Sami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

A

Course File

on

Object Oriented Programming (3IT4-06)

Program: B. Tech.
Semester: III
Session: 2020-21

Dolly Mittal
Assistant Professor
Information Technology

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Contents

Note:

1.*1st lecture of the course should cover prerequisite
2. **E: Easy, M: Moderate, D: Difficult
3. Format for Points 8-11 should be referred from AICTE's Recommendations for Examination Reforms

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Vision and Mission of Institute

Vision:

To promote higher learning in technology and industrial research to make our country a global player.

Mission:

To promote quality education, training and research in the field of engineering by establishing effective interface with industry and to encourage the faculty to undertake industry sponsored projects for the students.

Quality Policy of Institute

We are committed to 'achievement of quality' as an integral part of our institutional policy by continuous self-evaluation and striving to improve ourselves.

Institute would pursue quality in:

- All its endeavors like admissions, teaching- learning processes, examinations, extra and co-curricular activities, industry institution interaction, research & development, continuing education, and consultancy.

- Functional areas like teaching departments, Training & Placement Cell, library, administrative office, accounts office, hostels, canteen, security services, transport, maintenance section and all other services.

27

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Vision of IT Department

Vision of IT department is to:

"To design and deliver intelligent IT industry oriented education."

Mission of IT Department

Mission of IT department is to:

To prepare students to meet the need of users within an organizational and societal context through:

M2: Selection, creation, application, integration and administration of computing technologies

M3: Delivering student resource in the IT enabled domain.

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

RTU Scheme & Syllabus

RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

II Year-III Semester: B.Tech. Computer Science and Engineering

3CS4-06: Object Oriented Programming

Credit-3
3L+0T+0P

Max. Marks : 150 (IA:30,ETE:120)
End Term Exam: 3 Hours

SN	CONTENTS	Hours
1	Introduction to different programming paradigm, characteristics of OOP, Class, Object, data member, member function, structures in C++, different access specifiers, defining member function inside and outside class, array of objects.	8
2	Concept of reference, dynamic memory allocation using new and delete operators, inline functions, function overloading, function with default arguments, constructors and destructors, friend function and classes, using this pointer.	8
3	Inheritance, types of inheritance, multiple inheritance, virtual base class, function overriding, abstract class and pure virtual function	9
4	Constant data member and member function, static data member and member function, polymorphism, operator overloading, dynamic binding and virtual function	9
5	Exception handling, Template, Stream class, File handling.	6
	TOTAL	**40**

(4)

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Prerequisite of Course

1. Knowledge of C programming.
2. Basic knowledge of Turbo C++ or Dev C++ editors.

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

List of Text and Reference Books

1. Balagurusamy – Object Oriented Programming with C++, seventh edition, McGraw Education, 2017.
2. K.R. Venugopal, Rajkumar, T Ravishankar, "Mastering C++", McGraw Hill Publishing Co. Ltd, 2006.
3. Robert Lafore, Object Oriented Programming in Turbo C++, First Edition, Galgotia Publications.
4. Herbert Shildt, " C++ : The complete reference", Fourth Edition, McGraw Hill Education, 2017

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Time Table

Faculty Name : Dolly Mittal
Designation : Assistant Professor
Department : IT

LPW: 13 HRS

Day/Period	Lec. I 9-10	Lec. II 10-11	11-11:30	Lec. III 11:30-12:30	Lec. IV 12:30-1:30	Lec. V 1:30-2:30
Monday			L	Advance Java Lab 5ITB-G1		
Tuesday			U	Advance Java Lab 5ITA-G1		OOP Lecture 3ITA
Wednesday			N			OOP Lecture 3ITA
Thursday					OOP Lab 3ITA-G2	
Friday			C	Advance Java Lab 5ITB-G2		
Saturday	OOP Lecture 3ITA		H	Advance Java Lab 5ITA-G2		

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Syllabus Deployment: Course Plan & Coverage

Course Plan

SUBJECT: OOP (3IT4-06) **BRANCH: IT- A** **SEMESTER: III** **L/T/P: 3 / - / -**

UNIT	CONTENT
1	Prerequisite of Course, Introduction to different programming paradigm, characteristics of OOP, Class, Object, data member, member function, structures in C++, different access specifiers, defining member function inside and outside class, array of objects. **No. of Lectures Required: 08**
2	Concept of reference, dynamic memory allocation using new and delete operators, inline functions, function overloading, function with default arguments, constructors and destructors, friend function and classes, using this pointer. **No. of Lectures Required: 08**
3	Inheritance, types of inheritance, multiple inheritance, diamond problem, virtual base class, constructor and destructor in inheritance. **No. of Lectures Required: 09**
4	Constant data member and member function, static data member and member function, operator overloading, function overriding, polymorphism, dynamic binding and virtual function, pure virtual function and abstract class. **No. of Lectures Required: 09**
5	Exception handling, Template, Stream class, File handling **No. of Lectures Required: 06**

