTA		
unit-1	General Characters and Classification of Bryophyta	1
unit-2	Life History of <i>Marchantia</i> , <i>Anthoceros</i> and <i>Funaria</i>	3-8
unit-3	Evolution of Sporophyte in Bryophyta	9-31
BLOCK-II PTERIDOPHYTA		
unit-4	General Characters and Classification of Pteridophyta	32-38
unit-5	Life History of <i>Rhynia</i> , <i>Lycopodium</i> <i>Equisetum</i> and <i>Marsilea</i>	39
unit-6	Stelar Evolution and Heterospory and Seed Habit in Pteridophytes	41-49
BLOCK-III GYMNOSPERMS		
unit-7	General Characters and Classification of Gymnosperms	50-101
unit-8	<i>Pinus</i>	102-112
unit-9	<i>Gnetum</i>	113
BLOCK-IV PALAEO BOTANY		
unit-10	Palaeo botany : Fossils	115-127
unit-11	<i>Lyginopteris</i>	128-146
unit-12	<i>Williamsonia</i>	147-162
		163
		165-168
		169-174
		175-178

BOTANY

BS313BOT-E

B.Sc

SECOND YEAR SEMESTER-III

BOTANY

Plant Anatomy and Taxonomy



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

-Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2018**

CONTENTS

BLOCK-I: ANATOMY	1
Unit-1 : Meristems and Tissues	3
Unit-2 : Internal structure of leaf	33
Unit-3: Primary structure of stem and root	46
BLOCK-II: SECONDARY GROWTH & WOOD ANATOMY	65
Unit-4 : Secondary growth-Normal and Anomalous	67
Unit- 5: Anatomy of stem & root.	77
Unit-6 : Wood structure & Properties	89
BLOCK-3: PLANT TAXONOMY	97
Unit - 7: Principles of Plant taxonomy & Nomenclature	99
Unit - 8: Systems of Classification	108
Unit - 9: Recent trends in Plant Taxonomy	126
BLOCK-4: SYSTEMATIC TAXONOMY	143
Unit- 10 : Salient features & Economic importance of Polypetalae.	145
Unit- 11: Salient features & Economic importance of gamopetalae.	164
Unit- 12: Salient features & Economic importance of Monochlamydae and monocotyledons.	181
Model question paper	199

BS 413 BOT-E

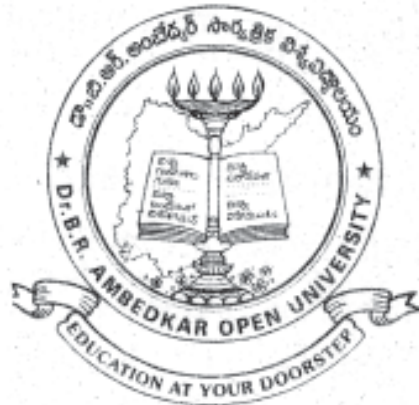
B.Sc

SECOND YEAR

SEMESTER-IV

BOTANY

**DEVELOPMENTAL BIOLOGY AND
MEDICINAL BOTANY**



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

BLOCK / Unit	Title	Page
Block:I	Developmental Biology	1
Unit-1	Microsporogenesis and Male Gametophyte Development	3-7
Unit-2	Megasporogenesis and Female Gametophyte Development	8-15
Unit-3	Polination	16-23
Block:II	Embryology and Palynology	25
Unit-4	Endosperm	27-32
Unit-5	Embryo	33-38
Unit-6	Palynology	39-53
Block:III	Ethnomedicine and Common Medicinal Plants	55
Unit-7	Ethnomedicine	57-60
Unit-8	Traditional Medicinal Systems	61-75
Unit-9	Plants in primary Health Care	76-89
Block:IV	Modern Medicine	91
Unit-10	Traditional Medicine Vs Modern Medicine	93-106
Unit-11	Pharmacognosy	107-114
Unit-12	Plant Crude Drugs	115-130

UG 405 SEE (1) BOT-E

B.Sc

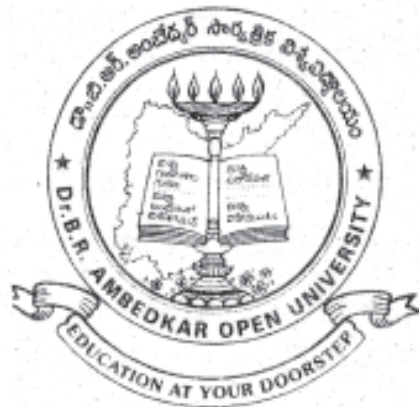
SECOND YEAR

SEMESTER-IV

BOTANY

SKILL ENHANCEMENT ELECTIVE COURSE-SEE-1

BIOFERTILIZERS TECHNOLOGY



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent....

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

BLOCK / Unit	Title	Page
Block :I	BIOFERTLIZERS-TYPES OF BIOINOCULANTS	1
Unit-1.	Nitrogen Fixing Bio- Fertilizers	3-16
Unit-2.	Phosphate Solubilizing Bio-Fertilizers	17-24
Unit-3.	Phosphate Mobilizing Biofertilizers - VAM	25-35
Unit-4.	Bio Fertilizers For Zinc and Silicate Solubilisation and Plant Growth Promoting Rhizobacteria	36-41
Block :II	BIOFERTLIZERS-TYPES OF TECHNOLOGY	43
Unit - 5.	Large Scale Production	45-54
Unit - 6	Biofertilizers - Sustainable Agriculture	55-65
Unit - 7	Biofertilizers - Low Cost Technologies	66-71
Unit - 8	Biofertilizers - Microbial Technology	72-83

BS405 SEE (2) BOT-E

B.Sc

SECOND YEAR

SEMESTER-IV

BOTANY

SKILL ENHANCEMENT ELECTIVE COURSE-SEE-2

FLORICULTURE



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent....

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

BLOCK / Unit	Title	Page
Block :I	Introduction, Turf Management, and Production Technology for cut and Loose Flowers in Floriculture	1
Unit-1.	Introduction, Scope and Importance of Floriculture	3-13
Unit-2.	Landscaping and Turf Management	14-32
Unit-3.	Protected Floriculture : Production Technology for Cut/Loose Flowers	33-53
Unit-4.	Production Technology for Cut and Loose Flowers	54-69
Block :II	Biotechnology, Disease Management, Value addition in Flower Crops and Computer Technology in Floriculture	71
Unit - 5.	Application of Biotechnology in Flower Crops	73-86
Unit - 6	Pest and Disease Management and Post Harvest Techniques of Flower Crops	87-109
Unit - 7	Value Addition in Flower Crops	110-124
Unit - 8	Computer Technology in Floriculture	125-135

BS 513 BOT-E

B.Sc

THIRD YEAR SEMESTER-V

BOTANY

CELL BIOLOGY AND GENETICS



"We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit"

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2023**

CONTENTS

<i>Block / Unit</i>	<i>Title</i>	<i>Page</i>
Block:I Cell Biology		1
Unit 1	Ultra Structure of Plant cell	3-14
Unit 2	Nucleus	15-18
Unit 3	Nucleic Acids	19-30
Block:II Chromosomes and Cell Division		31
Unit 4	Chromosomes	33-45
Unit 5	Extra Nuclear Genome	46-50
Unit 6	Cell Division	51-62
Block:III Genetics		63
Unit 7	Mendelism	65-90
Unit 8	Linkage and Crossing Over	81-91
Unit 9	Genetic Code and Protein Synthesis	92-104
Block:IV Mutations, Gene Organisation and Expression, Genetic Engineering		105
Unit-10	Mutations	107-119
Unit-11	Gene Organisation	120-131
Unit-12	Genetic Engineering	132-142
	Model Question Paper	144-144

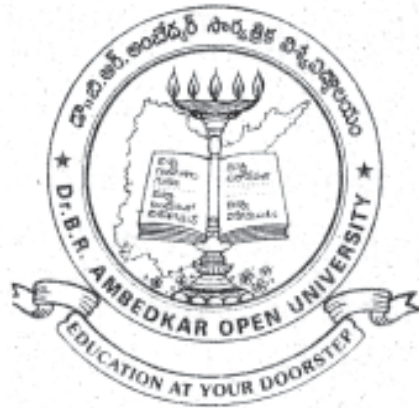
BS 513 BOT DSE (A) -E

B.Sc

THIRD YEAR SEMESTER-V

BOTANY

CROP PRODUCTION



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2020

CONTENTS

BLOCK / Unit	Title	Page
Block:I	Soil and Soil Fertility	1
Unit-1	Agricultural Meteorology	3-17
Unit-2	Soils	18-30
Unit-3	Manures and Fertilizers	31-39
Block:II	Tillage and Weed Control	41
Unit-4	Tilth and Tillage	43-55
Unit-5	Weeds and Weed Control	56-66
Unit-6	Cropping Systems	67-77
Block:III	Crop Production - I	79
Unit-7	Cereals and Millets	81-126
Unit-8	Oil seeds	127-155
Unit-9	Pulses	156-175
Block:IV	CROP PRODUCTION - II	177
Unit-10	Cash Crops	179-205
Unit-11	Spices	206-228
Unit-12	Fruits and Nuts	229-267
	Model Question Paper	268-269

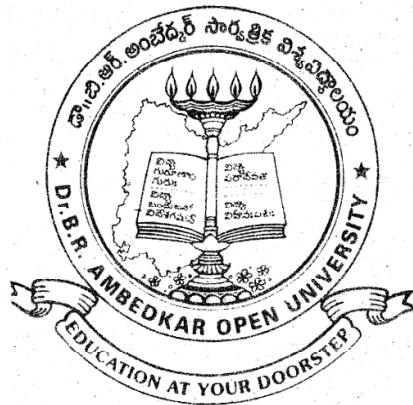
BS 513 BOT DSE (B) -E

B.Sc

THIRD YEAR SEMESTER-V

BOTANY

**PLANT TISSUE CULTURE
AND
GENETIC ENGINEERING**



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2021**

CONTENTS

BLOCK / Unit	Title	Page
Block:I Plant Tissue Culture Introduction and Application		1
Unit-1	Plant Tissue Culture	3-10
Unit-2	Applications of Plant Cell and Tissue Culture	11-15
Unit-3	Clonal Propagation	16-24
Block:II Invitro Culture		25
Unit-4	Invitro Morphogenesis	27-30
Unit-5	Types of Culture	31-39
Unit-6	Organ Culture	40-47
Block:III Gene Cloning		49
Unit-7	r-DNA Technology	51-59
Unit-8	Gene Clonning Vectors	60-66
Unit-9	Methods of Gene Transfer	67-79
Block:IV Transgenic Plants		81
Unit-10	Pathogen Free Plants	83-90
Unit-11	Improved Growth	91-97
Unit-12	Edible Vaccines	98-104
	Model Question Paper	105-106

BS 613 BOT - E

B.Sc

THIRD YEAR

SEMESTER-VI

BOTANY

**DISCIPLINE SPECIFIC COMPULSORY CORE COURSE
PLANT PHYSIOLOGY AND ECOLOGY**



" We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit..

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2023**

CONTENTS

BLOCK / Unit	Title	Page
Block:I	Plant Physiology & Plant Bio-Chemistry	1
Unit-1	Water Plant Relations	3-14
Unit-2	Mineral Nutrition	15-24
Unit-3	Enzymes.	25-35
Block:II	Photosynthesis	37
Unit-4	Photosynthesis	39-49
Unit-5	Carbon Assimilation Pathways	50-72
Unit-6	Translocation of Solutes	73-83
Block:III	Plant Respiration and Nitrogen Metabolism	85
Unit-7	Respiration	87-104
Unit-8	Nitrogen Metabolism and Nitrogen Fixation	105-121
Unit-9	Plant Growth, Growth Regulators Dormancy and Movement Physiology of Flowering	122-147
Block:IV	Ecology	149
Unit-10	Ecosystem	151-172
Unit-11	Ecological Successions	173-181
Unit-12	Environmental Pollution	182-193
Model Question Paper		194-195

BS 613 BOT DSE. (C)-E.

B.Sc

THIRD YEAR

SEMESTER-VI

BOTANY

DISCIPLINE SPECIFIC ELECTIVE - C
PLANT DISEASE MANAGEMENT



" We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit..

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD

2020

CONTENTS

BLOCK / Unit	Title	Page
Block:I	Major Diseases of Crop Plants	1
Unit-1	Fungal Diseases - I	3-27
Unit-2	Fungal Diseases - II	28-51
Unit-3	Bacterial, Viral and Mycoplasma Diseases.	52-75
Block:II	Major Insect Pest and Parasitic Plants of Crops	77
Unit-4	Cotton Pests	79-83
Unit-5	Paddy and Storage Pests	84-93
Unit-6	Angiosperm Parasitic Plants	94-102
Block:III	Plant Pest Management	103
Unit-7	Concepts and principles of pests and disease management practices	105-110
Unit-8	Cultural, Physical and Mechanical Methods	111-117
Unit-9	Chemical Methods	118-127
Block:IV	Ecological Pest Management Practices	129
Unit-10	Biological Control	131-137
Unit-11	Transgenic Technology for Management of Pests and Diseases.	138-151
Unit-12	Integrated pest Management	152-161
Model Question Paper		162-163

BS 613 BOT DSE (D)-E

B.Sc
THIRD YEAR SEMESTER-VI
BOTANY
SEED TECHNOLOGY



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2021**

CONTENTS

Block/Unit	Title	Pages
Block:I	Seed and Seed Storage	1-26
Unit-1:	Seed structure and dormancy	1
Unit-2:	Seed Storage	8
Unit-3:	physico and biochemical changes during seed storage	17
Block:II	Seed viability and Seed marketing and treatment	27-51
Unit-4:	seed viability and genetic erosion	27
Unit-5:	seed marketing	33
Unit-6:	seed treatment	43
Block:III	Structure of Pollen and Ovule, Seed development	51-76
Unit-7:	structure of pollen and ovule	51
Unit-8:	Principles of hybrid seed production	59
Unit-9:	Seed development and heterosis	65
Block:IV	Seed Production, Certification and Seed Banks	77-109
Unit-10:	seed testing	77
Unit-11:	seed certification	91
Unit-12:	seed banks	109
Model Question Paper	110

BS114CHE-T

B.Sc.

మొదటి సంవత్సరం

సెమిస్టర్ – I

రసాయన శాస్త్రం

అకర్బన, కర్బన & భౌతిక రసాయన శాస్త్రం-1



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

-Dr. B. R. Ambedkar

డా . బి. ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2017

విషయ సూచిక

ఖండం - 1 : అకర్షన రసాయన శాస్త్రం	1
భాగం - 1 : గ్రూపు - 1 A మూలక రసాయన శాస్త్రం.	2 - 22
భాగం - 2 : గ్రూపు - 11 A మూలక రసాయన శాస్త్రం.	23 - 38
భాగం - 3 : అంశిక సూక్ష్మ విశ్లేషణం - అంతర్గత సూత్రాలు	39 - 65
ఖండం - 2 : కర్షన రసాయన శాస్త్రం	66
భాగం - 4 : కర్షన రసాయన శాస్త్రం నిర్మాణ సిద్ధాంతం	67 - 97
భాగం - 5 : ఆల్కేన్లు మరియు సైక్లో ఆల్కేన్లు	98 - 125
భాగం - 6 : ఆల్కేన్లు మరియు ఆల్కాడైయిన్లు.	126 - 146
ఖండం - 3 : భౌతిక రసాయన శాస్త్రం - 1	147
భాగం : 1 పరమాణు నిర్మాణం -	148- 179
భాగం : 2 పరమాణు నిర్మాణం - 2	167 -180
భాగం : 3 వాయువులు	181 - 209
ఖండం - 4 : సాధారణ రసాయన శాస్త్రం	210
భాగం : 10. ఆవర్తన పట్టిక	211 - 242
భాగం : 11. రసాయన బంధం బావనలు మరియు రసాయన బంధ సిద్ధాంతాలు	233 - 243
భాగం : 12. అణు ఆర్బిటాల్ సిద్ధాంతం	244 - 259
మాధిరి ప్రశ్నా పత్రం	260 - 263

BS214CHE-T

B.Sc.

మొదటి సంవత్సరం

సెమిస్టర్ - 2

రసాయన శాస్త్రం

అకర్బన, కర్బన & భౌతిక రసాయన శాస్త్రం-2



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

-Dr. B. R. Ambedkar

డా . బి. ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2018

విషయ సూచిక

ఖండం - 1: అకర్పన రసాయన శాస్త్రం	1
భాగం - 1: గ్రూపు IIIA మూలకాల రసాయన శాస్త్రం	3
భాగం - 2: గ్రూపు IVA మూలకాల రసాయన శాస్త్రం	22
భాగం - 3: గ్రూపు IVA మూలకాల రసాయన శాస్త్రం మరియు టైట్రీమెట్రీ, భారాత్మక విశ్లేషణం - అంతర్గత సూత్రాలు.	43
ఖండం - 2: కర్పన రసాయన శాస్త్రం	79
భాగం - 4: ఆల్యైన్లు	81
భాగం - 5: బెంజీన్ - ఏరో మాటిక్ స్వభావం	93
భాగం - 6: ఎరీన్లు	101
ఖండం- 3: కర్పన రసాయన శాస్త్రం	119
భాగం - 7: హలోజన్ ఉత్పన్నాలు	120
భాగం - 8: హైడ్రాక్సీ ఉత్పన్నాలు (ఆల్కహాల్లు)	137
భాగం - 9: ఈథర్లు మరియు ఇపోక్సైడ్లు	182
ఖండం - 4: భౌతిక రసాయన శాస్త్రం	195
భాగం - 10: ద్రావణాలు	197
భాగం - 11 : ఘనస్థితి	224
భాగం - 12: ప్రావణ్య సమతాస్థితి - ఏక అనుఘటక వ్యవస్థ	243
మాదిరి ప్రశ్నా పత్రం	259

BS314CHE-T

B.Sc.

రెండవ సంవత్సరం

సెమిస్టర్ - 3

రసాయన శాస్త్రం

అకర్షణ, కర్షణ & భౌతిక రసాయన శాస్త్రం-3



” మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైనా వదులుకోవచ్చు గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు. ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేదీ లేదు ”

- డా. బి. ఆర్. అంబేద్కర్

డా . బి. ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2018

విషయ సూచిక

ఖండం -1 : అకర్పన రసాయన శాస్త్రం	1
భాగం - 1: గ్రూపు VIA మూలకాల రసాయన శాస్త్రం	3
భాగం - 2: గ్రూపు VIIA (హాలోజన్లు) మూలకాల రసాయన శాస్త్రం	31
భాగం - 3: గ్రూపు 'O' మూలకాలు వాటి సమ్మేళనాలు	56
ఖండం -2 : కర్పన రసాయన శాస్త్రం	71
భాగం - 4 : ఆల్మిహైడ్రైడ్లు మరియు కీటోన్లు	73
భాగం - 5 : కార్బాక్సిలిక్ ఆమ్లాలు మరియు వాటి ఉత్పన్నాలు	102
భాగం - 6 : కర్పన రసాయన చర్యలో కార్బోనయాన్ మాధ్యమ పదార్థాలు	143
ఖండం- 3 : భౌతిక రసాయన శాస్త్రం	155
భాగం - 7: ద్రవ స్థితి	157
భాగం - 8: కణాధార ధర్మాలు -1 : భాష్ప పీడన నిమ్నత	176
భాగం - 9 : కణాధార ధర్మాలు - 2 : ద్రవాభిసరణం లేదా ఆస్మాసిస్	195
ఖండం - 4 : సామాన్య రసాయన శాస్త్రం	215
భాగం - 10 : కేంద్రక రసాయన శాస్త్రము	217
భాగం - 11 : కొల్లాయిడ్ మరియు ఎమల్షన్లు	233
భాగం - 12 : అధిశోషణం	246
మాదిరి ప్రశ్నా పత్రం	261

BS414CHE-T

B.Sc.

రెండవ సంవత్సరం

సెమిస్టర్ - 4

రసాయన శాస్త్రం

అకర్బన, కర్బన & భౌతిక రసాయన శాస్త్రం - 4



” మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైనా వదులుకోవచ్చు గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు. ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేదీ లేదు”

- డా. బి. ఆర్. అంబేద్కర్

డా . బి. ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2019

విషయ సూచిక

ఖండం -1 : అకర్షన రసాయన శాస్త్రం	1
భాగం - 1 : d - బ్లాక్ మూలకాలు	3
భాగం - 2 : f - బ్లాక్ మూలకాలు	18
భాగం - 3 : సమన్వయ సమ్మేళనాలు - 1	29
ఖండం -2 : కర్షన రసాయన శాస్త్రం	49
భాగం - 4 : నైట్రోజన్ సమ్మేళనాలు - 1 (నైట్రో మరియు నైట్రైట్ సమ్మేళనాలు)	51
భాగం - 5 : నైట్రోజన్ సమ్మేళనాలు - 2 (అమీన్లు)	65
భాగం - 6 : కర్షన సమ్మేళనాల త్రిమితీయ సాదృశ్యాలు	88
ఖండం- 3 : కర్షన రసాయన శాస్త్రం	113
భాగం - 7: అనురూపాత్మక విశ్లేషణ - పరిచయము.	115
భాగం - 8: కార్బోహైడ్రేట్లు - 1	137
భాగం - 9: కార్బోహైడ్రేట్లు - 2	163
ఖండం - 4 : భౌతిక రసాయన శాస్త్రం	173
భాగం - 10 : రసాయన గతిక శాస్త్రం - చర్యల క్రమాంకం, అణుత	175
భాగం - 8 : ఉష్ణగతిక శాస్త్రం - I	199
భాగం - 9; విద్యుత్ రసాయన శాస్త్రం - I	224
మాదిరి ప్రశ్నా పత్రం	249

UG405 SEE (CHE1)-E

B.Sc.

CHEMISTRY

SECOND YEAR

SEMESTER – 4

SKILL ENHANCEMENT ELECTIVE COURSE - SEE-1

INSTRUMENTATION SKILLS



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B. R. Ambedkar

Dr. B. R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

Block - 1 : Instrumentation Skills - 1	1
Unit - 1 : General Principles of Instrumentation	3
Unit - 2 : Principles and Techniques of Gas Analysis	19
Unit - 3 : Principles and Techniques of Ph Meter and Potentiometer	31
Unit - 4 : Principles and Techniques of Conductometry	54
Block-2 : Instrumentation Skills - 2 (Spectroscopy)	71
Unit - 5 : Principles of Spectroscopy	73
Unit - 6 : Principles and Techniques of UV-Visible Spectroscopy	88
Unit - 7 : Principles and Techniques of Colorimetry	106
Unit – 8: Introduction to Infrared Spectroscopy	120
Model question papers	134

UG405 SEE (CHE2)-E

B.Sc.

CHEMISTRY

SECOND YEAR

SEMESTER – 4

SKILL ENHANCEMENT ELECTIVE COURSE - SEE-2

COSMETIC CHEMISTRY



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B. R. Ambedkar

Dr. B. R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

Block - 1 : Cosmetic chemistry - 1	1
Unit - 1 : Drugs and Cosmetics act and rules	3
Unit - 2 : Preservatives, vehicles used in cosmetics	15
Unit - 3 : Stabilizers antioxidants surfactants and humectants	30
Unit - 4 : Organoleptic additives	45
Block-2 : Cosmetic chemistry - 2	53
Unit - 5 : Basic knowledge of skin and hair	55
Unit - 6 : Skin preparations	64
Unit - 7 : Shampoos, conditioners and cosmetics for the face	75
Unit - 8 : Quality control of Cosmetic products	97
Model question papers	106

B.Sc.

మూడవ సంవత్సరం

సెమిస్టర్ - 5

రసాయన శాస్త్రం

అకర్బ, కర్బన & భౌతిక రసాయన శాస్త్రం - 5



” మనం నాగరికత నమకూర్చిన వస్త్రగత ప్రయోజనాలైనా వదులుకోవచ్చు గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు. ఎందుకంటే విద్యను వించిన వస్త్రగత ప్రయోజనమేదీ లేదు ”

- డా. బి. ఆర్. అంబేద్కర్

డా . బి. ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2020

విషయ సూచిక

ఖండం - 1: అకర్మ రసాయన శాస్త్రం	1
భాగం - 1 : సమన్వయ సమ్మేళనాలు-2	3
భాగం - 2: లోహ బంధం	15
భాగం - 3: లోహ సంగ్రహణ శాస్త్రం - సాధారణ సూత్రాలు.	28
ఖండం - 2: కర్మన రసాయన శాస్త్రం	55
భాగం - 4 : విజాతీయ చక్రియ సమ్మేళనాలు	57
భాగం - 5: అమీనో ఆమ్లాలు మరియు ప్రోటీన్లు	76
భాగం - 6: విటమిన్లు మరియు హార్మోన్లు	94
ఖండం- 3: భౌతిక రసాయన శాస్త్రం	115
భాగం - 7 : ఉష్ణగతిక శాస్త్రం-2.	117
భాగం - 8 : విద్యుత్ రసాయన శాస్త్రం - 2.	152
భాగం - 9 : కాంతి రసాయన శాస్త్రం.	172
ఖండం - 4: సాధారణ రసాయన శాస్త్రం - 2	189
భాగం - 10 : అతి నీలలోహిత-దృశ్య వర్ణ పటశాస్త్రం.	191
భాగం - 11 : పరారుణ వర్ణపట శాస్త్రం.	
భాగం - 12 : జీవ వ్యవస్థలలో లోహ అయాన్లు	208
మాదిరి ప్రశ్నా పత్రం	218

BS514CHEDSE(A)-E

B.Sc.

THIRD YEAR

SEMESTER – 5

CHEMISTRY

**Green Chemistry, Organic Chemistry,
Environmental Chemistry- DSE (A)**



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B. R. Ambedkar

Dr. B. R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2020

CONTENTS

BLOCK-1: GREEN CHEMISTRY	1
Unit - 1 : Introduction to Green Chemistry	3
Unit - 2 : Principles of Green Chemistry.	7
Unit - 3 : Green synthesis - Some Reactions.	22
BLOCK-2 : ORGANIC CHEMISTRY	31
Unit - 4 : Introduction to Pericyclic Reactions	33
Unit - 5 : Introduction to Assymmetric Synthesis	54
Unit - 6 : Structural Elucidation of Organic Compounds by Chemical Methods	70
BLOCK-3: ENVIRONMENTAL CHEMISTRY-1	83
Unit – 7: Atmosphere and its components	85
Unit – 8: Air Pollution	97
Unit – 9: Water Pollution	116
BLOCK-4 : ENVIRONMENTAL CHEMISTRY-2	127
Unit – 10: Soil and Sound Pollution.	129
Unit – 11: Chemical toxicology	138
Unit – 12: The state of global environment	149
Model question papers	159

BS514CHEDSE(B)-T

B.Sc.

మూడవ సంవత్సరం

సెమిస్టర్ - 5

రసాయన శాస్త్రం

వ్యవసాయ రసాయనాలు, కర్బన రసాయన శాస్త్రం

మరియు నానో సంఘేళనాలు - DSE(B)



” మనం నాగరికత నమకూర్చిన వస్తుగత ప్రయోజనాలైనా వదులుకోవచ్చు గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు. ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేదీ లేదు ”

- డా. బి. ఆర్. అంబేద్కర్

డా. బి. ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2021

విషయ సూచిక

ఖండం - 1 : వ్యవసాయరసాయనాలు - 1	1
భాగం - 1 : మొక్కల పోషకాల చరిత్రాత్మక అంశాలు	3
భాగం - 2 : మొక్కల తెగుళ్ళ నియంత్రణ రసాయనాల సంక్షిప్త పర్వేక్షణ	12
భాగం - 3 : కీటకనాశకాలు	26
ఖండం - 2 : వ్యవసాయరసాయనాలు - 2	33
భాగం - 4 : శిలీంధ్రనాశకాలు	41
భాగం - 5 : గుల్మనాశకాలు, రోడెంట్ నాశకాలు	50
భాగం - 6A : మొక్కల పెరుగుదల హార్మోన్లు	59
భాగం - 6B : చీడనాశక తయారీలు	65
ఖండం- 3 : కర్షణ రసాయన శాస్త్రం	71
భాగం - 7 : పెరిస్టెక్లిక్ చర్యలు - పరిచయం	73
భాగం - 8 : అసౌష్ఠ్య సంశ్లేషణం - పరిచయం	97
భాగం - 9 : రసాయన పద్ధతుల ద్వారా కర్షణ సమ్మేళనాల అణు నిర్మాణ నిర్ధారణ.	114
ఖండం - 4 : నానో పదార్థ రసాయన శాస్త్రం	129
భాగం - 10: నానో పదార్థాలు - పరిచయం	131
భాగం - 11: నానో పదార్థాల సంశ్లేషణం మరియు అభిలాక్షణికరణం	145
భాగం - 12: కర్షణ ఆధారిత నానో నిర్మాణాలు మరియు నానో పదార్థాల అనువర్తనాలు. 159	
మాదిరి ప్రశ్నా పత్రం	171

BS614CHE-E

B.Sc.

THIRD YEAR

SEMESTER - 6

CHEMISTRY

Inorganic, Organic & Physical Chemistry-6



"We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit"

-Dr. B. R. Ambedkar

Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD

CONTENTS

BLOCK-1: INORGANIC CHEMISTRY	1
Unit - 1: Metal Carbonyls and Nitrosyls	3
Unit - 2: Non- aqueous Solvents	20
Unit - 3: Organo metallic compounds	34
BLOCK-2 : ORGANIC CHEMISTRY	47
Unit - 4 : Alkaloids	49
Unit - 5 : Terpenoids	65
Unit -6 : Introduction to Synthetic strategies	77
BLOCK-3: PHYSICAL CHEMISTRY	95
Unit - 7 : Introduction and classification of Catalysis.	97
Unit - 8: Homogeneous Catalysis.	105
Unit - 9 : Heterogeneous Catalysis.	118
BLOCK-4 : GENERAL CHEMISTRY	129
Unit - 10 : Mass spectroscopy	131
Unit - 11 : H ¹ NMR Spectroscopy.	152
Unit - 12 : Introduction to Symmetry	171
Model question papers	185

BS614CHEDSE(C)-T

B.Sc.

మూడవ సంవత్సరం

సెమిస్టర్ - 6

రసాయన శాస్త్రం

ఔషధ రసాయన శాస్త్రం మరియు ఎంజైమ్లు - DSE (C)



” మనం నాగరికత నమకూర్చిన వస్తుగత ప్రయోజనాలైనా వదులుకోవచ్చు గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు. ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేదీ లేదు ”

- డా. బి. ఆర్. అంబేద్కర్

డా. బి. ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2020

విషయ సూచిక

ఖండం - 1 : ఔషధాల ప్రాథమిక అంశాలు	1
భాగం - 1 : ఔషధాల చరిత్ర మరియు సాంకేతిక పదాలు	3
భాగం - 2: ఔషధాల నామకరణం మరియు వర్గీకరణం	14
భాగం - 3: ఫార్మాసుటిక్స్ ప్రయోజనాలు మరియు మోతాదు రూపాలు.	23
ఖండం - 2 : ఔషధాల చర్యలు - 1	35
భాగం - 4: బాధానివారిణులు	37
భాగం - 5: సమ్మోహకాలు, ఉపశమనకారులు, ప్రశాంతకారులు	48
భాగం - 6: మలేరియా నివారిణులు.	55
ఖండం- 3 : ఔషధాల చర్యలు - 2	65
భాగం - 7 : బాక్టీరియా నిరోధక ఔషధాలు	67
భాగం - 8 : ఆంటి బయాటిక్స్	77
భాగం - 9 ; హృదయ ప్రసరణ ఔషధాలు మరియు కేంద్ర నాడీమండల వ్యవస్థ (CNS) ఉత్తేజకాలు	89
ఖండం - 4 : ఔషధాల చర్యలు - 3	99
భాగం -10 : ఆంటీ హెల్మింటిక్ ఔషధాలు	101
భాగం - 11 : HIV/AIDS ఔషధాలు	107
భాగం - 12 ; ఎంజైమ్లు	118
మాదిరి ప్రశ్నా పత్రం	137

BS614CHEDSE(D)-E

B.Sc.

THIRD YEAR

SEMESTER – 6

CHEMISTRY

**Polymer Chemistry, Computational Chemistry,
Separation Techniques and Chromatography -
DSE (D)**



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B. R. Ambedkar

Dr. B. R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2021

CONTENTS

BLOCK-1: POLYMER CHEMISTRY	1
Unit – 1: Introduction, Classification and Crystallinity of Polymers.	3
Unit – 2: Molecular weight determination of Polymers.	13
Unit – 3: Synthesis and applications of Polymers	29
BLOCK-2: COMPUTATIONAL CHEMISTRY & MOLECULAR MODELING	43
Unit – 4: Introduction to computational chemistry	45
Unit – 5: Molecular interactions and Energy calculations	55
Unit – 6: Molecular energy minimisation and molecular dynamics	66
BLOCK-3 : SEPARATION TECHNIQUES & CHROMATOGRAPHY-I	81
Unit - 7 : Separation techniques by solvent extraction	83
Unit - 8 : Introduction and principles of chromatography	102
Unit - 9 : Principles and techniques of paper chromatography	120
BLOCK-4 : CHROMATOGRAPHY - II	141
Unit – 10: Principles and techniques of Thinlayer Chromatography	143
Unit – 11: Principles and techniques of Column Chromatography.	166
Unit – 12: Principles and techniques of GC and HPLC	190
Model question papers	213

B.A/B.Com/B.Sc

FIRST YEAR

SEMESTER-I

DISCIPLINE SPECIFIC CORE COURSE-DSC-1

COMPUTER APPLICATIONS

COMPUTER FUNDAMENTALS



"We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit"

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2022

CONTENTS

BLOCK / Unit	Title	Page
BLOCK – I: COMPUTERS IN GENERAL		1
Unit-1:	Introduction to Computers	3-14
Unit-2:	Human-Computer interface	15-30
Unit-3:	Use of Computers in Education and Research	31-45
BLOCK – II: COMPUTER ACCESSORIES AND TECHNOLOGY		47
Unit-4:	Input & Output Technologies	49-68
Unit-5:	Memory	69-85
Unit-6:	Inside the Computer	86-116
BLOCK – III: HOW COMPUTER WORKS		117
Unit-7:	Number Systems	119-137
Unit-8:	Binary Arithmetic	138-151
Unit-9:	Central Processing Unit	152-164
BLOCK – IV: EMERGING SOFTWARE TECHNOLOGIES		165
Unit-10:	Computer Security threats	167-188
Unit-11:	Introduction to Windows-10	189-204
Unit-12:	Modern Computing	205-236

BS 215 CA-E

B.A/B.Com/B.Sc

FIRST YEAR

SEMESTER-II

DISCIPLINE SPECIFIC CORE COURSE-DSC-2

COMPUTER APPLICATIONS

PROGRAMMING WITH PYTHON



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2019

CONTENTS

Block/Unit	Title	Page
BLOCK – I: PYTHON FUNDAMENTALS		1
Unit-1:	Introduction to Python	3-24
Unit-2:	Data Types and Operators in Python	25-56
Unit-3:	Control Statements	57-73
BLOCK – II: PYTHON NUMERICAL AND CODE SEQUENCES		75
Unit-4:	Arrays in Python	77-106
Unit-5:	Strings and Characters	107-122
Unit-6:	Functions	123-152
BLOCK – III: SEQUENCES		153
Unit-7:	Lists	155-168
Unit-8:	Tuples	169-180
Unit-9:	Dictionaries	181-194
BLOCK – IV: ADVANCED PYTHON		195
Unit-10:	Object Oriented Concepts	197-218
Unit-11:	Graphic User Interfaces with Python	219-236
Unit-12:	Files and Database	237-251

BS 315 CA-E

B.A/B.Com/B.Sc

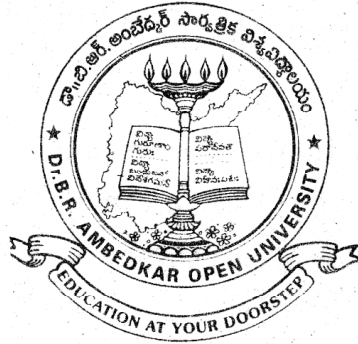
SECOND YEAR

SEMESTER-III

DISCIPLINE SPECIFIC CORE COURSE-DSC-3

COMPUTER APPLICATIONS

PROGRAMMING WITH C AND C++



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2019

COURSE TEAM

Course Design Team (CBCS)

1. Prof. A. Vinaya Babu
2. Prof. S.V.L.Narasimham
3. Prof. V. Vijay Kumar.
4. Prof. B. Vishnu Vardan.
5. Prof. S. Vishwanatha Raju
6. Mr. Vorsu Mallaiah
7. Mr. V. Giridhar.
8. Mr. D. Venkateswarlu
9. Prof. P. Madhusudhan Reddy.

Course Development Team (CBCS)

Editor & Writer

Prof. S.V.L.Narasimham

Associate Editor

Vorsu. Mallaiah

Cover Design

G. V. Swamy

First Edition : 2019

© 2019, Dr. B. R. Ambedkar Open University, Hyderabad, A.P.

All rights reserved. No part of this book may be reproduced in any form without the permission in writing from the University.

The text forms part of Dr. B. R. Ambedkar Open University Programme.

Further information on Dr. B. R. Ambedkar Open University courses may be obtained from the Director (Academic), Dr. B. R. Ambedkar Open University, Road No. 46, Prof. G. Ram Reddy Marg, Jubilee Hills, Hyderabad- 500033.

Web: www.braou.ac.in

E-mail: info@braou.ac.in

Printed on behalf of Dr. B. R. Ambedkar Open University, Hyderabad by the Registrar.

Lr. No.-

Printed

at:

.....

PREFACE

This book on “Programming with C and C++” is prepared for the Under Graduate Students of B.A./B.Com/B.Sc studying in Dr. B. R. Ambedkar Open University. This course is offered as a Discipline Specific Core Course (DSC) under CBCS being offered in III Semester as four credit course. For the convenience of the students, the entire syllabus is organized in to four blocks. Each block consists of three units. Each block covers a specific area of the subject. The Units are prepared by the expert in accordance with a format so designed as to enable the student to read and understand them without much difficulty. Each unit begins with a statement of the Objectives followed by an introduction, self check exercises in between and at the end model examination questions intended to test the student’s comprehension of its subject matter.

“Programming with C and C++” course aims at gaining the knowledge of Algorithms, Flowcharts, installing C++, opening C++ Editor, Editing C and C++ Programs, Saving source code as .c or .cpp based on whether it is C program or C++ Program, compiling and executing C or C++ Programs. And also enable the students to gain the knowledge of using C and C++ in various fields such as Learning, Assessing, teaching. The course explains each concept with crystal clear examples and images. C and C++ are powerful languages to learn, understand, programs easily by any beginners or experts in Computer Science. It enabled all the students to learn basics to advanced features of C and C++ languages. Today, we carry more computing power on our smart phones and tabs than was available in the early models. The past decade has seen the revolutionary development in the hardware and computer infrastructure. The software such as High level languages, Middleware, mobile operating systems, GPS and deep learning software enhanced the skills of the people in solving the challenging problems and living with emerging technologies. There is no doubt that the intelligent computing technologies has helped to change the world. Most websites are hosted on servers running free software. Even the mighty IBM, Google and Amazon also have their walls placed in the rock-solid foundations of GNU/Linux and C/C++ software.

This course consists of four blocks and 12 units. Block-1 deals with the Algorithms, Flowcharts, installing C++, compiling and executing C or C++ Programs, variables, data types, type conversion. Block-II introduces the flow of control such as sequencing, branching, iterations, unconditional branching, defining and calling functions, declaring pointers and pointer arithmetic, creating and performing various operations on strings. Block-III deals with creating, Reading, Printing one dimensional, two dimensional, and three dimensional arrays, writing programs using structures, unions and powerful files. Block – IV deals with various concepts of Object Oriented Programming Paradigms which are classes, objects, abstraction, encapsulation, inheritance, polymorphism, dynamic binding, templates, user defined data types, constructors, destructors, garbage management, multiple and multi-level inheritance, operator overloading, function overloading and overriding, passing objects to functions, this operator, friend functions, friend classes, static data members and member functions.

The University hopes that this material will help the student to get clear concepts of the Algorithms, program design with flowcharts. And also writing programs with C and C++ languages which enable the world progress to modern era with the emerging computing technologies.

CONTENTS

Block/Unit	Title	Page
BLOCK – I: FUNDAMENTALS OF C		1
Unit-1:	Program Design	3-11
Unit-2:	Evolution of C Language	12-18
Unit-3:	Basics of C Language	19-36
BLOCK – II: CONTROL STRUCTURES		37
Unit-4:	Flow of Control	39-51
Unit-5:	Functions	52-61
Unit-6:	Pointers and Strings	62-76
BLOCK – III: DERIVED DATA TYPES		77
Unit-7:	Arrays	79-91
Unit-8:	Structures and Unions	92-103
Unit-9:	Files	104-115
BLOCK – IV: INTRODUCTION TO C++		117
Unit-10:	Classes and Objects	119-143
Unit-11:	Inheritance	144-158
Unit-12:	Polymorphism	159-172
	Model Question Paper	173-174

BLOCK - I

FUNDAMENTALS OF C

This block gives an overall study on the writing algorithms on any kind of problems to plan the solution of the problems using step wise procedure. It also enables the students to open the C++ Editor, writing programs, compiling programs, executing programs. This block deals with program design with flowcharts in terms of graphical representation of the solution to the problem, and enable the students to write programs on variables, data types, constants, escape sequences, , statements, various operators in C such as arithmetic, assignment, relational, conditional, and type casting and type conversion, type coercion, library functions, and input and output..

The units included in the block are:

Unit-1: Program Design

Unit-2: Evolution of C Language

Unit-3: Basics of C Language

UNIT- 1: PROGRAM DESIGN

Contents

- 1.0 Objectives
- 1.1 Introduction
- 1.2 Definition and Examples of Algorithm
- 1.3 Flowcharts
- 1.4 Structured Programming
- 1.5 Summary
- 1.6 Check your progress – Model Answers
- 1.7 Model Examination Questions
- 1.8 Glossary

1.0 OBJECTIVES

After studying this unit, you should be able to

- understand concepts and examples of algorithms
- understand Various notations of flowchart
- explain structured programming with examples

1.1. INTRODUCTION

An algorithm is defined as “A precise rule (or set of rules) specifying how to solve some problem”. Algorithm is not related to any computer language but it is just the sequence of instructions which give clear idea on how to write the computer code. Every problem that can be solved using computer can be represented with an algorithm. Algorithms are generally language independent and hence, the algorithm can be implemented in any programming language. Flowchart can be used in representing process or planning in any area like business, education, hospitals. Old languages like BASIC (Beginner’s All Purpose Symbolic Instruction Code) and FORTRAN (Formula Translation) are examples of unstructured programs where the flow of steps are sequential as expected and can jump to an unexpected location with “goto” like statement. Structured languages use blocks of code (group of statements), control statements(decision making), iterative statements (repeating a block of code) and functions (module of code for reuse). All most all the modern languages like C, C++, Java, C#, Python etc. follow the structured programming approach.

1.2 DEFINITION AND EXAMPLES OF ALGORITHM

What is an algorithm?

The definition of an algorithm according to Webster dictionary is “A precise rule (or set of rules) specifying how to solve some problem”. Solution to a problem should be expressed in terms of unambiguous steps. The process of defining a solution in terms of discrete steps in a specific order to get a specific output is referred to as an algorithm. Algorithm is not related to

any computer language but it is just the sequence of instructions which give clear idea on how to write the computer code. Every problem that can be solved using computer can be represented with an algorithm. Algorithms are generally language independent and hence, the algorithm can be implemented in any programming language.

Characteristics of an Algorithm

- An algorithm has the following characteristics –
- Output: An algorithm should have *one or more* desired outputs
- Input: An algorithm may have *zero or more* well-defined inputs.
- Unambiguous –Each step of an algorithm should be clear and unambiguous.
- Finiteness: Algorithms should terminate after a finite number of steps.
- Language Independent: An algorithm should be independent of programming language.
- Writing Algorithms

Though there are no standards for writing algorithms, generally it is written between start and stop steps. Some may start it with the definition as “algorithm: <name> (input). The algorithm may contain basic code constructs like flow-control (if-else), loops (do, for, while) etc which are generally available in almost every language. The simplest example for an algorithm is addition of two numbers and display the result. Writing step numbers is optional and can be omitted if preferred so. Grouping of steps is represented with an indentation (pushed to right with tabs or spaces) so that all the steps indented with a level belong to the same group.

Step 1 “ START

Step 2 “ Declare three integers x, y, z

Step 3a – Read value and assign it to x

Step 3b – Read value and assign it to y

Step 4 “ Add values of x and y and assign it to z

Step 5 “ print z

Step 7 “ STOP

Here the assignment can be represented with a left arrow(\leftarrow) as $z \leftarrow x + y$

The input variables can be defined in the algorithm definition as well. Now the algorithm can be rewritten as:

Algorithm: add

Step 1 “ define x, y, z

Step 2a – $x \leftarrow$ read value

Step 2b - $y \leftarrow$ read value

Step 3 “ $z \leftarrow x + y$

Step 4 “ display z

Step 5 – STOP

Following section presents some basic algorithms which can be used in other larger algorithms as building blocks.

Example: Finding the maximum of two numbers given by user.

Start

Declare variables a and b

Read variables a and b

If $a > b$

 Display a is the maximum

Else

 Display b is the maximum

Stop

Example: Finding the maximum of three numbers given by user

Start

Declare variables a, b and c

Read variables a, b and c

If $a > b$

 If $a > c$

 Display a as maximum

 Else

 Display c as maximum

Else

 If $b > c$

 Display b as maximum

 Else

 Display c as maximum

Stop

Example: Finding the factorial of a number given by user.

Start

Declare variables n, f and i

Initialize variables

$f \leftarrow 1$

$i \leftarrow 1$

Read value of n

Repeat the next 2 steps until $i = n$

$f \leftarrow f * i$

$i \leftarrow i + 1$

Display f

Stop

Example: Finding roots of a quadratic equation $ax^2+bx+c=0$.

```

Start
Declare variables a, b, c, d, r1, r2, rp and ip;
 $d \leftarrow b^2 - 4ac$ 
If  $d \geq 0$ 
     $r1 \leftarrow (-b + \sqrt{d})/2a$ 
     $r2 \leftarrow (-b - \sqrt{d})/2a$ 
    Display r1 and r2 as roots.
Else
    Calculate real and imaginary parts
     $rp \leftarrow b/2a$ 
     $ip \leftarrow \sqrt{-d}/2a$ 
    Display roots  $rp + j(ip)$  and  $rp - j(ip)$ 
Stop

```

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. What is an algorithm?

.....

.....

.....

.....

1.3 FLOWCHARTS

Flowchart is another tool to explain the steps or process of a program. A flowchart can be defined as the “graphical or pictorial representation of an algorithm with the help of different predefined symbols, and arrows”. The program steps and sequence are represented with different basic graphic shapes like circle, oval, parallelogram, rectangle, diamond and arrows. While algorithms are generally used to represent a program, the flowchart can be used anywhere including program representation. Flowchart can be used in representing process or planning in any area like business, education, hospitals etc.

Symbols Used in Flowchart

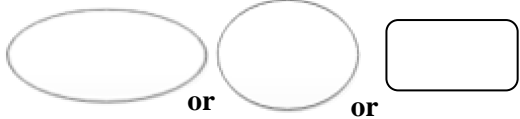



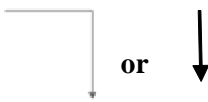
Symbol	Remarks
	Terminal - Start / End generally written inside this oval/circular, round rectangle
	Input / Output – What to read or what to output is written inside this parallelogram
	Process / Instruction steps are written inside this box
	Decision – The condition based on which a decision is to be taken is written inside this diamond
	Connector / Arrow – The flow from one step to another step or a group of step is connected through arrows

Table 1.1

Example: Print 1 to 10:

Algorithm: Print

Step 1: Start

Step 2: Initialize to 0

Step 3: Increment x by 1

Step 3: Print x

Step 4: If x is less than 10 then go to step 3

Step 5: Stop

Representation with Flowchart:

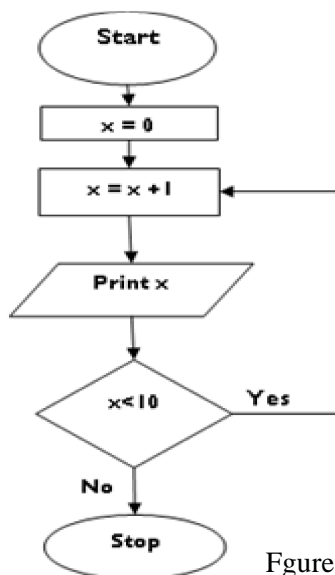


Figure 1.1

Program Design with Flowcharts:

Example: Finding Maximum of 3 numbers

The algorithm for finding maximum was given previously and its pictorial representation as flowchart is given below:

Flowchart:

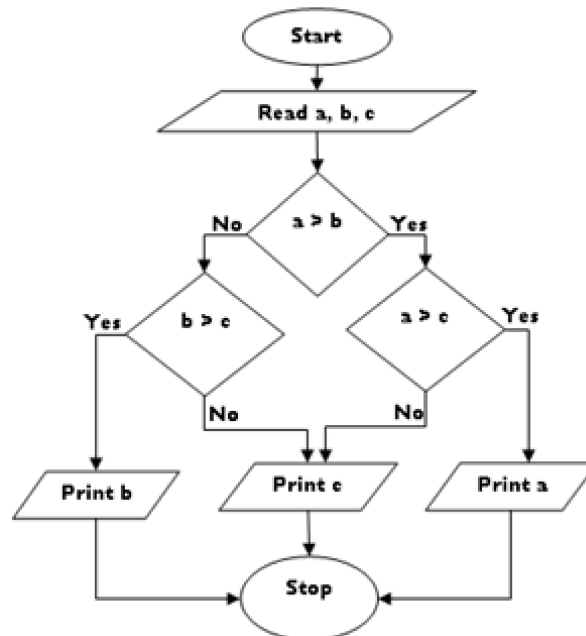


Figure 1.2

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

2. What is flowchart?

.....

.....

.....

1.4 STRUCTURED PROGRAMMING

Structured Programming Approach

Old languages like BASIC (Beginner’s All Purpose Symbolic Instruction Code) and FORTRAN (Formula Translation) are examples of unstructured programs where the flow of steps are sequential as expected and can jump to an unexpected location with “goto” like statement. Unstructured languages lack reusability and readability of the code, which are desirable features of any good programming language. A structured program can be defined as “a programming approach in which the program is made as a single structure”. It avoids the possibility of jumping to an arbitrary instruction inadvertently as is possible in unstructured languages. The instructions in this approach are executed in a well-defined sequential and structured manner. All most all the modern languages like C, C++, Java, C#, Python etc. follow the structured programming approach. Structured languages use blocks of code (group of statements), control

statements(decision making), iterative statements (repeating a block of code) and functions (module of code for reuse). A structured program has single entry point and terminates on exit. Structured programs are easier to read, understand, debug, maintain. Development of software also becomes easier and required less effort and time since code can be reused, less prone for errors and debugging in case of errors is easy.

Consider the following C program for finding the average of even numbers in an array of integers. The syntax and program are not important here and the focus is only on how a structured language looks like. However, reading comments (text after // in each line) will help in understanding the process.

```
#include <stdio.h>

int main(int argc, char *argv[])// Main method – Entry point
{
    int x[] = {1,2,3,4,5,6,7,8,9,10}; // Array of numbers
    int i, sum=0, cnt=0; // loop, sum and count variables
    float avg; // average variable
    int n = sizeof(x)/sizeof(int); // Find number of elements
    for(i=0; i<n; i++)// Iterate through the array variables
    {
        if(x[i]%2 == 0)// Check if remainder is 0 when divided by 2
        {
            sum += x[i];// Add to sum only if it is divisible by 2
            cnt++;// Increment count
        }
    }
    if(cnt > 0)// Check if there is at least one even numbers
    {
        avg = ((float)sum) /((float) cnt);// Compute average as real
        printf("Average is: %f\n", avg);
    }
    else// There are no even numbers in the array
    {
        printf("Average cannot be found if there are no elements\n");
    }
}
```

In the above program, each pair of curly braces ({}) represent a block. A block will be either executed or not executed depending on the condition. Within a block there can be other blocks (nested blocks). The main block contains declarations of some *variables* needed by the program, a *for* loop within which there is an ‘if’ block that checks whether the number is even, and if even, add it to sum and count is incremented by one. Then there is an ‘if’ block to print average if there is at least one number and an ‘else’ block that prints a message if there are no elements. 9
Once it reaches the end of main block, the program is terminated.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

3. What is structured programming?

.....

.....

.....

1.5 SUMMARY

An algorithm is A precise rule (or set of rules) specifying how to solve some problem. Though there are no standards for writing algorithms, generally it is written between start and stop steps. Some may start it with the definition as “algorithm: <name> (input). The algorithm may contain basic code constructs like flow-control (if-else), loops (do, for, while) etc which are generally available in almost every language. The simplest example for an algorithm is addition of two numbers and display the result. Writing step numbers is optional and can be omitted if preferred so. The program steps and sequence are represented with different basic graphic shapes like circle, oval, parallelogram, rectangle, diamond and arrows. While algorithms are generally used to represent a program, the flowchart can be used anywhere including program representation. Flowchart can be used in representing process or planning in any area like business, education, hospitals etc... Structured programs are easier to read, understand, debug, maintain. Development of software also becomes easier and required less effort and time since code can be reused, less prone for errors and debugging in case of errors is easy.

1.6 CHECK YOUR PROGRESS MODEL ANSWERS

1. A precise rule (or set of rules) specifying how to solve some problem
2. Graphical or pictorial representation of an algorithm with the help of different predefined symbols, and arrows
3. A programming approach in which the program is made as a single structure.

1.7 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Explain Algorithm with examples.
2. Describe the flowchart with an example.
3. What are the properties of structured programming?

II. Answer the following questions in about 15 lines each

1. Write an algorithm to find sum of numbers from 1 to n
2. Describe various symbols used in flowcharts.
3. Draw the flow chart to find the square of a given number

1.8 GLOSSARY

BASIC	:	Beginner's All Purpose Symbolic Instruction Code
FORTRAN	:	Formula Translation
COBOL	:	Common Business Oriented Language
SNOBOL	:	StriNg Oriented and symBolic Language
ALGOL	:	Algorithmic Language
Prolog	:	Programming in Logic.

UNIT- 2: EVOLUTION OF C LANGUAGE

Contents

- 2.0 Objectives
 - 2.1 Introduction
 - 2.2 History and features of C Language
 - 2.3 Structure and Execution of C program
 - 2.4 Summary
 - 2.5 Check your progress – Model Answers
 - 2.6 Model Examination Questions
 - 2.7 Glossary
-

2.0 OBJECTIVES

After studying this unit, you should be able to

- understand history and features of C language
 - explain how to write and execute c program
 - understand C language program
-

2.1 INTRODUCTION

C was developed by Dennis Ritchie in 1972 at the Bell Laboratories in the USA, where he was working on the development of the B language. This was the time when Thompson and Kernighan developed the Unix operating system and implemented it on a PDP 7 machine and wanted to move it to a PDP 11 machine. As C became popular, many versions emerged with different set of library functions. A good programming language should have certain qualities like simplicity, efficiency, compactness, well-structured etc. C is one of the first languages to have almost all good features. To avoid confusion, the ANSI (American National Standards Institute) standardized the language and any extensions can be made by individual vendors over and above the standards. In the Compilation stage, the preprocessed code is translated to human readable assembly instructions known as intermediate code which can be understood by the assembler specific to the machine.

2.2 HISTORY AND FEATURES OF C LANGUAGE

History of C Language

C was developed by Dennis Ritchie in 1972 at the Bell Laboratories in the USA, where he was working on the development of the B language. This was the time when Thompson and Kernighan developed the Unix operating system and implemented it on a PDP 7 machine and wanted to move it to a PDP 11 machine. They envisaged that the effort to port code to another machine should be minimized as the hardware is ever changing and improving. Therefore, they took a decision to write as much of the code as possible in a higher level language. They found that Ritchie was working on a language intended to undertake similar work.

After some modification to make it more suitable for systems work, C was used to write the first port of Unix. Ever since then, most of Unix has been written in C. Only specific small

portions of Unix system, which handles the hardware interface, is written in machine specific assembler.

C has become a popular language due to its simplicity, power, features and rich set of libraries. Nowadays, it is one of the main languages used for software development particularly in the areas of driver, network and system level programming.

As C became popular, many versions emerged with different set of library functions. To avoid confusion, the ANSI (American National Standards Institute) standardized the language and any extensions can be made by individual vendors over and above the standards. Hence, any program that sticks to ANSI standard C, is highly portable across platforms.

Why C Language?

A good programming language should have certain qualities like simplicity, efficiency, compactness, well structured etc. C is one of the first languages to have almost all good features like:

- Small set of keywords
- Strong input and output (referred to as I/O) capability
- Well-structured and modular
- Availability of supporting libraries
- User controllable memory management
- System level programming support
- Machine independence

These features make C one of the most popular languages for almost every application. However, because of its capability to access system resources like memory, storage etc., it is particularly popular with system programming in the areas of operating system, device drivers, network drivers, compilers, interpreters, embedded programming and system utilities.

Features of C language

- It is a robust (capable of handling the errors during execution and manage the incorrect input of data)
- Rich set of built-in functions and operators to express any complex equation/expression.
- Supports assembly language with features of a high-level language which is required by any system programming language
- C programs are efficient and run faster than many programs written in other languages due to its optimization of code and closeness to assembly language.
- C is almost portable which means that programs written on one machine can be run on another machine without modifications.
- Can be extended by way of libraries and defining new data types.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. Who developed C?

.....
.....
.....

2.3 STRUCTURE AND EXECUTION OF C PROGRAM

Structure of C Program

Every C program consists of one or more functions. Each of these functions performs its own tasks. The results generated by these tasks are passed between functions in several ways. A special function called main() should be declared which acts as the entry point for the entire set of functions. The execution of the program starts with the first line of this 'main' function and other functions are called from within main function directly or indirectly. There is no rule that the function main() should be the first function in a program.

Each statement within the block must be terminated by a semi-colon (;) known as "statement terminator". A block of related statements are enclosed within curly braces ({ }) which is known as compound statements. In C semi colon terminates the statements and hence a statement can span in more than one line, or many statements can be written in a single line.

Structure of a function

Each function of a C program consists of: A heading which includes the function name, list of arguments passed to that function and return data type declaration. A block that contains local variable definitions, statements which may include calls to other functions and may contain return statement.

For example, consider a function that adds two integers and returns the sum as an integer.

```
int add(int a, int b)
{
    int sum;
    sum = a+b;
    return sum;
}
```

Here the function add takes two integers a and b as parameters and inside the function it computes sum as a+b and then it returns the sum.

Once the function structure is clear, the structure of a c program can be given as follows:

Include Statements

Global Definitions

```
main ( )
```

```
{
```

```
    Statements;
```



```

}
function1 ( )
{
    Statements;
}
.....

```

Phases of Execution of a C Program

A program in any high level language is written as a simple text file with appropriate extension. Then it will be compiled to generate machine code which can be run by the user as a command. C also follows the same process but it has several stages before the final compilation takes place. The compilation and execution of a C program is done in four separate stages:

- Preprocessing
- Compilation
- Assembly
- Linking

Consider the simple C program that prints a message

```

#include <stdio.h>
intmain(void)
{
printf(“Hello, World!”);
return0;
}

```

In the above code, content within `/*` and `*/` is treated as a comment and acts as description for the reader, but not to the computer. Any line that starts with `#` symbol is treated as a preprocessor directive. The ‘main’ is the entry point for any program. The curly braces (`{ }`) denotes the block of code and the statements are written within a block. When this program is compiled, it follows the four stages of the compilation process

Preprocessing: In this stage, the text file is parsed and all comments are stripped, broken lines are joined etc. Then lines that start with a `#` character are interpreted by the preprocessor and appropriate modifications are made to the code. Preprocessing is a simple macro language with a small set of syntax and semantics, which helps in reducing code repetition. In this case the `#include` instructs the preprocessor to include the contents from ‘`stdio.h`’ file into the present source file at this location.

Compilation: In this stage, the preprocessed code is translated to human readable assembly instructions known as intermediate code which can be understood by the assembler specific to the machine.

Assembly: In this stage, an assembler is used to translate the assembly instructions to binary object code which consists of actual instructions to be run by the target processor.

Linking: The object code consists of machine instructions given in the code, but it doesn’t contain code from the libraries used in the program and some addresses may not have referenced properly or the order of code may not be proper (like a referenced function may be defined after it has been referenced). The linker will rearrange the code after adding the instructions

from library functions used by the program. In the example, the linker adds the code for printf function.

First C Program:

```
/*  
 * First C program that prints Hello, World  
 */  
#include <stdio.h>  
int main(void)  
{  
    printf("Welcome to C Programming"); // Print the message  
    return 0; // Terminate the program  
}
```

The above simple program that prints just a message “Welcome to C Programming”, has many constructs that give glimpses of a C program.

First, C has two types of comments, one is known as block comment (also known as multi line comment), surrounded with `/*` and `*/` and the other is line comment which comments all the text from the `//` to the end of that line. Comments can be placed anywhere in the code. These comments improves the readability of the program, but during the preprocessor stage, these are removed from the actual code before actual compilation starts.

Next there is a preprocessor directive that instructs the preprocessor to include all the text from the file “stdio.h”. The angular brackets around “stdio.h” indicate that the file resides in the standard header files directory. If angular brackets are replaced with double quotes (`#include “stdio.h”`) then, it searches for this file in the local directory first.

Next there is a function named as “main”, which is the entry point to the program. It takes no arguments (indicated by void inside the parenthesis) but returns an integer.

Next there is only one block of code for the main, which has two lines in it. The first line uses a built-in function printf, which is defined in stdio.h file, with the message as argument/parameter. The second line simply returns a zero to the shell that runs the program. After the execution of the last line of main block, the program terminates automatically.

Running the Program

If this program is run on Unix/Linux machine, simply issue the command

```
cc hello.c
```

This creates the output file “a.out” which is executable. To run a.out issue the following command:

```
./a.out
```

This produces the “Welcome to C Programming” message on the screen.

Note: On windows machine, GNU C Compiler is recommended for better portability. There is a myriad of IDEs (Integrated Development Environment) for Windows and Linux like NetBeans, Eclipse, Microsoft Visual Studio, Dev-C++ and so on. All these IDEs offer an editor for code editing and compile and run facility from within the IDE. If somebody is interested in the discontinued 16-bit Turbo C compiler from Borland (not recommended) then the Dev-C++ compiler is the closest to it (<https://www.bloodshed.net/dev/devcpp.html>).

Following is a simple program that illustrates the structure of a C program that has more than one function.

```
/* Example program to show basic structure. */
#include <stdio.h>
main ( )
{
    message ( );
}
message ( )
{
    printf ("HelloWorld!!! \n");
}
```

This program has two functions, viz., main() and message(). The entry point is main() where the first line is a call to message() function. So the control goes to message function, where the first line is a call to the standard library function printf(), which takes the message string as parameter and prints it on the screen. Then the printf() returns control back to message where it returns to main() since there are no more statements to execute. The main also terminates at this point since it has no more statements to execute. The '\n' in the message moves the cursor to the next line after printing "Hello World!!!" message. If this new line character (\n) is missing, the next output will continue in the same line after the message.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

2. Write the phases of compiling C program.

.....
.....
.....

2.4 SUMMARY

C was developed by Dennis Ritchie in 1972 at the Bell Laboratories in the USA, where he was working on the development of the B language. This was the time when Thompson and Kernighan developed the Unix operating system and implemented it on a PDP 7 machine and wanted to move it to a PDP 11 machine. C has become a popular language due to its simplicity, power, features and rich set of libraries. Nowadays, it is one of the main languages used for software development particularly in the areas of driver, network and system level programming. As C became popular, many versions emerged with different set of library functions. To avoid confusion, the ANSI (American National Standards Institute) standardized the language and any extensions can be made by individual vendors over and above the standards. Hence, any program that sticks to ANSI standard C, is highly portable across platforms. A program in any high level language is written as a simple text file with appropriate extension. Then it will be compiled to generate machine code which can be run by the user as a command. C also follows the same process but it has several stages before the final compilation takes place.

The compilation and execution of a C program is done in four separate stages known as Pre-processing, compiling, assembling, and linking. C has two types of comments, one is known as block comment (also known as multi line comment), surrounded with /* and */ and the other is line comment which comments all the text from the // to the end of that line. Comments can be placed anywhere in the code. These comments improves the readability of the program, but during the preprocessor stage, these are removed from the actual code before actual compilation starts.

2.5 CHECK YOUR PROGRESS MODEL ANSWERS

1. C was developed by Dennis Ritchie in 1972 at the Bell Laboratories in the USA
 2. Pre-processing, compilation, assembly, linking
-

2.6 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Explain various features of C language.
2. Describe structure of C program.
3. Explain the phases of Compiling C program?

II. Answer the following questions in about 15 lines each

1. Describe history of C language
 2. Describe comment lines in C.
 3. Explain Execution of C program with an example
-

2.7 GLOSSARY

Compiler	:	Translates source program into object program
Assembler	:	Translates source program into assembly language
Linker	:	Links pre-compiled library files to source code
Loader	:	Loads linked files into the processor for execution
.h file	:	C header files which is pre-compiled
main()	:	Main function which is executable by C directly.

UNIT- 3: BASICS OF C LANGUAGE

Contents

- 3.0 Objectives
 - 3.1 Introduction
 - 3.2 Character Set, Identifiers and Keywords, Variables
 - 3.3 Data Types, Escape Sequences, Statements
 - 3.4 Operators and Library Functions in C
 - 3.5 Input and Output
 - 3.6 Summary
 - 3.7 Check your progress – Model Answers
 - 3.8 Model Examination Questions
 - 3.9 Glossary
-

3.0 OBJECTIVES

After studying this unit, you should be able to

- understand character set, identifiers and keywords, variables in C
 - describedata types, escape sequences, statements in C language
 - explain how to work with various operators and library functions of C
 - understand various input and output statements in C
-

3.1 INTRODUCTION

C and C++ Languages uses the uppercase letters A to Z, the lowercase letters a to z, the digits 0 to 9, and certain special characters as building blocks to form basic program elements. In C, Identifiers are names given to various program elements, such as variables, constants, functions and arrays. An identifier should start with a letter (A-Z, a-z) or an underscore (_) followed by any number of letters and digits, in any order. Generally lowercase letters are used in C for variables and functions, though not mandatory. C is a case sensitive language. C supports several data types, to represent real world data in computer. Each data type requires a specific number of bytes in memory (size). Broadly the data can be classified as whole numbers (integers), fractional or real numbers, characters and series of characters (strings). C directly supports all these data types except strings. Strings are supported as a special case of array of characters. Constants are variables which cannot be altered, which can be integer, float, char and string. Characters below ASCII value 32 are non-printable control characters. If a string contains any of these values including double quote, apostrophe, question mark or backslash, it should be preceded by a backslash to escape it from interpretation which is known as escape sequence. An escape sequence always begins with a backslash and is followed by one or more special characters.

3.2 CHARACTER SET, IDENTIFIERS AND KEYWORDS, VARIABLES

Any language consists of certain alphabets, symbols, words, rules and conventions to follow. C is no exception for it. It has its own set of characters, variable name rules, data types, keywords, and so on. Following subsections describe these items.

The C Character Set

C uses the uppercase letters A to Z, the lowercase letters a to z, the digits 0 to 9, and certain special characters as building blocks to form basic program elements (e.g. constants, variables, operators, expressions). The special characters are listed below which have different meanings in the C language.

!	* +	\	“	>	#	(=	:	{	>
%) ~	;	}	/	^	-	[:	,	?
&	_]	‘	.	(space)						

Most versions of the language also allow certain other characters, such as @ and \$, to be included within strings (sequence of characters like a name) and comments.

Identifiers and Keywords

Identifiers are names given to various program elements, such as variables, constants, functions and arrays. An identifier should start with a letter (A-Z, a-z) or an underscore (_) followed by any number of letters and digits, in any order. Generally lowercase letters are used in C for variables and functions, though not mandatory. C is a case sensitive language, i.e., an uppercase letter is not equivalent to the corresponding lowercase letter. If the identifier name consists of many words, an underscore may be used to separate words for better clarity.

Examples: The following names are valid identifiers.

X	y12	sum_1	_temperature	this_is_a_very_long_Variable_Name
Names	area	tax_rate	TABLE	

The following names are not valid identifiers for the reasons stated.

4_th	the first character must be a letter
“x”	illegal characters (“
order-no	illegal character (-)
error flag	illegal character (space)

An identifier can be arbitrarily long but the significant length depends on the compiler (The ANSI standard recognizes 31 characters, though some recognize only the first eight characters). Additional characters are carried along for understandability and convenience of the programmer.

There are certain reserved words, called keywords that have standard, predefined meanings in C. These keywords cannot be used as user defined identifiers.

The standard keywords are

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	While

Some compilers may also include some or all of the following keywords:

ada	asm	entry	far	fortran	huge	near	pascal
-----	-----	-------	-----	---------	------	------	--------

Note that the keywords are all lowercase. Since uppercase and lowercase characters are not equivalent, it is possible to use an uppercase keyword as an identifier. Normally, however, this is not done; it is considered a poor programming practice.

Variables

A variable is an identifier used to represent a single data item of some specified type of data within a block of the program. A value is assigned to the variable at some point in the program and it can then be accessed or modified later in the program by referring to the variable name.

Declarations

A declaration consists of a data type, followed by one or more variable names, an optional value assigned to each variable, ending with a semi colon. Each array variable must be followed by a pair of square brackets, containing a positive integer which specifies the size (number of elements).

```
int a, b=20, c;
```

```
float root1, root2=1.234;
```

```
char flag, text[80];
```

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. List any four keywords in C.

.....

.....

.....

3.3 DATA TYPES, ESCAPE SEQUENCES, STATEMENTS

Data Types

C supports several data types, to represent real world data in computer. Each data type requires a specific number of bytes in memory (size). Broadly the data can be classified as whole numbers (integers), fractional or real numbers, characters and series of characters (strings). C directly supports all these data types except strings. Strings are supported as a special case of array of characters. Since each data type takes a fixed number of bytes in memory, the maximum value it can represent as binary value is $2^n - 1$ for integers and characters, where n is the number of bits. A character in C is internally represented as its ASCII (American Standard Code for Information Interchange), where each character is given a number between 0 and 127. Hence, a character can be used as a very small number in C which takes only one byte.

Integers are by default signed which means both negative and positive numbers can be represented. To represent the sign, the first bit of the data is taken and hence only $n-1$ bits are available for data. This essentially reduces the maximum value to half but it can be positive or negative. Size of memory required for an integer in C is compiler dependent which may take 2 or 4 bytes on 16 bit and 32 bit compilers respectively, which makes a C program non-portable and hence it can be explicitly declared as long (4 bytes always) or short (2 bytes always) by prefixing "long" or "short" keyword to "int" with a space in between. To represent only unsigned data (like count of people) the keyword "unsigned" can be prefixed to "int" data type with a space between these two key words. A proper modifier may be used before the declaration to avoid loss of data. Further, signed and unsigned are applicable only to integers and hence, int keyword may be omitted. For example consider the following where declarations in each line are equivalent.

int, signed int, signed
 unsigned int, unsigned
 signed long int, long int, signed long, long
 signed short int, short int, signed short, short
 unsigned long int, unsigned long
 unsigned short int, unsigned short

The basic data types along with memory requirement and range of values that can be represented with each type are given below.

Data Type	Description	Typical memory Requirements	Value Range
int	integer quantity	2 or 4 bytes (compilerdependent)	-32768 to 32767 (2 byte)- 2147483648 to 214743648 (4 byte)
short	Integer quantity	2 bytes	-32768 to 32767 (unsigned)
long	Integer quantity	4 bytes	-2147483648 to 214743648
unsigned	Integer quantity	2/4 bytes	0 to 65535 (2 byte) 0 to 4294967295 (4 bytes)
char	single character	1 byte	-128 to +127
float	floating number	4 bytes	
double	double-precision floating number	8 bytes	-1.7×10^{308} to $+1.7 \times 10^{308}$ with up to 16 decimal digits accuracy

Table 3.1

Constants

Constants are variables which cannot be altered, which can be integer, float, char and string. Moreover, there are several different kinds of integer and floating-point constants, as discussed below.

Integer and floating-point constants represent numbers which are generally referred to as numeric constants. The following rules apply to all numeric-type constants.

Commas and blank spaces cannot be included within the constant.

The constant can be preceded by a minus (-) sign if desired.

The value of constant cannot exceed specified minimum and maximum bounds which depend on compiler.

Integer Constants

An integer constant is an integer-valued number that consists of a sequence of digits. Integer constants can be written in three different number systems: decimal (base 10), octal (base 8) and hexadecimal (base 16). If a number starts with a zero, it is treated as octal number and if a number starts with 0x or 0X, it is treated as hexadecimal number. A suffix of L or l represents a long number, F or f represents a float number.

Example: Some valid numeric constants are shown below.

0 1 2.3f 23L -93 743 528l 32767 0X9A3E
0102

The following decimal constants are written incorrectly for the reasons stated.

12,245	illegal character (,)
36.0	illegal character (.)
10 2 0 30	illegal character (spaces)
192-123	illegal character (-)

Numerical Accuracy

Integer constants are exact quantities but floating-point constants are approximations due to its representation in binary format. Hence, the floating-point constant 1.0 might be represented in memory as 0.99999999 even though it might appear as 1.0 when it is displayed. Therefore, floating-point values cannot be used for certain purposes where exact values are required. For example, $x=1.0$, $y=1.0/1.0$; both values are 1.0 only. However, the expression $(x == y)$ may or may not result in true, since, the representation may be different for both internally. Hence, float numbers are never checked for equality but checked for their difference, for example like in $(x-y < 0.0001)$, where the accuracy required is .0001.

Character Constants

A character constant is a single character, enclosed in apostrophes (single quotes).

Example: Some character constants are shown below.

'A' 'x' '3' '\$' ''

The last constant consists of *space* enclosed in apostrophes.

String Constants

A string constant consists of any number of consecutive characters (including none) enclosed in (double) quotation marks.

Example: Some string constants are shown below.

"dog" "this is a test string" "street 12-13-24" "Line 1\nLine 2" ""

The fourth example above is printed in two lines since it contains a `\n` character, which moves the cursor to the next line before printing "Line2".

Boolean Constants

In C, value 0 (zero) is considered as false and any other value is considered as true. For example, following are some values for true: -1, -23456, 1, 12, 1222 ...

Symbolic Constants

Symbolic constants are usually defined at the beginning of a program and then may be used anywhere in the program in place of the numeric constants, character constants, and so on, that the symbolic constants represent. A symbolic constant is defined by writing.

```
#define name text
```

where *name* represents a symbolic name, typically written in uppercase letters, and *text* represents the sequence of characters associated with the symbolic name.

For example,

```
#define MAX 100
```

defines 100 as MAX. Wherever MAX is used, the compiler substitutes 100. Any change in the value of MAX will automatically reflect throughout the program.

Escape Sequences

Characters below ASCII value 32 are non-printable control characters. If a string contains any of these values including double quote, apostrophe, question mark or backslash, it should be preceded by a backslash to escape it from interpretation which is known as escape sequence. An escape sequence always begins with a backslash and is followed by one or more special characters. For example, a line feed (newline) is represented as \n. Such escape sequences always represent single characters, even though they are written in terms of two or more characters.

The commonly used escape sequences are listed below.

Character	Escape Sequence	ASCII Value
Bell (alert)	\a	007
Backspace	\b	008
Horizontal tab	\t	009
Vertical tab	\v	011
Newline (line feed)	\n	010
Form feed	\f	012
Carriage return	\r	013
Quotation mark(“)	\"	034
Apostrophe (‘)	\’	039
Question mark (?)	\?	063
Backslash (\)	\\	092
Null	\0	000

Table 3.2

Whitespace Characters: These are space (‘ ’), newline(‘\n’), carriage return(‘\r’) and tab(‘\t’). These characters except the space are nonprintable, but their effect can be felt when printed. The carriage return moves the cursor to the first character of the same line and hence, it overwrites the line. The newline character moves the cursor to the next line but will not move the current location to the first character and hence, produces a staircase effect. The handling of \n and \r are system dependent. Whitespace characters generally terminates the reading of strings in *scanf* function when %s is used.

Statements

A statement causes the computer to carry out some action. There are three different classes of statements in C. They are expression statements, compound statements and control statements.

An expression statement consists of an expression followed by a semicolon. The execution of an expression statement causes the expression to be evaluated. Some examples for expression statements are shown below.

```

a = 3;
c = a + b;
++i;
printf ("Area = %f", area );
// Empty or blank statement

```

Expressions:The expression consists of a single entity, such as a constant, a variable, an array element or a reference to a function. It may also consist of some combination of such entities interconnected by one or more operators. Some simple expressions are shown below.

```

a + b
x = y
c = a + b
x <= y
x == y
++i

```

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

2. Define a statement.

.....

.....

.....

3.4 OPERATORS AND LIBRARY FUNCTIONS IN C

C Language provides various operators to perrom the different operations based on the situations.

Arithmetic Operators

There are five arithmetic operations in C. They are

+ (addition), - (subtraction), * (multiplication), / (division), % (reaminder or modulus)

If a and b are two integer variables whose values are 10 and 3, respectively, then the following are true:

Expression	Value
$a + b$	13
$a - b$	7
$a * b$	30
a / b	3
$a \% b$	1

Table 3.3

Unary Operators

C has some operators which act upon a single operand like +, -, ++, --. The + and - operators can be used either as sign or as arithmetic operators as in +10, -20, 10+20, 4-7 etc. The increment or decrement operators (++ and --) increment or decrement the current value by one which is a short form for a = a+1. The ++ and - operators can be either on the left side or on the right side of the operand, which has different meanings when used in an expression. When the ++ precedes the operand, it conveys the meaning “increment first, and then use” and when it is suffixed to the operand it conveys the meaning “first use and then increment”. The expression b = a++; is equivalent to b = a; a = a+1; Similarly the expression b = ++a; is equivalent to a = a+1; b = a;

Following are some of the examples which explain the use of these operators.

```
int a = 10; // Declare and initialize a variable to 10
```

```
int b; // declare another variable
```

```
a++; // increment a by one which is equivalent to a= a+1 and value of a is 11 now
```

```
++a; // increment a by one which is equivalent to a=a+1 and value of a is 12 now
```

```
b=a++; // here a is assigned to b first and then a is incremented. Now b = 12 and a = 13
```

```
b = ++a; // Here a is incremented and then assigned to b. Now b = 14, a = 14
```

Relational and Logical Operators

Various relational operators available in C are:

< (less than) <= (less than or equal to) > (greater than)
 >= (greater than or equal to) == (equal to) != (not equal to)

The == and != operators fall into a separate precedence group, beneath the other relational operators. These operators also have a left-to-right associativity.

These operators are used to form logical expressions representing conditions that are either true (non-zero) or false (0).

Suppose that i=1, j=2 and k=3 are three integer variables. Several logical expressions involving these variables are shown below.

Expression	Interpretation	Value
i < j	true	1
(i + j) >= k	true	1
(j + k) > (i + 5)	false	0
k != 3	false	0
j == 2	true	1

Table 3.4

In addition to the relational and equality operators, C contains three logical operators. They are:

&& (logical and) || (logical or) ! (logical not)

The logical operators act upon operands that are themselves logical expressions. The net effect is to combine the individual logical expressions into more complex conditions that are either true or false. The result of a logical ‘and’ operation will be true only if both operands are true, whereas the result of a logical ‘or’ operation will be true if either or both operands are true.

The result of a logical ‘not’ operator is true if the result of the expression is false, and vice versa. Any nonzero value is interpreted as true.

Suppose that $i=7, j=5$ and $k=10$. Some complex logical expressions that make use of these variables are shown below.

Expression	Interpretation	Value
$(i \geq 6) \ \&\& \ (j == 5)$	true	1
$(i \geq 5) \ \ (k == 18)$	true	1
$(i < 11) \ \&\& \ (j > 100)$	false	0
$i != 10 \ \&\& \ j < 10 \ \&\& \ k > 5$	true	1
$!i$	false	0
$!(i > 30)$		true

Table 3.5

Assignment Operators

There are several different assignment operators in C. All of them are used to form assignment expressions, which assign the value of an expression to an identifier. The most commonly used assignment operator is $=$. Assignment expressions that make use of this operator are written in the form $identifier = expression$

Here are some typical assignment expressions:

```
a = 3;
x = y;
area = length * width;
delta = 0.001;
sum = a + b;
```

C permits multiple assignments of the form $identifier\ 1 = identifier\ 2 = \dots = expression$

Here the assignments are carried out from right to left. Thus, the multiple assignment $i = j = 5$ assigns 5 to j and then j to i .

C contains five additional assignment operators $+=$, $-=$, $*=$, $/=$ and $\%=$.

These are really short hand notation for

$\langle expression1 \rangle = \langle expression1 \rangle \langle operator \rangle \langle expression2 \rangle$

Usually, $expression1$ is an identifier, such as a variable or an array element.

$x += 1$ is equivalent to $x = x + 1$, $x /= 3$ is equivalent to $x = x / 3$

Operator precedence

Let $x = 10$ and $y = 5$ and now consider the expression $a = x + y * 5 + x$;

This can be evaluated in many ways as $(x+y) * (5+x)$ or $x + (y*5) + x$ or $((x+y)*5) + (x)$ etc.

Expressions in C are evaluated according to their precedence defined by C language. The above example is evaluated as $x + (y*5) + x$ since a $*$ is given higher precedence than $+$. Parentheses have the highest precedence and hence, C first evaluates the values within parentheses first. Most of the operators have left to right associativity meaning that they are associated from left to right, but some have right to left like $=$ where the right value is assigned to left variable.

Following is the precedence and associativity of the various C operators.

Operator Category	Operators	Associativity
unary operators	- ++ -- ! sizeof (type)	R → L
arithmetic multiply, divide and remainder	* / %	L → R
arithmetic add and subtract	+ -	L → R
relational operators	<<=>> =	L → R
equality operators	== !=	L → R
logical and	&&	L → R
logical or		L → R
assignment operators	= += -= *= /= %=	R → L
comma operator	,	L → R

Table 3.6

Operators with same precedence (within the same group) are processed according to their associativity. (Left to Right or Right to Left)

The Conditional Operators

Simple conditional operations can be carried out with the conditional operator (?:) which is also known as ternary operator. An expression that makes use of the conditional operator is called a conditional expression. Such an expression can be written in place of the more traditional *if-else* statement,

A conditional expression is written in the form

expression 1 ? expression 2 : expression 3

Here the value depends of the logical expression1. If expression1 is evaluated to true, expression2 is returned, otherwise, expression3 is returned. This is equivalent to

if(expression1 == true) return expression2 else return expression3;

For example, *min = (i < j) ? i : j;* assigns min=i if i < j, otherwise min=j

Type Casting and Type Coercion

The value of an expression can be converted to a different data type if desired. To do so, the expression must be preceded by the name of the desired data type, enclosed in parentheses; i.e.(data type) expression

This type of conversion from one data type to another type is known as type casting.

If no casting is done explicitly, C can automatically convert the data if necessary (called coercion)

For example,

```
int i;
```

```
float j;
```

```
i=1.23; /* automatically coerce the value 1.23 in to an integer 1 */
```

```
j=1; /* automatically converts inter 1 to float 1.0f */
```

```
i = (int) j; /* implicit type casting which assigns the integer part 1 to i */
```

Enumerations

An enumeration is a data type, where its members are constants that are written as identifiers, though they have signed integer values. These constants represent values that can be assigned to corresponding enumeration variables.

In general terms, an enumeration may be defined as

```
enum tag {member1, member2, ..., member m};
```

Where enum is a required keyword, tag is a name that identifies enumerations having this composition, and member 1, member 2 represents the individual identifiers that may be assigned to variables of this type (see below).

Example : Consider the enumeration defined in Example.

```
enum colors {black, blue, green, red, white};
```

The enumeration constants will represent the following integer values:

(black = 0, blue = 1, green = 2, red = 3 and white = 4

These automatic assignments can be overridden within the definition of the enumeration by explicitly assigning a value to it. Those constants that are not assigned explicit values will automatically be assigned values which increase successively by 1 from the last explicit assignment. This may cause two or more enumeration constants to have the same integer value.

Example:

```
enum colors {black = -1, blue, cyan, green, magenta, red = 2, white, yellow};
```

The enumeration constants will now represent the values (black=-1, blue=0, cyan=1, green=2, magenta=3, red=2, white=3, yellow=4)

To use these enumerations, an enumeration variable should be declared first and it can be assigned only those values from within the enumeration definition.

```
enum <enumeration type><identified>;
```

```
enum colors cl = red; // declares cl as colors type and assigns a value red to it
```

Library Functions

The C language comes with a very rich set of standard library functions that carry out various commonly used operations or calculations. Though these library functions are not a part of the language specification, every implementations of the language include them. Library functions are organized as sets according to their functionality and their definitions are given in header files generally have a “.h” extension.

A library function is accessed simply by writing the function name followed by a pair of parenthesis. If the function needs some information, the information is passed to the function by way of arguments. The arguments must be placed within the parentheses separated by commas. The arguments can be constants, variable names, or more complex expressions. A function that returns a data item can appear anywhere within an expression in place of a constant or an identifier (i.e., in place of a variable or an array element).

In order to use a library function it may be necessary to define certain constants and the function might have to be defined ahead of its usage. The information required to use a library function is placed in a text file called header file generally has an extension “.h”. These header files are placed in a specific folder known to the C compiler and depends on the operating system and compiler used. Hence, to use a function, it is common to include its header file in the main portion of the program. This is accomplished with the preprocessor statement *include*. The

syntax is:

`#include <filename>` where filename represents the name of a special header file.

Some commonly used functions are given in the following table. The column labeled “type” refers to the data type of the returned value from function. The *void* shown indicates that nothing is returned by this function. Hundreds of functions are supplied by all C compilers as Standard library functions organized into different library files. The C reference manual gives a list of all these functions and their usage.

Some Commonly Used Library Functions

Function	Type	Purpose
abs (i)	int	Return the absolute value of i.
ceil (d)	double	Round up to the next integer value (the smallest integer that is greater than or equal to d)
cos (d)	double	Return the cosine of d
cosh (d)	double	Return the hyperbolic cosine of d
exp (d)	double	Raise e to the power d (e=2.7182818... is the base of the natural (Naperian) system of logarithms)
fabs (d)	double	Return the absolute value of d
floor (d)	double	Round down to the next integer value (the largest integer that does not exceed d)
fmod (d1, d2)	double	Return the remainder (i.e. the noninteger part of the quotient) of d1/d2 with same sign as d1
getchar ()	int	Enter a character from the standard input device.
log (d)	double	Return the natural logarithm of d
pow (d1,d2)	double	Return the d1 raised to the d2 power
printf(...)	int	Send data items to the standard output device (arguments are complicated)
putchar (c)	int	Send a character to the standard output device
rand ()	int	Return a random positive integer
sin (d)	double	Return the sine of d
sqrt (d)	double	Return the square root of d
srand (u)	void	Initialize the random number generator
scanf(...)	int	Enter data items from the standard input device (arguments are complicated)
tan (d)	double	Return the tangent of d
toascii (c)	int	Convert value of argument to ASCII number
tolower (c)	int	Convert letter to lowercase
toupper (c)	int	Convert letter to uppercase

Table 3.7

Note: c denotes a character-type argument, i denotes an integer argument, d denotes a double-precision argument and u denotes an unsigned integer argument

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

3.What is an unary operator?

.....
.....
.....

3.5 INPUT AND OUTPUT

C provides a powerful set of input and output functions, which can do almost everything a program demands. An input/output function can be accessed from anywhere within a program by calling the appropriate function along with its arguments enclosed in parentheses. These functions may be used either as an expression, if they return something (e.g., `c = getchar();`), or as statements (e.g., `scanf(...);`).

Single Character Input

The `getchar` function is a part of the standard C language I/O library. It returns a single character from a standard input device (generally a keyboard). The function does not require any arguments. In general terms, a reference to the `getchar` function is written as

`character variable = getchar ();`

where `character variable` refers to some previously declared character variable.

Example:

```
char ch;  
ch = getchar();
```

Generic Input Data

Any type of data can be fed to the computer from a standard input device using the C standard library function `scanf()`. This function can be used to enter any combination of numerical values, single character and strings. The function returns the number of data items that have been entered successfully.

In general terms, the `scanf` function is written as

`scanf (control string, agr1, arg2, ..., argn)`

Where `control string` refers to a string containing certain required formatting information, and `arg1, arg2, ..., argn` are arguments that represent the addresses of individual input data items. The control string comprises the pattern of input that contains literals and variables represented by a % sign followed by data type representation (known as conversion characters). Frequently used conversion characters that follow the % symbol are listed in the following table.

Conversion Character	Meaning
c	Data item is a single character
d	Data item is a decimal integer
e	Data item is a floating-point value
f	Data item is a floating-point value
g	Data item is a floating-point value
h	Data item is a short integer
i	Data item is a deciman, hexadecimal or octal integer
o	Data item is an octal integer
s	Data item is a string followed by a whitespace character (the null character \0 will automatically be added at the end)
u	Data item is an unsigned decimal integer
x	Data item is a hexadecimal integer
[...]	Data item is a string which may include whitespace characters

Table 3.8

The arguments following the control string should match in number and data type of the conversion strings. Each variable name must be preceded by an ampersand (&) if it is not an array or pointer.

Example: Here is a typical application of a *scanf* function.

```
#include <stdio.h>

main () {
char item [20];
int partno;
float cost;
scanf ("%s %d %f", item, &partno, &cost);
....
}
```

This example reads a string with no white spaces in between, an integer and a float number from keyboard. The string should not contain any white space character. Note that the ampersand is missing for item, since it is an array of characters.

Output Function

Output data can be written from the computer onto a standard output device (typically a screen) using the library function *printf*. This function can be used to output any combination of numerical values, single characters and strings. It is similar to the input function *scanf*, except that it displays data on screen instead of reading from keyboard.

In general terms, the *printf* function is written as

```
printf (control string, arg1, arg2, ....., argn)
```

Where control string is the format information that contains literals and variable represented with %xxx, and arg1, arg2 ..., are arguments that represent the individual output data items corresponding to the %xxx conversion strings. The arguments can be constants, single variable or array names, or more complex expressions. In contrast to the scanf function the arguments in a printf function are not memory addresses and therefore they are not preceded by ampersands.

Some commonly Used Conversion Characters for Data Output are given below

Conversion Character	Meaning
c	Data item is displayed as a single character
d	Data item is displayed as a signed decimal integer
e	Data item is displayed as a floating-point value without an exponent
f	Data item is displayed as a floating-point value without an exponent
g	Data item is displayed as a floating-point value using either e-type or f-type conversion, depending on value; trailing zeros, trailing decimal point will not be displayed.
i	Data item is displayed as a signed decimal integer
o	Data item is displayed as an octal integer, without a leading zero
s	Data item is displayed as string
u	Data item is displayed as an unsigned decimal integer
x	Data item is displayed as a hexadecimal integer, without the leading 0x.

Table 3.9

A width modifier can be added before any of these characters to give width to use.

For example,

%4d prints the number in a width of 4 characters, right justified

%-4d prints the number in a width of 4 characters left justified

%04d prints the number in a width of 4 characters padded with preceding zeros if necessary

%10s prints the string in a width of 10 characters

%3.3s prints the string in exactly 3 characters, trimming it if longer

%5.2f prints a float number in a width of 5 characters, out of which 2 are for decimal, one for the dot.

In all the cases, if the width of data is more than the width specified, the width is overridden.

Example: Here is a simple program that uses the printf function.

```
#include <stdio.h>
#include <math.h>
main () /* print several floating point numbers */
{
    float i = 2.0, j = 3.0;
    printf ("Values: %f, %f, %f, %7.3f", i, j, i + j, sqrt ( i + j));
}
```

Example: Reading and Writing a Line of text

Following program reads a line of text and then writes it back, just as it was entered. The program illustrates the syntactic differences in reading and writing a string that contains a variety of characters, including whitespace characters.

```
# include <stdio.h>

main ( )          /* read and write a line of text */
{
    char line [80];
    scanf ("%^[^n]", line) ;
    printf ("%s", line);
}

```

The %^[^n] format specifies that all characters until it is not new line should be read. Similarly %[abcd] reads only one of out of the set a,b,c and d.

Commonly used format modifiers/flags are listed below:

Modifier	Meaning
-	Data item is left-justified within the field (blank spaces required to fill the minimum field width will be added after the data item rather than before the data item)
+	A sign (either + or -) will precede each signed numerical data item; without this flag, only negative data items are preceded by a sign
0	Causes leading zeros to appear instead of leading blanks; applies only to data items that are right-justified within a field whose minimum size larger than the data item.(Note : Some compilers consider the zero flag to be a part of the field width specification rather than an actual flag. This assures that the 0 is processed last, if multiple flags are present).
(blank space)	A blank space will precede each positive signed numerical data item; this flag is overridden by the + flag if both are present.
0 or 0X	Causes octal and hexadecimal data items to be preceded by 0 and 0x, respectively.
f, g	Causes a decimal point to be present in all floating-point numbers, even if the data item is a whole number, also prevents the truncation of trailing zeros in gtype conversion.

Table 3.10

Example: Here is a simple C program that illustrates the use of flags/modifiers with integer and floating-point quantities.

```
# include <stdio.h>

main ( ){          /* use of flags with integer and floating-point numbers */
    int I = 123;
    float x = 12.0, y = -3.3;
    printf (":%6d %7.0f %10.1e:\n\n", I,x, y);
}

```

```

printf (“:%-6d %-7.0f %-10.1e:\n\n”, I,x, y);
printf (“: %+6d %+7.0f %+10.1e:\n\n”, I, x, y )
printf (“: %-+6d %-7.0f %10.1e:\n\n”, I,x,y);
printf (“: %7.0f %#7.0f %7g %#7g:”, x x, y, y );
}

```

Reading and writing a line with gets and puts functions

The standard library functions gets and puts can be used to transfer of strings between the computer and the standard input/output devices.

Each of these functions accepts a single argument. The argument must be a data item that represents a string (e.g., a character array). The string may include whitespace characters. In the case of gets, the string will be entered from the keyboard, and will terminate with a newline character (i.e. the string will end when the user presses the RETURN key).

The gets and puts functions offer simple alternatives to the use of scanf and printf for reading and displaying strings, as illustrated in the following example.

```

#include <stdio.h>
main ( )      /* read and write a line of text */
{
    char line [80];
    gets (line);
    puts (line);
}

```

Check Your Progress

- Note: a) Space is given below for writing your answers
 b) Compare your answers with the one given at the end of the unit

4. What is getchar()?

.....

3.6 SUMMARY

Any language consists of certain alphabets, symbols, words, rules and conventions to follow. An identifier can be arbitrarily long but the significant length depends on the compiler (The ANSI standard recognizes 31 characters, though some recognize only the first eight characters). Additional characters are carried along for understandability and convenience of the programmer. A declaration consists of a data type, followed by one or more variable names, an optional value assigned to each variable, ending with a semi colon. The increment or decrement operators (++ and —) increment or decrement the current value by one which is a short form for a = a+1. The ++ and – operators can be either on the left side or on the right side of the operand, which has different meanings when used in an expression. Expressions in C are evaluated according to their precedence defined by C language.

The C language comes with a very rich set of standard library functions that carry out various commonly used operations or calculations. Though these library functions are not a part of the language specification, every implementations of the language include them. Library functions are organized as sets according to their functionality and their definitions are given in header files generally have a “.h” extension. Any type of data can be fed to the computer from a standard input device using the C standard library function scanf(). This function can be used to enter any combination of numerical values, single character and strings. The function returns the number of data items that have been entered successfully. The standard library functions gets and puts can be used to transfer of strings between the computer and the standard input/output devices.

3.7 CHECK YOUR PROGRESS MODEL ANSWERS

1. for, break, case, char
 2. A statement causes the computer to carry out some action.
 3. Operator which act upon a single operand.
 4. It returns a single character from a standard input device (generally a keyboard)
-

3.8 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Explain varoiops operators in C.
2. Describe the data types with examples.
3. Explain the input and output with examples?

II. Answer the following questions in about 15 lines each

1. Describe character set in C.
 2. Define identifier with examples and write the keywords in C.
 3. Describe the type casting, type coercion with examples
-

3.9 GLOSSARY

IDENTIFIER	:	Identifiers are names given to various program elements
VARIABLE	:	Logical name given to physical memory location
KEYWORD	:	Part of the C language vocabulary
PRINTF()	:	A built in function in C to print text, values, variables, expressions.
ASCII	:	American Standard Code for Information Interchange

BLOCK - II

CONTROL STRUCTURES

This block gives in-depth knowledge on the various concepts of flow of control in C and C++ such as sequencing, conditional branching using if, if-else, and if-else-elseif, switch statement with case statements which takes any number of branches and then unconditional branching with goto statement. Concepts of functions such as defining function, prototyping, invoking functions, parameter passing mechanisms are described very briefly. Pointers which store the address of variable, referencing, de-referencing, pointer arithmetic are explained with crystal clear examples. Strings which are stream of characters and operations on strings such as concatenation, padding, justification, reading, printing are explained using appropriate examples.

The units included in the block are:

Unit-4: Flow of Control

Unit-5: Functions

Unit-6: Pointers and Strings

UNIT- 4: FLOW OF CONTROL

Contents

- 4.0 Objectives
 - 4.1 Introduction
 - 4.2 Sequential and Branching Flow of Control
 - 4.3 Iterations- While, Do-While, For loops
 - 4.4 Summary
 - 4.5 Check your progress – Model Answers
 - 4.6 Model Examination Questions
 - 4.7 Glossary
-

4.0 OBJECTIVES

After studying this unit, you should be able to

- understand sequential flow of control in C
 - describe conditional branching and un-conditional branching in C language
 - explain how to work with various loops in C
 - understand break, continue, and exit in C
-

4.1 INTRODUCTION

In the sequential flow of control, the instructions were executed only once, in the same order in which they appeared in the program. In real world programsthis is unrealistic, since they do not include any logical control structures. A realistic program generally include tests to determine if certain conditions are true or false, require the repeated execution of groups of statements, and involve the execution of individual groups of statements on a selective basis. The else is generally applicable to the immediate preceding if condition. In some cases, where there are multiple if conditions, there may be a confusion to the user about this association. In such cases it is advised to use curly braces around the appropriate code to avoid confusion in reading the code. The go to statement is used to alter the normal sequence of program execution by transferring control to some other part of the same function without caring for the blockstructures of code, which breaks the structured nature of program. Hence, the usage of goto statement is highly discouraged and not seen in any of the example, but for its demonstration. This is used in case of deeply nested loops, and the control has to be moved out of all loops at once, as may be required in time critical applications. As a design principle, never use goto in structured programming unless it is an absolute must. Iterative statements are repeated based on a condition. As long as the condition is true, it continues to execute the set of statements within the block repeatedly. For example, the odd numbers within the range of 0 to 10 are to be printed. A value is considered to be odd if the remainder is 1 when divided by 2 ($x\%2$ should be 1). Programmatically.

4.2 SEQUENTIAL AND BRANCHING FLOW OF CONTROL

Sequential Flow of Control

In the programs so far, the instructions were executed only once, in the same order in which they appeared in the program. In real world programs this is unrealistic, since they do not include any logical control structures. A realistic program generally includes tests to determine if certain conditions are true or false, require the repeated execution of groups of statements, and involve the execution of individual groups of statements on a selective basis. In switch statement, each case is an entry point for the code execution based on its value. Default (denoted with a special keyword “default”) is used if the value doesn’t match any of the cases provided. Once, the entry point is selected, it continues to execute all statements till the end of switch block. That means, it executes all cases after the selection. To avoid this, a *break* statement is used to skip all the lines from this point to the end of switch block.