Total No. of Lectures Required: 40

(8)

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Course Coverage

3IT4-06 OBJECT ORIENTED PROGRAMMING

B.Tech.	Evaluation
Branch: Information Technology **Semester : III** **Lectures scheduled per week: 3**	**End Term Exam Time= 3 Hours** **Credits : 3** **Maximum Marks= 150** **[End Term Exam:120, Internal Assesment:30]**

Lecture No.	Date	Topic(s) Covered
1	1/07/20	Prerequisite of Course : Basic C Programming topics and pointers, structures.
2	3/07/20	Introduction to OOP, Basic Concepts of OOP: object, class
3	6/07/20	Encapsulation, inheritance, polymorphism
4	8/07/20	Dynamic binding, message passing, Introduction to C++, cin , cout
5	10/07/20	C++ : Token, data types, keywords, control structures, structures
6	13/07/20	C++ : Class, objects, data member, member functions, access specifiers, defining member function inside and outside class definition.
7	15/07/20	Memory allocation for objects, basic C++ program to illustrate working of object and class.
8	17/07/20	Instance variables and state of objects.
9	21/07/20	Member function calling other member function, private member functions.
10	24/07/20	Array of objects.
11	25/07/20	Default Arguments
12	28/07/20	Inline Functions
13	31/07/20	Function Overloading
14	1/08/20	Concept of reference variable
15	4/08/20	Function Calling mechanisms – Call by value, address and reference
16	7/08/20	Constructors
17	8/08/20	Constructor Overloading
18	10/08/20	Dynamic Initialization for Objects, Destructors
19	14/08/20	Dynamic Memory Allocation
20	18/08/20	Dynamic Objects, Dynamic Constructors & Destructors
21	21/08/20	Constant Data Members
22	22/08/20	Constant Member Functions
23	25/08/20	Static Data Members
24	28/08/20	Static Member Functions
25	29/08/20	this Pointer
26	1/09/20	Passing and Returning Objects
27	4/09/20	Friend Function: Introduction
28	5/09/20	Function Friendly to two or more classes
29	8/09/20	Friend class
30	11/09/20	Operator overloading : Binary operator

Swami Keshvanand Institute of Technology, Management &
Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Course Coverage

3IT4-06 OBJECT ORIENTED PROGRAMMING

B.Tech.	Evaluation
Branch: Information Technology **Semester : III** **Lectures scheduled per week: 3**	**End Term Exam Time= 3 Hours** **Credits : 3** **Maximum Marks= 150** **[End Term Exam:120, Internal Assesment:30]**

Lecture No.	Date	Topic(s) Covered
31	12/09/20	Operator overloading: Unary operator
32	15/09/20	Overloading by friend function
33	18/09/20	Overloading increment operators
34	19/09/20	Overloading insertion and extraction operators
35	22/09/20	Inheritance: Introduction
36	25/09/20	Single Inheritance
37	26/09/20	Hierarchical Inheritance
38	29/09/20	Multi-Level Inheritance
39	3/10/20	Multiple Inheritance
40	6/10/20	Ambiguity in multiple inheritance, hybrid inheritance, diamond problem
41	9/10/20	Virtual Base Class
42	10/10/20	Constructor and Destructor in Inheritance
41	13/10/20	Revision Class before Mid Term I
42	17/10/20	Function Overriding and Early Binding
43	20/10/20	Polymorphism, Dynamic Binding, Virtual Function

Program Outcome/Program Specific Outcome	Indicator	Statement
PO 1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialisation for the solution of complex engineering problems.	1.1.1	Apply mathematical techniques such as calculus, linear algebra, and statistics to solve problems
	1.1.2	Apply advanced mathematical techniques to model and solve computer science & engineering problems
	1.2.1	Apply laws of natural science to an engineering problem
	1.3.1	Apply fundamental engineering concepts to solve engineering problems
	1.4.1	Apply computer science & engineering concepts to solve engineering problems.
PO 2: Problem analysis: Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.	2.1.1	Articulate problem statements and identify objectives
	2.1.2	Identify engineering systems, variables, and parameters to solve the problems
	2.1.3	Identify the mathematical, engineering and other relevant knowledge that applies to a given problem
	2.2.1	Reframe complex problems into interconnected sub-problems
	2.2.2	problems Identify, assemble and evaluate information
	2.2.3	Identify existing processes/solution methods for solving the problem, including forming justified approximations and assumptions
	2.2.4	Compare and contrast alternative solution processes to select the best process.
	2.3.1	Combine scientific principles and engineering concepts to formulate model/s (mathematical or otherwise) of a system or process that is appropriate in terms of applicability and required accuracy.
	2.3.2	Identify assumptions (mathematical and physical) necessary to allow modeling of a system at the level of accuracy required.
	2.4.1	Apply engineering mathematics and computations to solve mathematical models
	2.4.2	Produce and validate results through skilful use of contemporary engineering tools and models
	2.4.3	Identify sources of error in the solution process, and limitations of the solution.
		Extract desired understanding and conclusions consistent with objectives and limitations of the analysis