Branching with The if, if-else, if-else-else if and switch Statement

The if – else statement is used to carry out a logical test and then take one of two possible actions, depending on the outcome of the test (i.e., whether the outcome is true or false). General form of if-else statement can be written as

```
if (condition) statement; else statement
```

Here, the statement can be a single statement or a compound statement that is enclosed in curly braces as {statement1; statement2; statement3; ... ;}. In practice, it is generally a compound statement which may include other control statements. The else part is optional.

The condition expression must be placed in parentheses. In this form, if the condition is true (non zero), the first statement will be executed otherwise, the second statement (after else) is executed. For example, consider a case where a pass mark is 50. Hence to declare the result the code may be

```
if(m>= 50) printf(“Pass”); else printf(“Failed”);
```

This may be written more legibly as follows:

```
if(m>=50)
{
    printf(“Pass”);
}
else
{
    printf(“Failed”);
}
```

There may be multiple if else statements or nested if else statements in many programs as in the following example. Consider grading of a student based on the mark. If the mark ≥ 75 , A grade, ≥ 60 , B grade, ≥ 50 , C grade otherwise F grade. This can be given with various if else statements as follows:

```
#include <stdio.h>

void main()
```

```

{
    int m; // Declare a variable m for marks
    printf("Enter marks: "); // Print a prompt statement for clarity
    scanf("%d", &m); // Read a number into m
    if(m>=75) {
        printf("A Grade");
    }
    else if(m >= 60) {
        printf("B Grade");
    }
    else if(m>=50) {
        printf("C Grade");
    }
    else {
        printf("F Grade");
    }
}

```

Dangling if-else

The else is generally applicable to the immediate preceding if condition. In some cases, where there are multiple if conditions, there may be a confusion to the user about this association. In such cases it is advised to use curly braces around the appropriate code to avoid confusion in reading the code.

```

if (condition)
    if (condition)
else
    printf("dangling else!\n");

```

In the above code, though our intension is to attach the else part with the first if condition, the computer attaches it to the second if condition. Hence, adding curly braces as given below, this is properly attached to the first if condition.

```

if (condition){
    if (condition)
}
else
    printf("dangling else!\n");

```

Multi-Branching with switch Statement

The switch statement causes a particular group of statements to be chosen from several available groups. The selection is based upon the current value of an expression that is included within the switch statement. It is a special case of more general if else chain of statements.

The general form of the switch statement is

```
switch (variable)
{
case <value1>: statement;
case <value2>: statement;
... ..
default: statement;
}
```

Where the variable is an integer or character type (an expression that results in an integer value may be used in place of the variable).

Each case is an entry point for the code execution based on its value. Default (denoted with a special keyword “default”) is used if the value doesn’t match any of the cases provided. Once, the entry point is selected, it continues to execute all statements till the end of switch block. That means, it executes all cases after the selection. To avoid this, a *break* statement is used to skip all the lines from this point to the end of switch block.

For example consider the following example, where a number is given by the user and based on the number a color is printed as follows:

0: Black, 1: Red, 2: Green, 3: Blue, 4: White and any other value is Yellow. The program with if else conditions is as below:

```
#include <stdio.h>

void main() {
    int value;
    printf(“Enter a number:”);
    scanf(“%d”,&value);
    if(value == 0)
        printf(“Black”);
    else if(value == 1)
        printf(“Red”);
    else if(value == 2)
        printf(“Green”);
    else if(value == 3)
        printf(“Blue”);
    else if(value == 4)
        printf(“White”);
    else
```

```
        printf("Yellow");
    }
```

The same program can be written using the switch-case statements is as follows:

```
#include <stdio.h>
void main() {
    int value;
    printf("Enter a number:");
    scanf("%d",&value);
    switch(value){
        case 0: printf("Black");
        case 1: printf("Red");
        case 2: printf("Green");
        case 3: printf("Blue");
        case 4: printf("White");
        default: printf("Yellow");
    }
}
```

In the above program, consider that a value 2 is given when running the program. Then the control goes to the line case 2: directly where it prints the "Green" and doesn't stop there. It also prints all the statements till the end of block, so it prints "Blue", "White" and "Yellow" as well, which is not correct. To avoid execution of other statements, after the "case 2" block of statements, a "break" statement must be introduced. The break statement at the end of default block is optional since, it is in any way the last statement within the switch block. The correct program now becomes:

```
#include <stdio.h>
void main() {
    int value;
    printf("Enter a number:");
    scanf("%d",&value);
    switch(value){
        case 0: printf("Black"); break;
        case 1: printf("Red"); break;
        case 2: printf("Green"); break;
        case 3: printf("Blue"); break;
        case 4: printf("White"); break;
        default: printf("Yellow");
    }
}
```

Unconditional Branching: The goto Statement

The go to statement is used to alter the normal sequence of program execution by transferring control to some other part of the same function without caring for the blockstructures of code, which breaks the structured nature of program. Hence, the usage of goto statement is highly discouraged and not seen in any of the example, but for its demonstration. This is used in case of deeply nested loops, and the control has to be moved out of all loops at once, as may be required in time critical applications. As a design principle, never use goto in structured programming unless it is an absolute must.

In its general form, the go to statement is written as

goto label;

where *label* is an identifier used to label the target statement to which control will be transferred.

label: statement

Each labeled statement within the current function must have a unique label, i.e., no two statements can have the same label.

Example: The following skeletal outline illustrates how the go to statement can be used to transfer control out of a loop if an unexpected condition arised.

```
/* main loop */
scanf ("%f", &x);
while (x <= 100) {
    ... ..
    if (x < 0) goto errorcheck;
    ... ..
    scanf ("%f", &x);
    ... ..
}
... ..
/* error detection part */
errorcheck : {
    printf("ERROR – NEGATIVE VALUE FOR X");
}
}
```

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. List out branching statements in C?

.....

.....

.....

4.3 ITERATIONS- WHILE, DO-WHILE, FOR LOOPS

Iterative Controls or Loops

Iterative statements are repeated based on a condition. As long as the condition is true, it continues to execute the set of statements within the block repeatedly. For example, the odd numbers within the range of 0 to 10 are to be printed. A value is considered to be odd if the remainder is 1 when divided by 2 ($x\%2$ should be 1). Programmatically,

```
if(x%2 == 1) printf("%d",x);
```

The above statement prints the value of x only if it is odd, otherwise, it doesn't print anything.

For small set of values, the above problem can be solved with a series of if statements as follows:

```
if(0%2 == 1) printf("%d",0);
```

```
if(1%2 == 1) printf("%d",1);
```

```
if(2%2 == 1) printf("%d",2);
```

... ..

```
if(9%2 == 1) printf("%d",9);
```

```
if(10%2 == 1) printf("%d",10);
```

This solution fails if the range is not known ahead, and also for large range it is not feasible. To avoid this we may repeat the if statement with different values in the range. The generic algorithm for this is

```
x = start;
```

Repeat:

```
    If( $x\%2$  is 1) then print  $x$ ;
```

```
    Increment  $x$  by one;
```

```
Till  $x < \text{end}$ ;
```

There are three iterative loop controls available in C language, viz., “*for*”, “*while*” and “*do-while*”. Theoretically all these can do the same thing with different syntax. However, in practice, a loop statement is selected based on the situation.

The while Statement

The “*while*” statement is used to carry out loop operations based on a condition. The general form of the statement is

```
while (expression) statement
```

The included *statement* will be executed repeatedly, as long as the value of *expression* is not zero (true). If the expression never becomes zero, it goes into an infinite loop effectively hanging the terminal. This statement can be simple or compound, though it is typically a compound statement. It must include some statement which alters the value of expression, thus providing a stopping condition for the loop.

Example: Consider that consecutive digits 0,1,2, ...,9, with one digit on each line are to be printed. This can be accomplished with the following program.

```
#include <stdio.h>
```

```

main ( )/* display the integers 0 through 9 */
{
    int digit = 0;
    while (digit <=9) // Condition
    {
        printf("%d\n", digit); // Print digit followed by a newline
        ++digit; // Increment digit
    }
}

```

Example: Consider that all odd numbers in the range 0 to 10 are to be printed.

```

#include <stdio.h>
main ( )/* display odd numbers between 0 and 10 */
{
    int digit = 0;
    while (digit <=10) // Condition
    {
        if (digit%2 == 1)
        {
            printf("%d\n", digit); // Print digit followed by a newline
        }
        ++digit; // Increment digit
    }
}

```

The do-while Statement

When a loop is constructed using the while statement, the test for continuation of the loop is carried out at the beginning of each pass. Sometimes, however, it is desirable to have a loop with the test for continuation at the end of each pass. This can be accomplished by means of the do – while statement. The general form of the do – while statement is

do statement; while (expression)

The included statement will be executed repeatedly, as long as the value of expression is not zero. Notice that statement will always be executed at least once, since the test for repetition does not occur until the end of the first pass through the loop. The statement can be either simple or compound though most applications will require it to be a compound statement. It must include some statement which alters the value of expression so that looping can terminate.

Example: Consider the same example given above for a while loop which will now be achieved with a do while loop.


```
#include <stdio.h>
main () /* display the integers 0 through 9 */
{
    int digit = 0;
    do
        printf ("%d\n", digit ++); // print value and then increment digit
    while (digit <=9);
}
```

Example: Consider that all odd numbers in the range 0 to 10 are to be printed.

```
#include <stdio.h>
main () /* display odd numbers between 0 and 10 */
{
    int digit = 0;
    do
    {
        if (digit%2 == 1) // check if digit is odd
        {
            printf("%d\n", digit); // Print digit followed by a newline
        }
        ++digit; // Increment digit
    }
    while (digit <=10) // Condition
}
```

The for Statement

Perhaps the most commonly used looping statement in C is the “for” statement. This statement includes an expression that specifies an initial value for one or more loop variables, another expression that provides a condition and a third expression that modifies the loop variable(s) at the end of each pass.

The general form of the for statement is

```
for (expression1; expression2; expression3) statement;
```

Where expression1 is used to initialize some parameter or parameters (called loop variable(s)) that control the looping action, expression2 represents a condition that must be satisfied for the loop to continue execution, and expression3 is used to alter the value of the parameter. Typically, expression1 is an assignment expression, expression 2 is a logical expression and expression3 is a unary expression or an expression. All the three expressions are optional which means that they can be empty as in “for(;;) statement” where there is no initialization, no condition and no modifier. This results in an infinite loop, which should be avoided unless needed.

When a for statement is executed, expression2 is evaluated and tested before each pass through the loop, and expression3 is evaluated at the end of each pass.

Example: Printing of consecutive numbers 0 to 9 can be written using a for loop as follows:

```
# include<stdio.h>

main ( )/* display the numbers 0 through 9 */
{
    int digit;
    for (digit = 0; digit <=9; ++ digit)
        printf ("%d\n", digit);
}
```

Example: Consider the same example that prints odd numbers between 0 and 10

```
# include <stdio.h>

main ( ) /* display the odd numbers between 0 and 10*/
{
    int digit;
    for (digit = 0; digit <=10; ++ digit)
    {
        if(digit%2 == 1)
            printf ("%d\n", digit);
    }
}
```

Nested Control Structures

Control structures like if-else, switch-case, for, while, do-while can be nested (i.e. embedded) one within another. The inner and outer loops need not be generated by the same type of control structure. It is mandatory that there should be no overlaps which means, one loop should be completely embedded within the other. Each loop must be controlled by a different variable, so that there is no clash among loop conditions.

Example: Printing sum of three, three consecutive numbers between 1 and 12, i.e. sums of (1,2,3),(4,5,6), (7,8,9) and (10,11,12). Essentially this is done using two for loops, where the first loop runs 4 times and second loop run three times.

```
#include <stdio.h>

void main()
{
    int i,j,sum; // Declare three variables
    for (i=0; i<4; i++) // Out for loop that iterates for 4 times with i = 0,1,2 and 3
    {
        sum = 0; // Every time initialize sum to zero
        for(j=0; j<3; j++) // Inner loop that iterates 3 times with j = 0,1 and 2
```

```

    {
    sum += (i*4 + j+1); // Add current number to sum
    }
    printf("Sum: %d\n", sum);
}
}

```

Here the inner for loop is executed for each iteration of outer for loop, every time changing value of j from 0 to 2. The i value changes from 0 to 3. The consecutive number can be obtained with the formula $i*4 + j + 1$.

The break and continue Statements

In some situations, we may skip an iteration based on some condition and we may abruptly terminate the loop. These two actions are achieved with continue and break statements respectively. The continue statement is used to bypass the remainder of the current pass through a loop. The loop does not terminate when a continue statement is encountered. Rather, the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop. On the other hand, the break statement abruptly ends the loop totally and goes to the next statement after the loop. The continue statement can be included within a for, while or a do-while statement. A break can also be used in switch-case apart from these three loops. The syntax for continue and break is:

```
continue;
```

```
break;
```

Example: Consider an application that reads numbers continuously from the keyboard and prints the sum of all even numbers not divisible by 4. The reading ends when the user enters a -1 as input.

```

#include <stdio.h>
void main()
{
    int sum = 0; // declare variable for sum and initialize
    int num; // Declare a variable for input
    while(1) // 1 is always true, hence it is an infinite loop
    {
        printf("Enter a number (-1 to Terminate):");
        scanf("%d",&num);
        if(num == -1)
        {
            break; // break out of loop only if value is -1
        }
        if(num%4 == 0)

```

```

    {
        continue; // skip when the number is divisible by 4
    }
    if(num%2 == 0)
    {
        sum += num; // add to sum only if the number is even
    }
}
printf("Sum: %d\n",sum);
}

```

The Comma Operator

This operator permits two different expressions to appear in situations where only one expression would ordinarily be used. For example, it is possible to write

```
for (expression1a, expression1b, expression2, expression3a, expression3b) statement;
```

Where expression1a and expression1b are the two expressions, separated by the comma operator, where only one expression (expression1) would normally appear. These two expressions would typically initialize two separate indices that would be used simultaneously within the for loop. Similarly expression3a and expression3b are combined with a comma operator, where a for loop expects only one expression.

Example:

```
for(i=0,j=10; i<10 && j>0; i++,j--) statement;
```

Here two integers i and j are initialized, tested as a compound logical expression and i is incremented and j is decremented after each pass.

Check Your Progress

Note: a) Space is given below for writing your answers

- b) Compare your answers with the one given at the end of the unit

2. List out iterative statements in C?

.....

.....

.....

4.4 SUMMARY

In C Language, a realistic program generally include tests to determine if certain conditions are true or false, require the repeated execution of groups of statements, and involve the execution of individual groups of statements on a selective basis. In switch statement, each case is an entry point for the code execution based on its value. Default (denoted with a special keyword “default”) is used if the value doesn’t match any of the cases provided. Once, the entry point is selected, it continues to execute all statements till the end of switch block. That means, it executes all cases after the selection. To avoid this, a *break* statement is used to skip

all the lines from this point to the end of switch block. Iterative statements are repeated based on a condition. As long as the condition is true, it continues to execute the set of statements within the block repeatedly. There are three iterative loop controls available in C language, viz., “for”, “while” and “do-while”. Theoretically all these can do the same thing with different syntax. However, in practice, a loop statement is selected based on the situation. In some situations, we may skip an iteration based on some condition and we may abruptly terminate the loop. These two actions are achieved with continue and break statements respectively. The continue statement is used to bypass the remainder of the current pass through a loop. The loop does not terminate when a continue statement is encountered. Rather, the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop. On the other hand, the break statement abruptly ends the loop totally and goes to the next statement after the loop. The continue statement can be included within a for, while or a do-while statement. A break can also be used in switch-case apart from these three loops. The syntax for continue and break is continue.

4.5 CHECK YOUR PROGRESS MODEL ANSWERS

1. if, if-else, if-else-elseif, switch
2. while, do-while, for

4.6 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Explain if, if-else, if-else if with examples.
2. Describe the switch with examples
3. Differentiate the while, do-while, for with examples?

II. Answer the following questions in about 15 lines each

1. Describe goto statement with examples.
2. Explain break and continue in C.
3. Describe the for loop examples

4.7 GLOSSARY

IF	:	Reserved word in C which execute a branch of statement if condition is true
ELSE	:	Reserved word in C which execute alternative branch of if
DEFAULT	:	Executes default case of switch statement
GOTO	:	Un-conditional branching statement in C
BREAK	:	Exit from the iterative statement or loop

UNIT- 5: FUNCTIONS

Contents

- 5.0 Objectives
- 5.1 Introduction
- 5.2 Basics of Functions
- 5.3 Recursive Functions
- 5.4 Pre-Processive Directives
- 5.5 Summary
- 5.6 Check your progress – Model Answers
- 5.7 Model Examination Questions
- 5.8 Glossary

5.0 OBJECTIVES

After studying this unit, you should be able to

- explain how to define and use functions in C
- describe various parameter passing mechanisms in C language
- explain how to write programs using recursive functions in C
- understand pre-processive directives in C

5.1 INTRODUCTION

A function is a self-contained program segment that carries out some specific, well-defined task. Every C program consists of one or more functions. One of these functions must be called main, from where the execution begins. Additional functions will be called either from main or from other functions. You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task. A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function. While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program. To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function. Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit. The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument. By default, C programming uses call by value to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function. The call by reference method of passing arguments to a

function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the parameter affect the passed argument. To pass a value by reference, argument pointers are passed to the functions just like any other value.

5.2 BASICS OF FUNCTIONS

A function is a self-contained program segment that carries out some specific, well-defined task. Every C program consists of one or more functions. One of these functions must be called main, from where the execution begins. Additional functions will be called either from main or from other functions. If a program contains multiple functions, their definitions may appear in any order. One function definition cannot be embedded within another.

A function will carry out its intended action whenever it is accessed (i.e. whenever the function is “called”) from some other portion of the program. The same function can be accessed from several different places within a program. Once the function has carried out its intended action, control will be returned to the point from which the function was accessed.

A function processes information passed to it by the calling statement and returns a single value. Information will be passed to the function via special identifiers called arguments or parameters and returned via the return statement. Both parameters and return values are optional which means some functions may take input but return nothing, may take nothing but returns a value or may take input and return a value.

Defining a Function

A function definition has different principal components: A return type, the function name, the argument declarations and the body of the function.

The first line of a function definition contains the type specification of the value returned by the function, followed by the function name, and (optionally) a set of arguments, separated by commas and enclosed in parentheses. If return type specification is omitted, it is assumed that it returns an integer. An empty pair of parentheses must follow the function name if the function definition does not include any arguments.

The general structure of a function is as follows:

```
<data-type><function name> (argument1, argument2, ... argument n)
<data type of argument1> argument1;
<data type of argument2> argument2;
... ..
<data type of argument n> argument n;
{
    statements;
    return <value>;
}
```

C also defines a convenient and more popular form of a function definition where the data types of each parameter is given in the argument list itself as follows, which is used throughout this text.

```
<data-type><function name> (type1 argument1, type2 argument2, ... type n argument n)
{
```

```

    statements;
    return <value>;
}

```

Where data-type represents the data type of the value which is returned, and name represents the function name. Each formal argument must be defined following the parentheses and before the curly brace begins.

The formal arguments or formal parameter allow information to be transferred from the calling portion of the program to the function. The corresponding arguments in the function reference are called actual arguments, since they define the information actually being transferred. The identifiers used as formal arguments are “local” in the sense that they are not recognized outside of the function. Hence, the names of the formal arguments may be the same as the names of other identifiers that appear outside the function definition.

Each formal argument must be of the same data type as the data item it receives from the calling portion of the program.

Information is returned from the function to the calling portion of the program via the return statement. The return statement also causes control to be returned to the point from which the function was accessed. In general terms, the return statement is written as

```
return expression;
```

where the expression must result in a value of the return data type of the function.

As an example, consider the function that takes two integers and returns their sum as an integer.

```

int add(a, b)
int a;
int b;
{
return a+b;
}

```

The same method may be written as

```

int add(int a, int b)
{
return a+b;
}

```

Consider another function that just prints a hello message and returns nothing (void).

```

void show_message()
{
printf(“Hello World!\n”);
}

```

Consider another function that takes no input but sends the value of PI.

```

float getPi()
{

```



```
    return 3.1415f;
}
```

Consider another function that takes a number as input and returns nothing.

```
float print_square(int n)
{
    printf(“%d\n”, n*n);
}
```

Accessing a Function

A function can be accessed (i.e. called) by specifying its name, followed by a list of arguments enclosed in parentheses and separated by commas. The function call may appear by itself (that is, it may comprise a simple expression), or it may be one of the operands within a more complex expression.

The arguments appearing in the function call are referred to as actual arguments, in contrast to the formal arguments that appear in the first line of the function definition. In a normal function call, there will be one actual argument for each formal argument. The actual arguments may be expressed as constants, single variables, or more complex expressions. However, each actual argument must be of the same data type as its corresponding formal argument.

Passing Arguments to a Function

When a value is passed to a function via an actual argument, the value of the actual argument is copied into the function. Therefore, the value of the corresponding formal argument can be altered within the function, but the value of the actual argument within the calling routine will not change. This procedure for passing the value of an argument to a function is known as passing by value. If the changes to a variable in the called function reflect back in the calling program, it is known as pass by reference, which is not directly supported by C language. This is achieved in C language by passing the address of the variable and this address is used in the function to manipulate the value (known as pointer which will be covered later).

Function Prototypes

Many C compilers support a more comprehensive system for handling argument specifications in function definitions. In particular, the proposed ANSI standard permits each of the argument data types within a function declaration to be followed by an argument name, that is,

```
<data type><function name> (<type 1><arg 1>, <type 2><arg 2>, ... <type n><arg n>);
```

Where arg 1, arg2, ... arg n refer to the first argument, the second argument, and so on.

Function declarations written in this form are called function prototypes.

Function prototypes are desirable (but not mandatory) because they facilitate error checking between the calls to a function and the corresponding function definitions.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. What is a function?

.....
.....
.....

5.3 RECURSIVE FUNCTIONS

Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied. The process is used for repetitive computations in which each action is stated in terms of a previous result. Many iterative (i.e. repetitive) problems can be written in this form.

In order to solve a problem recursively, two conditions must be satisfied. First, the problem must be written in a recursive form, and second, the problem statement must include a stopping condition. Consider calculation of factorial of a positive integer quantity. This is expressed as $n! = 1 \times 2 \times 3 \times \dots \times n$, where n is the specified positive integer. However, it can also be expressed in another way, by writing $n! = n \times (n-1)!$ where $n > 0$ and $0! = 1$. This is a recursive statement of the problem, in which the desired action (the calculation of $n!$) is expressed in terms of a previous result (the value of $(n-1)!$, which is assumed to be known). Hence, the recursive definition should move towards the known end condition for this to terminate properly. This last expression provides a stopping condition for the recursion.

Example: Calculating Factorials

```
# include <stdio.h>

/* calculate the factorial of an integer quantity using recursion */
main ( )
{
    int n;
    long int factorial (int n);
    /* read in the integer quantity */
    printf ("n = ");
    scanf ("%d", &n);
    /* calculate and display the factorial */
    printf("n! = %ld\n", factorial (n)); // Call to factorial function
}

long int factorial (int n) // calculate the factorial
{
    if (n ==1){
        return (1); // Termination condition
```

```

    }
    else {
        return (n * factorial (n - 1)); // Recursive statement
    }
}

```

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

2. What is a recursive function?

.....

.....

.....

5.4 PRE-PROCESSIVE DIRECTIVES

The C preprocessor is already introduced earlier and some of the useful directives are discussed here. The C preprocessor is a collection of special statements, called directives that are executed before the actual compilation process begins. Some of the preprocessor directives are #if, #elif, #else, #endif, #ifdef, #ifndef, #line and #undef. The preprocessor also includes three special operators namely defined, #, and ##. Thus, a preprocessor directive may appear anywhere within a program though generally they appear at the beginning of a program. However, the directive will apply only to the portion of the program following its appearance.

Macros

The preprocessor provides a convenient way to define shortcuts and constants using the directive #define. The format of a macro is:

```
#define <macroname><optional parameters in brackets><actual definition>
```

Once, a macro is defined, it can be used in the program, as illustrated in the following program:

```
#include <stdio.h>
#define PI 3.1415f
#define area(r) (PI*r*r)
```

```
int main()
{
    int radius;
    float a;

    printf("Enter the radius: ");
    scanf("%d", &radius);
    a = area(radius);
```

```

printf("Area of circle is %.2f", a);
return 0;
}

```

In the above program two macros are defined. The first one is a constant PI and the other is a formula to compute the area. When the program is compiled, these values are substituted in the program and then compiled.

Conditional operators

The `#if`, `#elif`, `#else` and `#endif` directives are used frequently. They permit conditional compilation of the source program, depending on the value of one or more true/false conditions. They are sometimes used in conjunction with the `defined` operator, which is used to determine whether or not a symbolic constant or a macro identifier has been defined within a program.

Example: The following preprocessor directives illustrate the conditional compilation of a C program. The conditional compilation depends on the status of the symbolic constant `FOREGROUND`.

```

#if defined (FOREGROUND)
    #define BACKGROUND 0
#else
    #define FOREGROUND 0
    #define BACKGROUND 7
#endif

```

Here, if `FOREGROUND` has already been defined, the symbolic constant `BACKGROUND` will represent the value 0. Otherwise, `FOREGROUND` and `BACKGROUND` will represent the values 0 and 7, respectively.

Example: Here is another example for conditional compilation. In this case the conditional compilation depends on the value represented by the symbolic constant `BACKGROUND`.

```

#if(BACKGROUND)== 7
    #define FOREGROUND0
#elif BACKGROUND==6
    #define FOREGROUND1
#else
    #define FOREGROUND6
#endif

```

The # operator

The operator `#` allows a formal argument within a macro definition to be converted to a string by prefixing a `#` symbol. The corresponding actual argument will automatically be enclosed in double quotes. Consecutive whitespace characters inside the actual argument will be replaced by a single blank space, and any special characters, such as `'`, `,` and `\`, will be replaced by their corresponding escape sequences, e.g. `\'`, `\,` and `\\`. In addition, the resulting string will automatically be concatenated (combined) with any adjacent strings.

Example: Here is an illustration of the use of the `#` operator

```
#define    display(text)    printf(#text "\n")
main ( )
{
...
    display (Please do not sleep in lclass);
    ...
    display (Please – don't snore during the professor's lecture!)
}
```

within main, the macros are equivalent to

```
printf ("please do not sleep in class. \n");
```

and

```
printf ("Please – don't snore during the professor's lecture!\n");
```

Notice that each actual argument is converted to a string within the “printf” function. Each argument is concatenated with a newline character (\n), which is written as a separate string within the macro definition. Also, notice that the consecutive blank spaces appearing in the second argument are replaced by single blank spaces, and each apostrophe (‘) is replaced by its corresponding escape sequence (\’).

The ## Operator

The “token-pasting” operator ## causes individual items within a macro definition to be concatenated, thus forming a single item. The various rules governing the use of this operator are somewhat complicated. However, the general purpose of the token-pasting operator is illustrated in the following example.

```
# define display(i)printf (“x” # i “ = %f\n”, x # # i)
```

Suppose this macro is accessed by writing

```
display (3);
```

The result will be

```
printf (“x3 = %f\n”, x3);
```

Thus, the expression x ## i becomes the variable x3, since 3 is the current value of the argument i.

Notice that this example illustrates the use of both the #and ## operators.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

3. How do you define a macro?

.....

.....

.....

5.5 SUMMARY

In C Language, A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions. You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task. A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function. The C standard library provides numerous built-in functions that your program can call. For example, `strcat()` to concatenate two strings, `memcpy()` to copy one memory location to another location, and many more functions. A function can also be referred as a method or a sub-routine or a procedure, etc. A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately. unction declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program. To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. f a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function. Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit. The “token-pasting” operator `##` causes individual items within a macro definition to be concatenated, thus forming a single item. The various rules governing the use of this operator are somewhat complicated.

5.6 CHECK YOUR PROGRESS MODEL ANSWERS

1. A function is a self-contained program segment that carries out some specific, well-defined task
 2. Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied
 3. `#define <macroname><optional parameters in brackets><actual definition>`
-

5.7 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Explain recursion with suitable examples.
2. Describe the macros with suitable examples
3. Write a function to print factorial a given number?

II. Answer the following questions in about 15 lines each

1. Describe advantages of functions in programming.
2. Explain how to define and call a function with an example.
3. Explain conditional compilation with an example.

5.8 GLOSSARY

PARAMETER	:	Variable or expression or value passed to/from function
ACTUAL PARAMETER	:	Variable or expression or value in the main()
FORMAL PARAMETER	:	Variable or expression or value in the definition of a function
CALL BY VALUE	:	values of actual parameters are copied into formal parameters
CALL BY REFERENCE	:	Address of actual parameter is passed to formal parameter

UNIT- 6: POINTERS AND STRINGS

Contents

- 6.0 Objectives
 - 6.1 Introduction
 - 6.2 Pointers in C
 - 6.3 Strings
 - 6.4 Summary
 - 6.5 Check your progress – Model Answers
 - 6.6 Model Examination Questions
 - 6.7 Glossary
-

6.0 OBJECTIVES

After studying this unit, you should be able to

- describe declaring pointers of various data types in C
 - explain how pointers can be manipulated in C language
 - understand how to declare strings in C
 - describe various operations on strings in C
-

6.1 INTRODUCTION

Computer stores data in bits and bytes only. Every data item occupies one or more contiguous memory cells (i.e., adjacent words or bytes). The number of memory cells required to store a data item depends on the type of data item. For example, a single character will typically be stored in 1 byte (8 bits) of memory; an integer usually requires two or four contiguous bytes, a floating-point number may require four contiguous bytes, and a double-precision quantity may require eight contiguous bytes. The memory cells are sequentially numbered from 0 to maximum memory available. The lower range of memory addresses are used by operating system components generally and the user space is after this range. Hence the addresses of variables are large numbers in general, expressed as hexadecimal numbers generally. Pointers are often passed to a function as arguments if the changes to the argument should reflect back in the calling program. This is referred to as reference (or by address or by location).

When an argument is passed by value, the data item is copied to the function. Thus, any alteration made to the data item within the function is not carried over into the calling function. When an argument is passed by reference, the address of a data item is passed to the function. Hence, any change that is made to the data item (i.e. to the contents of the address) will reflect in both the calling and called function. Thus, the use of a pointer as a function argument permits the corresponding data item to be altered globally from within the function. The String is defined as an array of characters which will have a size of 15, since the string length is 14 including spaces and the terminating '\0' character at the end. Both printf statements will print the same string while the printf("\n") provides the new line between them. In many cases, a string may be copied into another, which is taken care by strcpy function of C library. Internally, this is copying the elements of one array to another. The strcpy() function requires two arguments. The first is the address of the target array into which the string is to be copied, and the second is the address of the source array containing the original string.

6.2 POINTERS IN C

Fundamentals of Pointers

Computer stores data in bits and bytes only. Every data item occupies one or more contiguous memory cells (i.e., adjacent words or bytes). The number of memory cells required to store a data item depends on the type of data item. For example, a single character will typically be stored in 1 byte (8 bits) of memory; an integer usually requires two or four contiguous bytes, a floating-point number may require four contiguous bytes, and a double-precision quantity may require eight contiguous bytes. The memory cells are sequentially numbered from 0 to maximum memory available. The lower range of memory addresses are used by operating system components generally and the user space is after this range. Hence the addresses of variables are large numbers in general, expressed as hexadecimal numbers generally.

The human beings are used to the name convention where each object is given a name but there are cases where numbers are used. Typical example for a number convention is the hotel tables, where each table is internally numbered and the service is based on the table, but not on the persons. Similarly the rooms in a hotel are numbered. Suppose a person “X” stays in room 102. Now clean the room of X and clean room102 mean the same. In the same way if a variable *v* is assigned the space starting at 0x1f1f1f, then *v*=10 and put 10 at address 0x1f1f1f will both assign the value 10 to *v*. The “*v*=10” is named convention and “put 10 at address 0x1f1f1f” is numbered convention. Since the direct address is used in numbered convention, its effect can be seen from anywhere in the program, or even across programs.

Suppose *x* is a variable that represents some particular data item. The compiler will automatically assign memory cells for this data item. The data item can be accessed if the location of the first memory cell is known (i.e. the address). The address of *x*'s memory location can be obtained by the using & operator (called the address operator).

Now let us assign the address of *x* to another variable, *px*. Thus,

```
px = &x
```

This new variable is called a pointer to *x*, since it “points to the location where *x* is stored in memory. Note that *px* represents *x*'s address, not its value. Thus, *px* is referred to as a pointer variable. The relationship between *px* and *x* is illustrated below where *x* is located at an address “0Xaabbcc and *x* value is 25.

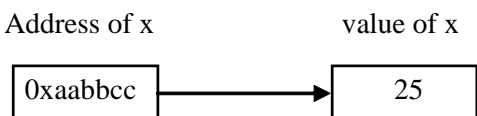


Figure 6.1

The data item represented by *x* (i.e., the data item stored in *x*'s memory cells) can be accessed by the expression **px*, where *** is a unary operator (called the indirection operator) that operates only on a pointer variable. Therefore, **px* and *x* both represent the same data item (i.e., the contents of the same memory cells). Furthermore, if we write *px = &x*; *u = *px*; then *u* and *v* will both represent the same value, i.e. the value of *x* will be assigned to *u* via a pointer indirectly.

Pointer Declarations

Pointer variables must be declared before they are used in a C program. The declaration is as follows:

```
<data-type> *<variable>
```

The interpretation of a pointer declaration is different than the interpretation of other variable declarations. When a pointer variable is declared, the variable name must be preceded by an asterisk (*) which informs the compiler that the variable is a pointer. The data type refers to the data type of the object stored at the address given by the pointer.

Example : A C program contains the following declarations :

```
float u, *v; // u is ordinary variable, v is a pointer to a float number
```

```
char *pv; // pv is a variable to a character
```

Example: Shown below is a simple program that illustrates the relationship between two integer variables, their corresponding addresses and their associated pointers.

```
#include <stdio.h>

main ( )
{
    int u = 3;
    int v;
    int pu;          /* pointer to an integer */
    int pv;          /* pointer to an integer */
    pu = &u; /* assign address of u to pu */
    v = pu;  /* assign value of u to v */
    pv = &v; /* assign address of v to pv */
    printf ("\nu=%d  &u=%x pu=%d",      u, &u, pu, *pu);
    print ("\n\nv=%d  &v=%x pv=%x  *pv=%d"      v, &v, pv, *pv);
}
```

Passing Pointers to a Function

Pointers are often passed to a function as arguments if the changes to the argument should reflect back in the calling program. This is referred to as reference (or by address or by location).

When an argument is passed by value, the data item is copied to the function. Thus, any alteration made to the data item within the function is not carried over into the calling function.

When an argument is passed by reference, the address of a data item is passed to the function. Hence, any change that is made to the data item (i.e. to the contents of the address) will reflect in both the calling and called function. Thus, the use of a pointer as a function argument permits the corresponding data item to be altered globally from within the function.

When pointers are used as arguments to a function, the formal arguments that are pointers must each be preceded by an asterisk. Also, if a function declaration is included in the calling portion of the program, the data type of each argument that corresponds to a pointer must be followed by an asterisk. Both of these points are illustrated in the following example.

```
#include<stdio.h>

void change(int *a); // Function Declaration

void main()
```

```

{
    int a = 10;
    change(a);
    printf("Value after calling change function: %d\n", a);
    change_ptr(&a);
    printf("Value after calling change_ptr function: %d\n",a);
}

void change(int a)
{
    a = a+1;
    printf("Value inside change function: %d\n", a);
}

void change_ptr(int *a)
{
    *a = (*a)+1;
    printf("Value inside change_ptr function: %d\n", a);
}

```

The main program first calls change() function with a copy of a, hence, within the function its value is incremented, but when it returns back, the value of original 'a' is still 10. Then it calls the second function change_ptr(), where it sends a copy address of 'a' and hence, the indirection operator manipulates data at the actual location where a is allocated space. So the change in value reflects back in the main function, and new value will be printed. The final output from the program is similar to the following:

```

Value inside change function: 11
Value after calling change function: 10
Value inside change_ptr function: 11
Value after calling change_ptr function: 11

```

Operations on Pointers (Pointer Arithmetic)

In the previous section addition of an integer to a pointer variable is illustrated. The integer value is interpreted as an array subscript; it represents the location of the desired array element relative to the first element in the array. This works because all of the array elements are of the same data type (so each array element occupies the same number of bytes), and all elements are allocated space contiguously. The actual number of memory cells separating the two array elements will depend on the data type of the array, though this is taken care of automatically by the compiler.

Suppose, for example, that px is a pointer variable representing the address of some variable x. We can write expressions such as ++px, --px, (px +3), (px+i) and (px-i), where I is an integer variable. Each expression will represent an address located some distance from the original address represented by px. The exact distance will be the product of the integer quantity and

the number of bytes associated with the data item to which px points. Suppose, for example, that px points to an integer quantity, and each integer quantity requires 2 bytes of memory. Then the expression (px+3) will result in an address 6 bytes beyond the integer to which px points.

Example: Consider the simple C program shown below.

```
#include <stdio.h >

main ( )
{
    int px; // pointer to an integer
    int I = 1;
    float f = 0.3;
    double d = 0.005;
    char c = '*';

    px = &I;
    printf ("values: i=%di f=%fd=%fc=%c\n", I, f, d, c);
    printf ("Addresses:&i=%x&d=%x&c=%x\n", &I, &f, &d, &c);
    printf ("Pointer values:px=%xpx + 1=%xpx + 2=%x, px +3=%x\n",
            px, px+1, px +2, px +3);
}
```

Passing Functions to Other Functions

A function's name gives the address of that function which means the name of the function being declared becomes a pointer to that function. The advantage of using pointers to functions is that a function can be passed to another function dynamically during runtime.

When a function accepts another function's name as an argument, a formal argument declaration must identify that argument as a pointer to another function. In its simplest form, a formal argument that is a pointer to a function can be declared as

```
<data type> (*function-name) ();
```

Where, datatype refers to the data type of the return value of the function. This function can then be accessed by means of the indirection operator. To do so, the indirection operator must precede the function name (i.e. the formal argument). Both the indirection operator and the function name must be enclosed in parentheses, that is,

```
(*function-name)(argument 1, argument 2, .... Argument n);
```

where argument 1, argument 2,, argument n refer to the arguments required in the function call.

More about Pointer Declarations

Before leaving this chapter we mention that pointer declarations can become complicated, and some care is required in their interpretation. This is especially true of declarations that involve functions or arrays.

One difficulty is the dual use of parentheses. In particular, parentheses are used to indicate functions, and they are used for nesting purposes (to establish precedence) within more complicated declarations. Thus, the declaration

```
int (*p)(int a);
```

indicates a pointer to a function that accepts an integer argument, and returns an integer. In this declaration, the first pair of parentheses is used for nesting, and the second pair is used to indicate a function.

The interpretation of more complex declarations can be increasingly difficult. For example, consider the declaration

```
int *(*p)(int (*a) [ ] );
```

In this declaration, (*p)(...) indicates a pointer to a function. Hence, int *(*p)(...) indicates a pointer to a function that returns a pointer to an integer. Within the last pair of parentheses (the function's argument specification), (*a)[] indicates a pointer to an array. As a result, int (*a)[] represents a pointer to an array of integers. Putting the pieces together, (*p) (int (*a) []) represents a pointer to a function whose argument is a pointer to an array of integers. Hence, the entire declaration

```
Int *(*p) (int (*a)[ ] );
```

represents a pointer to a function that accepts a pointer to an array of integers as an argument, and returns a pointer to an integer.

Remember that a left parenthesis immediately following an identifier name indicates that the identifier represents a function. Similarly, a left square bracket immediately following an identifier name indicates that the identifier represents an array. Parentheses that identify functions and square brackets that identify array have a higher precedence than the unary indirection operator (see Appendix C). Therefore, additional parentheses are required when declaring a pointer to a function or a pointer to an array.

The following example provides a number of illustrations.

Example: Several declarations ranging from simple to complex involving pointers are shown below.

Int *p;	p is a pointer to an integer quantity
int *p[10]	p is a 10-element array of pointers to integers
int (*p)[10]	p is a pointer to a 10-element integer array
int *p(void);	p is a function that returns a pointer to an integer
int p(char *a);	p is a function that accepts a pointer to a character and returns an integer
int *p(char *a);	p is a function that accepts a pointer to a character and returns a pointer to an integer
int (*p) (char *a)	p is a pointer to a function that accepts a pointer to a character and returns an integer
int *p(char (*a)[]);	p is a function that accepts a pointer to a character array and returns an integer
int *p(char *a[]);	p is a function that accepts an array of pointers to characters and returns an integer

<code>int *p (char *a[]);</code>	p is a function that accepts a character array and returns a pointer to an integer
<code>int *p(char (*a)[]);</code>	p is a function that accepts a pointer to a character array and returns a pointer to an integer
<code>int *(*p)(char(*a)[]);</code>	p is pointer to a function that accepts an array of pointers to characters and returns a pointer to an integer
<code>int (*p[10])(void);</code>	p is a 10-element array of pointers to functions where each function has no arguments and returns an integer
<code>int *(*p[10])(char *a);</code>	p is a 10-element array of pointers to functions where each function accepts a pointer to character, and returns a pointer to an integer

Table 6.1

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. What is a pointer?

.....

.....

.....

6.3 STRINGS

String is a chain of characters represented as an array of characters in C language. Hence, there is no special data type as string in c, except that they are treated differently in C. Following are some of the salient features of strings in C:

- Strings are enclosed in double quotes
- Two consecutive strings without anything between them are automatically concatenated
- Strings are represented as array of characters
- String name gives the address of the first character in that string (since it is an array)
- Numerous string operations are provided in standard library
- Special characters will be escaped with a reverse slash (\) as \n for new line, \a for audible bell, \t for tab character \" for double quote within double quotes ...
- All strings will have a NULL character (\0) appended at the end, which marks the termination of the string. Hence the size of string is one character more than length of the string.

Declaring a string is as simple as declaring an array of characters. The following declares a string that can take up to 19 characters long string (last one is for NULL character).

```
Char name[20];
```

If the string is initialized while declaring it, the dimension can be omitted as in

```
char name[] = "The C Tutorial";
```

where name is allocated the required size (15 bytes in this case) automatically by the compiler. Since one dimensional array can be represented with a pointer, the above declaration can be represented with

```
char *name = "The C Tutorial";
```

where name is a pointer to character type data and points to 'T' in the allocated space of 15 characters. Once, a pointer is declared, there is no difference between an array and a string except that the strings are terminated with a NULL ('\0') character in C.

Consider the following example:

```
charname[ ] = "The C Tutorial";
```

```
main ( )
{
printf (&name[0]);
printf("\n");
printf (name);
}
```

The String is defined as an array of characters which will have a size of 15, since the string length is 14 including spaces and the terminating '\0' character at the end. Both printf statements will print the same string while the printf("\n") provides the new line between them.

Putting and Getting Strings

Two very useful functions for printing and reading strings are puts() and gets().

The puts() function puts a string to the standard output device (generally the screen but could be re-directed) on your system. The gets() function will get a string from the standard input device (generally the keyboard but can be altered with redirection) on your system.

To put a string to the standard output the address of the array containing the string needs to be passed to the function. This can be done in one of two ways. Either you pass the array name or a pointer to the array.

```
#include<stdio.h>
```

```
char name [ ] = "The C Tutorial"
```

```
main ( )
{
    char *ptr;
    ptr = &name[0];
    puts (ptr);
    puts (name);
}
```

Copying Strings

In many cases, a string may be copied into another, which is taken care by strcpy function of C library. Internally, this is copying the elements of one array to another. The strcpy() function requires two arguments. The first is the address of the target array into which the string is to be copied, and the second is the address of the source array containing the original string.

```

#include <stdio.h>
char name [20];
main ( )
{
    char buffer [20];
    printf ("please enter your name");
    gets (buffer);
    strcpy ( name, buffer);
    puts (name);
}

```

The above program reads a name into buffer first and then copies it into name. Care must be taken to see that the target array size is at least equals the source size.

Joining Strings

Another important task with strings is joining (also known as concatenation) two strings. This is so common that many languages allow the use of + sign, but unfortunately is not. Concatenating two strings is done with another library function called strcat().

```

#include <stdio.h>
charname[] = "Kumar";
char first_name [] = "Ram";
charbuffer [80];
main ( )
{
    strcpy (buffer,first_name);
    strcat (buffer, "");
    strcat (buffer,name);
    puts (buffer);
}

```

The strcat function takes two arguments, where the first argument is the main string and the second argument is the string to add to the first argument. The first string should be large enough to contain both the strings and the NULL character at the end.

Finding the Length of Strings

Finding the length of a string is important in many cases like checking an empty string, checking if a string is too long etc. C provides a simple function strlen(). This takes only the string as argument for which length is to be obtained and will returns its length.

```

#include<stdio.h>
charname [80];
main ( )

```



```

{
    int length;
    puts("Enter your name");
    gets(name);
    length = strlen (name);
    printf ("Your name is %d characters long.\n",length);
}

```

Comparing Strings

Another standard function that is used frequently is comparing two strings using strcmp(). This is equivalent of == that is used in integer variables. It takes two strings and compares them. If they are same it returns 0, if the first string is greater than the second it returns a positive value or otherwise it returns a negative value. In fact it checks the difference between each character of first string with the corresponding second string's character and if there is a difference, it returns the difference. If both strings are identical, the difference even after the last character is zero, hence zero is returned.

```

#include<stdio.h>
char  answer [80];
main ( )
{
    static char capital [ ]="New Delhi";
    puts ("what is the capital city of the India?");
    gets (answer);
    if (strcmp ( capital, answer) == 0 )
        puts ("That is correct. ");
    else
    puts ("That is wrong");
}

```

Concatenation of Constant Strings

Strings constants are automatically concatenated when they appear side by side. For example:

```
char *cptr = "this is" " a test string";
```

This is equivalent to char *cptr = "this is a test string";

Obtaining parts of a String

Finding a substring from a given string is simply getting required characters into another array and then placing a NULL character at the end of the resulting string. Following example implements two functions to find the substring where the first one uses arrays and the other uses pointers.

Example:

```
char* substring1 (char src[], int start, int len, char dest[])
{

```

```

int I;
for(i=0; i<len; i++)
    {
        dest[i] = src[start+i]; // assign each character
    }
dest[i] = '\0'; // Termination of destination string
return dest;
}

char* substring2 (char* src, int start, int len, char* dest)
{
    int I;
    for(i=0; i<len; i++)
        {
            *(dest+i) = *(src+start+i); // assign each character
        }
    *(dest+i) = '\0'; // Termination of destination string
    return dest;
}

void main()
{
    char src[] = "This is a test string";
    char dest[21];
    char *p;
    p = substring1(src, 5,9, dest); // results in: is a test
    printf("Main String: %s\n", src);
    printf("Substring1 5,9: %s\n", dest);
    printf("Substring1 5,9 with pointer: %s\n", p);
    p = substring2(src,10,11, dest); // results in: test string
    printf("Substring2 10,11: %s\n", dest);
    printf("Substring2 10,11 with pointer: %s\n", p);
}

```

Care should be taken so that start+length is always less than the total length of the string. Similarly the dest array should have enough space to hold the substring plus NULL.

Following is a summary of various string functions available in C which are frequently used:

Function	Header File	Signature	Remarks
strcat	string.h	char *strcat(char *string1, char *string2);	Concatenates string2 to string1 and returns pointer to the first string
strcpy	string.h	char *strcpy(char *string1, char *string2);	Copies string2 into string1 and returns pointer to the first string.
Strncpy	string.h	char *strncpy(char *string1, char *string2, int count);	Copies up to count characters of string2 to string1 and returns pointer to the first string.
Strcmp	string.h	int strcmp(char *string1, char *string2);	Compares the value of string1 to string2 and returns -ve or 0 or +ve number.
Strncmp	string.h	int strncmp(char *string1, char *string2, int count);	Compares up to count characters of string1 and string2 and returns -ve or 0 or +ve number.
Strcasecmp	strings.h	int strcasecmp(char *string1, char *string2);	Compares strings without case sensitivity.
Strncasecmp	strings.h	int strncasecmp(char *string1, char *string2, int count);	Compares strings without case sensitivity up to count characters.
Strlen	string.h	int strlen(char *string);	Calculates and returns the length of string.
Strncat	string.h	char *strncat(char *string1, char *string2, int count);	Concatenates up to count characters of string2 to string1 and returns the first string.
Strchr	string.h	char *strchr(char *string, int c);	Locates the first occurrence of c in string and returns a pointer to that location.
Strrchr	string.h	char *strrchr(char *string, int c);	Locates the last occurrence of c in string and returns a pointer to that point.
Strstr	string.h	char *strstr(char *string1, char *string2);	Returns a pointer to the first occurrence of string2 in string1.

Table 6.2

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

2. What is a string?

.....

.....

.....

6.4 SUMMARY

Computer stores data in bits and bytes only. Every data item occupies one or more contiguous memory cells (i.e., adjacent words or bytes). The number of memory cells required to store a data item depends on the type of data item. For example, a single character will typically be stored in 1 byte (8 bits) of memory, an integer usually requires two or four contiguous bytes, a floating-point number may require four contiguous bytes, and a double-precision quantity may require eight contiguous bytes. The memory cells are sequentially numbered from 0 to maximum memory available. Pointer variables must be declared before they are used in a C program. The declaration is `<data-type> *<variable>` here the interpretation of a pointer declaration is different than the interpretation of other variable declarations. When a pointer variable is declared, the variable name must be preceded by an asterisk (*) which informs the compiler that the variable is a pointer. The data type refers to the data type of the object stored at the address given by the pointer. When pointers are used as arguments to a function, the formal arguments that are pointers must each be preceded by an asterisk. Also, if a function declaration is included in the calling portion of the program, the data type of each argument that corresponds to a pointer must be followed by an asterisk.

String is a chain of characters represented as an array of characters in C language. Hence, there is no special data type as string in c, except that they are treated differently in C. In many cases, a string may be copied into another, which is taken care by strcpy function of C library. Internally, this is copying the elements of one array to another. The strcpy() function requires two arguments. The first is the address of the target array into which the string is to be copied, and the second is the address of the source array containing the original string. Another standard function that is used frequently is comparing two strings using strcmp(). This is equivalent of == that is used in integer variables. It takes two strings and compares them. If they are same it returns 0, if the first string is greater than the second it returns a positive value or otherwise it returns a negative value. In fact it checks the difference between each character of first string with the corresponding second string's character and if there is a difference, it returns the difference. If both strings are identical, the difference even after the last character is zero, hence zero is returned.

6.5 CHECK YOUR PROGRESS MODEL ANSWERS

1. A pointer is an address variable which stores the address of a variable of particular data type
2. A string is stream of characters followed by an end-of-string

6.6 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Explain how to pass pointers to functions with an example.
2. Describe pointer arithmetic with examples
3. Write a program to copy, concatenate strings?

II. Answer the following questions in about 15 lines each

1. Describe advantages of pointers.
2. Explain how to declare pointers of different data types with examples.
3. Describe various built-in functions of string in the table format.

6.7 GLOSSARY

strcmp	:	Compares the value of string1 to string2 and returns -ve or 0 or +ve number
strlen	:	Calculates and returns the length of string
strcasecmp	:	Compares strings without case sensitivity up to count characters.
Strrchr	:	Locates the last occurrence of c in string and returns a pointer to that point.
Strstr	:	Returns a pointer to the first occurrence of string2 in string1

BLOCK - III

DERIVED DATA TYPES

This block gives a brief study on creating, reading, printing one dimensional, two dimensional and multi-dimensional arrays. Various operations on matrices such as addition, subtraction, multiplication, transpose are explained with code examples in C. The concepts such as creating, using the structures, accessing the members of the structures, creating and using the unions are explained with easy to understand code examples. Creating a file, opening a file, reading and writing on to files using different modes, copying, appending, sorting the contents of files are discussed using crystal clear examples. Files are powerful features of C language. Using files any kind of data can stored, manipulated, summarized, and analyzed very easily.

The units included in the block are:

Unit-7: Arrays

Unit-8: Structures and Unions

Unit-9: Files

UNIT- 7: ARRAYS

Contents

- 7.0 Objectives
 - 7.1 Introduction
 - 7.2 One Dimensional, Two Dimensional, Multi-Dimensional Arrays
 - 7.3 Matrix Operations with Arrays
 - 7.4 Storage Classes
 - 7.5 Summary
 - 7.6 Check your progress – Model Answers
 - 7.7 Model Examination Questions
 - 7.8 Glossary
-

7.0 OBJECTIVES

After studying this unit, you should be able to

- explain how to define arrays in C
 - describe various types of arrays in C language
 - explain how to write programs to perform matrix operations with arrays
 - understand storage classes in C
-

7.1 INTRODUCTION

A brief introduction to array is given earlier where the properties of an array are discussed. Many applications require the processing of sets of data items that have common properties (e.g., a set of numerical data, represented by $x_1, x_2 \dots x_n$). In such situations it is often convenient to represent the data as an array, where they will all share the same name (for example, x). The individual data items can be of any data type but should be homogeneous. Each array element is referred to by specifying the array name followed by non negative integer subscripts, where each subscript is enclosed in a pair of square brackets. A function returns a value, then you can store the returned value. If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal. Arrays are defined just like ordinary variables, except that each array name must be accompanied by a size specification (i.e. the number of elements). The expression is usually written as a positive integer constant. In general terms. Matrices are two dimensional arrays in C and all manipulations on matrices can be performed using index values for row and column to access an element. An individual array element within a multidimensional array can be accessed by repeatedly using the indirection operator. Usually, however this procedure is more awkward than the conventional method for accessing an array element. The following example illustrates the use of the indirection operator

7.2 ONE DIMENSIONAL, TWO DIMENSIONAL, MULTI-DIMENSIONAL ARRAYS

A brief introduction to array is given earlier where the properties of an array are discussed.

Many applications require the processing of sets of data items that have common properties (e.g., a set of numerical data, represented by $x_1, x_2 \dots x_n$). In such situations it is often convenient to represent the data as an array, where they will all share the same name (for example, x). The individual data items can be of any data type but should be homogeneous.

Each array element is referred to by specifying the array name followed by non negative integer subscripts, where each subscript is enclosed in a pair of square brackets. Thus, in the n -element array x , the array elements are $x[0], x[1], x[2], \dots, x[n-1]$, as illustrated below.

Element	$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$	$x[6]$	$x[7]$
value	20	30	10	5	0	5	2	1

Table 7.1

Multi Dimensional Array

Multi dimensional arrays can be defined in C with number of square bracket pairs, equal to dimension of the array. $A[]$ represents a single dimensional array, $A[][]$ represents a two dimensional array, $A[][][]$ represents a three dimensional array and so on. For example, $x[i]$ refers to an element in the one-dimensional array x , $y[i][j]$ refers to an element in the two dimensional array y .

Defining an Array

Arrays are defined just like ordinary variables, except that each array name must be accompanied by a size specification (i.e. the number of elements). The expression is usually written as a positive integer constant. In general terms, a multi-dimensional array may be defined as:

`<data-type><array-name>[expression][expression][expression]...;`

Where `data-type` is the data type, `array-name` is the array name, and `expression` is a positive valued integer expression that indicates the number of array elements.

Example:

```
int items[10]; // 10 integers
```

```
int items[10*15]; // 150 integers
```

```
float values[10][3]; // 30 float values as 10 rows x 3 columns
```

```
char cube[3][3][3]; // 27 characters as 3 rows x 3 columns x 3 depth levels
```

```
char text [80]; // 80 characters
```

Array definitions can include the initial values if desired. The initial values must appear in the same order of array elements, enclosed in braces and separated by commas. The general form is

`<data-type><array-name>[expression] = { value1, value2, valuen};`

Example:

```
int digits [10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

```
float x[6] = [0, 0.25, 0, -0.50, 0, 0];
```

```
char color [3] = ['R', 'E', 'D'];
```

```
int table[2][3] = {{ 1,2,3}, {4,5,6}}; // Multi dimensional array initialization
```

If array elements are not initialized, they assume unpredictable values (random) and may yield unpredictable results in the program.

Example : Consider the following array definitions.

```
int digits [5] = {3, 3, 3}; // Partially initialized array with first 3 values (0,1,2) initialized
```

Here the digits[3] and digits[4] are not assigned values, and hence, assume unpredictable values.

The array size need not be specified explicitly when initial values are included as a part of an array definition.

Example:

```
int digits [ ] = {1, 2, 3, 4, 5, 6};
```

```
float x[ ] = {0, 0.25, 0, -0.5};
```

```
char color [3] = "RED"
```

```
char color [ ] = "RED"
```

Passing Array to a Function

An array name without square brackets can be used as an argument to a function, which passes entire array to the function. When an array name is passed, actually passing the address of the first element to the function, thus, it is not same as passing an ordinary variable. The corresponding formal argument in the function declaration is written in the same manner. The last dimension may be omitted by using empty brackets, but others must be defined properly. For example, a one-dimensional array may be defined as a formal argument with empty brackets. The size of the array is not specified within the formal argument declaration.

Example: The following program outline illustrates the passing of an array from the main portion of the program to a function.

```
float average(int, float []); // Function declaration

main ( )
{
    int n=5; // variable declaration
    float avg;// variable declaration
    float list [n]={1.0f,2.0f,3.0f,4.0f,5.0f}; // array definition
    avg = average (n, list); // function call with arguments
    printf("Average: %f\n", avg);
}

// function definition with formal arguments

float average (int n, float x[])
{
    int i;
    float sum;
    for(i=0;i<n; i++)
{
```

```

        sum += x[i];
    }
    return (sum/(float)n);
}

```

Pointers and One Dimensional Arrays

The array name gives the address of the first element of the array, which means it is internally a pointer to the first element in that array. Hence, if x is a one-dimensional array, then $\&x[0]$ and x both refer to the address of the first array element. Moreover, the address of the second array element can be written as either $\&x[1]$ or as $(x + 1)$, and so on. In general, the address of the $(i + 1)^{\text{th}}$ array element can be expressed either as $\&x[i]$ or as $[x + i]$. Here adding i to x (the address of the first element) is known as pointer arithmetic, which has a different meaning than just arithmetic addition. An addition or deletion of i to pointer x gives the address of next or previous i^{th} element relative to x .

When writing the address of an array element in the form $(x+i)$, the C compiler uses the size of each element to advance to the next or previous element automatically. The programmer must specify only the address of the first array element (i.e. the name of the array) and the number of array elements beyond the first (i.e. a value for the subscript). The value of i is also referred to as an offset.

Since $\&x[i]$ and $(x+i)$ both represent the address of the i^{th} element of x , $x[i]$ and $*(x+i)$ both represent the contents of that address, i.e. the value of the i^{th} element of x . The two terms are interchangeable. Hence, either term can be used in any particular application. The choice depends upon the programmer's individual preferences. Following is an example that illustrates the relationship between array elements and their addresses

Example:

```

#include <stdio.h>

main ( )
{
    int x[10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
        int i;
    for (i = 0; i<10; ++i)
    {
        printf ("i= %d x[i]= %d *(x+i)= %d &x[i] = %x x+i=%x\n",
                i, x[i], *(x+i), &x[i], x+i);
    }
}

```

Above program when run, will give you values of each element of the array using both array and pointer representations respectively.

Multi Dimensional Arrays and Pointers

A two dimensional array is a collection of one-dimensional arrays. Therefore, a two-dimensional array can be defined as a pointer to a group of contiguous one-dimensional array. A two dimensional array declaration can be written as

82 <data type>(*<pointer variable>) [<expression2>];

rather than

```
<data type><array name> [<expression 1>][<expression 2>];
```

This concept can be generalized to higher dimensional arrays, that is,

```
<data type> (*<pointer variable>)[<expression 2>][<expression 3>]...[<expression n>];
```

Replaces

```
<datatype><array name> [<expression 1>][<expression 2>]...[<expression n>];
```

In these declarations data type refers to the data type of the array, pointer variable is the name of the pointer variable, array name is the corresponding array name, and expression 1, expression 2, ... expression n are positive valued integer expressions that indicate the maximum number of array elements associated with each subscript.

Notice that the asterisk and the pointer name are surrounded by parentheses. These parentheses must be present. Without them the program will define an array of pointers rather than a pointer to a group of arrays, since these particular symbols (i.e., the square brackets and the asterisk) would normally be evaluated right-to-left.

Example: Suppose that x is a two-dimensional integer array having 10 rows and 20 columns. We can declare x as

```
int(*x)[20];
```

Rather than

```
int x[10][20];
```

In the first declaration, x is defined to be a pointer to a group of contiguous, one-dimensional, 20-element integer arrays. Thus x points to the first 20 element array, which is actually the first row (row 0) of the original two dimensional array. Similarly, (x+1) points to the second 20-element array, which is the second row (row 1) of the original two-dimensional array, and so on, as illustrated below:

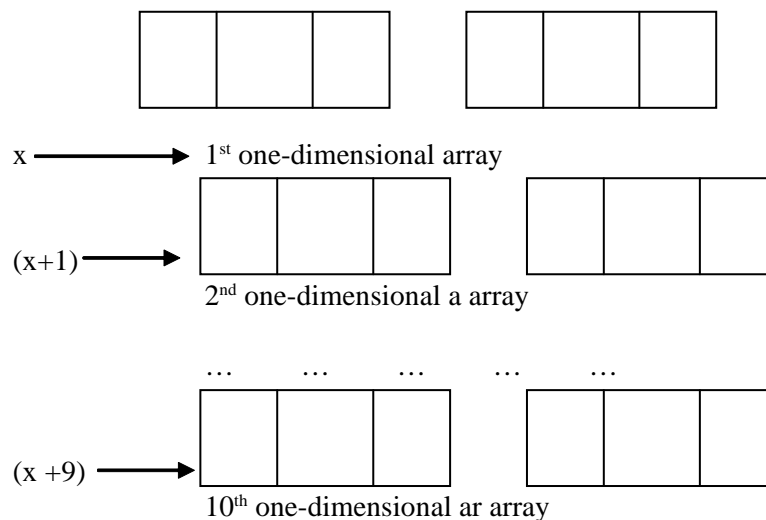


Figure 7.1

In the above example, if size of each element is 2 bytes, $x+1$ advances the pointer by $2 \times 20 = 40$ bytes.

Now consider a three-dimensional floating-point array t . This array can be defined as

```
float (*t) [20][30];
```

instead of

```
float t[10][20][30];
```

In the first declaration, t is defined as a group of contiguous, two-dimensional, 20×30 floating point arrays. Hence, t points to the first 20×30 array, $(t + 1)$ points to the second 20×30 array, and so on.

An individual array element within a multidimensional array can be accessed by repeatedly using the indirection operator. Usually, however this procedure is more awkward than the conventional method for accessing an array element. The following example illustrates the use of the indirection operator.

Example: Suppose that x is a two-dimensional integer array having 10 rows and 20 columns, as declared in the previous example. The item in row 2, column 5 can be accessed by writing either

```
x[2][5] or as *(*(x + 2) + 5)
```

The second form requires some explanation. First, note that $(x+2)$ is a pointer to row 2. Therefore, the object of this pointer, $*(x+2)$, refers to the entire row. Since row 2 is a one-dimensional array, $(x + 2)$ is actually a pointer to the first element in row 2. We now add 5 to this pointer. Hence, $((x+2) + 5)$ is a pointer to element 5 (the sixth element) in row 2. The object of this pointer, $*(*(x+2)+5)$, therefore refers to the item in column 5 of row 2, which is $x[2][5]$.

Programs that make use of multidimensional arrays can be written in several different ways. In particular, there are different ways to define the arrays, and different ways to process the individual array elements. The choice of one method over another is often a matter of personal preference. In applications involving numerical arrays, it is often easier to define the arrays in the conventional manner, thus avoiding any possible subtleties associated with initial memory assignments. The following examples, however, illustrates the use of pointer notation to process multidimensional numerical arrays

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. What is an Array?

.....
.....
.....

7.3 MATRIX OPERATIONS WITH ARRAYS

Matrices are two dimensional arrays in C and all manipulations on matrices can be performed using index values for row and column to access an element. Following example shows how to add or multiply two matrices. The program reads two matrices from keyboard and then adds them to get sum and multiplies them to get product.

```
84 #include <stdio.h>
```

```

// Function to read a given matrix with given rows and columns
void read_matrix(int mat[][], int rows, int cols)
{
    int r, c;
    for (r = 0; r < rows; r++)// for each row
    {
        for (c = 0; c < cols; c++)// for each column
        {
            printf("Enter value for [%d][%d]: ", r, c); // show message
            scanf("%d", &mat[r][c]); // read element
        }
    }
    return;
}

// Function that prints a given matrix with given rows and columns
void print_matrix(int mat[][], int rows, int cols)
{
    int r, c;
    for (r = 0; r < rows; r++) // for each row
    {
        for (c = 0; c < cols; c++) // for each column
        {
            printf("%d\t", mat[r][c]); // print element
        }
        printf("\n"); // end of a row, so print a newline
    }
    return;
}

// Matrix multiplication function that takes
// two input matrices and one output matrix
// row and column sizes of input matrices
// It returns 0 (false) on failure or true if successful and then res will be filled with data
int multiply_matrix(int mat1[][], int mat2[][], int res[][], int r1,int c1, int r2, int c2)
{

```

```

int i, j, k, sum;
if(c1 != r2)
{
    printf("First matrix columns not same as second matrix rows\n");
    return 0; // false
}
for (i = 0; i < r1; i++) for each row
{
    for (j = 0; j < c2; j++) // for each column
    {
        sum = 0;
        for (k = 0; k < c1; k++)
        {
            sum = sum + mat1[i][k]*mat2[k][j];
        }
        res[i][j] = sum;
    }
}
return 1; // true
}

// Matrix addition function that takes
// two input matrices and one output matrix
// row and column sizes of input matrices
// It returns 0 (false) on failure or true if successful and then res will be filled with data
int add_matrix(int mat1[][], int mat2[][], int res[][], int r1,int c1, int r2, int c2)
{
    int i, j, k, sum;
    if(r1 != r2 || c1 != c2)
    {
        printf("Matrix sizes are not same\n");
        return 0;// false
    }

    for (i = 0; i < r1; i++) // for each row
    {

```



```

        for (j = 0; j < c1; j++) // for each column
        {
            res[i][j] = mat1[i][j]+mat2[i][j];
        }
    }
    return 1;// true
}

int main()
{
    int r1, c1, r2,c2, res;
    int mat1[20][20], mat2[20][20], result[20][20];
printf("Array size should be less than 20\n");
printf("Enter number of rows and columns of mat1 matrix: ");
    scanf("%d%d", &r1, &c1);
    read_matrix(mat1, r1,c1);
    printf("Enter number of rows and columns of mat2 matrix: ");
    scanf("%d%d", &r2, &c2);
    read_matrix(mat2, r2,c2);
    printf("Matrix 1\n");
    print_matrix(mat1,r1,c1);
    printf("Matrix 2\n");
    print_matrix(mat2,r2,c2);
    res = multiply_matrix(mat1,mat2,result,r1,c1,r2,c2);
    if(res)
    {
        printf("Product\n");
        print_matrix(result, r1,c2);
        printf("\n");
    }

    res = add_matrix(mat1,mat2,result,r1,c1,r2,c2);
    if(res)
    {
        printf("Sum of Matrices\n");
        print_matrix(result, r1,c1);
    }
}

```

```

        printf("\n");
    }
    return 0;
}

```

The above program can be added the third function for subtraction by suitably modifying the add matrix function.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

2. What is a matrix?

.....

.....

.....

7.4 STORAGE CLASSES

Storage Class defines the scope, visibility and life of a variable during the runtime of a program. C language supports 4 storage classes:

- auto
- extern
- static
- register

Any of these can precede the actual variable definition to alter the storage class of that variable. For example, to make a variable static, it is declared as:

```
static int a = 0;
```

or more generally,

```
<storage class><data type><variable>
```

- **auto:** This is the default storage class for all the variables and hence it may be omitted in declaration. Auto variables are available only within the block and when the block ends, these variables are freed. They are uninitialized when created, which means the initial value is a garbage value (unknown or unpredictable).
- **extern:** Extern storage class allows a variable to be defined in someother block and is used in another block or even in another file of the same project (effectively they are global variables). These must be properly initialized where it is declared before it is used in another block. Generally external storage class is used for variables spanned in different files in large multi file programs.
- **static:** Static variables are declared as global variables with local scope (within block). Hence, they retain the value even after they are out of their scope if they are declared within a block. They are initialized only once and exist till the end of program. By default, they are initialized to 0 by the compiler.

- register:** Register variables are similar to the auto variables but they will be declared in the CPU registers if possible. Hence, it is a request but not guaranteed. If they find space in registers, they can be accessed faster and hence the overall performance will improve. Usually a few frequently used variables are declared as register keywords since the registers in a CPU are limited. Address of a register variable cannot be obtained and hence pointers cannot be used. Following table summarizes the properties of storage classes:

Storage class Specifier	Storage Area	Initial Value	Scope	Life
auto	Stack	Unspecified	Block	End of Block
extern	Data Segment	0	Global Multi-file	End of Program
static	Data Segment	0	Block	End of Program
register	CPU Register or Stack	Unspecified	Block	End of Block

Table 7.2

Example: The following example shows the usage of different storage classes:

```

#include <stdio.h>
int x; // Global variable, which will be used as extern later
void test()
{
    auto int a = 5; // auto is optional
    register int b = 10; // register variable
    static int c = 20;
    extern int x;
    printf("Value of a: %d\n",a);
    printf("Value of b: %d\n",b);
    printf("Value of c: %d\n",c);
    printf("Value of x: %d\n",x);
    a++;
    b++;
    c++;
    x++;
}

```

```

int main()
{
    x = 1; // Initialize x to 10
    test(); // call test function
    printf("Value of x: %d\n",x);
    x=20;
    test(); // call test again
    printf("Value of x: %d\n",x);
    return 0;
}

```

Pay attention to the values of variables printed. The static variable is initialized only once, hence the second call will print the incremented value. Similarly the extern variable can be modified from both *main* and *test* functions.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

3. What are the various storage classes in C?

.....

.....

.....

7.5 SUMMARY

In C, Each array element is referred to by specifying the array name followed by non negative integer subscripts, where each subscript is enclosed in a pair of square brackets. Thus, in the n-element array x, the array elements are x[0], x[1], x[2],... x[n-1]. Multi dimensional arrays can be defined in C with number of square bracket pairs, equal to dimension of the array. A[] represents a single dimensional array, A[][] represents a two dimensional array, A[][][] represents a three dimensional array and so on. Programs that make use of multidimensional arrays can be written in several different ways. In particular, there are different ways to define the arrays, and different ways to process the individual array elements. The choice of one method over another is often a matter of personal preference. In applications involving numerical arrays, it is often easier to define the arrays in the conventional manner, thus avoiding any possible subtleties associated with initial memory assignments. Storage Class defines the scope, visibility and life of a variable during the runtime of a program. C language supports 4 storage classes known as auto, extern, register and static. The array name gives the address of the first element of the array, which means it is internally a pointer to the first element in that array. Hence, if x is a one-dimensional array, then &x[0] and x both refers to the address of the first array element. Moreover, the address of the second array element can be written as either &x[1] or as (x + 1), and so on. In general, the address of the (i + 1)th array element can be expressed either as &x[i] or as [x + i]. Here adding i to x (the address of the first element) is known as pointer arithmetic, which has a different meaning than just arithmetic addition. An addition or deletion of i to pointer x gives the address of next or previous ith element relative to x.

7.6 CHECK YOUR PROGRESS MODEL ANSWERS

1. Arrays are defined just like ordinary variables, except that each array name must be accompanied by a size specification
2. A matrix is represented as a two dimensional array in C language
3. auto, extern, static, register

7.7 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Explain how to read, print one dimension, two dimensional arrays.
2. Describe pointers and arrays with examples programs
3. Write a program to add and multiply two matrices?

II. Answer the following questions in about 15 lines each

1. Write a program to pass arrays as function arguments.
2. Explain various storage classes in C.
3. Write a program to demonstrate various storage classes in C.

7.8 GLOSSARY

Array : A group of variables stored consecutively of similar data type

Two Dimensional Array: An array contains rows and columns just like a table

static : Storage class in C, which enable sharing a variable for all modules

Matrix : Represented as two dimensional array in C

UNIT- 8: STRUCTURES AND UNIONS

Contents

- 8.0 Objectives
- 8.1 Introduction
- 8.2 Structures
- 8.3 Unions
- 8.4 Summary
- 8.5 Check your progress – Model Answers
- 8.6 Model Examination Questions
- 8.7 Glossary

8.0 OBJECTIVES

After studying this unit, you should be able to

- explain how to define and use structures in C
- describe typedef statement in C language
- understand pointers to structures
- explain how to pass structures to functions in C

8.1 INTRODUCTION

Structure is a collection of heterogeneous data items. Consider that name, age and salary are the three attributes pertaining to a person. These can be represented with an array of characters (name), integer (age) and a float number (salary). In C language. The members of a structure can be processed as separate entities by accessing individual structure members. A structure member can be accessed with a dot as <variable>.<member> where variable is a structure type variable, and member refers to the name of one of its members. The dot between variable name and its member is an operator, which is in the highest precedence group, and its associativity is left-to-right. The beginning address of a structure can be accessed with ampersand (&) just like any other variable. A pointer to the structure type variable is defined as type*ptvar where type is a data type that identifies the composition of the structure, and ptvar represents the name of the pointer variable. A pointer can be assigned the beginning address of a structure variable using:ptvar = &variable. Unions resemble structures and contain members of different data types just like structures. However, unlike structures, these members share the same storage area in the memory, whereas each member within a structure is assigned its own unique storage area. They are useful where an application uses multiple variables but only one variable is used at a time, thus conserves space. Since a union shares common location for all its members, the size will be decided by the largest member. The user is responsible to keep track of what type of information is stored at any given point of time. An attempt to access the wrong type of information will produce meaningless results. Unions resemble structures and contain members of different data types just like structures. However, unlike structures, these members share the same storage area in the memory, whereas each member within a structure is assigned its own unique storage area. They are useful where an application uses multiple variables but only one variable is used at a time, thus conserves

space. Since a union shares common location for all its members, the size will be decided by the largest member.

8.2 STRUCTURES

Structure is a collection of heterogeneous data items. Consider that name, age and salary are the three attributes pertaining to a person. These can be represented with an array of characters (name), integer (age) and a float number (salary). In C language, this can be represented with

```
char name[80]; int age; float salary;
```

Now if there are many persons to be represented, one option is to declare each attribute as an array as

```
char name[10][80], int age[10], float salary[10];
```

where the number of persons is assumed to be 10.

However, to access the details of one person, each array has to be accessed with corresponding index, which is impractical and error prone if the number of attributes is large. C provides a convenient way to create a new data type within the program that contains other elements as members which is known as *structure*. A structure is handy in such situations, where the data is heterogeneous but related.

Defining a Structure

A structure must be defined in terms of its individual members as follows:

```
struct tag {  
    member 1;  
    member 2;  
    ...  
    member m;  
} [optional list of variables of this type separated by commas];
```

In this declaration, 'struct' is the required keyword, tag is a name that identifies structures of this type, and member 1, member 2 ..., member m are individual member declarations.

The individual members can be ordinary variables, pointers, arrays, or other structures. Following points may be remembered when using structures:

The member names within a particular structure must be distinct from one another. However, a member name can be the same as the name of a variable defined outside of the structure.

A storage class cannot be assigned to an individual member

Individual members cannot be initialized within a structure-type declaration.

Once the composition of the structure has been defined, individual structure-type variables can be declared as follows:

```
<storage class> struct tag <variable 1, variable 2, ....., variable n>;
```

Where storage-class is optional, *struct* is a required keyword, tag is the name that appeared in the structure type declaration, and variable 1, variable 2, ..., variable n are variables of type *struct tag*.

Example : A typical structure declaration is shown below.

```

struct person {
    char name[80];
    int age;
    float salary;
};

```

Usinga Structure

The members of a structure can be processed as separate entities by accessing individual structure members. A structure member can be accessed with a dot as <variable>.<member>

Where variable is a structuretype variable, and member refers to the name of one of its members. The dot between variable name and its member is an operator, which is in the highest precedence group, and its associativity is left-to-right.

Example: Consider the following structure declarations:

```

struct date {
    int day;
    int month;
    int year;
};

struct account {
    char name[80];
    int acct_no;
    char acct_type;
    float balance;
    struct date opened_date;
} customer;

```

In this example customer is a variable of type account and opened_date inside account is a type of date structure which has its own members (day month, and year). Any element within the customer structure can be accessed using a dot operator as follows:

customer, acct_no, customer.opened_date.year, customer.name and so on.

The dot operator takes precedence over the unary operators as well as the various arithmetic, relational, logical and assignment operators since it is a member of the highest precedence group.. For example, an expression of the form ++customer.acct_no is equivalent to ++(variable.member) which means, the ++ operator will apply to the structure member, not the entire structure variable. Similarly, the expression &variable.member is equivalent to &(variable member); thus, the expression accesses the address of the structure member, not the starting address of the structure variable.

User-Defined Data Types (typedef)

The typedefallows users to define new data types that are equivalent to existing data types. Once a user-defined data type is defined, new variables, arrays, structures, and so on, can be declared in terms of this new data. The syntax for typedef is:


```
typedef <existing data type><new data type>;
```

The new data type will be new in name only. In reality, this new data type will not be fundamentally different from one of the standard data types.

Example: Here is a simple declaration involving the use of typedef.

```
typedef int age ;
```

In this declaration age is a user-defined data type equivalent to type int. Hence, the variable declaration

```
agemale, female;
```

is equivalent to writing

```
int male, female;
```

The advantage of declaring age as integer is that if the data type is to be changed, it is done only at the definition, and it reflects throughout the program. For example, *typedef unsigned char age;* will automatically convert all variables to unsigned characters in one stroke. The typedef feature is very convenient when defining structures, since it eliminates the need to repeatedly write *struct tag* whenever a structure is referenced. As a result, the structure can be referenced more concisely. In addition, the name given to a user-defined structure type often suggests the purpose of the structure within the program. A user-defined structure type can be written as:

```
typedef struct{  
    member 1;  
    member 2;  
    . . .  
    member m;  
}new-type;
```

where new-type is the user-defined structure type. Structure variables can then be defined in terms of the new data type. Following example creates a new data type called “record” and declares some variables of type record.

```
typedef struct{  
    char name [80];  
    int acct-no;  
    char acct-type;  
    float balance;  
}record;
```

```
record oldcustomer, newcustomer;
```

A structure can be initialized while declaring a variable by assigning data in curly braces as follows:

```
record oldcustomer = {"Customer name", 111, 'R', 1000.1};
```

Here the members are assigned corresponding values in the same order of the member declaration.

Pointers to Structures

The beginning address of a structure can be accessed with ampersand (&) just like any other variable. A pointer to the structure type variable is defined as

```
type*ptvar;
```

Where type is a data type that identifies the composition of the structure, and ptvar represents the name of the pointer variable. A pointer can be assigned the beginning address of a structure variable using:

```
ptvar = &variable;
```

Example : Consider the following structure declaration:

```
typedef struct{
    char name [80];
    int acct-no;
    char lacct-type;
    float balance;
}    account;
accountcustomer,*pc;
```

In this example, customer is a structure variable of type account, and pc is a pointer variable whose object is a structure variable of type account. Thus, the beginning address of customer can be assigned to pc by writing

```
pc=&customer;
```

An individual structure member can be accessed in terms of its corresponding pointer variable by writing

```
ptvar->member which is a short form for (*ptvar).member
```

Example:

```
typedefstruct{
    int month;
    int day;
    int year;
}    date;
struct{
    int    acct_no;
    char acct_type;
    char name[80];
    float balance;
    date last_payment;
}    customer, *pc = &customer;
```

Notice that the pointer variable pc is initialized by assigning it the address of the structure variable customer. In other words, pc will point to customer (start address of customer to be

Now `acct_no` can be accessed in any of the following three methods:

```
customer.acct_no (or) pc->acct_no (or) (*pc).acct_no
```

The parentheses are required in the last expression because the period operator has a higher precedence than the indirection operator (*). Without the parentheses the compiler generates an error, because `pc` (a pointer) is not directly compatible with the dot operator.

Month of the last payment can be accessed by writing any of the following:

```
customer.last_payment.monthpc->last_payment.month(*pc).last_payment.month
```

Finally, the customer's name can be accessed by writing any of the following:

```
customer.name pc->name (*pc).name
```

Therefore, the third character of the customer's name can be accessed by writing any of the following.

```
customer.name[2]pc->name[2>(*pc).name[2]
```

```
*(customer.name +2)pc->(name +2)((*pc).name + 2)
```

A structure can also include one or more pointers as members. Thus, if `ptmember` is a pointer of a variable, then `*variable.ptmember` will access the value to which `ptmember` points. Similarly, if `ptvar` is a pointer variable that points to a structure and `ptmember` is a member of that structure, then `*ptvar->ptmember` will access the value to which `ptmember` points.

Example: Following example declares an array of new user defined data type "person". It then reads two records from user and prints them using pointers.

```
#include <stdio.h>

typedef struct _person {
    char name[80];
    long number; } person;

void main()
{
    person p[2];
    int i;
    for(i=0; i<2; i++)
    {
        printf("Enter person %d's name:", i+1);
        gets(p[i].name);
        printf("Enter person %d's Number:", i+1);
        scanf("%ld", &(p[i].number));
        getchar();
    }
}
```

```

    for(i=0;i<2;i++)
    {
        printf("Person %d's Name is: %s and number is %ld\n",
i+1, (p+i)->name, (p+i)->number);
        /*
        The above is same as
        printf("Person %d's Name is: %s and number is %ld\n",
i+1, p[i].name, p[i].number);
        (or)
        printf("Person %d's Name is: %s and number is %ld\n",
i+1, (*(p+i)).name, (*(p+i)).number);
        */
    }
}

```

The program first declares a new data type called person which has two members (name and number). It then declares an array of persons (p[2]). It then reads the data from keyboard using a for loop. Notice that there is a *getchar()* function call at the end of reading *for* loop which will exhaust the last enter character the user gives when entering the number. Care must be taken always to flush the keyboard enter character when reading strings, which otherwise takes the enter character as input and returns empty string before the user enters the actual data. It then prints the data using the second for loop. Alternative use of pointers is also shown in comments.

Passing Structures to Functions

A structure can be passed to a function as a parameter or can be returned from function. The element is passed to a function using call by value mechanism which means it sends a copy of the actual data and hence, any changes to the data in the function do not reflect back in the calling function. If the changes made to a structure argument are to reflect in called function, a pointer to the structure should be passed. The above program can be rewritten as follows to demonstrate the structure passed to a function.

```

#include <stdio.h>
typedef struct _person {
    char name[80];
    long number; } person;
void printData(person, int); // Function declaration
void readData(person *, int); // Function declaration
void main()
{
    person p[2];

```

```

    int i;
    for(i=0; i<2; i++)
    {
        readData(p+i, i);
    }
    for(i=0; i<2; i++)
    {
        printData(p[i], i);
    }
}
void readData(person *p, int i)
{
    printf("Enter person %d's name:", i+1);
    gets(p->name);
    printf("Enter person %d's Number:", i+1);
    scanf("%ld", &(p->number));
    getchar();
    return;
}
void printData(person p, int i)
{
    printf("Person %d's Name is: %s and number is %ld\n", i+1, p.name, p.number);
}

```

In the above program, the reading and writing parts are moved to functions. Since the reading requires changes to be reflected back in main program, it is sent as a pointer. The printing part doesn't change anything and hence data is sent as a value. Since the two functions are declared after the main function, which uses these functions, C assumes them to return integer and the functions are declared to return nothing (void) later, which is a conflict. To resolve, we declare the function as prototype above main as given in the program.

Self Referential Structures

It is sometimes necessary to include a pointer of the same type within the definition of the structure. Note that a pointer can only be declared inside the structure definition but not a variable of the same type. If a variable is declared, its size becomes infinity and the compiler terminates with a stack overflow message.

```

structtag {
    member 1;
    member2;
    ...
}

```

```

    struct tag *name;
};

```

where name refers to the name of a person and next references an element of the same tag type variable. Thus, the structure of type tag will contain a member that points to another structure of type tag. Such structures are known as self-referential structures. This type of definition is used extensively in linked lists, trees and other data structures which beyond the scope of this text.

Example:

```

struct list_element{
    char name[40];
    struct list_element*next;
};

```

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. What is structure in C?

.....

.....

.....

8.3 UNIONS

Unions resemble structures and contain members of different data types just like structures. However, unlike structures, these members share the same storage area in the memory, whereas each member within a structure is assigned its own unique storage area. They are useful where an application uses multiple variables but only one variable is used at a time, thus conserves space. Since a union shares common location for all its members, the size will be decided by the largest member.

The user is responsible to keep track of what type of information is stored at any given point of time. An attempt to access the wrong type of information will produce meaningless results. In general terms, the composition of a union may be defined as:

```

union tag {
    member 1;
    member 2;
    ...
    member m;
};

```

where union is the required keyword and the others are members of the union just as in case of a structure definition. The tag is optional in this type of declaration. Variables of this type can be defined like structure type variables as *union tag x*; If typedef is used, the new type can be used to declare variables.

Arrays of unions can also be declared and used as in case of structures. A pointer to the union also acts just like a pointer to a structure, except that it always points to the start address for all the elements. Following example explains the basic usage of union.

```
#include <stdio.h>

typedef union _tag
{
    long int intdata;
    char strdata[20];
    float floatdata;
    char chardata;
} storage;

void main()
{
    storage st, *p;
    p = &st; // Assign address of structure to pointer p
    printf("Address of int: %x\n", &(st.intdata));
    printf("Address of float: %x\n", &(st.floatdata));
    printf("Address of int: %x\n", &(st.strdata));
    printf("Address of int: %x\n", &(st.chardata));

    printf("Size of union: %d\n", sizeof(st));
    printf("Enter a name: ");
    gets(st.strdata);
    printf("Name: %s\n", st.strdata);
    printf("Name with pointer: %s\n", p->strdata);
    printf("Enter a float number: ");
    scanf("%f", &st.floatdata);
    printf("Float: %f\n", st.floatdata);
    printf("Float with pointer: %f\n", p->floatdata);
    printf("Enter an integer: ");
    scanf("%d", &st.intdata);
    printf("Integer: %d\n", st.intdata);
    printf("Integer using pointer: %d\n", p->intdata);
    printf("Character: %c\n", st.chardata);
    // prints the first byte of integer as character, and has no meaning
}
```

Here we have four union variables, `intdata`, `strdata`, `floatdata` and `chardata`. The size is computed based on the largest element, which is `strdata` which takes 20 bytes. Hence, the size of union is 20 bytes. If it were a structure, it would have taken $(4+20+4+1 = 29)$ bytes but many compilers use word boundary (divisible by word size which is normally 2 or 4 bytes depending on the compiler) which makes it 30 or 32 bytes depending on the compiler. The `sizeof` operator returns the actual size computed by the compiler. Access to members is illustrated both with dot notation and with a pointer notation.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

2. What is union?

.....
.....
.....

8.4 SUMMARY

Structure is a collection of heterogeneous data items. Consider that name, age and salary are the three attributes pertaining to a person. These can be represented with an array of characters (name), integer (age) and a float number (salary). An individual structure member can be accessed in terms of its corresponding pointer variable by writing `ptvar->member` which is a short form for `(*ptvar).member`. A structure can also include one or more pointers as members. Thus, if `ptmember` is a pointer of a variable, then `*variable.ptmember` will access the value to which `ptmember` points. Similarly, if `ptvar` is a pointer variable that points to a structure and `ptmember` is a member of that structure, then `*ptvar->ptmember` will access the value to which `ptmember` points. A structure can be passed to a function as a parameter or can be returned from function. The element is passed to a function using call by value mechanism which means it sends a copy of the actual data and hence, any changes to the data in the function do not reflect back in the calling function. If the changes made to a structure argument are to reflect in called function, a pointer to the structure should be passed. It is sometimes necessary to include a pointer of the same type within the definition of the structure. Note that a pointer can only be declared inside the structure definition but not a variable of the same type. If a variable is declared, its size becomes infinity and the compiler terminates with a stack overflow message. Unions resemble structures and contain members of different data types just like structures. However, unlike structures, these members share the same storage area in the memory, whereas each member within a structure is assigned its own unique storage area. They are useful where an application uses multiple variables but only one variable is used at a time, thus conserves space. Since a union shares common location for all its members, the size will be decided by the largest member.

8.5 CHECK YOUR PROGRESS MODEL ANSWERS

1. Structure is a collection of heterogeneous data items called as members
2. Unions resemble structures and contain members of different data types just like structures. However, unlike structures, these members share the same storage area in the memory

8.6 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Write a program to demonstrate pointers and structures
2. Explain how to pass structures as arguments to functions with code examples
3. Write a program to demonstrate unions?

II. Answer the following questions in about 15 lines each

1. Describe self referential structures.
 2. Explain typedef in C
 3. Write a program to demonstrate defin and use structure
-

8.7 GLOSSARY

Structure	:	Structure is a collection of heterogeneous data items called as members
Union	:	Unions resemble structures and contain members of different data types just like structures. However, unlike structures, these members share the same storage area in the memory
typedef	:	Defines user defined data types
ACCU	:	Association of C and C++ Users.

UNIT- 9: FILES

Contents

- 9.0 Objectives
 - 9.1 Introduction
 - 9.2 File Basics
 - 9.3 Operations on Files and Command line Arguments
 - 9.4 Summary
 - 9.5 Check your progress – Model Answers
 - 9.6 Model Examination Questions
 - 9.7 Glossary
-

9.0 OBJECTIVES

After studying this unit, you should be able to

- explain how to create, open, read files in C
 - explain how to find location of content in files
 - understand copying, concatenating, sorting files
 - understand commandline arguments in C
-

9.1 INTRODUCTION

A file is a sequence of data stored on the permanent storage medium like hard disk. The programmer has freedom to choose where to store the file. In fact, in C, all input and output is handled as files. For C the input comes from a special file which is designated as the standard input device (generally the keyboard), and the output is sent to a special file known as standard output device (generally the screen). A file can be opened in many ways as shown in the table below. A file type can also be specified as text file or binary file. A text file is human readable and the binary files are raw files, generally not readable by human beings. Every file is appended with a special imaginary symbol known as “End of File”, defined as an integer constant in `stdio.h` as EOF (generally represented with -1) to ensure that the program will terminate up on encountering EOF. Every program should handle this EOF character to see that the program avoids reading beyond the last character of the file. It is often useful to know where the current location is in a file. This is especially important in case of random accessing of a file. The function `ftell()` returns the offset of the file pointer from the start of the file as a long number. It is very common to pass arguments to a program when the program is invoked from the command line. For example, the `ls` program in Unix or `dir` command in Windows can take arguments as `ls a*` or `dir a*` where `ls` or `dir` is the command and `a*` is the argument. This is supported in C by modifying the main function’s parameter list. Information about command line arguments is passed by the operating system to the entry function of the invoked program by way of parameters, which is the `main()` function.

9.2 FILE BASICS

A file is a sequence of data stored on the permanent storage medium like hard disk. The programmer has freedom to choose where to store the file. In fact, in C, all input and output is

handled as files. For C the input comes from a special file which is designated as the standard input device (generally the keyboard), and the output is sent to a special file known as standard output device (generally the screen).

There are three operations that can be performed with a file, viz., open it, access it, and close it. A file is opened explicitly to read, or to be written or both. C provides a standard library for all file based operations.

A file must be opened before it is accessed and must be closed when nothing more need to be done. When a file is opened, C provides a pointer to a special structure called FILE, which holds information pertaining to the file like the name, size, current location etc. This pointer is now the handle for the file, and all operations require this pointer as reference, including the close operation. The reading and writing operations on file are done exactly the same way as they are done with keyboard and screen except that the function names are fprintf, fscanf, fputs, fgets, fgetc, fputc etc. All these functions take the pointer to FILE structure as an extra parameter.

Opening and Closing a File

A file can be opened using fopen command and can be closed with fclose command.

```
FILE *<pointer> = fopen(<file name>, <opening mode>);
```

```
fclose(<pointer>);
```

where pointer is the pointer variable that holds the address of a FILE structure if properly opened, file name is the name of the file with full path and extension and opening mode is a string describing the mode to open.

File opening modes

A file can be opened in many ways as shown in the table below. A file type can also be specified as text file or binary file. A text file is human readable and the binary files are raw files, generally not readable by human beings.

File – Type Specifications

File-Type	Meaning
“r”	Open an existing file for reading only.
“w”	Open a new file for writing only. If a file with the specified file-name currently exists, it will be destroyed and a new file created in its place.
“a”	Open an existing file for appending (i.e. for adding new information at the end of the file). A new file will be created if the file with the specified file-name does not exist.
“rw”	Open an existing text file for read and write
“r+”	Open an existing file for both reading and writing.
“w+”	Open a new file for both reading and writing. If a file with the specified file-name currently exists, it will be destroyed and a new file created in its place.
“a+”	Open an existing file for both reading and appending. A new file will be created if the file with the specified file-name does not exist.
“rb”	Open an existing binary file for reading
“wb”	Open a new file for binary writing
“ab”	Open a binary file for appending
“r+b” or “w+b”	Open a binary file for read and write

Table 9.1

Example: Following code snippet gives an example of how a file can be opened:

```
File *fp;
fp = fopen("abc.txt", "r"); // Open the file as a text file for reading only
fp = fopen("abc.txt", "w"); // File will be opened for writing. Any existing file will be lost
fp = fopen("abc.txt", "r+w"); // File will be created if doesn't exist, or opened if exists.
fp = fopen("abc.txt", "a"); // File will be created or opened to append data at the end
fp = fopen("abc.exe", "rb"); // File will be opened in binary read only mode
fp = fopen("abc.exe", "wb"); // File will be opened in binary write mode
fp = fopen("abc.exe", "a+b"); // File will be opened in binary append and read mode
....
```

In all the cases, the next step is to check whether the file is opened or failed to open. This can be done by checking if fp is NULL, which indicates a failure.

```
if(fp == NULL)
{
    printf("Cannot open the file\n");
    exit(1);
}
```

If it is not NULL, the file is properly opened and ready for read/write operations. Finally the file must be closed with fclose function:

```
fclose(fp); // Close the file associated with pointer fp.
```

A skeletal program to use a file is given below:

```
#include <stdio.h>
FILE *fpt;

fpt = fopen("sample.dat", "r+");
if (fpt == NULL)
{
    printf ("\nERROR – Cannot open the file\n");
}
else {
    . . . // Actual file operations like read/write goes here
    fclose (fpt);
}
```

End of File (EOF)

Every file is appended with a special imaginary symbol known as “End of File”, defined as an integer constant in stdio.h as EOF (generally represented with -1) to ensure that the program will terminate up on encountering EOF. Every program should handle this EOF character to see that the program avoids reading beyond the last character of the file.

Implicit Files

C opens three files automatically when the `stdio.h` is included and the pointers to these three files are available with standard names. These are “`stdin`” for input, “`stdout`” for output and “`stderr`” for error output. Generally `stderr` and `stdout` refer to the screen (console). These are automatically closed when the program ends and there is no necessity to explicitly close these files.

Example: Following program creates a new file and writes some data into it. The file is created in the default directory if path is not specified.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    FILE *fp;
    char *fname = “test.txt”;
    fp = fopen(fname, “w”);
    if(fp == NULL)
    {
        printf(“Can not open file %s\n”, fname);
        exit(1);
    }
    fprintf(fp, “This is a test file line 1\n”);
    fprintf(fp, “This is a test file line 2\n”);
    fprintf(fp, “This is a test file line 3\n”);
    fprintf(fp, “This is a test file line 4\n”);
    fprintf(fp, “This is a test file line 5\n”);
    fclose(fp);
}
```

In the above program, a file is opened with a name taken from a variable (`fname`) and some lines are printed to it using `fprintf` and finally the file is closed. After the run of this program, the file “`test.txt`” can be found in the current directory (where the exe file is created) with the given content.

Example: Following is a full example that reads the file contents and prints the data on the screen. Open a text editor and create the text file “`test.txt`” in the same folder where the compiled executable file exists or give the full path to the `test.txt` file something like “`C:\\users\\admin\\Desktop\\test.txt`”. Note that the ‘`\`’ is escaped with one more ‘`\`’ as it has a special meaning in strings in C.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
```

```

FILE *fp;
char *fname = "test.txt";
int ch;

fp = fopen(fname, "r");
if(fp == NULL)
{
    printf("Can not open file %s\n", fname);
    exit(1); // Exit the program with error code 1 (which may be used by the shell)
}
while((ch=fgetc(fp)) != EOF)
{
    putchar(ch, stdout);
}
fclose(fp);
}

```

In the above program the `stdlib.h` is included to get the definition of "`exit(int n)`" function, which terminates the C program at that point. After opening the file, the program enters a while loop where each character is read into `ch` and checked for EOF. If it is not EOF, print that character using `putc(character, file pointer)` method, where the output file pointer is `stdout`, which is by default available to all programs. Finally close the file after the while loop terminates on encountering EOF.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. What is EOF?

.....

.....

.....

9.3 OPERATIONS ON FILES AND COMMND LINE ARGUMENTS

Following program copies a file to another file by opening two files one for reading and the other for writing, then copies character by character from source to destination.

```

#include <stdio.h>
#include <stdlib.h>
void main()
{
    FILE *fpin, *fpout;

```

```

char *finname = "test.txt";
char *foutname = "test-out.txt";
int ch;

fpin = fopen(finname, "r");
if(fpin == NULL)
{
    printf("Can not open file %s\n", finname);
    exit(1);
}

fpout = fopen(foutname, "w");
if(fpout == NULL)
{
    printf("Can not open file %s\n", foutname);
    exit(1);
}

while((ch=fgetc(fpin)) != EOF)
{
    fputc(ch, fpout);
}

fclose(fpin);
fclose(fpout);
}

```

Example: Following program appends one file to another using append mode. It is almost equivalent to the file copy except that the output file is opened in append mode instead of write mode.

```

#include <stdio.h>
#include <stdlib.h>
void main()
{
    FILE *fpin, *fpout;
    char *finname = "test.txt";
    char *foutname = "test-out.txt";
    int ch;

```

```

fpin = fopen(finname, "r");
if(fpin == NULL)
{
    printf("Can not open file %s\n", finname);
    exit(1);
}

fpout = fopen(foutname, "a");
if(fpout == NULL)
{
    printf("Can not open file %s\n", foutname);
    exit(1);
}

while((ch=fgetc(fpin)) != EOF)
{
    fputc(ch, fpout);
}
fclose(fpin);
fclose(fpout);
}

```

Finding the Current Location

It is often useful to know where the current location is in a file. This is especially important in case of random accessing of a file. The function `ftell()` returns the offset of the file pointer from the start of the file as a long number.

```
long <offset variable> = ftell(<file pointer>);
```

Random Access

All the file operations in C are designed to read or write data sequentially starting from first byte onwards, which is known as sequential access. The only variation is the append mode which automatically goes to the end of the file so that any writing will append data to the end. Sometimes it is useful to be able to read or write something from a given location of the file. This ability to directly go to a specific location of a file is called random or direct access. The function `fseek()` enables a program to go directly to a specific location of the file and the file operations can be performed starting at that location. To do this, `fseek()` requires three parameters: the filepointer, an offset, and a value telling it how to use the offset. The standard syntax is :

```
fseek ( filepointer, offset, how);
```

There are three constants defined in `stdio.h` as `SEEK_START`, `SEEK_CUR`, `SEEK_END` which can be used with `how`. If `how` is defined as `SEEK_START`, the offset is measured from the start

of the file. If it is defined as `SEEK_CUR`, then the offset is measured from the current location and if it is defined as `SEEK_END`, the offset is measured from the end of the file.

Example: Following program opens a file in read+write mode and shows the 6th character from the file and then changes the first character (0th character) to X.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    FILE *fp;
    char *fname = "test.txt";
    int ch;

    fp = fopen(fname, "r+"); // read and write mode
    if(fp == NULL)
    {
        printf("Can not open file %s\n", fname);
        exit(1);
    }
    // Get 6th character in the file
    fseek(fp, 6, SEEK_SET);
    printf("6th character in the file is: %c\n", fgetc(fp));

    // Change 1st character to X
    fseek(fp, 0, SEEK_SET);
    fputc('X', fp);

    fclose(fp);
}
```

Example: Following program reads numbers from a file and sorts them using the library function `qsort` defined in `stdlib.h`. `qsort()` is the function that can be called with a pointer to the array of data, the number of elements in that array, the size of each element and a comparison function.

```
#include <stdio.h>
#include <stdlib.h>
int compare (const void *val1, const void *val2);

void main()
```

```

{
FILE *fp;
int data[20];

char *fname = "data.txt";
int val, count, i;
char line[80];

fp = fopen(fname, "r");
if(fp == NULL)
{
    printf("Can not open file %s\n", fname);
    exit(1);
}
/* —— Data reading part —— */
count = 0;
for(i=0;i<20;i++)
{
    iffeof(fp)
    {
        break; // If end of file, break from the loop
    }
    fgets(line, 80, fp);
    if(strlen(line) == 0)
    {
        break; // If line is NULL, break from the loop
    }
    sscanf(line, "%d", &val); // Read one integer from string
    data[count] = val; // Add the integer to the data
    count++; // and increment the count by one
}
fclose(fp);
/* — Print original data — */
printf("Data before sort\n—————\n");
for(i=0;i<count;i++)
{

```

```

        printf("%d\n", data[i]);
    }
    qsort(data,count,sizeof(int),compare); // Actual sort function call
    /* — Print sorted data — */
    printf("Data after sort\n—————\n");
    for(i=0;i<count;i++)
    {
        printf("%d\n", data[i]);
    }
}

int compare (const void *val1, const void *val2)
{
    int *p1 = (int *) val1;
    int *p2 = (int *) val2;
    if (*p1 > *p2) return 1;
    if (*p1 < *p2) return -1;
    return 0;
}

```

The file is opened and each line is read using fgets function which takes three arguments, the character array into which data is read, maximum length of array (including NULL at the end) and the file pointer. Before reading a line it checks if EOF is encountered by using the function feof(file pointer) function which returns zero if not EOF or non-zero (true) if EOF is reached. From the string, a number is read using sscanf which acts like scanf except that it reads data from a string instead of keyboard (stdin). The number is then copied into data array and the counter is incremented.

The function “compare” uses pointers of type void *, which is defined by the library function. So each pointer is to be casted as pointer of type int * before the values can be compared. The compare function should return +1 if first value is greater than second, -1 if second value is greater than first, or zero if both are equal.

To run this program a file “data.txt” is to be created with any text editor and add some numbers, one per line. The file should be in the same directory where the executable file is created.

Command Line Arguments

It is very common to pass arguments to a program when the program is invoked from the command line. For example, the ls program in Unix or dir command in Windows can take arguments as ls a* or dir a* where ls or dir is the command and a* is the argument. This is supported in C by modifying the main function’s parameter list. Information about command line arguments is passed by the operating system to the entry function of the invoked program by way of parameters, which is the main() function.

To make use of the command line arguments which are passed to main(), it needs to be modified accordingly. C supports arguments to main which is not explored so far. It takes two arguments. The first one is a number that gives how many arguments there are and the second one is an array of strings containing the actual arguments. Thesetwo arguments are traditionally given the names argc and argv though they can be changed. To utilize the arguments, change the main program as follows:

```
main (int argc, char *argv [ ] )
```

Following program lists all the arguments passed from the command line:

```
#include <stdio.h>

int main (int argc, char *argv[])
{
    int i;
    printf ("Number of arguments: %d\n", argc);
    for(i=0; i<argc; i++)
    {
        printf("Argument %d: %s\n", i, argv[i]);
    }
}
```

In the above program, argc is the count of arguments passed, and argv is an array of pointers to strings.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

2. What is command line argument?

.....

.....

.....

9.4 SUMMARY

In general, A file is a sequence of data stored on the permanent storage medium like hard disk. A file must be opened before it is accessed and must be closed when nothing more need to be done. When a file is opened, C provides a pointer to a special structure called FILE, which holds information pertaining to the file like the name, size, current location etc. This pointer is now the handle for the file, and all operations require this pointer as reference, including the close operation. The reading and writing operations on file are done exactly the same way as they are done with keyboard and screen except that the function names are fprintf, fscanf, fputs, fgets, fgetc, fputc etc. All these functions take the pointer to FILE structure as an extra parameter. Every file is appended with a special imaginary symbol known as “End of File”, defined as an integer constant in stdio.h as EOF (generally represented with -1) to ensure that the program will terminate up on encountering EOF. Every program should handle this EOF character to see that the program avoids reading beyond the last character of the file. All the

onwards, which is known as sequential access. The only variation is the append mode which automatically goes to the end of the file so that any writing will append data to the end. Sometimes it is useful to be able to read or write something from a given location of the file. This ability to directly go to a specific location of a file is called random or direct access. The function `fseek()` enables a program to go directly to a specific location of the file and the file operations can be performed starting at that location. To do this, `fseek()` requires three parameters: the filepointer, an offset, and a value telling it how to use the offset. It is very common to pass arguments to a program when the program is invoked from the command line. For example, the `ls` program in Unix or `dir` command in Windows can take arguments as `ls a*` or `dir a*` where `ls` or `dir` is the command and `a*` is the argument. This is supported in C by modifying the main function's parameter list. Information about command line arguments is passed by the operating system to the entry function of the invoked program by way of parameters, which is the `main()` function.

9.5 CHECK YOUR PROGRESS MODEL ANSWERS

1. Every file is appended with a special imaginary symbol known as "End of File"
 2. Argument(s) passed to command prompt of operating system through C program
-

9.6 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Write a program to demonstrate creating, opening, reading the file
2. Explain concatenation of two files using a code example
3. Write a program to sort the content of a file?

II. Answer the following questions in about 15 lines each

1. Describe accessing a file randomly.
 2. Explain how to copy contents of a file to another file with code example
 3. Write a program to demonstrate command line arguments
-

9.7 GLOSSARY

File	:	A file is a sequence of data stored on the permanent storage medium like hard disk.
EOF	:	Every file is appended with a special imaginary symbol known as "End of File"
Bitfield	:	A number of bits in a word made accessible as a struct member.
bitset	:	A standard library "almost container" holding N bits and providing logical operations on those.

BLOCK - IV

INTRODUCTION TO C++

This block gives an overall knowledge Object Oriented Programming with C++. Various concepts of Object Oriented Programming Paradigms such as class, object, abstraction, encapsulation, inheritance, polymorphism, and dynamic binding are described with examples. Defining Class, instantiating objects of class, access specifiers such as public, private, protected are explained with appropriate examples. Constructors, destructors, function overloading, function overriding, pointers to objects, passing objects to functions are described with crystal clear examples. Overloading of arithmetic operators, input and output operators are explained with appropriate examples. Inheritance concepts such as base class, derived class, multiple inheritances, and multi-level inheritance are explained diagrammatically with suitable example. Virtual functions, virtual base class, templates are described with crystal clear code examples.

The units included in the block are:

Unit-10: Classes and Objects

Unit-11: Inheritance

Unit-12: Polymorphism

UNIT- 10: CLASSES AND OBJECTS

Contents

- 10.0 Objectives
 - 10.1 Introduction
 - 10.2 OOPs and C++
 - 10.3 C++ Classes, Objects, I/O Statements, Access Specifiers
 - 10.4 Data Members, Member Functions, Constructor, Destructor
 - 10.5 Function Over Loading, Garbage Collection
 - 10.6 Summary
 - 10.7 Check Your Progress – Model Answers
 - 10.8 Model Examination Questions
 - 10.9 Glossary
-

10.0 OBJECTIVES

After studying this unit, you should be able to

- understand various concepts of OOPs in C++
 - explain how to create classes, objects, Access specifiers
 - describe data members, member functions, constructor, destructor, I/O statements
 - understand function over loading, overriding, garbage management
-

10.1 INTRODUCTION

When computers were invented, programming was done by binary instructions, which was feasible as long as the programs were just a few hundred instructions long. As the programs grew in size, writing code in binary format became almost impossible and assembly language was invented, which uses symbolic representation of machine instructions. This made programs a little understandable. As programs continued to grow, high level languages were introduced like FORTRAN and BASIC. These were impressive in the beginning and as the program length increases, it becomes very difficult to understand the program. Then structured programs like C and Pascal were introduced which modularized the programming by using functions and multiple files. Moderately complex programs can be developed with these languages with ease. Once the project grows beyond certain size, even this method also fails because of too many functions, variables and structures. All along the journey of development of languages, methods were created to allow the programmer to deal with increasingly greater complexity. Every new language or approach took the best part of the previous languages and added new methods to handle complexity of the program. Today many of the projects are near or at the point where the structured approach no longer works. To solve this problem, object oriented programming was invented which modularizes the program where each module is almost self contained and independent of other.

10.2 OOPs AND C++

It is the latest way and the most natural way of solving a problem. This example explains why 119

this concept is natural way and productive. A manager of an office asked his clerk to get a new pen. As far as the manager is concerned, his object is to get a new pen. He doesn't bother how a pen can be purchased following the rules of that company. (May be an indent is placed, Quotations are called, Lowest quotation is sent to the purchases manager for approval, permission is granted after 2 or 3 queries, order is placed, reminders are sent to the supplier as he could not supply in time, and so on. Finally the pen is purchased and given to the manager). i.e., each work is done by the individual concerned, but for a common objective, getting a pen. In the same way, in OOP, types of operations are classified and each class is responsible for its data and related functions. Using OOP, a problem can be decomposed into subprograms of related parts of the problem. Then using the actual language, each part can be translated into self-contained units, called Objects. All the OOP languages have three things in common: Objects, Polymorphism and Inheritance, which are the essences of human behavior.

Object and Data Encapsulation

An object is a logical entity that contains data and methods that manage that data. Some code and/or data may be private to the object (inaccessible directly from outside the object), and some data and/or code is public (accessible from anywhere in the program). This type of linkage of code and data is known as data encapsulation. It is a good practice to provide manipulation methods for the data so that data is not directly accessed by the user. For simplicity an object may be viewed as a user defined type variable. Class is the definition of how an object should be. It can be viewed as a blueprint or a design that is used to create objects. For example, the blueprint of a car can be viewed as a class while the car itself made out of this design is an object. An object of a given class type can be created either statically using a variable, or dynamically using the 'new' keyword.

```
<className><varName(optional initialization data);
```

or

```
<varNamePtr> = new <className>(optional initialization data);
```

Where varName is a variable of a given className and varNamePtr is a pointer of type className.

Polymorphism

All OOPs support polymorphism, which means that one name, can be used for several related but slightly different purposes. Consider the operation of a stack. We need two functions push () and pop () for stack operation. Depending on the data type (say integer, long, float, double ...) we should write separate functions, which should do the same thing on the given data. Hence the same function names (PUSH and POP) are appropriate for all the data types. The compiler will select the appropriate push or pop function depending on the data type passed as input. For this reason all functions in C++ must be prototyped. Another example is adding two numbers. A plus (+) symbol must work correctly for all data types like adding two strings, two complex numbers and so on. The program should be developed in such a way that it takes the right spirit of the expression, with the + symbol. This behavior is known as overloading. A human being can automatically select appropriate method depending on the situation and computer must be programmed to do so.

Inheritance

It is the process by which one class can acquire the properties of another class. Let an "Animal" is defined as a being with 4 legs and a tail. Now a "Cow" can be defined as an "Animal" with 2 horns. A "Tiger" can be defined as an "Animal" with claws. Thus both Tiger and Cow are animals and so they should have 4 legs and a tail apart from their distinct characters. By doing this redefinition of animal is eliminated. Here the "Animal" is known as the *base* class and

“Cow” and “Tiger” are known to be *derived* from Animal. By analogy, it is equivalent to a father and son, where all the assets of the father will be inherited by the son in addition to any asset acquired by him exclusively.

Aggregation

In addition to the three concepts mentioned above (which apply to classes), there are also aggregation relationships (which apply to objects). These apply when objects of one class are created and used in objects of another class. Aggregation of this kind is more flexible than inheritance because they can be more readily reorganized.

C++ Reserved Words

The Following are reserved and should not be used as function names or variable names. Most of these are C reserved words as well.

asm	else	new	This	auto
enum	operator	throw	bool	explicit
private	true	break	export	protected
try	case	extern	public	typedef
catch	false	register	typeid	char
float	reinterpret_cast	typename	class	for
return	union	const	friend	short
unsigned	const_cast	goto	signed	using
continue	if	sizeof	virtual	default
inline	static	void	delete	int
static_cast	volatile	do	long	struct
wchar_t	double	mutable	switch	while
dynamic_cast	namespace	template		

Function Prototyping

In C a function prototype is optional and half prototyping (parameter definition is omitted) is sufficient. But in C++, full prototyping (parameters should also be defined) is essential and compiler gives an error message if a function is not defined. Consider a function add, which takes two integers as arguments and returns their sum.

```
int add(); //Half prototype
```

```
int add(int, int); // Full prototype
```

This is justified because C++ supports polymorphism. So each function declares its parameters and their types well ahead. Otherwise the compiler can't choose the correct function depending on the types of parameters passed.

New data types in C++

C++ introduces a new data type '*long double*', which takes 80 bits which can represent real numbers of 3.4×10^{-4932} to 3.4×10^{4932} , which is sufficient to represent practically anything. C++ also gives a new data type called *class*, which is similar to structure in C. The C++ also enhanced the structure to contain data as well as its related functions (in C, structure cannot contain code). Another data type exclusive for C++ is the reference variable, which is an alias of another variable.

Limitation of goto in C++

The *goto* statement is limited to a block (pair of parentheses) in C++. This is necessary because, if it goes out of block, the object is no more valid, but its destructor (will be discussed later) is not run. To avoid this, scope of *goto* is limited to a block.

Namespace in C++

C++ uses namespace to distinguish methods of same signature but belong to different regions of a program or files.. Different namespace blocks can have identifiers of same name. All declarations within those blocks are declared in the named scope. To declare a namespace globally, a directive “using namespace” is used at the top of the program just below the #include directives.

```
using namespace std;
```

The *std* is a standard namespace that is used generally and is seen in almost all the programs. All modern compilers make the namespace mandatory. If a block is to be given a separate namespace, it can be declared as

```
namespace nmsp1
{
    <type><varname>;
    ...
}
Namespace nmsp2
{
    <type><varname>;
    ...
}
```

To access a variable from a namespace, the scope resolution operator ‘::’ is used.

```
namespaceName::variableName
```

where namespaceName is the name of the namespace and variableName should be defined within that namespace block.

```
// Creating namespaces
#include <iostream>
using namespace std;
namespace ns1
{
    int x = 10;
int value()
    {
        return x;
    }
}
```

```

}
namespace ns2
{
double x = 2.5;
    double value()
    {
        return x*x;
    }
}

int main()
{
    // Access value function within ns1
    printf("Value from ns1: %d\n",ns1::value());
    // Access value function within ns2
    printf("Value from ns2: %f\n",ns2::value());
    // Access variable x directly
    printf("x from ns1 and ns2: %d, %f\n",ns1::x, ns2::x);
    return 0;
}

```

The output of the program will be:

```

Value from ns1: 10
Value from ns2: 6.250000
x from ns1 and ns2: 10, 2.500000

```

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. What is namespace?

.....

.....

.....

10.3 C++ CLASSES, OBJECTS, I/O STATEMENTS, ACCESS SPECIFIERS

Class and Structure

C++ supports both struct and class data types, which are almost similar. Both have data and functions as members of that class or structure. The syntax for both is given below:

```

class <className> {
<optional scope specifier><data type><variableName>;
...
<member functions>
};

```

```

struct <structureName> {
<optional scope specifier><data type><variableName>;
...
<member functions>
};

```

Except for the class/struct keyword, there is no difference between these two definitions. In fact, these can be used interchangeably in C++. The only difference between them is the default scope, which is public by default in structures (accessed from anywhere) and private by default in classes (accessed only from within the class). Consider the following example:

```

class Adder {
    int a, b;
    public:
void printData() {
    printf("a=%d, b=%d, a+b=%d\n", a, b, a+b);
}
    void setData(int x, int y) {
        a = x;
        b = y;
    }
}
int main(int argc, char *argv[])
{
    Adder adder = new Adder();
    adder.setData(20, 30);
    adder.printData();
    adder.a = 10; // Gives an error because a is private by default.
    adder.printData();
}

```

In the above example, a new class/structure is defined as having two variables a and b and two functions that act up on the data. By default the variables are private, which means, they cannot be accessed from outside the class if it declared as 'class'. Hence, the last line gives an

error unless a is made public or class is changed to struct where all variables are public by default.

C++ provides a new facility to write functions outside the class definitions but can be declared as members of a class by using the scope resolve operator (::). The advantage is that the class definition only contains the data and function declarations, and the actual code that manipulates the data can be written outside the class for clarity. Following example demonstrates the same class defined above, with the functions printData and setData declared inside the class and defined outside the class with scope resolution operator. It is important to note that the function declarations are necessary inside the class and can be implemented (defined) outside the class with scope resolution operator.

```
class Adder {
    int a, b;
    public:
void printData(void);
    void setData(int , int );
}
void Adder::printData(void) {
    printf("a=%d, b=%d, a+b=%d\n", a, b, a+b);
}
void Adder::setData(int x, int y) {
    a = x;
    b = y;
}
int main(int argc, char *argv[])
{
    ... // same as above
}
```

Access Specifier

In C++, a variable can be declared to be a public variable, a private variable or a protected variable, just by preceding its definition with the access specifier (public, private or protected) followed by a colon. Following table gives the visibility of each specifier.

Specifier	Access
public	The variable is accessible from anywhere in the program.
private	The variable is accessible only from within the class it is defined or its friend functions. It is not available even to its derived classes.
protected	The variable is available from within the class and its sub-classes (derived classes).

Example:

```
public:
int x;
private:
float y;
protected:
char name[80];
```

Once a specifier is encountered, all the variable declared next will use the same specifier until the compiler encounters another specifier. For example, the following code declares two public variables (x and y) and one private variable (z):

```
public:
int x;
int y;
private:
float z;
```

Input and Output Using C++'s I/O Functions

C's I/O system is unparalleled and any kind of format can be achieved using the library functions (printf, scanf etc.). Though the power of C's I/O is more than adequate, C++ has its own I/O functions defined as streams. This is to facilitate the feature of extending the I/O to any device, which may come in future. For example, text can be sent through MODEM, by extending the output stream in C++. A second reason for such a special system is to develop new formats. Consider printing complex numbers of the format $x + iy$. In C, this can be done with standard format like

`printf("(%d +i%d)",x.real, x.imag);` where, x is a structure having 2 integer values as real and imaginary parts of the complex number. There is no way that we can print the complex number in one stroke, like

`printf("%complex", x);` where complex is a new format.

In C++ overloading the << operator can easily do this It is perfectly correct to write:

`cout << x;` where x is a complex number and '*cout*' is the standard C++ output directive similar to `stdout` in C. Similarly, we can write a function to get the input from the user like:

```
cin >> x;
```

The equivalent of this in C is:

```
scanf("%d%d",&x.real, &x.imag);
```

Here `cin` is the standard input function which is equivalent to `stdin` in C. It is possible to use `printf` and `scanf` in C++ but using *cout* and *cin* functions is natural.

Example: Following is the first C++ program that shows the hello world message on the screen.

```
#include <iostream>
using namespace std;
void main()
```



```

{
cout << "\n"
<< "Hello World"
<< "\n";
}

```

This prints the message 'Hello World' on the screen. Here the left shift operator '<<' (called *insertion operator*) is overloaded as an operator to input data to the object `cout`, which outputs this on to the screen. Similarly the right shift operator '>>' (called *extraction operator*) is overloaded as an operator to input data to the object `cin` which inputs the data to the variable. Inclusion of the header file '*iostream*' is necessary for accessing various definitions and hence it is almost seen in every C++ program. All C++ header files are included without the .h extension.

Example: Following example reads some data from the user and prints it on the screen.

```

#include <iostream>
using namespace std;
void main()

```

```

{
int x;
char string[20];
cout << "\nEnter a value:";
cin >> x;
cout << "\nEnter your name:";
/*

```

First method: reads only up to the first white space

```

*/
cin >> string; //Ends at the first space.

```

```

/*

```

Second method: Reads full text up to 19 characters or new line whichever comes first

```

*/
//cin.get(string, 20, '\n');

```

```

cout << "\nHellow"
<< string
<< "you have entered "
<< x
<< endl;
}

```

Here many extractors are cascaded which can also be done with a *cout* to each *extractor*.

```
cout << "this " << "is " << "a string" << endl; is equivalent to
```

```
cout << "this ";cout <<"is "; cout <<"a string";cout <<endl;
```

The 'endl' is a macro defined in *iostream* header file as newline character for that operating system (\n for Unix, \n\r for Windows, \r for Mac etc.). The stream *cin* reads a string only up to the first white space. If a string with white spaces is to be read, the member function 'get(char *, int, char) may be used which takes a pointer to the string into which data is to be copied, number of characters to read (a NULL is appended at the end automatically) and the delimiter to use. Hence, if the first method is commented and second method is uncommented, the program works properly.

Formatted Input and Output

We can obtain formatted input and output in two methods.

i) Using ios member functions

ii) Using manipulator functions

Using ios member functions :

in *iostream* the following enumeration is declared.

```
enum {  
    skipws      = 0x0001,  
    left        = 0x0002,  
    right       = 0x0004,  
    internal    = 0x0008,  
    dec         = 0x0010,  
    oct         = 0x0020,  
    hex         = 0x0040,  
    showbase    = 0x0080,  
    showpoint   = 0x0100,  
    uppercase   = 0x0200,  
    showpos     = 0x0400,  
    scientific  = 0x0800,  
    fixed       = 0x1000,  
    unitbuf     = 0x2000,  
    stdio       = 0x4000
```

128 };

skipws:	When set, leading white-space characters are discarded when performing input on a stream. If it is unset, these are not discarded.
left, right:	Sets the justification.
internal:	When set, a numeric value is padded to fill a field by inserting spaces between any sign or base character.
dec, oct, hex,	By default, numeric values are output in the base as they are
showbase, uppercase	represented. However, we can override this default by setting appropriate flag.
showbase	Causes the output to be prefixed with its base symbol (for example 80 hex is printed as 0x80. If upper case is set, the same value is printed as 0X80).
scientific, fixed	By default, the system takes care of the notation of the floating format. But at times we wish to print the value in a particular format. To print a value in scientific notation, set scientific flag. To print in standard floating point notation set fixed flag. For example a value 1234.5678 will be printed as 1.234e3 by default, with a floating-point precision of 2 (since precision is less than the actual digits). If fixed is set, the same is printed as 1234.57.
unitbuf	When set, the C++ I/O system performance is improved. By default this is set.
stdio	When set, each stream is flushed after each output.

Table 10.1

General format of setting or unsetting is

```
stream.setf(long flags);
```

```
stream.unsetf(long flags);
```

if stream is standard I/O then

```
cout.setf(ios::<flagname>);
```

```
cout.unsetf(ios::<flagname>);
```

For example to set both showbase and uppercase it may be written as:

```
cout.setf(ios::showbase | ios::uppercase);
```

To set width of a field to n use width() function as:

```
cout.width(n);
```

To set precision with to 2, use precision() function as:

```
cout.precision(2); // sets 2 digits after decimal point
```

Using I/O Manipulator Functions:

This method uses special functions called manipulators, which can be included in an I/O statement. These are defined in *iomanip* header file. The general format is

stream << manipulator_ function;

The following table gives a list of various manipulator functions.

Manipulator	Purpose	Input/Output
dec	Format numeric data in decimal	I & O
endl	Output a new line char and flush the stream	O
ends	Output a null	O
flush	Flush a stream	O
hex	Format numeric data in hex	I & O
oct	Format numeric data in octal	I & O
resetiosflags(long f)	Turn off flags specified in f	I & O
setbase(int base)	Set the number base to base	O
setfill(int ch)	Set the fill char to ch	I & O
setiosflags(long f)	Turn on flags specified in f	I & O
setprecision(int p)	Set No. of digits displayed after the decimal point	I & O
setw(int w)	Set the field width to w	I & O
ws	Skip white spaces	I

Table 10.2

The following example gives a brief overview of how to use various functions.

```
#include <iostream>
#include <iomanip> // for manipulator functions
using namespace std;
int main()
{
int x=12345;
float y=1234.5678;
char *z = "This is a test string";
/* Using Manipulator functions from iomanip.h */
cout << "using manipulator functions" << endl;
cout << "using width 3 justification = right :";
cout << setw(3) << x <<endl; // ignores this because width is > 3
cout << "using width 15 justification= right :";
cout << setw(15) << x <<endl; // sets the width to 15
cout << setiosflags ios::left); // set left justification on
cout << "using width 15 justification = left :";
```

```

cout << setw(15) << x << endl;
cout << resetiosflags(ios::left); // Turn of left justification
cout << "default :." << y << endl
<< setw(8) << setprecision(2) << setiosflags(ios::fixed)
<< "Prec. 2 :." << y << endl // prints 2 digits after decimal piont
<< setprecision(5)
<< "Prec. 5 :." << y << endl
<< setiosflags(ios::scientific) // Prints in exponential notation
<< y << endl
<< setiosflags(ios::uppercase) // use E & X for Exp. and Hex
<< y << endl;
cout << setw(10) << z << endl // overrides the width function
<< setw(40) << z << endl // right justified
<< setiosflags(ios::left) << z << endl; // left justified
cout << "Now using ios flags to get required format" << endl;
cout.setf(ios::hex|ios::showbase|ios::uppercase);
cout << x << endl;
cout.unsetf(ios::hex);
cout << x << endl
<< "End of Test Program\n";
return 0;
}

```

Local Variables in C++

Traditionally all local variables must be declared at the start of a block in C which is not mandatory in C++, where local variables can be declared anywhere in the program before it is first used. For example the following is incorrect in C but is perfectly acceptable in C++.

```

f() {
int i;
i=10;
int j; // error in C
....
....
}

```

In fact, it is convenient to declare the variables as close to the place where it is first encountered as possible. This makes the life easy when debugging a program. This is also convenient to encapsulate that function and the variables at a later date. It also helps us avoid accidental side effects. Even variables are initialized dynamically, by equating it to an expression, say `int p=(strlen(x)+1)/blanks(x)`; Here, `p` is assigned a value which is the result of an expression.

Now if x is read from the keyboard at run time, p is initialized using the length and blanks in the string.

Reference Variables and Passing Variables by Reference:

Reference variables are aliases to other data items. They are not allocated memory directly. Instead, they point to other variables' addresses, hence they cannot be just declared. They must be initialized to another variable. Reference variables are declared with an ampersand (&) before the variable. For example,

```
int x;  
int &y = x;
```

This declares x as an integer, and y as a reference variable (alias to x), which is also sharing the memory allocated to x. Hence, you can change the value of x by either x or by y.

If a variable is passed to a function by *reference*, then in the function where it is used, we need not use the pointer notation. We simply use the variable as normal variable. The only difference is during the declaration. Consider the following example, where the variables are sent by address (pointer notation) and as reference variables. The result of both the callings is identical, but the way the parameters are sent while calling and receiving is different.

```
#include <iostream.h>  
  
int main (void)  
{  
int a=10, b=20;  
int c=10, d=20;  
swap_ptr(&a, &b);  
swap_ref(c,d);  
cout << "a = " << a << " b = " << b << "\n";  
cout << "c = " << c << " d = " << d << "\n";  
}  
  
swap_ptr(int *x, int *y)  
{  
int t;  
t = *x;  
*x = *y;  
*y = t;  
}  
  
swap_ref(int &x, int &y)  
{  
int t;  
t = x;  
x = y;  
y = t;  
}  
}
```

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

2. What is access specifier?

.....
.....
.....

10.4 DATA MEMEBERS, MEMBER FUNCTIONS, CONSTRUCTOR, DESTRUCTOR

Initializing Variables of a Class

Variables of a class can be initialized using the ‘:’ operator between the closing bracket of the parameters list and the block beginning ({}). In the following code, variables a and b, are assigned values, i.e., no value is assigned during creation of variables.

```
class Test
{
    int a,b;
public:
    Test(int x, int y) { a= x; b=y;} // assignment (not initialization)
    ....
};
```

Here is the way to initialize the variables during their creation:

```
class Test
{
    int a,b;
public:
    Test(int x, int y) : a(x), b(y) { } // Initialization of a and b
    ....
};
```

Here a and b are private and test is a public function (method) that initializes a and b while they are created.

Constructor and Destructor

A constructor is a special member function of a class which is automatically called just after the object is created which can be used for initialization of objects of a class. A Constructor is a special functions which has the same name as that of the class. It doesn't declare any return type. If a constructor is not defined, the compiler generates a default constructor with empty body and with no parameters. Destructor is also a special function that is automatically just before an object is destroyed or deleted from memory. It also has the same name as that of the class except that it is preceded by a '~' (tilde character). Hence, if a class name is A, then the constructor's name is A() and destructor's name is ~A().

There are three types of constructors namely Default, Parameterized and Copy constructors.

Default Constructors

Default constructor is a constructor with no parameters. If the default constructor is not defined by the user, it will be automatically created by the compiler with empty body.

Parameterized Constructors

Constructor can be overloaded using unique parameters list for each constructor. The parameters passed to the constructor are used to initialize the object when it is created.

Copy Constructor: A copy constructor is a constructor that takes a reference to an object of same type as parameter and initializes the current object.

See the following example below which illustrates all the three types of constructors and the destructor:

```
#include <iostream>
#include <string.h>
using namespace std;
class Person {
    char name[80];
    int age;
public:
    Person()
    {
        cout << "Default constructor called.\n";
        strcpy(name,"XXXXXX");
        age = 999;
    }
    Person (const char * nm, int a)
    {
        cout << "Two arguments constructor called with " << nm
        << ", " << a << endl;
        strcpy(name, nm);
        age=a;
    }
    Person(const Person &p)
    {
        cout << "Copy Constructor called with " << p.name << ", "
        << p.age << endl;
        strcpy(name, p.name);
        age = p.age;
    }
};
```



```

    }
    ~Person()
    {
        cout << "Destructor called for " << name << endl;
        strcpy(name, "");
        age = 0;
    }
    void print()
    {
        cout << "Name: " << name << " Age: " << age << endl;
    }
};

int main()
{
    cout<< "Creating p with default values..." << endl;
    Person p;
    p.print();
    cout << "Creating q with arguments ..." << endl;
    Person q("ABCDE", 30);
    q.print();
    cout << "Creating Pointer object ..." << endl;
    Person *ptr;
    ptr = new Person("GHIJK", 25);
    ptr->print();
    cout << "Assigning q to a new variable r " << endl;
    Person r = q;
    r.print();
    delete ptr;
    return 0;
}

```

In the above program the assignment `r = q` during declaration calls the copy constructor. Further, the constructor is overloaded using different parameters.

Passing Objects as parameters

An object may be passed to a function just in same way as any other data type. When an object is passed to the function, a copy of it only is sent. Hence, any change to the data in the function doesn't affect the data of the object. If the function needs to change the data really, a pointer to the object or a reference to the object must be sent and the function needs to be modified

accordingly.

```
#include <iostream>
using namespace std;
class test {
    int i;
public:
    void set(int x) { i=x;}
    void out() {cout << i << " ";}
};
void f1(test x)
{
    x.set(100);
    cout << "\nValue inside the function f1 is : ";
    x.out();
}
void f2(test *x)
{
    x->set(100);
    cout << "\nValue inside the function f2 is : ";
    x->out();
}
void main()
{
    test x;
    x.set(10);
    cout << "\nvalue at the starting is :";
    x.out();
    f1(x);
    cout << "\nValue after sending to f1 (call by value) is : ";
    x.out();
    f2(&x);
    cout << "\nValue after sending to f2 (call by reference) is : ";
    x.out();
}
```

The output is:

value at the starting is :10

Value inside the function f1 is : 100

Value after sending to f1 (call by value) is : 10

Value inside the function f2 is : 100

Value after sending to f2 (call by reference) is : 100

Observe that the call to f1 doesn't change the value but a call to f2 will change the value since the address is sent.

Arrays of objects

An array of objects can be created just like an array of any other data type. For example:

```
#include <iostream.h>
using namespace std;
class count {
    int x;
public:
    void show() { cout << "\nThe Number is : " << x; }
    void set(int i) { x=i;}
};
void main()
{
    count c[10];
    for(int i=0;i<10;i++) c[i].set(i);
    for(i=0;i<10;i++) c[i].show();
}
```

Pointers to Objects

The data or functions of an object can be accessed through pointers, just as in case of members of a structure through pointer to a structure. Following example illustrates the use of pointers to objects:

```
#include <iostream.h>
using namespace std;
class Ptrobj
{
    int n;
public:
    void set_num(int x) { n= x;}
    void show_num();
};
void Ptrobj :: show_num() {cout << n << "\n"; }
```

```

void main()
{
Ptrobj obj, *ptr; // an object and a pointer are created
obj.set_num(10);
obj.show_num();
ptr = &obj; // Pointer is assigned the address of obj
ptr->set_num(100);
ptr->show_num();
}

```

Dynamic Creation of Objects

An object may be allocated memory dynamically using the operator 'new' and the memory can be released by using the operator 'delete'. These functions eliminate the use of malloc() and free() functions, though they can be used in C++ also. The main advantages of using *new* instead of *malloc()* are

Size of the variable/ object is automatically calculated.

initializing an object is possible.

Automatic type casting

The new allocates memory from the heap and if memory is not available, it returns *NULL*, just like its counterpart 'malloc'. *Delete* operator frees the memory previously allocated. Note that it is highly dangerous to try to delete an object that doesn't exist. This may lead to unpredictable behavior including a system crash. The syntax is:

Allocating memory without initializing

```

<ptr variable> = new <variable type>;
delete <ptr variable>;

```

Allocating memory with initializing

```

<ptr variable> = new <variable type>(parameters);

```

Allocating an array of variables

```

<ptr variable> = new <variable type> [ <size> ];
delete [ <size> ] <variable type>;

```

Note that when array is allocated memory with *new*, it can't be initialized. In some modern C++ compilers size need not be specified in *delete* statement.

```

#include <iostream.h>

```

```

using namespace std;

```

```

void main()

```

```

{

```

```

int *i;

```

```

float *p;
p = new float [10]; // allocating 10 float values
i = new int (15); // allocating integer and initialize to 15;
if(!p || !i) { cout << "\nError in allocating memory"; return;}
for(int j=0; j<10;j++) p[i]=100.0+j;
cout << "\n" << *i << "\n";
for(j=0;j<10;j++) cout << p[i] << "\n";
delete i;
delete [10] p;
}

```

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

3. What is parameterized constructor?

.....

.....

.....

10.5 FUNCTION OVERLOADING, GARBAGE COLLECTION

Default Function Arguments

C++ allows a function to assign a default value to a parameter when no argument corresponding to that parameter is specified in a call to that function. For example consider the function f below:

```
void f(int i=1) { ... }
```

can be called with one or no arguments as

```
f(10); // Pass an explicit value of 10
```

```
f(); // Let the function use its default value i.e., 1
```

Example :

```
#include <iostream>
```

```
using namespace std;
```

```
void print_size(int x =0, int y =0, int h=0, int w=0)
```

```
{
```

```
cout << "Window starts at row , col: " << x << ", " << y << " ";
```

```
cout << "With height, width: " << h << ", " << w << endl;
```

```
}
```

```
int main()
```

```

{
print_size();    // 0 arguments ( x=y=h=w=0)
print_size(1);      // 1 argument (x=1, y=h=w=0)
print_size(1,2);    // 2 arguments (x=1, y=2, h=w=0)
print_size(1,2,10); // 3 arguments (x=1, y=2, h=10, w=0)
print_size(1,2,10,5); // 4 arguments (x=1, y=2, h=10, w=5)
return 0;
}

```

The output will be

```

Window starts at row, col: 0, 0 With height, width: 0, 0
Window starts at row, col: 1, 0 With height, width: 0, 0
Window starts at row, col: 1, 2 With height, width: 0, 0
Window starts at row, col: 1, 2 With height, width: 10, 0
Window starts at row, col: 1, 2 With height, width: 10, 5

```

In general, any number of arguments can be set to defaulted values, and the only rule is that they must all be declared after the parameters without default. The following function declaration is incorrect.

```
func(int i, int j=0, int k, int l=0) //j had default but declared before k.
```

It should be declared as:

```
func(int i, int k, int j=0, int l=0)
```

Function Overloading

C++ supports function overloading where the same function name can be used for two functions provided their parameters are different. C++ can select appropriate function when called, depending on the parameters it receives during the call. The only rule is that there should be no ambiguity in the parameter types. Any function can be overloaded in C++ except the main method, which should be unique.

```

#include <iostream>
#include <string.h>
#include <malloc.h>
using namespace std;
int add(int a, int b)
{
    return a+b;
}
float add(float a, float b)
{
    return a+b;
}

```

```

}
char* add(const char *a, const char *b)
{
    int l = strlen(a)+strlen(b)+1;
    char *s = (char *) malloc(l);
    strcpy(s,a);
    strcat(s,b);
    return s;
}
int main()
{
    char *s = add("abc", "def"); // calls the char * version of add
    cout<< add(10, 20)<<endl;
    cout<< add(1.2f, 2.3f)<<endl;
    cout <<s<<endl;
    free(s);
}

```

The output of the function will be as expected:

30

3.5

abcdef

The string version of add allocates memory dynamically using malloc function.

Garbage collection

Garbage collection is a form of automatic memory management. The garbage collector or collector attempts to reclaim garbage, or memory used by objects that will never be accessed or mutated again by the application. C++ doesn't provide garbage collection directly. Hence, every object that the user creates must be cleared off when it is not necessary anymore. This can be done with new and delete keywords which will allocate and release memory allocated to an object. For example, consider the following program in C++:

```

#include<iostream.h>
classA{
    intx;
    public:
    A(){x=0;}
};

intmain(){

```

```

    A*a=newA();
    Deletea;
}

```

In the above program, a new object of type A is created and its address is assigned to a. If delete is not used, the variable's assigned memory will not be cleared and will be hanging around as long as the program is running. This is called memory leakage. Hence, every object created dynamically should be deleted by the user so that there is no memory leakage during the run of the program.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

4. What is garbage collection?

.....

.....

.....

.....

10.6 SUMMARY

It is the latest way and the most natural way of solving a problem. This example explains why this concept is natural way and productive. A manager of an office asked his clerk to get a new pen. As far as the manager is concerned, his object is to get a new pen. He doesn't bother how a pen can be purchased following the rules of that company. (May be an indent is placed, Quotations are called, Lowest quotation is sent to the purchases manager for approval, permission is granted after 2 or 3 queries, order is placed, reminders are sent to the supplier as he could not supply in time, and so on. Finally the pen is purchased and given to the manager). i.e., each work is done by the individual concerned, but for a common objective, getting a pen. In the same way, in OOP, types of operations are classified and each class is responsible for its data and related functions. Using OOP, a problem can be decomposed into subprograms of related parts of the problem. Then using the actual language, each part can be translated into self-contained units, called Objects. All the OOP languages have three things in common: Objects, Polymorphism and Inheritance, which are the essences of human behavior.

A constructor is a special member function of a class which is automatically called just after the object is created which can be used for initialization of objects of a class. A Constructor is a special functions which has the same name as that of the class. It doesn't declare any return type. If a constructor is not defined, the compiler generates a default constructor with empty body and with no parameters. Destructor is also a special function that is automatically just before an object is destroyed or deleted from memory. It also has the same name as that of the class except that it is preceded by a '~' (tilde character). Hence, if a class name is A, then the constructor's name is A() and destructor's name is ~A().Garbage collection is a form of automatic memory management. The garbage collector or collector attempts to reclaim garbage, or memory used by objects that will never be accessed or mutated again by the application. C++ doesn't provide garbage collection directly. Hence, every object that the user creates must be cleared off when it is not necessary anymore. This can be done with new and delete keywords which will allocate and release memory allocated to an object.

10.7 CHECK YOUR PROGRESS MODEL ANSWERS

1. A namespace declares a region that provides a scope to the identifiers (names of the types, function, variables etc) inside it
 2. It tells the way of accessing the members of class
 3. Overloaded using unique parameters list for each constructor.
 4. Garbage collection is a form of automatic memory management
-

10.8 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Write a program to create class and objects with data and functions
2. Explain different types of constructors and destructor with code examples
3. Write a program to pass objects as parameters?

II. Answer the following questions in about 15 lines each

1. Describe the concepts of abstraction, encapsulation, and polymorphism.
 2. Explain how to create array of objects with a programming examples
 3. Write a program to demonstrate garbage collection
-

10.9 GLOSSARY

Class	:	An instantiable entity which encapsulates state and behaviour.
Object	:	Instance of a class with unique id
Encapsulation	:	Wrapping different things into one entity.
Constructr	:	A member function with the class name to initialize the data of objects
Destructor	:	A member function with the class name to destruct the objects

UNIT- 11: INHERITANCE

Contents

- 11.0 Objectives
 - 11.1 Introduction
 - 11.2 Inheritance
 - 11.3 Static Data Members, Static Methods, This Operator, Friend Classes
 - 11.4 Summary
 - 11.5 Check your progress – Model Answers
 - 11.6 Model Examination Questions
 - 11.7 Glossary
-

11.0 OBJECTIVES

After studying this unit, you should be able to

- explain how to define base class and derived class in C++
 - describe multiple and multi level inheritance in C++
 - understand friend classes and friend functions
 - describe static methods and static data members
-

11.1 INTRODUCTION

Inheritance is one of the important features of object oriented programming, through which the capabilities of a class can be extended by deriving a new class from an existing class. By doing so, the new class inherits all the public and protected properties of the existing class and the user can add more data and methods to extend the capabilities of a class. The class from which the new class is extended is known as the base class and the new class is called derived class. A new class can be derived from more than one base class in C++, though it is discouraged due to its associated problems. However it is a powerful feature, through which code can be aggressively reused. The general syntax for multiple inheritance is: `<newClass> : <access><baseClass1> ,<access><baseClass2> ...{ ... };` Where `newClass` is derived from more than one class (`baseClass1`, `baseClass2`, ...) with the given access (public or protected or private). If access is omitted for any base class in the list, it is taken as private base class for the new class. Static Variables are guaranteed a zero if not initialized to other values. Also an object need not be instantiated to use the static functions. This feature is used to generate wrapper classes. Wrapper classes are useful in pure object oriented languages, where one cannot write general purpose functions (say $\sin(x)$) outside of the classes. An object need not be created to call static methods or static members since they belong to the class but not to the object. Hence `className.methodName()` should be sufficient to invoke a method

11.2 INHERITANCE

Inheritance is one of the important features of object oriented programming, through which the capabilities of a class can be extended by deriving a new class from an existing class. By doing so, the new class inherits all the public and protected properties of the existing class and the user can add more data and methods to extend the capabilities of a class. The class from

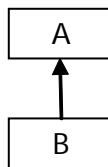
which the new class is extended is known as the base class and the new class is called derived class.

```
class A {
  int x,y;
  public:
  ....
  ....
};
```

To create another class B, which contains all the properties of class A, then B is declared as below:

```
class B: public A {
  int z;
  public:
  ....
  ....
};
```

Diagrammatically it is represented as



Here class B gets all the properties (or inherits all the properties) of A and class B is said to be derived from class A. Class A is called *base class* and class B is called *derived class*. Though A is base class of B, the private parts of A are not accessible by B. For example, if any function of B tries to access x or y, compiler will issue an error message. To make x and y of A accessible to B, they must be declared as protected so that they are not accessible to others except the derived classes. Following table gives an idea of the security level in OOP.

Access Type	Accessible from			
	same class	derived class	friend function	Others
private	yes	no	yes	no
protected	yes	yes	yes	no
public	yes	yes	yes	yes

Table 11.1

In the previous definition of B, nothing is mentioned about the type of A, hence it is taken as a *private* part of B. Now consider that a new class C is derived from B.

```
class C : B {  
    .....  
};
```

From C any function or variable that belongs to A cannot be accessed. To overcome this problem, any base class must be made *public*, if this new class is going to form a Base for another class in future. The following is the correct way to declare derived classes.

```
class A {  
    protected int x,y;  
    public :  
    .....  
};
```

```
class B : public A {  
    protected int z;  
    public:  
    .....  
};
```

The general form to define a derived class is

```
class <class name> : <access type><base>, <base>,..... {  
    private data  
    public:  
    public data  
} <list of variables>;
```

As a practical example for inheritance, consider a shape with the following definition:

```
class Shape {  
    protected:  
    float x;  
    public:  
    Shape() { x = 0;}  
    Shape(float v) { x = v;}  
    void set(float v) { x = v}  
    float getX(void) { return x;}  
    float area(void) { return 0;}  
    void printArea() { cout << " Area of the shape: " << area() << endl;
```

Here, the class has one variable x, which can be set with constructor, or through its member function set(). Now consider that an application needs a method area() which computes the area of a given shape. For this the method needs to be inserted into the class. If the source code is available it can be edited and new code can be inserted as required. Further, the code now need to be tested from the beginning since, during editing, some errors might have crept in. If it is not available, the only way to extend the class is through inheritance by which the Shape can be extended. Consider a Rectangle which extends Shape (Rectangle is derived from Shape)

```
class Rectangle : public Shape
{
    float y;
    public:
    float area() { return x*y;}
};
```

Now an object of type Rectangle can make use of both set() and area() methods to set dimensions and compute the area. There are some issues with inheritance like, how to call the base class constructor with arguments, what happens when the base class and the inherited class have the same function, etc.

Calling Base Class Constructor

In the Rectangle class there is no constructor and hence, the default constructor is automatically created and nothing will be done, except that it calls the default constructor of base class (Shape()). To enable initialization, a constructor can be defined in Rectangle that just calls the Shape's constructor with the parameters. This done with

```
Rectangle(float v1, float v2) : Shape(v1, v2) { }
```

Here constructor automatically takes the same access level as its base class constructor by default which is public in this case, since Shape's constructor is declared as public.

```
class Rectangle : public Shape
{
    Rectangle(float v1, float v2) : Shape(v1) {
        y = v2; // v1 is passed to base class constructor
    }
    public:
    float area() { return x*y;}
};
```

Now consider another shape Circle, which requires only radius.

```
class Circle : public Shape
{
    const float pi = 3.1415;
    Circle(float v) : Shape(v) { }
    public:
```

```

float area() { return pi*x*x;};
};

```

In both cases, a function area is required with no arguments. While extending a class users may use different names for area like computeArea(), shapeArea(), area() and so on, which are all meaningful. Further, the base class has the function printArea which uses area() method only. This is where the base class should mandate all its derived classes to implement the area() function with exact signature, so that it can be used to get the area of the shape properly. The area() function in base class is meaningless and it is there only to make the compiler to properly compile the program. It is at this point, that the base class may just declare a function with a special keyword “virtual” and assigned a value 0 to it. This virtual function is known as pure virtual function, and should be implemented by the derived class.

Abstract Base Class

An abstract base class is a class where not all methods are implemented i.e., it has at least one pure virtual function. Since there is missing code for the pure virtual function, an object of this type cannot be created. A new class must be derived from this abstract base class to create objects of this type where the missing virtual functions should be defined. Here is the new version of the shape class that makes it abstract and all the derived classes implement the area function.

Now both Rectangle and Circle classes should implement area() function and Shape can use the appropriate area() function defined in the respective classes. Now consider the following full program:

```

#include <iostream>
using namespace std;
class Shape {
    protected:
    float x;
    public:
    Shape() { x = 0;}
    Shape(float v) { x = v;}
    void set(float v) { x = v;}
    float getX(void) { return x;}
    void printArea() { cout << “ Area of the shape: “ << area() << endl;}
    virtual float area(void) = 0; // Pure virtual function
};
class Rectangle : public Shape
{
    float y;
    public:
    Rectangle(float v1, float v2) : Shape(v1) {y = v2;} // v1 is passed to base class constructor
    float area(void) { return x*y;}
};
148

```

```

class Circle : public Shape
{
    static const float pi = 3.1415;
    public:
    Circle(float v) : Shape(v) {}
    float area(void) { return pi*x*x;};
};

int main(void)
{
    Rectangle r(10,20);
    Circle c(10);
    r.printArea();
    c.printArea();
}

```

The area of r and c are printed properly by using appropriate area function defined in their respective class definitions.

Multiple Inheritances

A new class can be derived from more than one base class in C++, though it is discouraged due to its associated problems. However it is a powerful feature, through which code can be aggressively reused. The general syntax for multiple inheritance is:

```
<newClass> : <access><baseClass1> ,<access><baseClass2> ...{ ... };
```

Where newClass is derived from more than one class (baseClass1, baseClass2, ...) with the given access (public or protected or private). If access is omitted for any base class in the list, it is taken as private base class for the newClass.

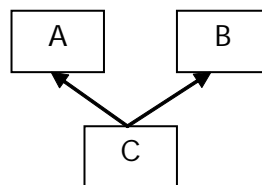


Figure 11.1

The above diagram implies that there are two classes A and B, from which a third class C is derived. If there is a function f() in both A and B and not implemented by C and a call to this function from C is ambiguous since, it can access both A,f() and B.f(). To further complicate the things, consider the following example.

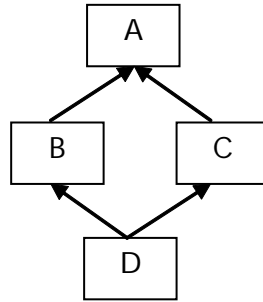


Figure 11.2

Here, B and C are derived from A and D is derived from B and C. If a variable is available in A which needs to be initialized during the creation of D, which path to choose is ambiguous. via B and via C. Similarly the ambiguity during selecting a function remains same. Similarly the base class constructor is called twice if a variable of type D is declared (one via B and the other via C).

See the below example...

```

#include <iostream>
using namespace std;
class A {
    public:
    int x;
    public:
    A() { x = 0; cout << "A Default Constructor called " << endl;}
    A(int v) { x = v; cout << "A 1 Arg Constructor called with " << v << endl;}
    void print() { cout << "x from A: " << x << endl;}
    ~A(){cout << "A Destrutor called " << endl;}
};

class B : public A
{
    int x;
    public:
    B():A(3){x = 10; cout << "B Default Constructor called.." << endl;}
    B(int v) : A(50) {x = v; cout << "B 1 Arg Constructor called with " << v << endl;}
    void print() { cout << "x from B: " << x << endl;}
    ~B(){cout << "B Destrutor called " << endl;}
};
  
```



```

class C : public A
{
    int x;
    public:
    C():A(5){x = 20; cout << "C Default Constructor called.." << endl;}
    C(int v) : A(60) {x = v; cout << "C 1 Arg Constructor called with " << v << endl;}
    void print() { cout << "x from C: " << x << endl;}
    ~C(){cout << "C Destructor called " << endl;}
};

```

```

class D : public B, public C
{
    //int x;
    public:
    D():B(15), C(25){cout << "D Default Constructor called.." << endl;}
    D(int v) : B(v-5), C(v+100) {cout << "D 1 Arg Constructor called.." << endl;}
    ~D(){cout << "D Destructor called " << endl;}
};

```

```

int main(void)
{
    D d;
    d.B::print();
    d.C::print();
    // d.A::print(); // Ambiguous base class for D hence error
    * p = &d;
    p->print();
}

```

The output of the program is :

A 1 Arg Constructor called with 50

B 1 Arg Constructor called with 15

A 1 Arg Constructor called with 60

C 1 Arg Constructor called with 25

D Default Constructor called..

x from B: 15

x from C: 25

x from B: 15

D Destructor called

C Destructor called

A Destructor called

B Destructor called

A Destructor called

By careful observation, following points can be noted

- The constructors are recursively called until the final base class constructor is encountered.
- Constructors are called in the order of definition of the derived class (B's constructor is called before C's constructor) or left to right order.
- Destructors are called in the reverse order of the constructor calls.
- Each base class is specified with its own access
- When there is an ambiguity, it is resolved with scope operator ::
- When there is further ambiguity, compiler gives an error as in (*d.A::print();*) since A has two paths to reach.
- A base class pointer can be assigned address of a derived class.

Multi Level Inheritance

C++ supports hierarchical inheritance where the inheritance is a sequential chain of multiple base classes but is different from multiple inheritance as shown below:

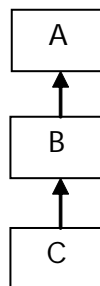


Figure 11.3

Here, C is derived from B, which is derived from A. This eliminates the ambiguities associated with multiple inheritance structure. This is much widely used in many object oriented programming language other than C++. If a function exists in All the 3 classes, still accessing a variable is not an issue using a base class of that type or using scope operator.

The Implicit this Pointer (Keyword):

All objects have an implicit pointer available to every member function of the class. This pointer is given a name “this” which points to the object itself and is available to all member functions including constructors and destructor. Consider the class,

```
#include <iostream>
using namespace std;
```

```

class A {
    int x;
    public:
    A(int x) { this->x = x;}
    void print() { cout << "x = : " << x << endl;}
};

int main(void){
    A a(10);
    a.print();
}

```

Without this->, x = x is meaningless since it will always refers to the local variable, which is different from object variable x. The this pointer is useful wherever there is an ambiguity with local variables, or to refer to the current object.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. What is inheritance?

.....

.....

.....

11.3 STATIC DATA MEMBERS, STATIC METHODS, THIS OPERATOR, FRIEND CLASSES

Static Members and Static Functions

Static members are the members of the class itself and not the members of the objects. It means that all the objects of that type share these variables. Any member function can access the value. If any object changes the value, all the others will represent the new value. Static functions are the functions, which can access only static data. They are not passed this pointer and hence, cannot access any other functions or data of the class. The static data items must be declared in the program explicitly, with the scope resolution, as

```
<data type><class name> :: <data member> [ = <initial value> ] ;
```

Static Variables are guaranteed a zero if not initialized to other values. Also an object need not be instantiated to use the static functions. This feature is used to generate wrapper classes. Wrapper classes are useful in pure object oriented languages, where one cannot write general purpose functions (say $\sin(x)$) outside of the classes. An object need not be created to call static methods or static members since they belong to the class but not to the object. Hence `className.methodName()` should be sufficient to invoke a method. See the example below, which counts the objects that are created during the run of the program.

```
#include <iostream.h>
class test {
```

```

private:
    static int count;
    int value;
public:
    test(int a=0) { value = a; count++;}
    setCount(int a) { count = a;}
    setValue(int a) { value = a;}
    show(){cout << "Count = " << count << " Value = " << value << '\n';}
    ~test(){count--; cout <<"Destructor run.. Now the count = "<< count<<"\n";}
};
int test::value; // value can be initialized by assigning a value here
main()
{
test a(5),b(10),c(15);
a.show(); b.show();c.show();
test d;
a.show();
a.setCount(10);
d.show();
{
test a(100),b(50);
a.show();b.show();
}
}

```

The output of the program is:

```

Count = 3 Value = 5
Count = 3 Value = 10
Count = 3 Value = 15
Count = 4 Value = 5
Count = 10 Value = 0
Count = 12 Value = 100
Count = 12 Value = 50
Destructor run.. Now the count = 11
Destructor run.. Now the count = 10
Destructor run.. Now the count = 9
Destructor run.. Now the count = 8

```

Destructor run.. Now the count = 7

Destructor run.. Now the count = 6

Note that the static count reflects the current count always whatever object calls the show method. Following example illustrates the use of wrapper classes.

```
#include <iostream.h>

class wrapper
{
public:
static double PI;
static int MAX_SHORT;
double sin(double x)
{
double a;
a = 1 - (x*x*x)/(1*2*3) + (x*x*x*x*x)/(1*2*3*4*5); // 1 - x3/3! + x5/5! ...
return a;
}
};

int wrapper::MAX_SHORT = 32767;
double wrapper::PI = 3.14159;

main()
{
double x= 60.0; // degrees
int a;
a = wrapper.MAX_SHORT;
x = wrapper.sin(x*wrapper.PI/180.0);
cout << "x = " << x << " a = " << a << "\n";
}
}
```

Friend Classes and Functions

A class cannot access the private members of another class. Similarly a class cannot access protected members of another class unless it is inherited from that class. One exception in C++ is that a friend class or function can access any member of the any other class that declares this class or function as a friend. This is necessary if a class members are to be accessed legitimately by another class, as in case of a print class which prints the data.

Friend Class

A friend class is a class that can access the private and protected members of a class in which it is declared as friend. This is needed when we want to allow a particular class to access the private and protected members of a class.

Example:

In this example there are two classes A and B. The A class has two private data members x and y, this class declares B as friend class. This means that B can access the private members of A. This is demonstrated with function disp() of B which prints private members x and y of A.

```
#include <iostream>
using namespace std;
class A {
private:
    char x='A';
    int y = 10;
public:
    friend class B;
};
class B {
public:
    void disp(A obj){
        cout<<obj.x<< " , " <<obj.y<<endl;
    }
};
int main() {
    B obj;
    A obj2;
    obj.disp(obj2);
    return 0;
}
```

The output is as expected:

A, 10

Friend Function

Similar to friend class, this function can access the private and protected members of another class. A global function can also be declared as friend as shown in the example below:

```
#include <iostream>
using namespace std;
class A {
private:
    char x='A';
    int y=10;
public:
```

```

    friend void disp(A obj);
};
//Global Function
void disp(A obj){
    cout<<obj.x<< " , " <<obj.y<<endl;
}
int main() {
    A obj;
    disp(obj);
    return 0;
}

```

Here the global function disp() is made friend to class A and hence, the private members of the object passed to it can be directly accessed from within the function. This feature is used in operator overloading of << and >> which are explained later in this text when polymorphism and overloading are explained.

Check Your Progress

- Note: a) Space is given below for writing your answers
 b) Compare your answers with the one given at the end of the unit

2. What are static members?

.....

11.4 SUMMARY

Inheritance is one of the important features of object oriented programming, through which the capabilities of a class can be extended by deriving a new class from an existing class. By doing so, the new class inherits all the public and protected properties of the existing class and the user can add more data and methods to extend the capabilities of a class. The class from which the new class is extended is known as the base class and the new class is called derived class. An abstract base class is a class where not all methods are implemented i.e., it has at least one pure virtual function. Since there is missing code for the pure virtual function, an object of this type cannot be created. A new class must be derived from this abstract base class to create objects of this type where the missing virtual functions should be defined. Here is the new version of the shape class that makes it abstract and all the derived classes implement the area function. C++ supports hierarchical inherence where the inheritance is a sequential chain of multiple base classes but is different from multiple inheritance. Static Variables are guaranteed a zero if not initialized to other values. Also an object need not be instantiated to use the static functions. This feature is used to generate wrapper classes. Wrapper classes are useful in pure object oriented languages, where one cannot write general purpose functions (say $\sin(x)$) outside of the classes. An object need not be created to call static methods or static members since they belong to the class but not to the object. Hence className.methodName() should be sufficient to invoke a method. A class cannot access the private members of another class. Similarly a class cannot access protected members of another class unless it is inherited from that class.

One exception in C++ is that a friend class or function can access any member of the any other class that declares this class or function as a friend. This is necessary if a class members are to be accessed legitimately by another class, as in case of a print class which prints the data.

11.5 CHECK YOUR PROGRESS MODEL ANSWERS

1. Is process of extending the features of existing class by creating a new class
 2. Static members are the members of the class itself and not the members of the objects
-

11.6 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Explain inheritance with code examples
2. Describe friend classes and friend functions with code examples
3. Write a program to demonstrate multiple inheritance?

II. Answer the following questions in about 15 lines each

1. Describe abstract base class.
 2. Explain static methods and static data members with examples
 3. Describe conflict resolution in multiple inheritance
-

11.7 GLOSSARY

Base Class	:	Class from which a new class is derives using inheritance
Derived Class	:	Class which is extended from base class using inheritance
Abstract class	:	Class whose member functions are not implemented
Static data member	:	Data member of a class which is shared by all the objects of a class.
Friend functions	:	Function which can access the private and protected members of another class

UNIT- 12: POLYMORPHISM

Contents

- 12.0 Objectives
 - 12.1 Introduction
 - 12.2 Virtual Functions
 - 12.3 Operator Overloading
 - 12.4 Templates
 - 12.5 Summary
 - 12.6 Check your progress – Model Answers
 - 12.7 Model Examination Questions
 - 12.8 Glossary
-

12.0 OBJECTIVES

After studying this unit, you should be able to

- describe polymorphism in C++
 - explain how to implement virtual functions in C++
 - describe operator overloading
 - understand templates
-

12.1 INTRODUCTION

The definition for polymorphism is “having many forms”. Typically, polymorphism occurs when there is a hierarchy of classes which are related by inheritance and the methods have the same signature. Since a base class pointer can point to a derived class object, a call to a member function using a base class pointer should select the proper derived class method instead of base class method. This selection is achieved in C++ with the support of polymorphism. Consider the case of diamond inheritance where class A is the base class of classes B and C and D is derived from class B and class C. Now if an object of type D is declared, constructor A() is run 2 times. Similarly, destructor of A (~A()) is also run twice. This causes serious problems. To avoid this, class A is declared as virtual base class of B and C. Then, there will be only one instance of A. In C++, any function name can be used any number of times except *main()*, with different argument list. This feature is known as overloading of a function. This was discussed earlier in. Similarly most of the operators can also be overloaded to give special meaning to the operators when used with those operators. In fact it is a must to over load the operators, if new data types are to be created and operations on them are to be defined.

12.2 VIRTUAL FUNCTIONS

Polymorphism

The definition for polymorphism is “having many forms”. Typically, polymorphism occurs when there is a hierarchy of classes which are related by inheritance and the methods have the same signature. Since a base class pointer can point to a derived class object, a call to a

member function using a base class pointer should select the proper derived class method instead of base class method. This selection is achieved in C++ with the support of polymorphism.

Consider the following example, which has a base class called Shape and the Rectangle and Triangle are derived from it. A pointer to base class pointer is used to call the area in the main function.

```
#include <iostream>
using namespace std;
class Shape {
protected: int w, h;
public:
Shape( int w = 0, int h = 0){
    this->w = w;
    this->h = h;
}
void area() {
    cout << "Base class area: —" << endl; // No generic formula for area
}
};
class Rectangle: public Shape {
public:
Rectangle( int w = 0, int h = 0):Shape(w, h) { }
void area () {
    cout << "Rectangle area: " << w*h << endl;
}
};

class Triangle: public Shape {
public:
Triangle( int w = 0, int h = 0):Shape(w, h) { }
void area () {
    cout << "Triangle area :"<< (w*h/2.0f)<<endl;
}
};

class Circle: public Shape {
public:
```

```

Circle(int r=0):Shape(r) { }
void area () {
cout << "Circle area :." << (3.14157*w*w)<<endl;
}
};

// Main function for the program
int main() {
    Shape *shape;
    Rectangle r(10,5);
    Triangle t(10,5);
    Circle c(10);
    shape = &r;
    shape->area();
    shape = &t;
    shape->area();
    shape = &c;
    shape->area();
    return 0;
}

```

In all the cases it only takes the base class area which prints “Base class area: —” message three times. This happens because the type information is bound to the pointer during compile time and it only knows the base class. To get the correct results, just add the ‘virtual’ keyword to the area() function in the base class.

```
virtual void area() { ...
```

This kind of automatic selection of appropriate function with base class pointer is known as runtime polymorphism, where the function is selected at runtime rather than at compile time as in case of non-virtual function.

Virtual Destructor

Consider A is the base class of B. Let B is allocating some memory in the constructor and freeing it in the destructor. . If a base class type pointer is used to refer the derived class, and then if we delete the object, then it tries to run the base class constructor only. To avoid this memory leakage, it is necessary that the derived class constructor be also run when the base class pointer is pointing to derived class objects. For this to run, the base class destructor is made virtual. See the code snippet below:

```

class A {
    A() { .. }
    virtual ~A() { .. }
};

```

```

class B: public A {
    B() { .. }
    ~B() { .. }
};
main()
{
    ....
    A *p;
    p = new B(); // Pointer of type base class pointing to derived class object
    ....
    delete p; // If ~A() is not virtual, then only ~A() is run.
}

```

Virtual Base Class

Consider the case of diamond inheritance where class A is the base class of classes B and C and D is derived from class B and class C. Now if an object of type D is declared, constructor A() is run 2 times. Similarly, destructor of A (~A()) is also run twice. This causes serious problems. To avoid this, class A is declared as virtual base class of B and C. Then, there will be only one instance of A. Consider the following example, where, the virtual base class is used. Try the same program without the virtual keyword.

```

#include <iostream.h>
using namespace std;
class A
{
public:
    A() { cout << "Base class A's constructor \n";}
    ~A() { cout << "Base class A's destructor \n"; }
};
class B: public virtual A
{
public:
    B() { cout << "Base class B's constructor \n";}
    ~A() { cout << "Base class B's destructor \n"; }
};
class C: virtual public A
{
public:
    C() { cout << "Base class C's constructor \n";}
    ~C() { cout << "Base class C's destructor \n"; }
}

```

```

};
class D: public B, public C
{
public:
    D() { cout << "Base class D's constructor \n";}
    ~D() { cout << "Base class D's destructor \n"; }
};
main()
{
    D x;
}

```

In the above program, observe virtual is used on either side of the public symbol. When one of the base classes is virtual, its constructor is run first, irrespective of the place where it is declared. Otherwise, the constructors are run in the order the base classes are declared. In this case, constructor of A is run only once.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

1. What is polymorphism?

.....

.....

.....

12.3 OPERATOR OVERLOADING

Operator Overloading

In C++, any function name can be used any number of times except main(), with different argument list. This feature is known as overloading of a function. This was discussed earlier in. Similarly most of the operators can also be overloaded to give special meaning to the operators when used with those operators. In fact it is a must to over load the operators, if new data types are to be created and operations on them are to be defined. The syntax is:

```

<type><class name>::operator <symbol>(argument list)
{
    operations defined relative to the class
}

```

Example: Following example creates a new user defined data type for complex numbers along with the basic operations on it + and =:

```

#include <iostream.h>

```

```

class complex
{
int x,y;
public:
complex operator + (complex);
complex operator = (complex);
void show(void);
void assign(int p, int q);
};
complex complex ::operator + (complex p)
{
complex temp;
temp.x = x+p.x;// x is short form of this->x
temp.y = y+p.y;
return temp;
}
complex complex::operator = (complex p)
{
x=p.x;
y=p.y;
return *this; // this is returned to facilitate cascading of =
}
void complex::show(void) { cout << "\n" << x << " +i " << y; }
void complex::assign(int p, int q=0) { x=p; y=q; }
void main()
{
complex a,b,c;
a.assign(2,3);
b.assign(4,5);
a.show();
b.show();
c=a+b;
c.show();
c=a+b+c;
c.show();
a=b=c;
}

```

```

a.show();
b.show();
c.show();
}

```

Overloading I/O Operators (Insertion (<<) and Extraction (>>))

In some instances, the information hiding access rules are too prohibitive. The friend mechanism gives nonmembers of the class access to the non-public members of a class. As an example, let us consider the insertion operator, which is overloaded to print an object. Let the object is of type A.

```

class A { ... }
A t;
cout << t;

```

In this case, the operator << takes the objects on either side as parameters. ostream& and A&. So the operator takes the form << (ostream&, A&). To print the private members of A, either the data must be public, or the private members of the class must be accessible to <<, by means of a friend keyword. The syntax is:

```

istream &operator >>(istream &<variable1>,<type>&<variable2>)
{
<your code here>
.....
return <variable1>;
}

```

```

ostream &operator <<(ostream &<variable1>,<type>&<variable2>)
{
<your code here>
.....
return <variable1>;
}

```

Observe the over loading of these operators in the examples.

```

#include <iostream.h>
using namespace std;
class complex
{
int x,y;
public:
friend istream& operator >> (istream& s, complex& c);
friend ostream& operator << (ostream& s, complex& c);

```

```

};
ostream & operator <<(ostream& s, complex& p)
{
s << "\n" << p.x << " +i " << p.y;
return s;
}
istream& operator >> (istream& s, complex& p)
{
cout << "\nEnter real and imaginary parts : ";
s >> p.x >> p.y;
return s;
}
void main()
{
complex a;
cin >> a;
cout << a;
}

```

Note that in the original class, these streams are declared as friend functions. This is necessary to have access of the private data that must be input or output.

Over Loading Unary Operators

Unary operators act on only one operand. For example ++ operates on only one operator like x++ or ++x. In this case, since the 'this' operator is automatically sent to all the member functions, it is not necessary to send any parameters. Chief disadvantage with unary increment and decrement operators is that they have no left/right sensitivity.

For example, if $i=1$, $j=i++$ gives $j=1$, $i=2$ and $j= ++i$ gives $i=2$, $j=2$. But in C++, if it is overloaded, it always gives the result of $j= ++i$.

A new way to distinguish these two cases is introduced later, where, for post increment, a dummy integer is taken as a parameter. See the example below

```

#include <iostream.h>
class counter
{
int count;
public:
counter() { count=0;} // no argument constructor
counter(int c) { count=c;} // one argument constructor
int get_count() {return count;}
counter operator ++ (); // pre-increment

```



```

counter operator ++(int); // post-increment
};
counter counter::operator ++()
{
count ++; // this->count incremented
counter temp; // new counter declared.
temp.count = count;
return temp;
}
counter counter::operator ++(int dummy)
{
counter temp;
temp.count = count; // this->count is assigned to temp.count
this->count++;
return temp;
}
void main()
{
counter c1, c2(10);
cout << "\n C1 = " << c1.get_count();
cout << "\n C2 = " << c2.get_count();
c1= c2++;
cout << "\n C1 = " << c1.get_count();
cout << "\n C2 = " << c2.get_count();
c1= ++c2;
cout << "\n C1 = " << c1.get_count();
cout << "\n C2 = " << c2.get_count();
}

```

User defined Data Types

C++ supports the user defined data types with either struct or class. The main difference between C user defined types and C++ user defined types is that the operators can be overloaded appropriately in C++, which is not possible in C. An example of user defined data type is presented above in the operator overloading section where complex user defined type is presented.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

2. What is function overloading?

.....
.....
.....

12.4 TEMPLATES

Templates are powerful features of C++ which allows you to write generic programs. Using templates, it is easy to create a single function or a class to work with different data types. These are often used in larger coding projects for the purpose of code reusability and flexibility of the programs.

The concept of templates can be used in two different ways:

- Function Templates
- Class Templates

Function Templates

A function template works in a similar to a normal function, with one key difference. A single function template can work with different data types at once but, a single normal function can only work with one set of data types. Normally, if you need to perform identical operations on two or more types of data, you use function overloading to create two functions with the required function declaration. However, a better approach would be to use function templates because you can perform the same task writing less and maintainable code.

Stax of funxtion templates

template<**class** T>

T someFunction(T arg)

```
{  
    ... ..  
}
```

In the above code, *T* is a template argument that accepts different data types (int, float), and **class** is a keyword. You can also use keyword **typename** instead of class in the above example. When, an argument of a data type is passed to someFunction(), compiler generates a new version of someFunction() for the given data type.

Example: Finding smallest of two generic variables

```
using namespace std;  
// template function  
template <class T>  
T Small(T g1, T g2)  
{
```

```

return (g1 < g2) ? g1 : g2;
}
int main()
{
int t1, t2;
float f1, f2;
char c1, c2;
cout << "Enter two integers:\n";
cin >> t1 >> t2;
cout << Small(i1, i2) <<" is Smaller." << endl;
cout << "\nEnter two floating-point numbers:\n";
cin >> f1 >> f2;
cout <<Small(f1, f2) <<" is Smaller." << endl;
cout << "\nEnter two characters:\n";
cin >> c1 >> c2;
cout <<Small(c1, c2) << " has smaller ASCII value.";
return 0;
}

```

Class Templates

Like function templates, class templates can be created for generic class operations. These are useful on the occasions where a class is implemented for all classes, only the data types used are different. Genarelly we create a different class for each data type or create different member variables and functions within a single class. This will unnecessarily increase the source code and will be hard to maintain, as the change is one class/function should be performed on all classes/functions. Hence class templates make it easy to reuse the same code for all data types.

Syntax of class template

template<**class** T>

```

class className
{
... ..
public:
    T var;
    T someOperation(T arg);
... ..
};

```

Where T is the template argument which is a placeholder for the data type used. Inside the class body, a member variable *var* and a member function *someOperation()* are both of type T.

Syntax of creating a class template object

```
className<dataType> classObject;
```

Example

```
className<int> classObject;
```

```
className<float> classObject;
```

```
className<string> classObject;
```

Example: Simple calculator using class template

```
#include <iostream>
```

```
using namespace std;
```

```
template <class T>
```

```
class Calculator
```

```
{
```

```
private:
```

```
    T num1, num2;
```

```
public:
```

```
    Calculator(T n1, T n2)
```

```
    {
```

```
        num1 = n1;
```

```
        num2 = n2;
```

```
    }
```

```
    void displayResult()
```

```
    {
```

```
        cout << "Numbers are: " << num1 << " and " << num2 << "." << endl;
```

```
        cout << "Addition is: " << add() << endl;
```

```
        cout << "Subtraction is: " << subtract() << endl;
```

```
        cout << "Product is: " << multiply() << endl;
```

```
        cout << "Division is: " << divide() << endl;
```

```
    }
```

```
    T add() { return num1 + num2; }
```

```
    T subtract() { return num1 - num2; }
```

```

    T multiply() { return num1 * num2; }

    T divide() { return num1 / num2; }
};
int main()
{
    Calculator<int> intCalc(2, 1);
    Calculator<float> floatCalc(2.4, 1.2);

    cout << "Int results:" << endl;
    intCalc.displayResult();

    cout << endl << "Float results:" << endl;
    floatCalc.displayResult();

    return 0;
}

```

In the above program, the class contains two private members of type T: *num1* & *num2*, and a constructor to initialize the members. It also contains public member functions to calculate the addition, subtraction, multiplication and division of the numbers which return the value of data type defined by the user. Likewise, a function *displayResult()* to display the final output to the screen. In the *main()* function, two different Calculator objects *intCalc* and *floatCalc* are created for data types: *int* and *float* respectively. The values are initialized using the constructor.

Check Your Progress

Note: a) Space is given below for writing your answers

b) Compare your answers with the one given at the end of the unit

3. What is a template?

.....

.....

.....

12.5 SUMMARY

Virtual Function is a function in base class, which is overridden in the derived class, and which tells the compiler to perform Late Binding on this function. In Late Binding function call is resolved at runtime. Hence, now compiler determines the type of object at runtime, and then binds the function call. Late Binding is also called Dynamic Binding or Runtime Binding. We can call private function of derived class from the base class pointer with the help of virtual keyword. Compiler checks for access specifier only at compile time. So at run time when late binding occurs it does not check whether we are calling the private function or public function. On using Virtual keyword with Base class's function, Late Binding takes place and the derived version of function will be called, because base class pointer points to Derived class object.

Templates are powerful features of C++ which allows you to write generic programs. Using templates, it is easy to create a single function or a class to work with different data types. These are often used in larger coding projects for the purpose of code reusability and flexibility of the programs. A function template works in a similar to a normal function, with one key difference. A single function template can work with different data types at once but, a single normal function can only work with one set of data types. Normally, if you need to perform identical operations on two or more types of data, you use function overloading to create two functions with the required function declaration. However, a better approach would be to use function templates because you can perform the same task writing less and maintainable code.

12.6 CHECK YOUR PROGRESS MODEL ANSWERS

1. The definition for polymorphism is “having many forms”. Typically, polymorphism occurs when there is a hierarchy of classes which are related by inheritance and the methods have the same signature
2. In C++, any function name can be used any number of times except main(), with different argument list. This feature is known as overloading of a function
3. Template creates a single function or a class to work with different data types

12.7 MODEL EXAMINATION QUESTIONS

I. Answer the following questions in about 30 lines each

1. Explain virtual functions with an example
2. Explain overloading of unary operator with code examples
3. Write a program to demonstrate function template

II. Answer the following questions in about 15 lines each

1. Describe virtual base class with examples
2. Explain operator overloading with + operator with a code example
3. Write a program to demonstrate class template

12.8 GLOSSARY

Virtual function	:	Virtual Function is a function in base class, which is overridden in the derived class, and which tells the compiler to perform Late Binding
Template	:	Template creates a single function or a class to work with different data types.
C++98	:	The ISO C++ standard.
Const_cast	:	A C++ keyword used as a style of cast for explicitly casting away const.
Dynamic type	:	The type of an object as determined at run-time.

. B.R. Ambedkar Open University
B.Sc / B.Com./ B.A.
2nd Year 3rd Semester (3 year degree course)
MODEL QUESTION PAPER
COMPUTER APPLICATIONS DSC-3
Programming with C and C++

[Time: 3 hours]

[Max. Marks: 80]

Section – A
[Short Answer Questions]
[Marks: 4x5=20]

Note: a) Answer any four of the following in about 10 lines each
b) Each question carries **5 marks**

1. **[Block-I]** Explain Algorithm with an example.
2. **[Block-I]** Describe the primitive data types in C.
3. **[Block-II]** Describe do-while with a code example.
4. **[Block-II]** Write a program to find the factorial of a number using recursive function.
5. **[Block-III]** Write a program to read, print two dimensional array.
6. **[Block-III]** Discuss passing structures to functions with a code example.
7. **[Block-IV]** Write a program to demonstrate function templates.
8. **[Block-IV]** Explain multiple inheritance with an example

Section –B
[Essay type]
[Marks: 4x10=40]

Note: a) Answer any **four** of the following in about 30 lines each
b) Each question carries **10 marks**

9. **[Block-I]** Draw the flowchart to print prime numbers between 1 and 100.
(Or)
10. **[Block-I]** Write a program to demonstrate Arithmetic and assignment operators.
11. **[Block-II]** Explain switch and while with an example.
(Or)
12. **[Block-II]** Write a program to demonstrate operations on strings along with output
13. **[Block-III]** Describe Unions with an example.
(Or)
14. **[Block-III]** Write a program to copy one file to another file
15. **[Block-IV]** Describe the types of constructors and destructor with examples.
(Or)
16. **[Block-IV]** Write a program to overload +, - operators.

Section –C

[Objective type questions]

[Marks: 20]

Total Number of questions 20-[15 from (Theory) and 5 from (Practical's)]

I. Multiple choice questions (10 marks)

1. Abbreviation of Programming in Logic..... (3)
1) ProLang 2) Prologic 3) Prolog 4) LogicPro
2. Abbreviation of Common Business Oriented Language (2)
1) COMBIL 2) COBOL 3) ALGOL 4) BASIC
3. Which of the followings is automatically added to every class, if we do not write our own. (1)
1) Constructor 2) Member funciton 3) Class name 4) Private
4. Operator overloading is supported byfeature of C++ (2)
1) Encapsulation 2) Polymorphism
3) Late Management System 4) Inheritance
5. A member function can always access the data in _____, (in C++). (3)
1) Public member 2) Private member
3) Members of same class 4) Members of same object
6.is size of float data type in C (2)
1) 2 Bytes 2) 4 Bytes 3) 6 Bytes 4) 8 Bytes
7. In scanf () or printf() %u indicates..... (3)
1) Float 2) double 3) Unsigned int 4) Signed int
8. Out put of printf("i= %d" , 123.45678 is (1)
1) 123 2) 123.45 3) 123.45678 4) .45678
9. if int *p,i=25; p=&I; then Out put of printf(*p) is (2)
1) Comilation error 2) 25 3) 0xA2356B 4)) Runtime error
10. A member function with the class name deletes the objects (3)
1) Constructor 2) Delete operator 3) Destructor 4) This operator

II. Match the following (5 marks)

- | | | |
|-------------------------|-------|---------------------|
| 1. Encapsulation | (c) | a. Polymorphism |
| 2. Pointer | (e) | b. Data type |
| 3. Operator Overloading | (a) | c. Class |
| 4. int | (b) | d. Generic Class |
| 5. Template | (d) | e. Address variable |
| | () | f. Object |

III. Fill in the blanks (5 marks)

1. Full form of SNOBOL StriNg Oriented and symBOLic Language
2. Full form of ALGOL is Algorithmic Language.
3. A pointer can hold Address of a variable.
4. A member function with class name which initialize members is Constructor.
5. In C++ members of structure are by default Public.

BS 415 CA-E

B.A/B.Com/B.Sc

SECOND YEAR

SEMESTER-IV

DISCIPLINE SPECIFIC CORE COURSE-DSC-4

COMPUTER APPLICATIONS

DATA BASE MANAGEMENT SYSTEM



"We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit"

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

CONTENTS

Block/Unit	Title	Page
BLOCK-I: INTRODUCTION TO DBMS		
Unit-1:	Fundamentals of DBMS	3
Unit-2:	Entity-Relationship model	3 - 12
Unit-3:	Relational Model	13 - 29
		30 - 43
BLOCK-II: DBMS ARCHITECTURES		
Unit-4:	Normalization	45
Unit-5:	Data Base System Architectures and Data Models	47 - 57
Unit-6:	Storage Structures	58 - 81
		82 - 93
BLOCK-III: FILE STRUCTURES AND TRANSACTION MANAGEMENT		
Unit-7:	File Organization	95
Unit-8:	Storage Access	97 - 115
Unit-9:	Data Base Transactions	116 - 129
		130 - 144
BLOCK-IV: STRUCTURED QUERY LANGUAGE		
Unit-10:	Introduction to SQL	145
Unit-11:	SQL Data Manipulation Language	147 - 164
Unit-12:	Advanced SQL	165 - 178
		179 - 186
	Model Question Paper	187 - 188

UG 405 SEE (CA 1)-E

B.A/B.Com/B.Sc

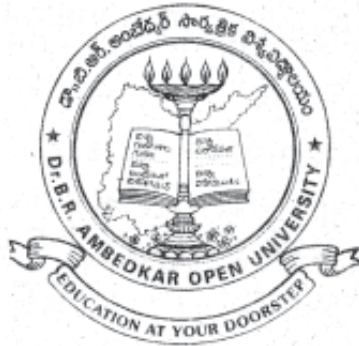
SECOND YEAR

SEMESTER-IV

SKILL ENHANCEMENT COMPULSORY COURSE-SECC-1

COMPUTER APPLICATIONS

MULTIMEDIA APPLICATIONS USING GIMP



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2020

CONTENTS

Block/Unit	Title	Page
BLOCK – I: THEORY OF MULTIMEDIA		1
Unit-1:	Fundamentals of GIMP and Multimedia:	3 -15
Unit-2:	Working with Images	16 -29
Unit-3:	Sound and Video	30 -47
Unit-4:	Making Multimedia	48 -60
BLOCK – II: MULTIMEDIA WITH GIMP		61
Unit-5:	Basics of GIMP	63 -87
Unit-6:	Quick Mask, Layer Mask, Layers, Paths	88 -111
Unit-7:	Advanced Features of GIMP	112 -145
Unit-8:	Animation with GIMP	146 -162
	Model Question paper	163-164

UG 405 SEE (CA 2)-E

B.A/B.Com/B.Sc

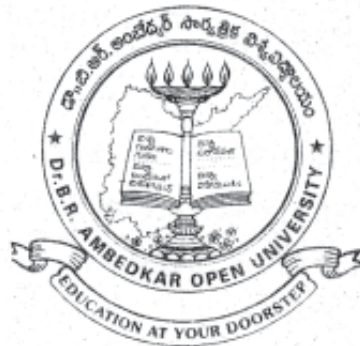
SECOND YEAR

SEMESTER-IV

SKILL ENHANCEMENT COMPULSORY COURSE-SEE-CA-2

COMPUTER APPLICATIONS

MULTIMEDIA APPLICATIONS USING BLENDER



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2020

CONTENTS

Block/Unit	Title	Page
BLOCK – I: FUNDAMENTALS OF BLENDER		3
Unit-1:	Blender User Interface:	3-20
Unit-2:	Rendering with Blender	21-37
Unit-3:	Shading and Rendering	38-62
Unit-4:	Blender Layers and Passes	63-76
BLOCK – II: ADVANCED FEATURES OF BLENDER		77
Unit-5:	Blender Modeling	79-94
Unit-6:	Animation and Rigging	95-116
Unit-7:	Visual Effects and Simulation	117-136
Unit-8:	Video Editing	137-155
	Model Question Paper	156-157

BS 515 CA-E

B.A/B.Com/B.Sc

THIRD YEAR

SEMESTER-V

DISCIPLINE SPECIFIC CORE COURSE-DSC-5

COMPUTER APPLICATIONS

PROGRAMMING WITH JAVA



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2023

CONTENTS

Block/Unit	Title	Page
BLOCK – I: BASIC FEATURE OF JAVA		1
Unit-1:	Introduction to Java Language	3-14
Unit-2:	Data Types	15 - 22
Unit-3:	Java Operators and Control Structures	23 - 40
BLOCK – II: JAVA AND OOPs		41
Unit-4:	Introduction to Classes and Objects	43 - 55
Unit-5:	Additional Features of Java OOPs	56 - 69
Unit-6:	Inheritance and Abstract Classes	70 - 81
BLOCK – III: ADDITIONAL FEATURES OF JAVA		83
Unit-7:	Packages and Interfaces	85 - 94
Unit-8:	Java Exception Handling	95 - 109
Unit-9:	Java Collections	110 - 127
BLOCK – IV: BACK-END AND FRONT-END TOOLS OF JAVA		129
Unit-10:	Java Data Base Connectivity	131 - 144
Unit-11:	Java AWT and Event Handling	145 - 158
Unit-12:	Java Visual Programming with Swing	159 - 184
Model Question Paper		185 - 187

[BS 515 CA DSE (A) - E]

B.A/B.Com/B.Sc

THIRD YEAR

SEMESTER-V

DISCIPLINE SPECIFIC ELECTIVE COURSE-DSEC-1

COMPUTER APPLICATIONS

OPERATING SYSTEMS



"We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit"

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2023**

CONTENTS

Block/Unit	Title	Page
BLOCK – I: INTRODUCTON TO OPERATING SYSTEMS		1
Unit-1:	Fundamentals of Operating Systems	3-15
Unit-2:	Processes and Threads	16-40
Unit-3:	Memory Management	41-65
BLOCK – II: MANAGEMENT O F FILES, INPUT/OUTPUT		67
Unit-4:	File Systems	69-93
Unit-5:	Input/Output	94-125
Unit-6:	Dead Locks	126-144
BLOCK – III: VIRTUALIZATION AND CLOUD MANAGEMENT		145
Unit-7:	Virtualization and Cloud - I	147-159
Unit-8:	Virtualization and Cloud-II	160-185
Unit-9:	Multiple Processor Systems	186-196
BLOCK – IV: OPERATING SYSTEM SECURITY		197
Unit-10:	Security-I	199-215
Unit-11:	Security-II	216-233
Unit-12:	Android Case Study	234-255
Model Question Paper		256-257

BS615CA-E

B.A/B.COM/B.Sc

THIRD YEAR SEMESTER-VI

COMPUTER APPLICATIONS

INTERNET TECHNOLOGIES AND HTML5



"We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit"

-Dr. B. R. Ambedkar

Dr. B. R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2022

CONTENTS

BLOCK-1: Introduction to Data Communication and Networks	1
Unit - 1 : Introduction to data communication	3
Unit - 2 : Network Types	13
Unit - 3 : Transmission Media	24
BLOCK-2: LAN Technologies and OSI Layers	37
Unit – 4: LAN Technologies and Overview of OSI Layers: Physical and Data Link Layers	39
Unit – 5: OSI Layers: Network and Transport Layers	49
Unit – 6: OSI Layers: Session, Presentation and Application Layers	57
BLOCK-3 : Hyper Text Markup Language – HTML - 1	63
Unit - 7 : HTML and HTML Formatting	65
Unit - 8 : HTML Styles	75
Unit - 9 : HTML Forms	92
BLOCK-4 : Hyper Text Markup Language - HTML - 2	103
Unit – 10: HTML Media	105
Unit – 11: CSS	115
Unit – 12: CSS Advance	155
Model question papers	195

BS 615 CA DSE (C) - E

B.A/B.Com/B.Sc

THIRD YEAR

SEMESTER-VI

COMPUTER APPLICATIONS

DISCIPLINE SPECIFIC ELECTIVE COURSE - DSE(C)

DATA STRUCTURES USING JAVA



"We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit"

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2023

CONTENTS

Block/Unit	Title	Page
BLOCK – I: LISTS AND STACKS		1
Unit-1:	Mathematical Review on Analysis of Algorithms	3-18
Unit-2:	Stack Data Structure	19-40
Unit-3:	List Data Structure	41-67
BLOCK – II: QUEUES AND TREES		67
Unit-4:	Queue Data Structure	71-96
Unit-5:	Tree Data Structure	97-124
Unit-6:	Specialised Trees	125-159
BLOCK – III: HASHING AND HEAPING		161
Unit-7 :	Hashing	163-186
Unit-8 :	Priority Queues (Heaps)	187-196
Unit-9 :	Sorting - I	197-212
BLOCK – IV: SEARCHING AND SORTING		213
Unit-10 :	Sorting II	215-227
Unit-11 :	Searching	228-239
Unit-12 :	Graphs	240-257
Model Question Paper		258-259

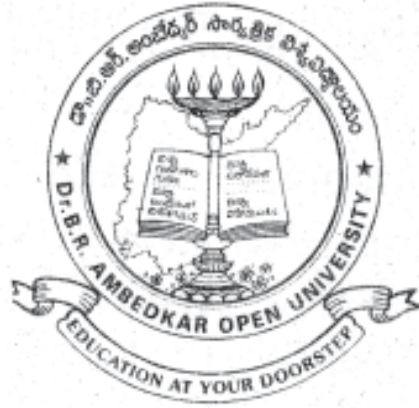
బి.ఎస్.సి

మొదటి సంవత్సరం

మొదటి సెమిస్టర్

భూవిజ్ఞాన శాస్త్రం

సాధారణ లేక భౌతిక భూవిజ్ఞాన శాస్త్రం



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ,
సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను,
హక్కును మాత్రం కోల్పోకూడదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2017

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజి.నెం.
ఖండం-1	సాధారణ భూవిజ్ఞాన శాస్త్రం	1
భాగం-1	భూవిజ్ఞాన శాస్త్రం పరిధి, అభివృద్ధి	3-12
భాగం-2	సౌర వ్యవస్థ	13-21
భాగం-3	భూమి	22-32
ఖండం -2	అంతర్గత భౌమ చర్యలు	33
భాగం-4	పర్వతాలు	35-41
భాగం-5	భూకంపాలు	42-48
భాగం-6	అగ్నిపర్వతాలు	49-58
ఖండం -3	బహిర్గత భౌమ చర్యలు-1	59
భాగం-7	శిలాశైథిల్యం	60 -69
భాగం-8	నదులు	70-88
భాగం-9	భూగర్భజలం	89-100
ఖండం -4	బహిర్గత భౌమ చర్యలు -2	101
భాగం-10	హిమానీ నదాలు	103-115
భాగం-11	సరస్సులు, సముద్రాలు	116-134
భాగం-12	పవనాలు	135-144

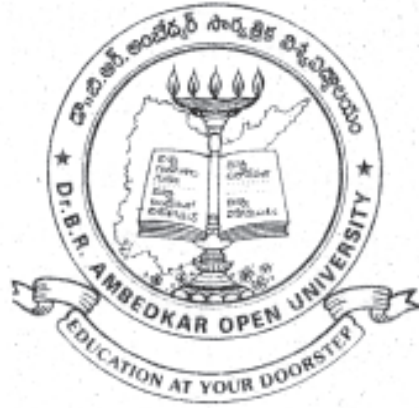
బి.ఎస్.సి

మొదటి సంవత్సరం

రెండవ సెమిస్టర్

భూవిజ్ఞాన శాస్త్రం

స్ఫటిక శాస్త్రం, ఖనిజ శాస్త్రం



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ,
సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను,
హక్కును మాత్రం కోల్పోకూడదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2018

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజి.నెం.
ఖండం-1	స్పటికశాస్త్రం	1
భాగం-1 :	స్పటికశాస్త్రం	3-25
భాగం-2 :	సమాక్ష, చతుష్కోణ, విషమాక్ష వ్యవస్థలు	26-36
భాగం-3 :	ఏకనత, త్రినత, షట్కోణ వ్యవస్థలు	37-45
ఖండం-2	ఖనిజ శాస్త్రం	47
భాగం-4 :	ఖనిజ శాస్త్ర పరిచయం	49-57
భాగం-5 :	ఖనిజాల వర్గీకరణ	57-66
భాగం-6 :	ఖనిజాల భౌతిక, రసాయన ధర్మాలు	67-79
ఖండం-3	ఖనిజ కుటుంబాలు	81
భాగం-7 :	నీసో, సారో, సైక్లో, ఐనో సిలికేటులు	83 -91
భాగం-8 :	ఫిల్టో, టెక్టోసిలికేటులు, ఇతర ముఖ్య ఖనిజాలు	92-99
భాగం-9 :	నాన్ సిలికేట్ ఖనిజాలు	100-114
ఖండం-4	ప్రకాశ ఖనిజ శాస్త్రం	115
భాగం-10 :	ప్రకాశ ఖనిజశాస్త్రం అభివృద్ధి	117-127
భాగం-11 :	ధృవణ సూక్ష్మదర్శిని - వర్ణన	128-135
భాగం-12 :	ఖనిజ ప్రకాశ ధర్మాలను నిర్ధారించడం	136-147

BS316 GEO-T

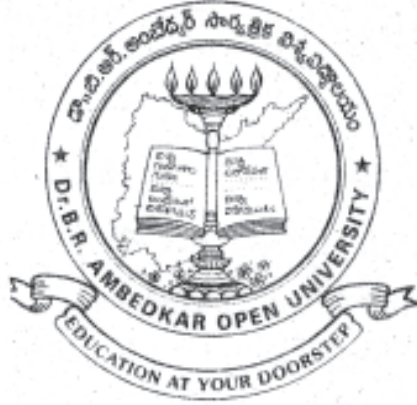
బి.ఎస్.సి

రెండవ సంవత్సరం

మూడవ సెమిస్టర్

భూవిజ్ఞాన శాస్త్రం

అగ్నిమయ శిలాశాస్త్రం



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు.

ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేది లేదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2018

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజి.నెం.
ఖండం-1 శిలాశాస్త్రం - పరిచయం		1
భాగం-1 :	శిలా శాస్త్రం - శిలల వర్గీకరణ	3-10
భాగం-2 :	మాగ్నాల ఆవిర్భావము, రకాలు	11-19
భాగం-3 :	మాగ్నాలు, వాటి స్వభావం	20-27
ఖండం-2 అగ్నిశిలల లక్షణాలు		29
భాగం-4 :	అగ్నిశిలల రూపాలు	31-42
భాగం-5 :	అగ్నిశిలల నిర్మితులు	43-49
భాగం-6 :	అగ్నిశిలల వయనాలు, సూక్ష్మనిర్మితులు	50-69
ఖండం-3 అగ్నిశిలల ఆవిర్భావం		71
భాగం-7 :	మాగ్నాల స్ఫటికీకరణ	73 -81
భాగం-8 :	బవెన్ ప్రతిచర్య నియమం	82-85
భాగం-9 :	అగ్నిశిలల ఉద్భవం	86-92
ఖండం-4 అగ్నిశిలల వర్గీకరణ - రకాలు		93
భాగం-10 :	అగ్నిశిలల వర్గీకరణ	95-105
భాగం-11 :	అగ్నిశిలల వర్ణన - పాతాళ అగ్నిశిలలు	106-112
భాగం-12 :	అగ్నిశిలల వర్ణన -ఉపపాతాళ, అగ్నిపర్వత శిలలు	113-119

BS 416 GEO -T

బి.ఎస్సి

రెండవ సంవత్సరం

నాల్గవ సెమిస్టర్

భూవిజ్ఞాన శాస్త్రం

అవక్షేప, రూపాంతర ప్రాప్తి శిలాశాస్త్రం



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు.

ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేది లేదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2019

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజి.నెం.
ఖండం-1 అవక్షేప శిలా శాస్త్రం		1
భాగం-1	: అవక్షేప శిలల ఆవిర్భావం	3-11
భాగం-2	: అవక్షేప శిలల లక్షణాలు	11-22
భాగం-3	: అవక్షేప శిలల వర్గీకరణ	23-41
ఖండం-2 రూపాంతర ప్రాప్తిశిలా శాస్త్రం -1		43
భాగం-4	: రూపాంతర ప్రాప్తి కారకాలు, రీతులు,	45-52
భాగం-5	: రూపాంతర ప్రాప్తి శిలల ఖనిజ లక్షణాలు	53-59
భాగం-6	: రూపాంతర ప్రాప్తి శిలల వయనాలు, నిర్మితులు, వర్గీకరణ	60-67
ఖండం-2 రూపాంతర ప్రాప్తిశిలా శాస్త్రం -2		69
భాగం-7	: మాగ్మాలు - రూపాంతర ప్రాప్తి	71-76
భాగం-8	: కెటాక్లాస్టిక్ రూపాంతర ప్రాప్తి	77-81
భాగం-9	: ఉష్ణీయ రూపాంతర ప్రాప్తి	82-87
ఖండం-2 రూపాంతర ప్రాప్తిశిలా శాస్త్రం -3		89
భాగం-10	: గతశిల - ఉష్ణీయ రూపాంతర ప్రాప్తి	91-96
భాగం-11	: పాతాళ రూపాంతర ప్రాప్తి	97-102
భాగం-12	: ప్రత్యేకమైన భారతదేశపు శిలలు	103-108

UG 405 SEE (GEO 1)-E

B.Sc

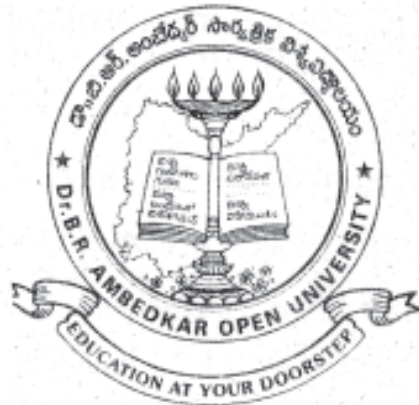
SECOND YEAR

SEMESTER-IV

GEOLOGY

SKILL ENHANCEMENT ELECTIVE COURSE-SEE-1

REMOTE SENSING, GIS & GPS



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

BLOCK / Unit	Title	Page
BLOCK – I: REMOTE SENSING		
Unit-1	Introduction to Geoinformatics : RS, GIS and GPS	3-17
Unit-2	Remote Sensing Data Acquisition, Platforms and Sensors	18-31
Unit-3	Remote Sensing Data Analysis	32-44
Unit-4	Global Positioning System (GPS)	45-51
Block - II GIS & GPS		
Unit-5	Introduction to GIS and Its Linkage to RS	55-71
Unit-6	GIS Data Products : Spatial and Attribute Data	72-81
Unit-7	Spatial Data Measurements and Analysis	82-100
Unit-8	Integration of RS, GIS, GPS and Applications	101-111

UG 405 SEE (GEO 2)-E

B.Sc

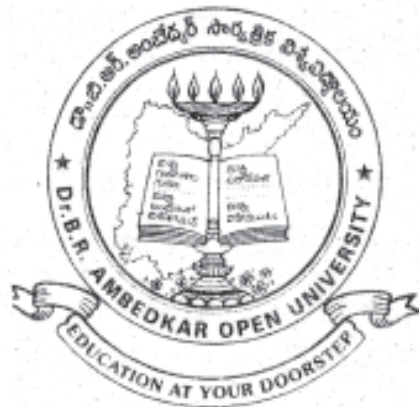
SECOND YEAR

SEMESTER-IV

GEOLOGY

SKILL ENHANCEMENT ELECTIVE COURSE-SEE-2

GEOCHEMISTRY



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

BLOCK / Unit	Title	Page
BLOCK- I Geochemistry-1		1
Unit-1	Basic Concepts of Geochemistry	3-12
Unit-2	Periodic table	13-22
Unit-3	Composition of Planets, Meteorites and Earth	23-29
Unit-4	Analytical Techniques –I	30-38
BLOCK-II Geochemistry-2		39
Unit-5	Geochemical Cycles	41-50
Unit-6	Geochemical classification and Distribution of elements	51-57
Unit-7	Water quality	58-66
Unit-8	Analytical Techniques –II	67-79

BS 516 GEO-E

B.Sc

THIRD YEAR

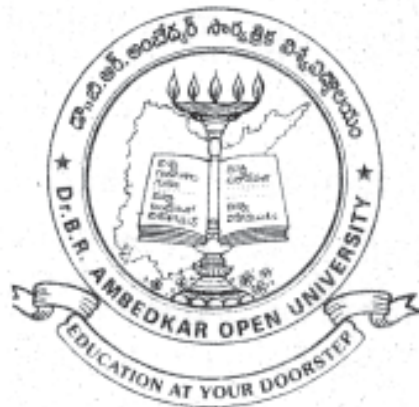
SEMESTER-V

GEOLOGY

DISCIPLINE SPECIFIC COMPULSORY CORE COURSE

STRUCTURAL GEOLOGY

ECONOMIC GEOLOGY



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit..

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2020

CONTENTS

BLOCK / Unit	Title	Page
BLOCK-I	STRUCTURAL GEOLOGY - FUNDAMENTALS	1
Unit-1	: Structural Geology - An Introduction	3-8
Unit-2	: Mechanical Principles and Field Procedures	9-18
Unit-3	: Primary Structures	19-29
BLOCK-II	IMPORTANT STRUCTURES	31
Unit-4	: Unconformities and Joints	33-42
Unit-5	: Folds	43-55
Unit-6	: Faults	56-70
BLOCK-III	ECONOMIC GEOLOGY PROCESS OF FORMATION OF ORES	71
Unit-7	: Scope of Economic Geology	73-77
Unit-8	: Igeous Process or Primary Processes	78-92
Unit-9	: Secondary, Organic and Metamorphic Processes	93-106
BLOCK- IV	DESCRIPTION OF ECONOMIC MINERAL DEPOSITS	107
Unit-10	: Metallic Minerals	109-119
Unit-11	: Non - Metallic Minerals	120-128
Unit-12	: Fules and Radio - Active Minerals.	129-135
	Model Question Paper	136-137

BS 516 GEO DSE (A)-E

B.Sc

THIRD YEAR

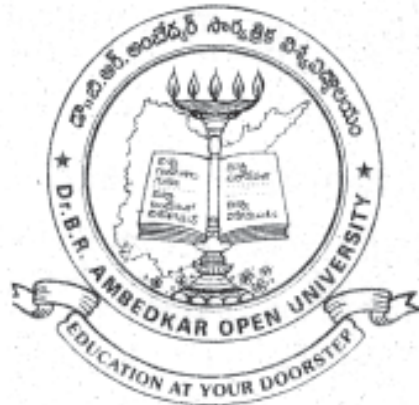
SEMESTER-V

GEOLOGY

DISCIPLINE SPECIFIC ELECTIVE - A

MINERAL EXPLORATION

MINERAL ECONOMICS



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit..

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2020

CONTENTS

BLOCK / Unit	Title	Page
BLOCK-I	MINERAL EXPLORATION - I	1
Unit-1	: Principles and Strategies of Mineral Exploration	3-13
Unit-2	: Geological Exploration I - Physiographic and Mineralogical Guides	14-24
Unit-3	: Geological ExplorationII-Stratigraphic, Lithological & Structural Guides	25-41
BLOCK-II	MINERAL EXPLORATION - II	43
Unit-4	: Geophysical Methods	45-51
Unit-5	: Geochemical and Geobotanical Methods	52-60
Unit-6	: Drilling Methods, Sampling Methods and Estimation of Ore Deposit	61-76
BLOCK-III	MINERAL ECONOMICS - I	77
Unit-7	: Principles of Mineral Economics	79-86
Unit-8	: Methods of Mineral Dressing	87-94
Unit-9	: Classification of War Minerals : Strategic, Critical and Essential.	95-108
BLOCK-IV	MINERAL ECONOMICS - II	109
Unit-10	: National Mineral Policy	111-119
Unit-11	: Conservation and Substitution of Minerals	120-129
Unit-12	: Growth of Mineral Industry - Mineral Legislation	130-144
	Model Question Paper	145-146

BS 516 GEO DSE (B) -E

B.Sc

THIRD YEAR

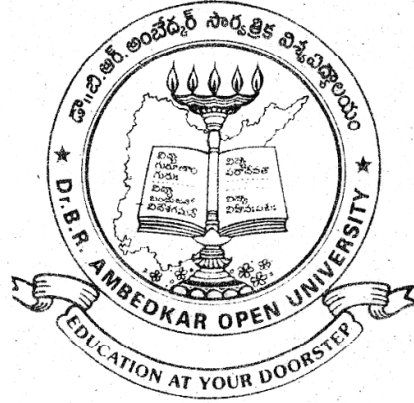
SEMESTER-V

GEOLOGY

DISCIPLINE SPECIFIC ELECTIVE - B

MINING GEOLOGY

ORE DRESSING



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit..

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2021

CONTENTS

BLOCK / Unit	Title	Page
BLOCK-I MINING GEOLOGY - 1		1
Unit-1	: Introduction - Mining Geology	3-8
Unit-2	: Methods of Breaking Rocks	9-16
Unit-3	: Open Cost Mining Methods	17-25
BLOCK-II MINING GEOLOGY - 2		27
Unit-4	: Underground Mining Methods	29-45
Unit-5	: Mine Drainage and Pumping	46-58
Unit-6	: Mining Environmental Issues	59-70
BLOCK-III ORE DRESSING - 1		71
Unit-7	: Introduction to Ore Dressing	73-82
Unit-8	: Crushing, Grinding and Seizing Methods	83-103
Unit-9	: Classification and Concentration Methods for Ore Dressing.	104-116
BLOCK-IV ORE DRESSING - 2		117
Unit-10	: Gravity and Flotation Concentration Methods	119-135
Unit-11	: Magnetic and Electric Separation Methods	136-144
Unit-12	: Flow sheets for Important ores	145-155
	Model Question Paper	156-157

BS 616 GEO - T

బి.ఎస్.సి

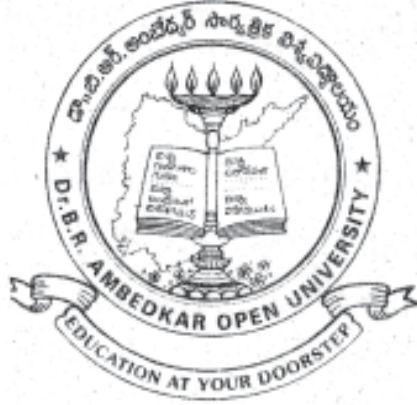
మూడవ సంవత్సరం

ఆరవ సెమిస్టర్

భూవిజ్ఞాన శాస్త్రం

భారతదేశ భూవిజ్ఞాన శాస్త్రం

పురాణీయ శాస్త్రం



“మనం నాగరికత సమకూర్చిన పస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు.

ఎందుకంటే విద్యను మించిన పస్తుగత ప్రయోజనమేది లేదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2020

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజి.నెం.
ఖండం-1	భారతదేశ భూవిజ్ఞాన శాస్త్రం-I	1
భాగం-1 :	స్తరశాస్త్ర నియమాలు, భౌమ కాలపట్టిక మరియు భారతదేశ నైసర్గిక స్వరూపం	3-21
భాగం-2 :	ఆర్కియన్ సముదాయం	22-34
భాగం-3 :	ప్రోటిరోజోయిక్ సముదాయం	35-48
ఖండం-2	భారతదేశ భూవిజ్ఞాన శాస్త్రం- II	49
భాగం-4 :	పేలియోజోయిక్, మీసోజోయిక్ మరియు గోండ్వానా సముదాయం	51-67
భాగం-5 :	దక్కను నాపలు, టెర్షియరీ మరియు తరుణ విన్యాసాలు	68-83
భాగం-6 :	ఆంధ్రప్రదేశ్ మరియు తెలంగాణా భూ విజ్ఞానం	84-95
ఖండం-3	పురాజీవ శాస్త్రం-I	97
భాగం-7 :	పురా జీవశాస్త్రం, శిలాజీకరణ పరిస్థితులు, శిలాజాల ఉపయోగాలు	99-108
భాగం-8 :	జీవుల వర్గీకరణ	109-120
భాగం-9 :	ఫోరామినిఫెరా, ప్రవాళాలు	121-142
ఖండం-4	పురాజీవ శాస్త్రం - II	143
భాగం-10 :	గాస్ట్రోపాడ్, సెఫలోపాడ్ జీవులు	145-156
భాగం-11 :	పెలిసిపాడ్, బ్రాకియోపాడ్ జీవులు	157-186
భాగం-12 :	ఏకినోడెర్మాటా, ట్రైలోబైట్లు, గ్రాఫ్టోలైట్లు	187-203
	మాదిరి ప్రశ్నా పత్రం	204-206

BS 616 GEO DSE (C) - T

బి.ఎస్.సి

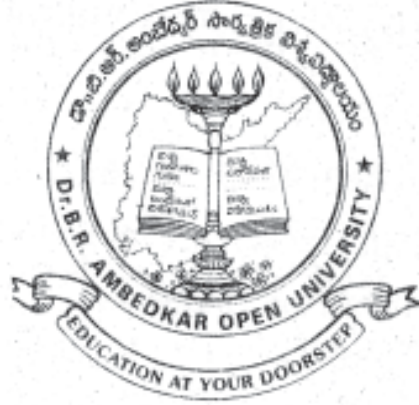
మూడవ సంవత్సరం

ఆరవ సెమిస్టర్

భూవిజ్ఞాన శాస్త్రం

డిసిప్లిన్ స్పెసిఫిక్ ఎలక్టివ్ కోర్సు - సి

భూగర్భ జల శాస్త్రం



“మనం నాగరికత సమకూర్చిన పస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు.

ఎందుకంటే విద్యను మించిన పస్తుగత ప్రయోజనమేది లేదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2020

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజి.నెం.
ఖండం-1	భూగర్భ జల శాస్త్ర భావనలు	1
భాగం-1 :	జలసంబంధ వలయము	3-11
భాగం-2 :	భూగర్భ జలాల ఉనికి, వితరణ	12-18
భాగం-3 :	జలస్తర ధర్మాలు - డార్చీ నియమం	19-27
ఖండం-2	భూగర్భజల అన్వేషణ	29
భాగం-4 :	భూగర్భజలాల భూగర్భ శాస్త్రీయ అన్వేషణ	30-43
భాగం-5 :	భూగర్భ జలాల భూభౌతిక అన్వేషణ	44-64
భాగం-6 :	భూగర్భ జలాల అధ్యయనానికి రిమోట్ సెన్సింగ్, GIS పద్ధతులు	65-71
ఖండం-3	భూగర్భజల కాలుష్యం	73
భాగం-7 :	భూగర్భ జలాల గుణం	75-90
భాగం-8 :	నీటి కాలుష్యం	91-103
భాగం-9 :	సముద్ర జల అంతర్గమనం - నియంత్రణ పద్ధతులు	104-114
ఖండం-4	భూగర్భజలాల యాజమాన్యం	115
భాగం-10 :	భూగర్భ జలాల యాజమాన్యం	117-127
భాగం-11 :	జలవిభాజక క్షేత్రం (వాటర్ షెడ్) యాజమాన్యం	128-140
భాగం-12 :	భూగర్భ జలాల నమూనాకరణం	141-160
	మాదిరి ప్రశ్నా పత్రం	161-163

BS 616 GEO DSE (D) -E

B.Sc

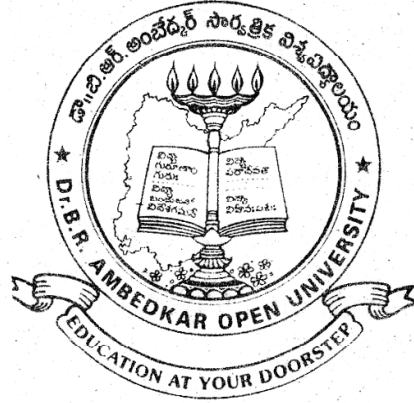
THIRD YEAR

SEMESTER-VI

GEOLOGY

DISCIPLINE SPECIFIC ELECTIVE - D

ENVIRONMENTAL GEOLOGY



“ We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit..

-Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2021

CONTENTS

BLOCK / Unit	Title	Page
BLOCK-I ENVIRONMENTAL GEOLOGY- CONCEPTS		1
Unit-1	: Principles of Environmental geology	3-8
Unit-2	: Earth and its Spheres, Earth's Materials	9-16
Unit-3	: Thermal Environments of Earth's Surface.	17-24
BLOCK-II GEOLOGICAL HAZARDS		25
Unit-4	: Volcanoes, Earthquakes and Tsunamis	27-64
Unit-5	: Land slides and Subsidence	65-81
Unit-6	: Floods, cyclones and Drought	82-95
BLOCK-III RESOURCES AND ENVIRONMENTAL ISSUES		97
Unit-7	: Energy Resources - Environmental Issues	99-113
Unit-8	: Mineral Resources - Environmental Issues	114-124
Unit-9	: Water Resources - Environmental Issues..	125-140
BLOCK- IV GLOBAL ENVIRONMENTAL ISSUES		141
Unit-10	: Global Warming - Climate Change	143-160
Unit-11	: Ozone Layer Depletion	161-172
Unit-12	: Acid Rain	173-181
	Model Question Paper	182-183

BS117 MAT-T

బి.ఎస్సీ.

మొదటి సంవత్సరం సెమెస్టర్ - 1

గణితశాస్త్రం

అవకలన గణితం



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr.B.R.Ambedkar

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాదు

2017

విషయసూచిక

ఖండం/భాగం	శీర్షిక	పుట
ఖండం-I:	అవధులు, అవిచ్ఛిన్నత	1
భాగం - 1:	అవధులు, అవిచ్ఛిన్నత	2
భాగం - 2:	ప్రమేయపు అవకలనీయత	28
భాగం - 3:	అవకలన పద్ధతులు	52
ఖండం-II:	పారంపర్య, పాక్షిక అవకలనం	72
భాగం - 4:	పారంపర్య అవకలనం, లైబ్రిటీ సిద్ధాంతం	73
భాగం - 5:	పాక్షిక అవకలనం, సమఘాత ప్రమేయాలకు ఆయిలర్ సిద్ధాంతం	95
భాగం - 6:	మధ్యమ విలువ సిద్ధాంతాలు	129
ఖండం-III:	టేలర్ బహుపది, అనిశ్చితరూప అవధుల పరిగణన, ఉజ్జాయింపు	156
భాగం - 7:	టేలర్ సిద్ధాంతం, లెగ్రాంజ్ మరియు కోషీ శిష్టరూపాలు	157
భాగం - 8:	అనిశ్చితా రూపాలు, లోపితా సూత్రం	175
భాగం - 9:	అవకలనీయతా అనువర్తనాలు - 1: దోషాలు, ఉజ్జాయింపు, స్పర్శరేఖ, అభిలంబరేఖ	192
ఖండం-IV:	అంత్యవిలువలు, వక్రత, అనంత స్పర్శరేఖలు మరియు వక్రనిర్మితి	214
భాగం - 10:	అవకలనీయతా అనువర్తనాలు - 2: గరిష్ఠం, కనిష్ఠం, వక్రత	215
భాగం - 11:	అసామాన్య బిందువులు మరియు అనంత స్పర్శరేఖలు	243
భాగం - 12:	వక్ర నిర్మితి	258
మాదిరి ప్రశ్నాప్రత్నం		290

బి.ఎన్.సి.

మొదటి సంవత్సరం రెండవ సెమిస్టర్

గణితశాస్త్రం

అవకలన సమీకరణాలు



“మనం నాగరికత నమకూర్చిన వస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ,
సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను,
హక్కును మాత్రం కోల్పోకూడదు”.

- డా॥ బి.ఆర్. అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2017

విషయసూచిక

ఖండం/భాగం	పాఠ్యాంశం	పుట
ఖండం-I:	సమాకలనం	1
భాగం - 1:	సమాకలన పద్ధతులు - I	2
భాగం - 2:	సమాకలన పద్ధతులు - II	42
భాగం - 3:	సమాకలన అనువర్తనాలు - వైశాల్యములు	61
ఖండం-II:	మొదటి పరిమాణ అవకలన సమీకరణాలు	78
భాగం - 4:	పరిచయం, అవకలన సమీకరణాల రచన	79
భాగం - 5:	మొదటి పరిమాణ మొదటి తరగతి అవకలన సమీకరణాలు (విభజనీయ చలరాశులు ; సమఘాతీయ మరియు అసమఘాతీయ సమీకరణాలు; ఏకఘాత మరియు బెర్నోలీ సమీకరణాలు)	87
భాగం - 6:	యదార్థ అవకలన సమీకరణాలు	109
ఖండం-III:	ఏకపరిమాణ, హెచ్చుతరగతి అవకలన సమీకరణాలు	128
భాగం - 7:	ఏకపరిమాణ, ఒకటికి మించిన తరగతి గల అవకలన సమీకరణాలు - లంబ సంబంధములు	129
భాగం - 8:	సమకాలిక అవకలన సమీకరణాలు మరియు పూర్ణ అవకలన సమీకరణాలు	161
భాగం - 9:	స్థిర గుణకాలతో ద్వితీయ మరియు హెచ్చు పరిమాణం గల సమఘాతీయ ఏకఘాత అవకలన సమీకరణాలు	188
ఖండం-IV:	ద్వితీయ మరియు హెచ్చు పరిమాణ అసమఘాతీయ సమీకరణాలు - పాక్షిక అవకలన సమీకరణాలు	207
భాగం - 10:	స్థిరగుణకాలతో ద్వితీయ మరియు హెచ్చు పరిమాణం గల అసమఘాతీయ ఏకఘాత అవకలన సమీకరణాలు	208
భాగం - 11:	చలరాశి గుణకాలతో ఏకఘాత అవకలన సమీకరణాలు	229
భాగం - 12:	పాక్షిక అవకలన సమీకరణాలు - వర్గీకరణ - లెగ్రాంజ్ పద్ధతి ద్వారా సాధన	226
మాదిరి ప్రశ్నాపత్రం		273

BS317 MAT-T

బి.ఎస్సీ.

రెండవ సంవత్సరం సెమెస్టర్ - 3

గణితశాస్త్రం

గణిత విశ్లేషణం



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

డా॥ బి.ఆర్. అంబేద్కర్ సార్యత్రిక విశ్వవిద్యాలయం

హైదరాబాదు

2018

విషయసూచిక

ఖండం/భాగం	శీర్షిక	పుట
ఖండం-I: వాస్తవ సంఖ్యలు, అనుక్రమాలు		1
భాగం - 1: వాస్తవ సంఖ్యా వ్యవస్థ - దాని పూరణ		2-21
భాగం - 2: R లో వివృత, సంవృత సమితులు		22-34
భాగం - 3: వాస్తవ అనుక్రమాలు, అభిసరణ		35-52
ఖండం-II: అనంతశ్రేణులు		53
భాగం - 4: ధనపదాల అనంతశ్రేణులు		54-63
భాగం - 5: అభిసరణ పరీక్షలు		64-77
భాగం - 6: ఏకాంతర శ్రేణి		78-88
ఖండం-III: రీమాన్ సమాకలనం		89
భాగం - 7: రీమాన్ సమాకలనీయత		90-104
భాగం - 8: సమాకలన మూలసిద్ధాంతం		105-111
భాగం - 9: సమాకలన గణిత మధ్యమ విలువ సిద్ధాంతాలు		112-119
ఖండం-IV: ప్రమేయాల అనుక్రమాలు, శ్రేణులు		120
భాగం - 10: బిందుపర, ఏకరూప అభిసరణలు		121-133
భాగం - 11: ప్రమేయాల అనంతశ్రేణులు		134-143
భాగం - 12: ఘాతశ్రేణి		144-154
మాదిరి పరీక్షా ప్రశ్నాప్రత్యం		155-158

BS417 MAT-T

బి.ఎస్.సి.

రెండవ సంవత్సరం నాలుగవ సెమిస్టర్

గణితశాస్త్రం

బీజగణితం



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలవైనా వదులుకోవచ్చునేమో గాని, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కులను మాత్రం కోల్పోకూడదు.....”

- డా॥ బి.ఆర్. అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాదు

2019

విషయసూచిక

ఖండం / భాగం	పాఠ్యాంశం	పుటసంఖ్య
ఖండం - I	సమూహాలు - I	1
భాగం - 1	సమితులు, సంబంధాలు, ప్రమేయాలు	3 - 24
భాగం - 2	సమూహాలు	25 - 45
భాగం - 3	ఉపసమూహాలు, చక్రీయ సమూహాలు	46 - 63
ఖండం - II	సమూహాలు - II	65
భాగం - 4	సహసమితులు, పరిమిత సమూహాలకు లెగ్రాంజ్ సిద్ధాంతం	67 - 82
భాగం - 5	అభిలంబ ఉపసమూహాలు, వ్యుత్పన్న సమూహాలు	83 - 103
భాగం - 6	సమరూపతలు, ప్రస్తార సమూహాలు	104 - 136
ఖండం - III	వలయాలు - I	137
భాగం - 7	వలయాలు, ఉపవలయాలు	139 - 166
భాగం - 8	పూర్ణాంక ప్రదేశాలు, క్షేత్రాలు	167 - 184
భాగం - 9	ఆదర్శాలు, వ్యుత్పన్న వలయాలు	185 - 205
ఖండం - IV	వలయాలు - II	207
భాగం - 10	వలయ సమరూపత	209 - 225
భాగం - 11	యూక్లిడియన్ ప్రదేశాలు, బహుపది వలయాలు	226 - 244
భాగం - 12	ఏకైక కారణాంక ప్రదేశాలు	245 - 257
	మాదిరి ప్రశ్నా పత్రం	258 - 261

UG 405 SEE (MAT 1) - E

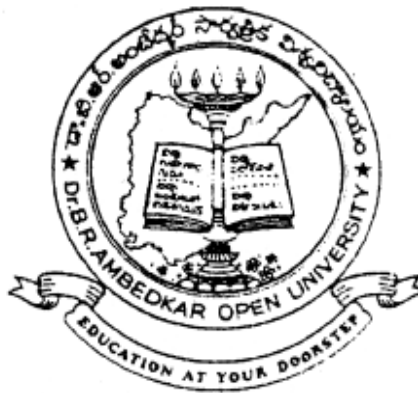
B.Sc.

SECOND YEAR SEMESTER - IV

MATHEMATICS

SKILL ENHANCEMENT ELECTIVE COURSE - SEE - 1

PROBABILITY & STATISTICS



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

BLOCK/UNIT	TITLE	PAGE
Block-I: Measures of Central Tendency		1
Unit -1: Measures of Central Tendency - I		2-22
Unit-2: Measures of Central Tendency-II		23-42
Unit - 3: Measure of Dispersion - I		43-59
Unit-4: Measures of Dispersion-II		60-74
Block-II: Probability		75
Unit - 5: Probability		76-91
Unit-6: Some Theorems on Probability		92-104
Unit-7: Independent and Dependent Events		105-116
Unit-8: Bayes' Theorem		117-126
Model Question Paper		127-129

UG 405 SEE (MAT 2) - E

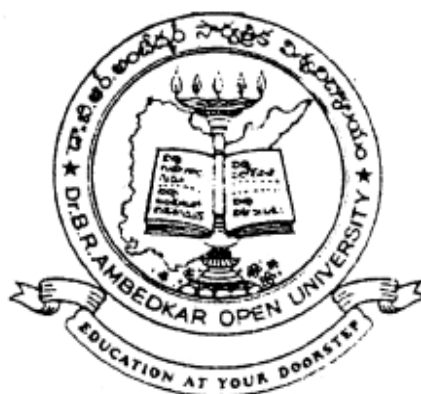
B.Sc.

SECOND YEAR SEMESTER - IV

MATHEMATICS

SKILL ENHANCEMENT ELECTIVE COURSE-SEE-2

GRAPH THEORY



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr.B.R.Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

BLOCK/UNIT	TITLE	PAGE
Block-I: Connectedness and Traversability in Graphs		1
Unit -1: Graphs - Terminology and Basic Concepts		2-23
Unit -2: Connectedness - Complete, Regular and Bipartite Graphs		24-41
Unit -3: Vertex and Edge Connectivity in Graphs		42-54
Unit - 4: Traversability: Eulerian and Hamiltonian Graphs		55-67
Block-II: Planarity, Matrix Representations and Shortest Paths in Graphs		68
Unit - 5: Matrix Representation and Isomorphism in Graphs		69-90
Unit - 6: Directed Graphs		91-112
Unit - 7: Planar Graphs and Graph Colouring		113-128
Unit - 8: Shortest Path and Shortest Circuit Problems		129-145
	Model Examination Question Paper	146-148

BS 517 MAT – T

బి.ఎన్.సి.

మూడవ సంవత్సరం సెమిస్టర్ - 5

గణితశాస్త్రం

రుజు బీజ గణితం



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలను వదులుకోవొచ్చునేమోగాని, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కులను మాత్రం కోల్పోకూడదు”.

- డా॥ బి.ఆర్. అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాదు

2020

విషయసూచిక

క్రమ సంఖ్య	ఖండం/భాగం	పుట సంఖ్య
ఖండం - I	: సదిశాంతరాళాలు	1-34
భాగం - 1	: సదిశాంతరాళాలు - ఉపాంతరాళాలు	2-13
భాగం - 2	: ఆధారము మరియు పరిమాణము	14-26
భాగం - 3	: వ్యుత్పన్న అంతరాళాలు	27-34
ఖండం - II	: ఋజు పరివర్తన, దాని మాత్రిక	35-79
భాగం - 4	: ఋజు పరివర్తనలు	36-49
భాగం - 5	: కోటి మరియు శూన్యత	50-62
భాగం - 6	: ఋజు పరివర్తన యొక్క మాత్రిక	63-80
ఖండం - III	: లాక్షణిక విలువలు మరియు లాక్షణిక సదిశలు	81-117
భాగం - 7	: ప్రాథమిక పరివర్తనలు మరియు సాధారణ రూపమునకు లఘూకరించుట (కుదించుట)	82-96
భాగం - 8	: ఏకఘాత సమీకరణ వ్యవస్థ	97-109
భాగం - 9	: లాక్షణిక మూలాలు, సదిశలు - కేలే హామిల్టన్ సిద్ధాంతము	110-117
ఖండం - IV	: ద్విఘాత రూపాలు మరియు అంతరలబ్ధ అంతరాళాలు	118-153
భాగం - 10	: ద్విఘాత రూపాలు	119-130
భాగం - 11	: అంతర్లబ్ధ అంతరాళాలు	131-140
భాగం - 12	: లంబాత్మకత మరియు గ్రాం-స్మిత్ లంబీకరణ పద్ధతి మాదరి పరీక్షా ప్రశ్నా పత్రం	141-153 154-157

BS517MATDSE(A)-E

B. Sc.

THIRD YEAR SEMESTER - V

MATHEMATICS

DISCIPLINE SPECIFIC ELECTIVE COURSE - A

THREE DIMENSIONAL GEOMETRY



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit.”

Dr. B.R. Ambedkar

Dr.B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2020

CONTENTS

BLOCK / UNIT	TITLE	PAGE No.
BLOCK - I : Cartesian Co-ordinates and Plane		1
Unit - 1	: Cartesian Co-ordinates, Direction Cosines and Direction Ratios	3 - 40
Unit - 2	: The Plane and Various Forms of the Equation of a Plane	41 - 66
Unit - 3	: Bisectors of the Angles Between Two Planes and Two Sides of a Plane	67 - 96
BLOCK – II : The Straight Line		97
Unit - 4	: Straight Line and Various Forms of the Equation of a Straight Line	99 - 127
Unit - 5	: Skew Lines and Shortest Line Segment Between Two Skew Lines	128 - 153
Unit - 6	: Change of Axes	154 - 176
BLOCK – III : The Sphere		177
Unit - 7	: Sphere, Circle, Intersection of Sphere and a Line	179 - 194
Unit - 8	: Tangent Plane, Normal Plane and Polar Planes of a Sphere	195 - 215
Unit - 9	: Radical Plane and Coaxial System of Spheres	216 - 227
BLOCK – IV : Cone and Cylinder		229
Unit - 10	: Cone, Intersection of a Cone with a Plane and a Line	231 - 249
Unit - 11	: Enveloping Cone, Reciprocal Cone and Right Circular Cone	250 - 267
Unit - 12	: The Cyclinder and the Right Circular Cylinder	268 - 280
	Model Question Paper	281 - 284

BS 517 MAT DSE (B)-E

B.Sc.

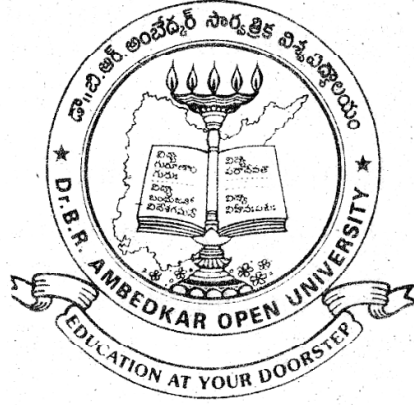
THIRD YEAR

SEMESTER – 5

MATHEMATICS

DISCIPLINE SPECIFIC ELECTIVE COURSE - B

LINEAR PROGRAMMING PROBLEMS



“We may forgo material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2021

CONTENTS

BLOCK / Unit	Title	Page
BLOCK-I LPP, Methods of Solving		1-56
Unit-1	: Formation of Linear Programming Problem	2-17
Unit-2	: Graphical Method of Solving LPP	18-33
Unit-3	: Simplex Method of Solving LPP, Basic Simplex Method	34-56
BLOCK-II Degeneracy, Big - M Method		57-89
Unit-4	: Degeneracy in LPP	57-66
Unit-5	: Big - M Method	67-79
Unit-6	: Two Phase Simplex Method	80-89
BLOCK-III Primal, Dual LPP		90-115
Unit-7	: Primal and Dual LPPs	91 - 98
Unit-8	: Relationship Between Primal and Dual LPP's	99 -106
Unit-9	: Solution by Dual Simplex Method	107 - 115
BLOCK-IV Transportation, Assignment Problems		116-160
Unit-10	: Transportation and Assignment Problems	117 - 128
Unit-11	: Solution of the Transportation problem, IBFS	129 - 141
Unit-12	: Test of optimality, Degeneracy, Resolution	142 - 155
	Model Question Paper	156 - 160

:

BS617MAT - T

బి.ఎస్.సి.

ఘాడవ సంవత్సరం ఆరవ సెమిస్టర్

గణితశాస్త్రం

సంఖ్యాత్మక పద్ధతులు



“ఘనం నాగరికత సఢకూర్చిన వస్తుగత ప్రయోజనాలవైనా వదులుకోవచ్చునేఢో గాని, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కులను మాత్రం కోల్పోకూడదు. ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనఢేదీ లేదు.”

- డా॥ బి.ఆర్. అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం
ఘైదరాబాదు

2020

విషయసూచిక

ఖండం / భాగం	పాఠ్యాంశం	పుటసంఖ్య
ఖండం - I	: సమీకరణాల సాధనా పద్ధతులు	1
భాగం - 1	: బీజీయ, బీజాతీత సమీకరణాల సాధనలు	3 - 34
భాగం - 2	: ఏకకాలిక రుజు సమీకరణాలకు సంఖ్యాత్మక సాధన - ప్రత్యక్ష పద్ధతులు	35 - 51
భాగం - 3	: ఏకకాలిక రుజు సమీకరణాలకు సంఖ్యాత్మక సాధన - పునరుక్త పద్ధతులు	52 - 63
ఖండం - II	: అంతర్వేశనం - I	65
భాగం - 4	: పరిమిత భేద పరికర్తలు	67 - 96
భాగం - 5	: న్యూటన్ పురోగమన, తిరోగమన భేద అంతర్వేశనం	97 - 124
భాగం - 6	: కేంద్రీయ భేద అంతర్వేశనం	125 - 151
ఖండం - III	: అంతర్వేశనం - II	153
భాగం - 7	: లెగ్రాంజ్ అంతర్వేశనం, న్యూటన్ విభాజిత భేద అంతర్వేశనం	155 - 176
భాగం - 8	: విలోమ అంతర్వేశనం	177 - 197
భాగం - 9	: కనిష్ఠ వర్గాల ఉజ్జాయింపు	198 - 214
ఖండం - IV	: సంఖ్యాత్మక అవకలనం మరియు సమాకలనం	215
భాగం - 10	: సంఖ్యాత్మక అవకలనం	217 - 234
భాగం - 11	: సంఖ్యాత్మక సమాకలనం - I	235- 254
భాగం - 12	: సంఖ్యాత్మక సమాకలనం - II	255- 273
	మాదిరి ప్రశ్నా పత్రం	275-279

BS 617 MAT DSE(C)-T

బి.ఎస్.సి.

మూడవ సంవత్సరం సెమిస్టర్ - VI

గణితశాస్త్రం

డిసిప్లిన స్పెసిఫిక్ ఎలక్టివ్ కోర్సు-సి

విరళ గణిత నిర్మితులు



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైనా వదులుకోవొచ్చునేమోగాని, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కులను మాత్రం కోల్పోకూడదు”.

- డా॥ బి.ఆర్. అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాదు

2020

విషయసూచిక

క్రమ సంఖ్య	ఖండం/భాగం	పుట నం.
ఖండం - I	: తర్కము యొక్క మూల సిద్ధాంతాలు	1-60
భాగం - 1	: ప్రవచనాలు మరియు సంధానములు (సంయోజకాలు)	2-16
భాగం - 2	: సూత్రముల సమత్యులత మరియు సామాన్య రూపములు	17-42
భాగం - 3	: ప్రవచన కలనగణితంనకు అనుమితి సిద్ధాంతం	43-60
ఖండం - II	: ముఖ్య నియమాలు, జనన్ ప్రమేయాలు	61-117
భాగం - 4	: అంతర్భాగ మరియు వర్జన నియమం మరియు గణితానుగమన సూత్రము	62-81
భాగం - 5	: జనక ప్రమేయాలు	82-96
భాగం - 6	: ఆవృత్త సంబంధాలు	97-117
ఖండం - III	: గ్రాఫ్ల మరొక భావనలు, సంధానితత్వము	118-186
భాగం - 7	: గ్రాఫ్లు - కొన్ని మౌలిక భావనలు, రకములు	119-139
భాగం - 8	: ఉపగ్రాఫ్లు, మాత్రిక వర్జన, గ్రాఫ్ల తుల్య రూపత	140-164
భాగం - 9	: సంధానితత్వము, ఆయులీరియన్ మరియు హామిల్టోనియన్ గ్రాఫ్లు	164-186
ఖండం - IV	: సమతలీయ గ్రాఫ్లు, ప్రవాహాలు	187-244
భాగం - 10	: వృక్షములు, బైనెరి వృక్షములు, విస్తృత వృక్షములు	188-211
భాగం - 11	: సమతలీయ గ్రాఫ్లు, గ్రాఫ్ వర్గీకరణ	212-231
భాగం - 12	: జాలము నందు ప్రవాహములు	232-249
	మాదిరి పరీక్షా ప్రశ్నా పత్రం	250-253

BS 617 MAT DSE (D) - E

B.Sc.

THIRD YEAR SEMESTER - VI

MATHEMATICS

DISCIPLINE SPECIFIC ELECTIVE COURSE - D

VECTOR CALCULUS



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B.R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2021

CONTENTS

BLOCK/UNIT	TITLE	PAGE
Block - I : Vector Differential Calculus		1
Unit - 1: Basic Vector Algebra		2-54
Unit - 2: Differentiation of Vector Point Functions, Curves, Tangents, Arc Length		55-86
Unit - 3: Gradient, Divergence, Curl Operators and their Algebra		87-129
Block - II : Multiple Integrals		130
Unit - 4: Transformations, Polar, Spherical Polar, Cylindrical Polar Coordinates		131-151
Unit - 5: Double Integrals, Change of Order of Integration		152-172
Unit - 6: Triple Integrals, Applications of Multiple Integrals		173-194
Block - III : Vector Integration		195
Unit - 7: Line Integrals and Surface Integrals		196-227
Unit - 8: Volume Integrals and Applications of Vector Integration		228-246
Unit - 9: Curvilinear Coordinates		247-265
Block - IV : Integral Theorems		266
Unit - 10: Green's Theorem and its Applications		267-294
Unit - 11: Stoke's Theorem and its Applications		295-318
Unit - 12: Gauss's Divergence Theorem and its Applications		319-343
Model Question Paper		344-347

BS118PHY-E

B.Sc.

FIRST YEAR SEMESTER – I

PHYSICS

COURSE-1: MECHANICS



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2022

CONTENTS

BLOCK-I MECHANICS OF PARTICLES	1
UNIT-1: Introduction to Vectors	2 - 9
UNIT-2: Vector Calculus	10 - 27
UNIT-3: Linear Momentum and Collisions	28 - 39
UNIT-4: Kinematics	40 - 51
BLOCK-II MECHANICS OF RIGID BODIES	52
UNIT-5: Centre of Mass, Motion of Centre of Mass, Reduced Mass	53 - 65
UNIT-6: Torque and Rotational Motion	66- 845
UNIT-7: Conservation of Angular Momentum	86 - 103
BLOCK-III CENTRAL FORCES	104
UNIT-8: Introduction to Central Forces	105 - 118
UNIT-9: Motion of Planets and Satellites - Kepler's Laws.	119 - 129
UNIT-10: Gravitational Field and Gravitational Potential	130 - 142
BLOCK-IV RELATIVITY	143
UNIT-11: Special Theory of Relativity	144 - 151
UNIT-12: Applications of special theory of Relativity	152 - 168
Model paper	169 - 171

B.Sc218PHY-T

బి.ఎస్.సి.

మొదటి సంవత్సరం సెమిస్టర్-2

భౌతిక శాస్త్రము

కోర్సు-2 తరంగాలు మరియు డోలనాలు



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

- Dr. B. R. Ambedkar

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాదు 2018

విషయసూచిక

ఖండం - I : ప్రాథమిక కంపనాలు మరియు డోలనాలు

భాగం - 1	: సరళ హరాత్మక చలనం	09-23
భాగం - 2	: సరళ హరాత్మక చలనాల సమ్మేళనం	24-31
భాగం - 3	: అవరుద్ధ హరాత్మక డోలనాలు	32-48
భాగం - 4	: బలాత్కృత డోలనాలు	49-74

ఖండం - II : స్థితిస్థాపక యానకంలో తరంగాలు - I

భాగం - 5	: పురోగమన తరంగాలు - రకాలు	77-84
భాగం - 6	: డాప్లర్ ఫలితం	85-91

ఖండం - III : స్థితిస్థాపక యానకంలో తరంగాలు - II

భాగం - 7	: తీగలలో కంపనాలు	95-111
భాగం - 8	: కడ్డీలలో కంపనాలు	112-130
భాగం - 9	: గాలిలో అనుదైర్ఘ్య తరంగాలు	131-148

ఖండం - IV: సంక్లిష్ట కంపనాలు మరియు అతిధ్వనులు

భాగం - 10	: పురియో సిద్ధాంతము	155-168
భాగం - 11	: పురియో విశ్లేషణ	169-186
భాగం - 12	: అతిధ్వనులు	195-206

బి.ఎస్సీ.

రెండవ సంవత్సరం సెమెస్టర్ - 3

భౌతికశాస్త్రం

కోర్సు-3: ఉష్ణగతికశాస్త్రం



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

డా॥ బి.ఆర్. అంబేద్కర్ సార్యత్రిక విశ్వవిద్యాలయం

హైదరాబాదు

2018

విషయసూచిక

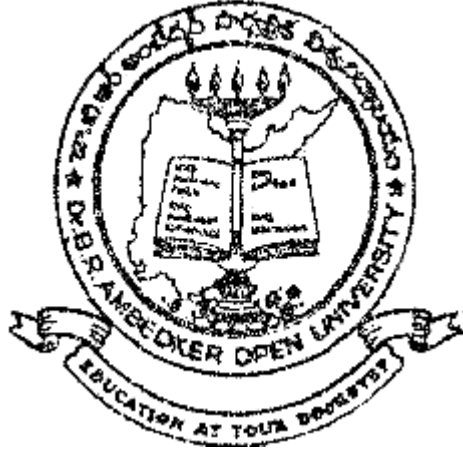
ఖండం/భాగం	శీర్షిక	పుట
ఖండం-I:	ఉష్ణగతిక నియమాలు - 1	1
భాగం-1:	వాయువుల అణుచలన సిద్ధాంతం	2-28
భాగం-2:	సున్న, మొదటి ఉష్ణగతికశాస్త్ర నియమాలు	29-38
భాగం-3:	ద్విగత, అద్విగత ప్రక్రియలు	39-44
భాగం-4:	కార్నో యంత్రం	45-57
ఖండం-II:	ఉష్ణగతిక నియమాలు - 2	58
భాగం-5:	ఉష్ణగతికశాస్త్ర రెండవ నియమం	59-67
భాగం-6:	ఎంట్రపీ	68-79
ఖండం-III:	ఉష్ణగతిక శక్త్యాలు	80
భాగం-7:	ఉష్ణగతికశాస్త్రపు శక్త్యాలు	81-87
భాగం-8:	మాక్స్ వెల్ ఉష్ణగతికశాస్త్ర సమీకరణాలు - అనువర్తనాలు	88-94
ఖండం-IV:	క్వాంటం వికరణ సిద్ధాంతం	95
భాగం-9:	అల్ప ఉష్ణోగ్రత భౌతికశాస్త్రం	96-114
భాగం-10:	అల్ప ఉష్ణోగ్రత భౌతికశాస్త్ర అనువర్తనములు	115-128
భాగం-11:	కృష్ణవస్తువు వికరణం (క్లాసికల్ పద్ధతి)	129-139
భాగం-12:	కృష్ణవస్తువు వికరణం (క్వాంటీకరణ పద్ధతి)	140-155
మాదిరి పరీక్షా ప్రశ్నాప్రతుం		156-159

BS 418 PHY-T

బి.ఎన్.సి.

రెండవ సంవత్సరం సెమిస్టర్-4

భౌతిక శాస్త్రము
కోర్సు-4 దృశాశాస్త్రం



We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent.....”

Dr. B. R. Ambedkar

డా॥ బి.ఆర్.అంబేద్కర్ సార్యత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2019

విషయ సూచిక

	పేజీలు
ఖండం-I: వ్యతికరణం	7
భాగం-1: హైగన్స్ సూత్రం మరియు యంగ్స్ ప్రయోగము	9
భాగం-2: వ్యతికరణము - అనువర్తనాలు	38
ఖండం-II: వివర్తనము	49
భాగం-3: ఫ్రెనెల్ ఫ్రాన్ హాఫర్ వివర్తనం	51
భాగం-4: తిన్నని అంచువద్ద ఏర్పడే ఫ్రెనెల్ వివర్తనం	60
భాగం-5: వివర్తన గ్రేటింగ్ - కాంతి తరంగదైర్ఘ్యం కనుక్కోవడం	66
భాగం-6: గ్రేటింగ్ విశ్లేషణ, పృథక్కరణ సామర్థ్యం	71
ఖండం-III: ద్రువణము	77
భాగం-7: సమతల ద్రువణం, పోలరాయిడ్, పరావర్తన ద్రువణం	79
భాగం-8: వివిధ రకాల ద్రువిత కాంతుల ఉత్పన్నం - విశ్లేషణం	93
భాగం-9: భ్రమణ ద్రువణం	99
ఖండం-IV: లేజర్లు మరియు హోలోగ్రఫీ	107
భాగం-10: లేజర్లు	109
భాగం-11: హోలోగ్రఫీ	123
భాగం-12: దృశాతంతువు	127

UG 405 SEE (PHY1)-T

బి. ఎస్.సి.

రెండవ సంవత్సరం

సెమిస్టర్ - IV

స్కీల్ ఎన్హాన్స్‌మెంట్ ఎలక్టివ్ కోర్స్ (SEE) - 1

భౌతిక శాస్త్రం

విద్యుత్ పరికరాల ప్రాథమిక నైపుణ్యాలు



” మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైనా వదులుకోవచ్చు గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు. ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేదీ లేదు”

- డా. బి. ఆర్. అంబేద్కర్

డా . బి. ఆర్. అంబేద్కర్ సార్యత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2019

విషయ సూచిక

ఖండం - I: విద్యుత పరికరాల ప్రాథమిక నైపుణ్యాలు - 1	1
భాగం - 1: విద్యుత్ దాని చరిత్ర	3
భాగం - 2: విద్యుత్ ఘాతానికి చికిత్స	12
భాగం - 3: ప్రవాహాలు మరియు నిరోధాలు	20
భాగం - 4: పూజలు మరియు M.C.B లలోని రకాలు	30
ఖండం - II : విద్యుత పరికరాల ప్రాథమిక నైపుణ్యాలు - 2	41
భాగం - 5: ఎర్లింగ్	43
భాగం - 6: హాస్ వైరింగ్	47
భాగం - 7: ప్రతిదీప్త దీపం దాని నిర్మాణం	62
భాగం - 8: ట్రాన్స్‌ఫార్మర్లు (విద్యుచ్ఛాలక బలము మార్చునట్టి యంత్రము)	69
గుర్తులు మరియు చిహ్నాలు	79
మాదిరి పరీక్షా పత్రం	82

UG 405 SEE (PHY 2)-E

B.Sc.

PHYSICS

SECOND YEAR SEMESTER - IV

SKILL ENHANCEMENT ELECTIVE COURSE - SEE-2

**MOBILE PHONE REPAIRING AND
MAINTENANCE**



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

- Dr. B. R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

	Page No.
Block-I: Basic Electronics	1
Unit 1: Introduction to Basic Electronics	3
Unit 2: Types of Mobile Phones and Potential Hazards	13
Unit 3: Mobile phones technology	19
Unit 4: Cellular concepts and Block Diagram of mobile phones	31
Block-II: Mobile phones repairing and maintenance	43
Unit 5: Parts of a Conventional Mobile Phone	45
Unit 6: Assembling and Disassembling a Mobile Cell Phone	60
Unit 7: Diagnosing and Repairing Mobile Phone	64
Unit 8: Faults & Causes of mobile phones	74

BS 518 PHY-E

B.Sc.

THIRD YEAR SEMESTER – V

**PHYSICS
ELECTROMAGNETISM**



“We may forego material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2020**

CONTENTS

Block/Unit No.	Title	Page No.
Block I: Electrostatics		1-66
Unit-1: Electric Field and Gauss Theorem		2-16
Unit-2: Electric Potential		17-31
Unit-3: Capacitance		32-41
Unit-4: Parallel Plate Condenser		42-66
Block II: Current Electricity		67-115
Unit - 5: Electrical Conductivity		68-82
Unit - 6: Kirchoff's Laws		83-94
Unit - 7: Network Theorems		95-115
Block III: Magnetostatics		116-144
Unit - 8: Ampere's Law		117-125
Unit - 9: Biot-Savart's Law		126-137
Unit - 10: Magnetic Field and Magnetic Force on a Circuit – Torque		138-144
Block IV: Electromagnetic Induction		145-166
Unit - 11: Self Induction and Mutual Inductances		146-151
Unit - 12: Faraday's Law and Lenz's Law		152-166
• Model Examination Question Paper		167-169

BS 518 PHY DSE (A) - E

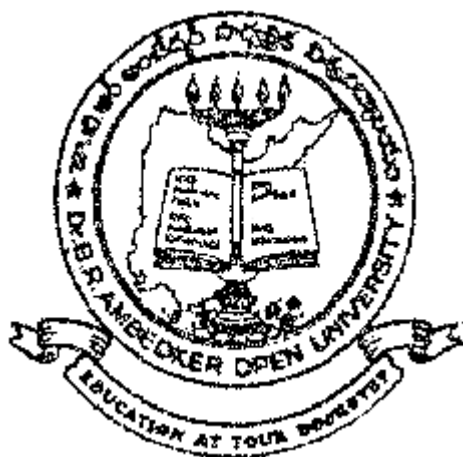
B.Sc.
PHYSICS

THIRD YEAR

SEMESTER-V

BASIC ELECTRONICS

Discipline Specific Elective (DSE) - A



We may forgo material benefits of Civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent.....”

Dr. B. R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2020

CONTENTS

	Pages
BLOCK-I SEMI CONDUCTORS.....	5
Unit-1: Semi Conductors - PN Junction diodes.....	7
Unit-2: Transistor Configurations and Characteristics	26
Unit-3: Special Semiconductor Devices.....	46
BLOCK-II TRANSISTOR AMPLIFIERS	63
Unit-4: Transistor Biasing and Load Line Analysis	65
Unit-5: An Introduction Amplifiers.....	73
Unit-6: Common Emitter Amplifier	85
Unit-7: Power Amplifier	93
BLOCK-III OSCILLATORS AND MULTIVIBRATORS	103
Unit-8: Oscillators: Barkhausen's Criterion And L-C Oscillators	105
Unit-9: Multivibrators	116
BLOCK-IV POWER SUPPLIES AND REGULATION.....	127
Unit-10: Rectifiers and Filters	129
Unit-11: Voltage Regulation	140
Unit-12: Functioning of Cathode Ray Oscilloscope	154

BS 518 PHY - (DSE-B)

B.Sc

THIRD YEAR

SEMESTER-V

DISCIPLINE SPECIFIC ELECTIVE COURSE-B

PHYSICS

MATERIAL SCIENCE



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B.R.Ambedkar

**DR. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2021

CONTENTS

	Pages
BLOCK-I Properties of Materials	1-53
Unit-1: Mechanical properties	1-24
Unit-2: Elastic behavior of materials.	25-32
Unit-3: Deformation of crystalline materials.	33-40
Unit-4: Corrosion	41-53
BLOCK-II Introduction to Thin films	57-88
Unit-5: Vacuum techniques	57-71
Unit-6: Thin film deposition methods	72-80
Unit-7: Thin film formation	81-88
BLOCK-III Thin Films and Their Properties	91-138
Unit-8: Electrical properties of thin metal film	91-112
Unit-9: Dielectric properties of thin film	113-126
Unit-10: Optical Properties of thin Film	127-138
BLOCK-IV Instrumentation.....	141-155
Unit-11: Techniques and Measurements-I.	141-149
Unit-12: Techniques and Measurements-II.	150-155

BS 618 PHY - TM

భౌతికశాస్త్రం

మాడవ సంవత్సరం

సెమిస్టర్-6

ఆధునిక భౌతికశాస్త్రం



We may forgo material benefits of Civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent.....”

Dr. B. R. Ambedkar

డా॥ బి.ఆర్.అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2020

విషయ సూచిక

పేజీలు

ఖండం-I:	స్పెక్ట్రోస్కోపీ	
భాగం-1:	బోర్ సిద్ధాంతం మరియు రేఖీయ వర్ణపటం	1
భాగం-2:	క్వాంటం సంఖ్యలు	21
ఖండం-II:	పరమాణాత్మక సిద్ధాంతం	
భాగం-3:	ఫోటో విద్యుత్ ఫలితం	35
భాగం-4:	ప్రాడింగర్ తరంగ సమీకరణం	67
భాగం-5:	ప్రాడింగర్ తరంగ సమీకరణానికి అనువర్తనాలు	74
ఖండం-III:	కేంద్రక నిర్మాణం	
భాగం-6:	రేడియో ధార్మికత	85
భాగం-7:	మూలకాల పరివర్తనం	106
భాగం-8:	వికిరణ శోధకాలు	124
భాగం-9:	కృత్రిమ రేడియో ధార్మికత	141
ఖండం-IV:	కేంద్రక ధర్మాలు	
భాగం-10:	కేంద్రక నమూనాలు	157
భాగం-11:	కేంద్రక విచ్ఛిత్తి సంతానం	171
భాగం-12:	ప్రాథమిక కణాలు	189

BS 618 PHY DSE (C) - E

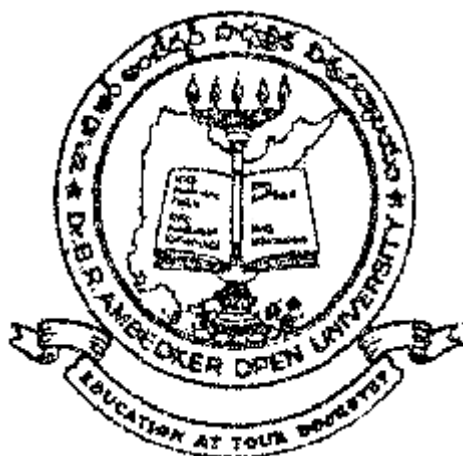
B.Sc.
PHYSICS

THIRD YEAR

SEMESTER-VI

**DIGITAL ELECTRONICS AND
COMMUNICATIONS**

Discipline Specific Elective (DSE-C)



We may forgo material benefits of Civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent.....”

Dr. B. R. Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2020

CONTENTS

	Pages
BLOCK-I Operational Amplifiers	
Unit-1: Difference Amplifier	7
Unit-2: Operational Amplifier and its Characteristic Parameters	13
Unit-3: Operational Amplifier - Configurations and Analysis.....	21
BLOCK-II Measuring Instruments	
Unit-4: Analog measuring Instruments	33
Unit-5: Digital measuring Instruments	46
BLOCK-III Digital Electronics	
Unit-6: Number system and Logic gates	59
Unit-7: Combinational & Sequential Logic Circuits.....	73
Unit-8: Fundamentals of Microprocessors	85
BLOCK-IV Modulation and Demodulation	
Unit-9: Amplitude Modulation and Demodulation	101
Unit-10: Frequency Modulation and Demodulation	110
Unit-11: Elements of Superheterodyne Receiver.....	117
Unit-12: Television: Transmission and Reception	123

BS618 PHY DSE(D)–EM

B.Sc.

THIRD YEAR SEMESTER – VI

PHYSICS

Discipline Specific Elective Course-D

PHYSICS OF NANOMATERIALS



“We may forgo material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2021

CONTENTS

Block/Unit No.	Title	Page No.
Block - I:	Fundamentals of Nanomaterials	1-35
Unit-1:	An introduction to Nanomaterials	2-9
Unit-2:	Synthesis of Nanomaterials	10-18
Unit-3:	Characteristic Techniques	19-35
Block - II:	Properties of Nanomaterials-I	36-132
Unit-4:	Electronic Properties	37-71
Unit-5:	Dielectric Properties	72-101
Unit-6:	Magnetic Properties	102-132
Block - III:	Properties of Nanomaterials-II	133-155
Unit-7:	Optical Properties	134-139
Unit-8:	Thermal Properties	140-145
Unit-9:	Mechanical Properties	146-155
Block - IV:	Nano Devices and Sensors	156-197
Unit-10:	Quantum Devices	157-166
Unit-11:	Super Conducting Devices	167-178
Unit-12:	Introduction of Nano Sensors	179-197
•	Model Question Paper	198-200

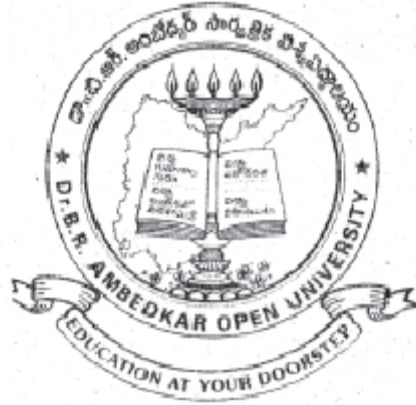
బి.ఎస్సి

మొదటి సంవత్సరం

మొదటి సెమిస్టర్

జంతు శాస్త్రం

జంతు వైవిధ్యం - అకశేరుకాలు



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ,
సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను,
హక్కును మాత్రం కోల్పోకూడదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2017

విషయ సూచిక

కోర్సు-1 : జంతువైవిధ్యం - అకశేరుకాలు

ఖండము / భాగం	పాఠ్యాంశం	పేజి.నెం.
ఖండం-1 ప్రోటోజోవా, ఫోరి ఫెరా, నిడేరియా		
యూనిట్-1	జంతు సామ్రాజ్యము - ఉపోద్ఘాతము	1-9
యూనిట్-2	ప్రోటోజోవా - సామాన్య లక్షణములు, వర్గీకరణము, నమూనా అధ్యయనము - ఎల్ఫీడియం, ప్రోటోజోవా వ్యాధులు	10-41
యూనిట్-3	ఫోరి ఫెరా - సామాన్య లక్షణములు, వర్గీకరణము నమూనా అధ్యయనము - సైకాస్, ఫోరి ఫెరా కుల్యా వ్యవస్థ	42-61
యూనిట్-4	నిడేరియా - సామాన్య లక్షణములు, వర్గీకరణము, నమూనా అధ్యయనము - అరీలియా, నిడేరియాలో బహురూపకత.	62-89
ఖండం-2 ప్లాటి హెల్మింథిస్, ఆస్కె హెల్మింథిస్		90
యూనిట్-5	ప్లాటి హెల్మింథిస్ - సామాన్య లక్షణములు, వర్గీకరణము, ముఖ్యమైన ట్రిమటోడా మరియు సిస్టోడా పరాన్నజీవులు - నమూనా అధ్యయనం.	91-121
యూనిట్-6	ఆస్కె హెల్మింథిస్ - సామాన్య లక్షణములు, వర్గీకరణము, నమూనా అధ్యయనము ఆస్కారిస్ లుంబ్రికాయిడిస్, హెల్మింథ్ల పరాన్నజీవుల అనుకూలనములు	122-143
ఖండం-3 అనెలిడా, ఆర్ట్రోపోడా		144
యూనిట్-7	అనెలిడా - సామాన్య లక్షణములు, వర్గీకరణము నమూనా అధ్యయనము - హైరుడినేరియా గ్రాస్యలోసా, సమఖండ విన్యాసము, వృక్కములు, శరీరకుహర వాహికలు	144-196
యూనిట్-8	ఆర్ట్రోపోడా - సామాన్య లక్షణములు, వర్గీకరణము, ఒనైకోఫోరా - (పెరిపేటస్) సంబంధ బాంధవ్యాలు కీటకాలు - ఆర్థిక ప్రాముఖ్యత	197-222
యూనిట్-9	నమూనా అధ్యయనము - పేలిమాన్	223-262
ఖండం-4 మొలస్కా ఇక్టెనోడెర్మేటా, హెమికార్డేటా		263
యూనిట్-10	మొలస్కా-సామాన్య లక్షణములు, వర్గీకరణము నమూనా అధ్యయనము - ఫైలాగ్లోబోసా	264-290
యూనిట్-11	ఇక్టెనోడెర్మేటా-సామాన్య లక్షణములు, వర్గీకరణము, నమూనా అధ్యయనము-వీప్టీరియాస్	291-320
యూనిట్-12	హెమికార్డేటా - సామాన్య లక్షణములు, వర్గీకరణము, నమూనా అధ్యయనము - బెలనోగ్లాసస్ - సంబంధ బాంధవ్యములు.	321-335

BS219 ZOO-T

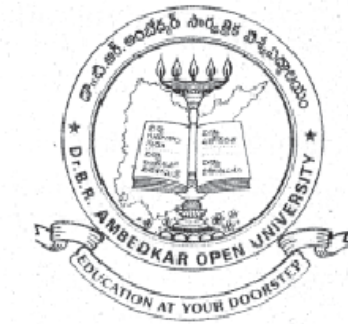
బి.ఎస్సి

మొదటి సంవత్సరం

రెండవ సెమిస్టర్

జంతు శాస్త్రం

జంతు వైవిధ్యం - సకసేరుకాలు



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ,
సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను,
హక్కును మాత్రం కోల్పోకూడదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2018

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజి.నెం.
ఖండం-1	సకసేరుకాలు, ప్రాటోకార్డేటా మరియు సైక్లోస్టామేట	1
భాగం-1 :	సకసేరుకాలు సాధారణ లక్షణాలు, ప్రాటోకార్డేటా, హెర్డ్మేనియ- నమూనా అధ్యయనం	3-38
భాగం-2 :	సెఫాలోకార్డేటా - నమూనా అధ్యయనం - ఆంఫియాక్స్	39-58
భాగం-3 :	వర్టిబ్రేట సాధారణ లక్షణాలు, వర్గీకరణ మరియు సైక్లోస్టామేట	59-80
ఖండం-2	చేపలు మరియు ఉభయచరాలు	81
భాగం-4 :	చేపలు-సాధారణ లక్షణాలు, వర్గీకరణ; మరియు డిప్నాయ్ చేపలు	83-99
భాగం-5 :	సారచేప - నమూనా అధ్యయనం	100-126
భాగం-6 :	ఉభయచరాలు (ఆంఫిబియా) - సాధారణ లక్షణాలు, వర్గీకరణ, సంతానపాలన	127-139
ఖండం-3	సరీసృపాలు మరియు పక్షులు	1411
భాగం-7 :	సరీసృపాలు-సాధారణ లక్షణాలు, వర్గీకరణ; డైనోసార్స్; విషసర్పాలు మరియు విషరహిత సర్పాలు	143-162
భాగం-8 :	కెలోటిస్ (తొండ) - నమూనా అధ్యయనం	163-180
భాగం-9 :	పక్షులు-సాధారణ లక్షణాలు, వర్గీకరణ, వైహాయన అనుకూలనాలు, పక్షుల వలస	181-197
ఖండం-4	పక్షులు మరియు క్షీరదాలు	199
భాగం-10 :	పావురం - నమూనా అధ్యయనం	201-220
భాగం-11 :	క్షీరదాలు-సాధారణ లక్షణాలు, వర్గీకరణ, జలావాస అనుకూలనాలు	221-233
భాగం-12 :	కుందేలు - నమూనా అధ్యయనం	234-287

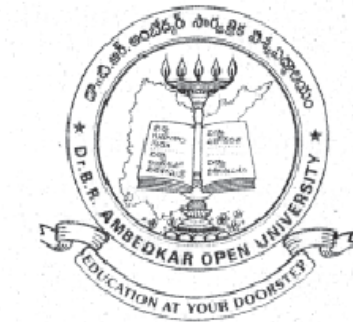
బి.ఎస్సి

రెండవ సంవత్సరం

మూడవ సెమిస్టర్

జంతు శాస్త్రం

జీవావరణ శాస్త్రం, జంతు భౌగోళిక శాస్త్రం మరియు జీవాభివృద్ధి శాస్త్రం



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు.

ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేదీ లేదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2018

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజీ.నెం.
ఖండం-I	జీవావరణ శాస్త్రం-1	1
భాగం -1	: ఆవరణ వ్యవస్థ నిర్మాణం మరియు విధులు, ఆవరణ వ్యవస్థ రకాలు - జల మరియు భౌమ్య ఆవరణలు	3-25
భాగం-2	: జీవభౌమ్య రసాయన వలయాలు	26-38
భాగం-3	: సమాజ నిర్మాణం - అనుక్రమం, జనాభా ఆవరణ శాస్త్రం, శక్తి ప్రవాహం	39-68
ఖండం-II	: జీవావరణ శాస్త్రం -II	69
భాగం-4	: జంతువుల సంబంధాలు	71-83
భాగం-5	: ఆవరణ శాస్త్ర అనుకూలనాలు	84-97
భాగం-6	: వన్యప్రాణుల సంరక్షణ, భారతదేశంలో జాతీయపార్కులు మరియు అభయారణ్యాలు	98-156
ఖండం-2	: జంతు బౌగోళిక శాస్త్రం - III	157
భాగం-7	: జంతు బౌగోళిక ప్రాంతాలు	159-173
భాగం-8	: వాలెస్ రేఖ మరియు విచ్ఛిన్న విస్తరణ	174-183
భాగం-9	: ఖండాల కదలిక	184-193
ఖండం - IV	: జీవాభివృద్ధి శాస్త్రం	195
భాగం-10	: అండముల రకాలు, ఫలదీకరణం, గాస్ట్రులేషన్	197-212
భాగం-11	: అవయవోత్పత్తి	213-221
భాగం-12	: కోడి - పిండత్యచాలు, క్షీరదాలలో జరాయువు స్థితి	222-236

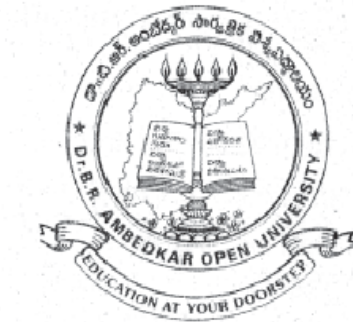
బి.ఎస్సి

రెండవ సంవత్సరం

నాలుగవ సెమిస్టర్

జంతు శాస్త్రం

కణజీవ శాస్త్రం, అణుజీవ శాస్త్రం, జన్యుశాస్త్రం మరియు జీవపరిణామం



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు.

ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేదీ లేదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2019

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజి.నెం.
ఖండం-I	కణజీవశాస్త్రం	1
భాగం -1	కణజీవశాస్త్ర ఉపోద్ఘాతం మరియు జంతుకణ సూక్ష్మ నిర్మాణం, కణాంగాల నిర్మాణం మరియు విధులు - ప్లాస్మాత్వచం, అంతర్జీవ ద్రవ్యజాలకం, గాల్జి సంక్లిష్టం మరియు లైసోసోములు	3-35
భాగం-2	కణాంగాల నిర్మాణం మరియు విధులు - మైట్రోకాండ్రీయం, తారావత్కేంద్రాలు, లైసోసోములు కేంద్రకం, మరియు క్రోమోసోములు	36-62
భాగం-3	కణవిభజన, సంయోగ బీజ జననం, అనిషేక జననం	63-84
ఖండం-II	అణుజీవ శాస్త్రం	85
భాగం-4	కేంద్రకామ్లాలు - DNA నిర్మాణం, RNA నిర్మాణం, DNA ప్రతికృతి	87-108
భాగం-5	మాంసకృత్తుల సంశ్లేషణ - అనులేఖనం, అనునాదం	109-115
భాగం-6	జన్యుక్రమత - జన్యుసంకేతం, ఒపెరాన్ భావన	116-130
ఖండం-III	జన్యుశాస్త్రం	131
భాగం-7	మెండల్ అనువంశిక సూత్రాలు మరియు మెండల్ కాని అనువంశిక సూత్రాలు, సహలగ్నత & వినిమయం, లింగనిర్ధారణ మరియు లింగ సహలగ్నత అనువంశికత	133-176
భాగం-8	ఉత్పరివర్తనాలు : క్రోమోసోముల ఉత్పరివర్తనాలు - తొలగింపులు, స్థానభ్రంశం, విలోమాలు, ప్రతిస్థాపన, ఎన్యుఫ్లాయిడి మరియు పాలిఫ్లాయిడి.	177-199
భాగం-9	జీవక్రియల అంతఃస్పిద్ధ దోషాలు, ఒక జన్యువు / పాలిపెప్టైడ్.	200-209
ఖండం - IV	పరిణామం	211
భాగం-10	జీవ ఆవిర్భావం మరియు జీవపరిణామ సిద్ధాంతాల ఉపోద్ఘాతం: లామార్క్, నియోలామార్క్, డార్విన్, నియోడార్విన్, ఆధునిక కృత్రిమ సిద్ధాంతం.	213-229
భాగం-11	జీవపరిణామానికి ప్రత్యేక నిదర్శనాలు : శిలాజాల రకాలు, శిలాజాల డేటింగ్, మానవుడు మరియు గుర్రం పరిణామం.	230-266
భాగం-12	జాతిభావన : వివక్షత - సంభోగ పూర్వ మరియు సంభోగ అనంతర యాంత్రికత, జాతుల ఉత్పత్తి - విధానాలు	267-278
	మాదిరి పరీక్షా పత్రం	279-281

UG419 SEE(1)-ZOO-E

B.Sc.

SECOND YEAR SEMESTER-IV

ZOOLOGY

SKILL ENHANCEMENT ELECTIVE COURSE-SEE-1

APICULTURE



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2019**

CONTENTS

Block/Unit	Title	Page
BLOCK- I BEE COLONY MAINTENANCE		
Unit –1	: Bee species and Distribution.....	1 - 15
Unit –2	: Life history and Hive.....	16 - 32
Unit –3	: Bee Keeping Equipment.....	33 - 40
Unit– 4	: Harvesting and Processing of Honey.....	41 - 57
BLOCK -II APICULTURE MANAGEMENT		
Unit –5	: Bee Diseases	58 – 72
Unit –6	: Pests and Parasites.....	73 - 78
Unit –7	: Products of Apiculture Industry and its uses.....	79 – 87
Unit –8	: Economics.....	88 – 92
	Model Question Paper.....	93- 94

UG419 SEE(2)-ZOO-E

B.Sc.

**SECOND YEAR SEMESTER-IV
ZOOLOGY**

SKILL ENHANCEMENT ELECTIVE COURSE-SEE-2

VERMICULTURE



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2019**

CONTENTS

Block/Unit	Title	Page
BLOCK- I VERMI COMPOSTING		
Unit –1	: Selection of Site and Earthworms.....	1 - 16
Unit –2	: Composting Systems and Techniques.....	17 - 28
Unit - 3	: Bedding.....	29 - 41
Unit– 4	: Essential Parameters of Vermiculture.....	42 - 51
BLOCK -II MANAGEMENT AND ECONOMICS		
Unit –5	: Harvesting.....	52 – 61
Unit –6	: Worm Composting Pests and its Management.....	62 - 69
Unit –7	: Economic Importance of Vermiculture.....	70 - 78
Unit –8	: Economics.....	79 - 85
	Model Question Paper.....	86- 87

BS 519 ZOO -T

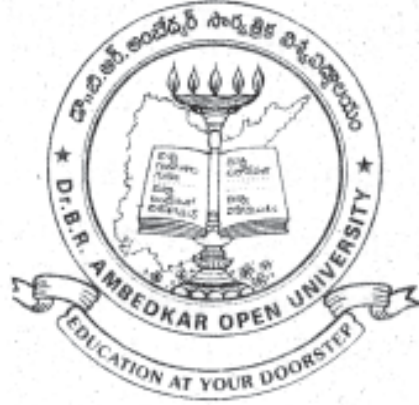
బి.ఎన్.సి

మూడవ సంవత్సరం

బదవ సెమిస్టర్

జంతు శాస్త్రం

శరీరధర్మశాస్త్రం మరియు జీవరసాయన శాస్త్రం



“మనం నాగరికత సమకూర్చిన పస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు.

ఎందుకంటే విద్యను మించిన పస్తుగత ప్రయోజనమేది లేదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2020

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజి.నెం.
ఖండం-1	శరీర ధర్మశాస్త్రం - I	1
భాగం-1 :	జీర్ణక్రియ	3-17
భాగం-2 :	శ్వాసక్రియ	18-31
భాగం-3 :	ప్రసరణ	32-49
ఖండం-2	శరీర ధర్మశాస్త్రం - II	51
భాగం-4 :	విసర్జన	53-62
భాగం-5 :	కండర సంకోచం	63-80
భాగం-6 :	నాడీ ప్రచోదనం	81-97
ఖండం-3	అంతఃస్రావక గ్రంథులు మరియు ఎంజైములు	99
భాగం-7 :	అంతః స్రావక వ్యవస్థ	101-121
భాగం-8 :	ద్రవాభిసరణ క్రమత	122-131
భాగం-9 :	ఎంజైములు	132-133
ఖండం-4	జీవరసాయన శాస్త్రం మరియు జీవాణువులు	139
భాగం-10 :	పిండి పదార్థాలు	141-159
భాగం-11 :	మాంసకృత్తులు	160-172
భాగం-12 :	లిపిడ్లు	173-180
మాదిరి పరీక్షా ప్రశ్న		181-182

B.Sc.

**SECOND YEAR SEMESTER-V
ZOOLOGY**

**PRINCIPLES OF AQUACULTURE
DISCIPLINE SPECIFIC ELECTIVE COURSE-A**



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2020**

CONTENTS

Block/Unit	Title	Page
BLOCK- I BIOLOGY OF FISHES		
Unit –1	: Biology of Fishes.....	1-15
Unit –2	: Edible Aquatic Forms.....	16-33
Unit- 3	: Life History of Cultivated fish and prawns.....	34-42
BLOCK -II FISHERY SCIENCE		
Unit –4	: Migration of Fishes.....	43-50
Unit –5	: Post Harvest Technology.....	51-62
Unit –6	: Crafts and Gears.....	63-78
BLOCK- III FISH CULTURE SYSTEMS		
Unit –7	: Composite and Air Breathing Fish Culture.....	79-88
Unit –8	: Fresh water Culture Systems.....	89-114
Unit –9	: Coastal Aquaculture and Mariculture.....	115-128
BLOCK -IV FISH FARM MAINTENANCE		
Unit –10	: Farm Pond and Management.....	129-149
Unit –11	: Fish Nutrition.....	150-157
Unit –12	: Fish Diseases.....	158-171
	MODEL QUESTION PAPER	172

BS 519 ZOO DSE (A) - T

బి.ఎస్.సి

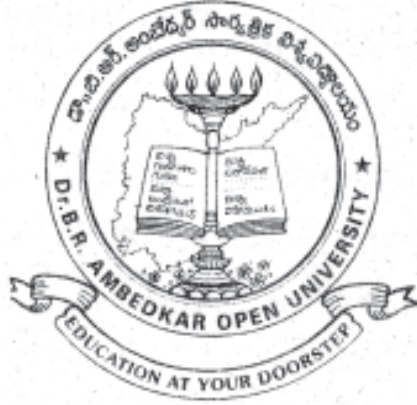
మూడవ సంవత్సరం

బదవ సెమిస్టర్

జంతుశాస్త్రం

డిసిప్లినీన్ స్పెసిఫిక్ ఎలక్టివ్ కోర్సు - ఎ

పట్టు పరిశ్రమ



“మనం నాగరికత సమకూర్చిన పస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు.

ఎందుకంటే విద్యను మించిన పస్తుగత ప్రయోజనమేది లేదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2020

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పేజీ.నెం.
ఖండం-1	సాదారణ పట్టుపరిశ్రమ	1
భాగం-1 :	పట్టుపరిశ్రమ చరిత్ర	3-9
భాగం-2 :	పట్టుపురుగుల రకాలు	10-22
భాగం-3 :	మల్బరీ వర్గీకరణ	23-34
ఖండం-2	మల్బరీ సాగు మరియు క్షేత్ర నిర్వహణ	35
భాగం-4 :	మల్బరీ ఉత్పత్తి	37-57
భాగం-5 :	సాగు పద్ధతులు	58-83
భాగం-6 :	మల్బరీ వ్యాధులు మరియు చీడలు	84-101
ఖండం-3	పట్టుపురుగు జీవశాస్త్రం	103
భాగం-7 :	పట్టుపురుగు జీవశాస్త్రం	105-131
భాగం-8 :	పట్టుపురుగు పోషణ మరియు పట్టుగ్రంథి	132-139
భాగం-9 :	పట్టు పురుగుల పెంపకం	140-182
ఖండం-4	పెంపకం మరియు రీలింగు సాంకేతికత	183
భాగం-10 :	పట్టుపురుగు వ్యాధులు మరియు చీడలు	185-202
భాగం-11 :	పట్టుకాయల రీలింగ్	203-233
భాగం-12 :	ముడి పట్టు	234-253
మాదిరి పరీక్షా పత్రం		254-255

BS 619 ZOO-T

బి.ఎస్.సి.

మూడవ సంవత్సరం సెమిస్టర్ - VI

జంతుశాస్త్రం

వ్యాధినిరోధక శాస్త్రం మరియు జంతుజీవసాంకేతికశాస్త్రం



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైనా వదులుకోవచ్చునేమోగాని, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కులను మాత్రం కోల్పోకూడదు. ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేమీ లేదు”.

- డా॥ బి.ఆర్. అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాదు

2020

విషయసూచిక

ఖండం/భాగం	పాఠ్యాంశం	పేజీ నం.
ఖండం - I	వ్యాధినిరోధక శాస్త్రం-I	
భాగం - 1	వ్యాధి/రోగనిరోధక వ్యవస్థలో ప్రాథమిక అంశాలు	1-15
భాగం - 2	ప్రతిజనకాలు	16-24
భాగం - 3	ప్రతిరక్షకాలు/ప్రతిదేహాలు/ప్రతిజీవకాలు (Antibodies)	25-41
ఖండం - II	వ్యాధినిరోధక శాస్త్రం- II	
భాగం - 4	రోగనిరోధక విధుల క్రమత	42-52
భాగం - 5	ఆరోగ్యం మరియు వ్యాధిలో రోగనిరోధక వ్యవస్థ	53-60
భాగం - 6	టీకాలు	61-68
ఖండం - III	జంతుజీవసాంకేతిక శాస్త్రం- I	
భాగం - 7	జంతుజీవసాంకేతిక శాస్త్రం భావనలు మరియు పరిధి	69-74
భాగం - 8	జన్యు తారుమారు (Manipulations)	75-86
భాగం - 9	జంతుకణ వర్ణనం	87-96
ఖండం - IV	జంతుజీవసాంకేతిక శాస్త్రం- II	
భాగం - 10	జన్యుమార్పిడి / ట్రాన్స్ జెనిసిస్ (Transgenesis)	97-113
భాగం - 11	మూలకణాలు	114-126
భాగం - 12	జన్యుచికిత్స/థెరపీ	127-131
	నమూనా ప్రశ్నపత్రం	132-134

B.Sc.

**THIRD YEAR SEMESTER-VI
ZOOLOGY**

**FISH GENETICS AND SEED TECHNOLOGY
DISCIPLINE SPECIFIC ELECTIVE COURSE-C**



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD
2020**

CONTENTS

Block/Unit	Title	Page
BLOCK- I FISH GENETICS-I		
Unit –1	Chromosomes in Fishes.....	1- 9
Unit –2	Inheritance of Qualitative Morphological Traits in Fish.....	10-23
Unit- 3	Inheritance of Quantitative Traits in Fish.....	24-30
BLOCK -II FISH GENETICS-II		
Unit –4	Chromosome Manipulation.....	31-42
Unit –5	Transgenic Fish Production.....	43-54
Unit –6	Cryopreservation of Gametes.....	55-65
BLOCK- III SEED TECHNOLOGY-I		
Unit –7	Fish Seed Collection.....	66-72
Unit –8	Bundh breeding.....	73-79
Unit –9	Induced breeding.....	80-91
BLOCK -IV SEED TECHNOLOGY-II		
Unit –10	Carp hatcheries, transportation of breeders and seed	92-114
Unit –11	Shrimp Hatchery technology	115-136
Unit –12	Fishery institutes and extension in India	137-143
	MODEL QUESTION PAPER	144

బి.ఎన్.సి

మూడవ సంవత్సరం

ఆరవ సెమిస్టర్

జంతుశాస్త్రం

పట్టుపురుగుల ప్రజననం మరియు గ్రెయినేజ్
డిసిప్లిన్ స్పెసిఫిక్ ఎలక్టివ్ కోర్సు-డి



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలైన వదులుకోవచ్చునేమో గానీ, సర్వోత్పన్నమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు. ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేదీ లేదు.”

-డా॥ బి.ఆర్.అంబేద్కర్

డా. బి.ఆర్.అంబేద్కర్ సార్వత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2020

విషయ సూచిక

ఖండం/భాగం	అంశం	పేజీలు
ఖండం-I: పట్టుపురుగుల ప్రజననం		
భాగం-1:	పట్టుపురుగుల ప్రజననం	1
భాగం-2:	ఎంపిక పద్ధతులు	10
భాగం-3:	హిటరోసిస్ మరియు సంకరతేజం	24
ఖండం-II: గ్రెయినేజ్-I		
భాగం-4:	జనకతరాలు	31
భాగం-5:	ప్రజనన కేంద్రాలు	39
భాగం-6:	విత్తన పట్టుకాయల ఎంపిక	48
ఖండం-III: గ్రెయినేజ్-II		
భాగం-7:	గ్రెయినేజ్	57
భాగం-8:	పట్టుపురుగుల గుడ్ల ఉత్పత్తి	75
భాగం-9:	గుడ్ల సంరక్షణ మరియు నిద్రావస్థ	86
ఖండం-IV: గ్రెయినేజ్-III		
భాగం-10:	అమ్మ చికిత్స	95
భాగం-11:	ఉప ఉత్పత్తులు	102
భాగం-12:	అర్థికాంశాలు	108
మాదిరి పరీక్షా ప్రశ్నలు		113

BS127 STAT-E

B.Sc.

FIRST YEAR SEMESTER - I

STATISTICS

**DESCRIPTIVE STATISTICS &
PROBABILITY**



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr.B.R.Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

CONTENTS

BLOCK/UNIT	TITLE	PAGE
Block-I:	Data	1
Unit-1:	Method of Collection and Editing of Primary Data	3-13
Unit-2:	Method of Collection and Editing of Secondary Data	14-21
Unit-3:	Classification and Tabulation of the Data	22-38
Block-II:	Measures of Central Tendency and Dispersion	39
Unit -4:	Mean, Median, Mode, GM, HM	41-71
Unit -5:	Range, QD, MD, SD	72-95
Unit -6:	Moments, Central and Non-central Moments, Skewness	96-113
Block-III:	Addition Rule	115
Unit - 7:	Definition and Basic Concepts	117-130
Unit - 8:	Addition Theorem of Probability	131-141
Unit - 9:	Some More Important Theorems	142-147
Block-IV:	Baye's Theorem	149
Unit - 10:	Conditional Probability	151-158
Unit - 11:	Independent Events	159-169
Unit - 12:	Baye's Theorem	170-176
	Model Examination Question Paper	177-180

BS227 STAT - E

B.Sc.

FIRST YEAR SEMESTER - II

STATISTICS

MATHEMATICAL EXPECTATION

&

MOMENT INEQUALITY



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr.B.R.Ambedkar

Dr. B.R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

CONTENTS

BLOCK/UNIT	TITLE	PAGE
Block-I: Random Variables		185
Unit-1: Definition –discrete and continuous random variables		187 - 194
Unit- 2: Probability mass function, Probability density function and distribution functions		195 - 215
Unit-3: Transformation of one-dimensional random variable.		216 - 228
Block - II: Bivariate Random Variables		229
Unit-4: Bivariate Distribution & Properties		231 - 238
Unit-5: Joint, Marginal and Conditional Distribution		239 - 246
Unit-6: Independence of Random Variables		247 - 251
Block - III: Expectation, Moments and co-variance		253
Unit-7: Expectation of Function of Random Variable		255 - 266
Unit-8: Central Moments and Covariance		267 - 277
Unit-9: Addition and Multiplication Theorems		278 - 284
Block - IV: Generating Functions		285
Unit-10: Moment generating function and Cumulant generating function		287 - 301
Unit- 11: Probability Generating function and Characteristic Generating Function		302 - 311
Unit-12: Cauchy –Schwartz inequality		312 - 317
Model Question Paper		318 - 320

BS 327 STAT – E

B.Sc.

SECOND YEAR SEMESTER – III

STATISTICS

DISCRETE DISTRIBUTIONS



“We may forego material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

Dr. B. R. AMBEDKAR OPEN UNIVERSITY

HYDERABAD

2019

CONTENTS

Block/Unit No.	Title	Page No.
Block - I:	Discrete Distributions - I	1-42
Unit-1:	Bernoulli and Binomial Distributions	2-22
Unit-2:	Poisson Distribution	23-36
Unit-3:	Discrete Uniform Distribution	37-42
Block - II:	Discrete Distributions - II	43-69
Unit-4:	Negative Binomial distribution	44-53
Unit-5:	Geometric Distribution	54-62
Unit-6:	Hypergeometric Distribution	63-69
Block - III:	Applications - I	70-93
Unit-7:	Applications of Bernoulli and Binomial Distributions	71-80
Unit-8:	Application of Poisson Distribution	81-87
Unit-9:	Applications of Negative Binomial Distribution	88-93
Block - IV:	Applications - II	94-118
Unit-10:	Problems on Hypergeometric Distribution	95-100
Unit-11:	Problems on Uniform Distribution	101-107
Unit-12:	Miscellaneous Problems on Discrete Distribution	108-118
	Model Question Paper	119-207

BS 427 STAT – E

B.Sc.

SECOND YEAR SEMESTER – IV

STATISTICS

CONTINUOUS DISTRIBUTIONS



“We may forego material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2020

CONTENTS

Block/Unit No.	Title	Page No.
Block - I:	Continuous Distributions - I	1-38
Unit-1:	Rectangular Distribution	2-9
Unit-2:	Normal Distribution - I	10-27
Unit-3:	Normal Distribution - II	28-38
Block - II:	Continuous Distribution – II	39-70
Unit-4:	Gamma Distribution	40-49
Unit-5:	Beta Distribution	50-62
Unit-6:	Cauchy Distribution	63-70
Block - III:	Applications - I	71-109
Unit-7:	Applications of Rectangular Distribution	72-80
Unit-8:	Applications of Normal Distribution – I	81-97
Unit-9:	Applications of Normal Distributions – II	98-109
Block - IV:	Applications - II	110-134
Unit-10:	Applications of Gamma Distribution	111-116
Unit-11:	Applications of Beta Distribution	117-124
Unit-12:	Applications of Cauchy Distribution	125-134
•	Model Question Paper	135-138
•	Normal Distribution Tables	139-140

UG 405 SEE (STAT 1) –E

B.Sc.

SECOND YEAR SEMESTER – IV

STATISTICS

SKILL ENHANCEMENT ELECTIVE COURSE-SEE-I

**STATISTICAL DATA ANALYSIS
USING MS EXCEL**



“We may forgo material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2020

CONTENTS

Block/Unit No.	Title	Page No.
Block - I: Excel and Functions		1-52
Unit-1:	Introduction to Excel	2-14
Unit-2:	Basic Mathematical Functions	15-31
Unit-3:	Statistical Functions - Measures of Central Tendency MEAN, MEDIAN, MODE, GEOMETRIC MEAN(GM), HARMONIC MEAN(HM)	32-41
Unit-4:	Statistical Function - Measures of DISPERSION, SKEWNESS, KURTOSIS, COVARIANCE	42-52
Block - II: Built-in Functions		53-102
	RSQ, NORDIST etc,	
Unit-5:	RSQ, RANK, FORECAST, TRIMMEAN functions	54-64
Unit-6:	NORMDIST, NORMINV, NORMSDIST, NORMSINV functions	65-72
Unit-7:	CONFIDENCE , VAR, ZTEST, TDIST, TINV, tTEST functions	73-89
Unit-8:	FDIST, FINV, CHIDIST, CHIINV, CHITEST functions	90-102
•	Model Question Paper	103-105

UG 405 SEE (STAT2)–E

B.Sc.

SECOND YEAR SEMESTER – IV

STATISTICS

SKILL ENHANCEMENT ELECTIVE COURSE - SEE-2

**STATISTICAL TECHNIQUES FOR
RESEARCH METHODS**



“We may forgo material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2020

CONTENTS

Block/Unit No.	Title	Page No.
Block - I:	Research and Sampling	1-42
Unit-1:	Introduction to Research	2-10
Unit-2:	Research Process	11-20
Unit-3:	Sampling Designs	21-29
Unit-4:	Data Collection and Presentation	30-42
Block - II:	Model and Report	43-90
Unit-5:	Data Analysis	44-56
Unit-6:	Models and Model Building	57-63
Unit-7:	Interpretation and Report Writing	64-80
Unit-8:	Presentation of Reports	81-90
•	Model Question Paper	91-92

BS 527 STAT – E

B.Sc.

THIRD YEAR SEMESTER – V

STATISTICS

**CORRELATION,
REGRESSION, SAMPLING**



“We may forgo material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2021

CONTENTS

Block/Unit No.	Title	Page No.
Block - I:	Correlation	1-52
Unit-1:	Correlation	2-19
Unit-2:	Rank Correlation	20-36
Unit-3:	Correlation Ratio	37-52
Block - II:	Correlation, Regression	53-79
Unit-4:	Simple Linear Regression	54-66
Unit-5:	Regression for 3 Variables	67-73
Unit-6:	Multiple and Partial Correlations	74-79
Block - III:	Data Analysis	80-115
Unit-7:	Introduction to Categorical Data Analysis	81-91
Unit-8:	Why Categorical data Analysis ?	92-97
Unit-9:	Independence and Association of Attributes	98-115
Block - IV:	Sampling	116-152
Unit-10:	Simple Random Sampling	117-129
Unit-11:	Stratified Random Sampling	130-146
Unit-12:	Systematic Sampling	147-152
•	Model Question Paper	153-156

BS 527 STAT DSE (A)-E

B.Sc.

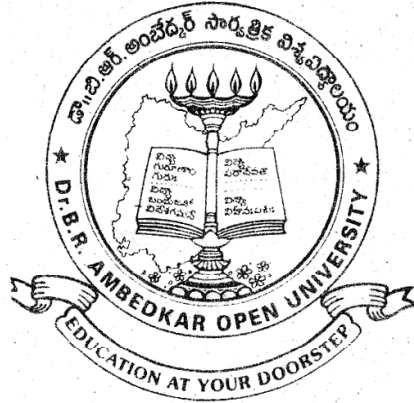
THIRD YEAR

SEMESTER – V

STATISTICS

DISCIPLINE SPECIFIC ELECTIVE COURSE - A

DESIGN OF EXPERIMENTS



“We may forgo material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2021

CONTENTS

BLOCK / Unit	Title	Page
BLOCK-I	Analysis of Variance one-way Classification	1-41
Unit-1	: Introduction, Cochran's Theorem	2-16
Unit-2	: Analysis of Variance one-way Classification Mathematical Model, Estimation of sum of squares	17-29
Unit-3	: Analysis of Variance One way classification F-Test	30-41
BLOCK-II	Anova Two - way classification	42-75
Unit-4	: Introduction - Analysis of Variance Two-way Classification	43-48
Unit-5	: Anova Two-way classification - Mathematical Model, Estimation of Sum of Squares.	49-62
Unit-6	: Analysis of Variance, Two-Way Classification, F- Test	63-75
BLOCK-III	Analysis of Design	76-112
Unit-7	: Principles of Experimentation	77 - 89
Unit-8	: Completely Randomized Design (CRD)	90 - 99
Unit-9	: E(MS), F-Test	100 - 112
BLOCK-IV	Randomised Block Design and Latin Square Design	113-162
Unit-10	: Introduction to RBD	114 - 122
Unit-11	: Analysis and Model Interpretation, E(MS), F-Tests, One Missing Observation and Its Estimation in RBD	123- 141
Unit-12	: Latin Square Design	142 - 162
	Practical Model Question Paper	163 - 166

BS 527 STAT DSE (B) – E

B.Sc.

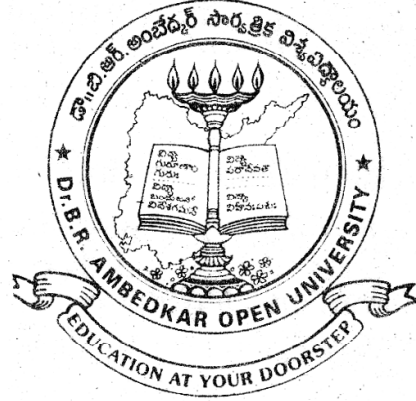
THIRD YEAR

SEMESTER – 5

STATISTICS

DISCIPLINE SPECIFIC ELECTIVE COURSE - B

BIostatISTICS



“We may forgo material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2021

CONTENTS

BLOCK / Unit	Title	Page
BLOCK-I Bio Assay		1-38
Unit-1	: Purpose and structure of biological Assay, Types of biological assays	2-14
Unit-2	: Direct Assays, Ratio Estimates	15-27
Unit-3	: Asymptotic Distributions, Feller's theorem and Regression approach to estimate dose – response and relationships	28-38
BLOCK-II Logit & Probit Approaches		39-121
Unit-4	: Logit and Probit approaches when dose response curve for standard preparation is unknown, quantal responses	40-52
Unit-5	: Methods of estimation of parameters, estimation of extreme quantiles	53-78
Unit-6	: Dose allocation schemes, polychotomous quantal response, estimation of points on the quantal response function	79-121
BLOCK-III Statistical Genetics - I		122-152
Unit-7	: Concepts of Gene and Genotypes	123 - 129
Unit-8	: Mendel's & Hardy Weinberg Laws	130 -139
Unit-9	: Estimation of Allele Frequency (Dominant/ Codominant Cases), Multiple Alleles	140 - 152
BLOCK-IV Statistical Genetics - II		153-181
Unit-10	: Approach to Equilibrium for X-Linked Gene	154 - 161
Unit-11	: Natural Selection, Mutation and Genetic Drift	162 - 175
Unit-12	: Equilibrium When Both Natural Selection and Mutation Operative	176 - 181
	Model Examination Question Paper	182 - 185

BS 627 STAT – T

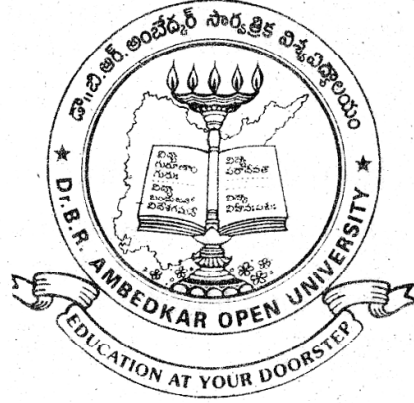
బి.ఎస్.సి

మూడవ సంవత్సరం

ఆరవ సెమిస్టర్

గణాంకశాస్త్రం

పరికల్పన అంచనా, పరీక్ష



“మనం నాగరికత సమకూర్చిన వస్తుగత ప్రయోజనాలనైన వదులుకోవచ్చునేమో గానీ, సర్వోత్కృష్టమైన విద్య అందించే ఫలాలను సంపూర్ణంగా అనుభవించే అవకాశాలను, హక్కును మాత్రం కోల్పోకూడదు.

ఎందుకంటే విద్యను మించిన వస్తుగత ప్రయోజనమేది లేదు”

-డా॥బి.ఆర్.అంబేద్కర్

డా॥ బి.ఆర్. అంబేద్కర్ సార్యత్రిక విశ్వవిద్యాలయం

హైదరాబాద్

2021

విషయ సూచిక

ఖండము / భాగం	పాఠ్యాంశం	పుట.నం.
ఖండం-I	అంచనా సిద్ధాంతము	1-58
భాగం-1 :	అంచనా యొక్క ప్రాథమిక భావనలు, ఉత్తమ అంచనాదారుని లక్షణాలు - పటిష్ఠత, నిష్పాక్షికత మరియు సమర్థత.	2-22
భాగం-2 :	ఉత్తమ అంచనాకారి యొక్క లక్షణాలు - పర్యాప్తము, కనిష్ఠ విస్తృతి నిష్పాక్షిక అంచనాకారి	23-40
భాగం-3 :	అంచనా పద్ధతులు - గరిష్ఠ సంభావిత అంచనా పద్ధతి, భ్రామక పద్ధతి మరియు కనిష్ఠ విస్తృతి పద్ధతి.	41-58
ఖండం-II	గణాంక పరికల్పన యొక్క పరీక్ష	59-98
భాగం-4 :	పరికల్పన పరీక్ష యొక్క భావనలు	60-72
భాగం-5 :	నేమాన్ పియర్సన్ లెమ్మా ఆధారిత సమస్యలు	73-85
భాగం-6 :	ఏక సగటు మరియు నిష్పత్తి కొరకు పరీక్షలు	86-98
ఖండం-III	సరాసరి, నిష్పత్తి మరియు క్రమ విచలనాల కొరకు పరీక్షలు	99-164
భాగం-7 :	సగటుల కొరకు, నిష్పత్తుల మరియు క్రమ విచలనాలకు Z - పరీక్ష	100-119
భాగం-8 :	సగటుల t - పరీక్ష జంట నమూనాల t - పరీక్ష	120-141
భాగం-9 :	F - పరీక్ష మరియు ఖై వర్గాల పరీక్ష	142-164
ఖండం-IV	అపరామితియ పద్ధతులు	165-217
భాగం-10 :	సంజ్ఞ మరియు రన్ (పరుగు) పరీక్ష	166-181
భాగం-11 :	విల్కోగ్నన్ సంజ్ఞాయుత స్థల పరీక్ష మరియు మాన్ విట్టె విల్కోగ్నన్ U పరీక్ష	182-202
భాగం-12 :	మధ్యగత పరీక్ష, కొల్మోగోవ్ - స్మిర్నోవ్ పరీక్ష & క్యూస్కల్ వల్లీ పరీక్ష	203-217
	మాదిరి పరీక్షా ప్రశ్నా పత్రం	218-221
	టేబుల్స్	222-228

BS 627 STAT DSE (C)-E

B.Sc.

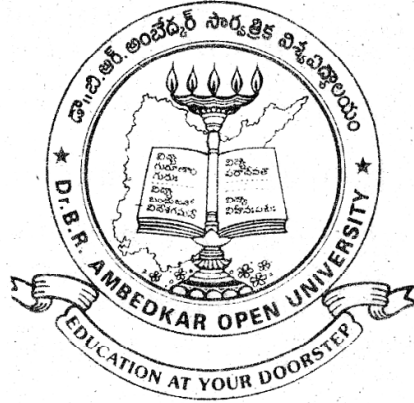
THIRD YEAR

SEMESTER – 6

STATISTICS

DISCIPLINE SPECIFIC ELECTIVE COURSE - C

OPERATIONS RESEARCH



“We may forgo material benefits of civilization, but we cannot forego our right and opportunity to reap the benefits of the highest education to the fullest extent...”

Dr. B. R. Ambedkar

**Dr. B. R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2021

CONTENTS

BLOCK / Unit	Title	Page
BLOCK-I	Linear Programming Problems	1-49
Unit-1	: Formulation of LPP	2-15
Unit-2	: Graphical Method	16-36
Unit-3	: Theorems Related to Optimal Vertex Point	37-49
BLOCK-II	Surplus etc variables, BIG M-Method	50-109
Unit-4	: Slack, Surplus, Artificial Variables	51-69
Unit-5	: Big-M Method	70-91
Unit-6	: Two Phase Simplex Method	92-109
BLOCK-III	DUAL, TP	110-170
Unit-7	Dual Simplex Method:	111 - 132
Unit-8	: Initial Basic Feasible Solution (BFS) By NWC, Least Cost Method, VAM	133 -152
Unit-9	: Unbalanced Transportation Problem	153 - 170
BLOCK-IV	Assignment Problem	171-212
Unit-10	: Assignment Problem Hungarian Method Travelling Salesman Problem	172 - 193
Unit-11	: Two Machine Job Sequencing Problem	194 - 204
Unit-12	: Three Machine Job Sequencing Problems	205 - 212
	Model Question Paper	213 - 218

:

BS 627 STAT DSE (D) – E

B.Sc.

THIRD YEAR

SEMESTER – 6

STATISTICS

DISCIPLINE SPECIFIC ELECTIVE COURSE - D

STATISTICAL COMPUTING

USING C/C++ PROGRAMMING



“We may forgo material benefits of civilization, but we cannot forgo our right and opportunity to reap the benefits of the highest education to the fullest extent as the education is the greatest material benefit”

-Dr. B.R. Ambedkar

**Dr. B.R. AMBEDKAR OPEN UNIVERSITY
HYDERABAD**

2022

CONTENTS

<u>Block/Unit</u>	<u>Title</u>	<u>Page</u>
BLOCK I: FUNDAMENTALS OF C		1-4
Unit-1	Program Design	1-3
Unit-2	Evolution of C Language	
Unit-3	Basics of C Language	2-4
BLOCK II: CONTROL STRUCTURES		5-8
Unit-4	Flow of Control	5-11
Unit-5	Functions	5-12
Unit-6	Pointers and Strings	7-8
BLOCK III: DERIVED DATA TYPES		9-15
Unit-7	Arrays	9-10
Unit-8	Structures and Unions	10-11
Unit-9	Files	12-13
BLOCK - IV: INTRODUCTION TO C++		14-23
Unit-10	Classes and Objects	14-16
Unit-11	Inheritance	16-21
Unit-12	Polymorphism	21-23
Model Examination Question Paper		24-26