	2.4.4	
PO 3: Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.	3.1.1	Recognize that need analysis is key to good problem definition
	3.1.2	Elicit and document, engineering requirements from stakeholders
	3.1.3	Synthesize engineering requirements from a review of the state-of-the-art
	3.1.4	Extract engineering requirements from relevant engineering Codes and Standards such as IEEE, ACM, ISO etc.
	3.1.5	Explore and synthesize engineering requirements considering health, safety risks, environmental, cultural and societal issues
	3.1.6	Determine design objectives, functional requirements and arrive at specifications
	3.2.1	Apply formal idea generation tools to develop multiple engineering design solutions
	3.2.2	Build models/prototypes to develop diverse set of design solutions
	3.2.3	Identify suitable criteria for evaluation of alternate design solutions
	3.3.1	Apply formal decision making tools to select optimal engineering design solutions for further development
	3.3.2	Consult with domain experts and stakeholders to select candidate engineering design solution for further development
	3.4.1	Refine a conceptual design into a detailed design within the existing constraints (of the resources)
	3.4.2	Generate information through appropriate tests to improve or revise design
PO 4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.	4.1.1	Define a problem, its scope and importance for purposes of investigation
	4.1.2	Examine the relevant methods, tools and techniques of experiment design, system calibration, data acquisition, analysis and presentation
	4.1.3	Apply appropriate instrumentation and/or software tools to make measurements of physical quantities
	4.1.4	Establish a relationship between measured data and underlying physical principles.
	4.2.1	Design and develop experimental approach, specify appropriate equipment and procedures

	4.2.2	Understand the importance of statistical design of experiments and choose an appropriate experimental design plan based on the study objectives
	4.3.1	Use appropriate procedures, tools, and techniques to conduct experiments and collect data
	4.3.2	Analyze data for trends and correlations, stating possible errors and limitations
	4.3.3	Represent data (in tabular and/or graphical forms) so as to facilitate analysis and explanation of the data, and drawing of conclusions
	4.3.4	Synthesize information and knowledge about the problem from the raw data to reach appropriate conclusions
PO 5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.	5.1.1	Identify modern engineering tools, techniques and resources for engineering activities
	5.1.2	Create/adapt/modify/extend tools and techniques to solve engineering problems
	5.2.1	Identify the strengths and limitations of tools for (i) acquiring information, (ii) modelling and simulating, (iii) monitoring system performance, and (iv) creating engineering designs.
	5.2.2	Demonstrate proficiency in using discipline specific tools
	5.3.1	Discuss limitations and validate tools, techniques and resources
	5.3.2	Verify the credibility of results from tool use with reference to the accuracy and limitations, and the assumptions inherent in their use.
PO 6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.	6.1.1	Identify and describe various engineering roles; particularly as pertains to protection of the public and public interest at global, regional and local level
	6.2.1	Interpret legislation, regulations, codes, and standards relevant to your discipline and explain its contribution to the protection of the public
PO 7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.	7.1.1	Identify risks/impacts in the life-cycle of an engineering product or activity
	7.1.2	Understand the relationship between the technical, socio economic and environmental dimensions of sustainability
	7.2.1	Describe management techniques for sustainable development
	7.2.2	Apply principles of preventive engineering and sustainable development to an engineering activity or product relevant to the discipline

PO 8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.	8.1.1	Identify situations of unethical professional conduct and propose ethical alternatives
	8.2.1	Identify tenets of the ASME professional code of ethics
	8.2.2	Examine and apply moral & ethical principles to known case studies
PO 9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.	9.1.1	Recognize a variety of working and learning preferences; appreciate the value of diversity on a team
	9.1.2	Implement the norms of practice (e.g. rules, roles, charters, agendas, etc.) of effective team work, to accomplish a goal.
	9.2.1	Demonstrate effective communication, problem solving, conflict resolution and leadership skills
	9.2.2	Treat other team members respectfully
	9.2.3	Listen to other members
	9.2.4	Maintain composure in difficult situations
	9.3.1	Present results as a team, with smooth integration of contributions from all individual efforts
PO 10: Communication: Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.	10.1.1	Read, understand and interpret technical and non-technical information
	10.1.2	Produce clear, well-constructed, and well- supported written engineering documents
	10.1.3	Create flow in a document or presentation
	10.2.1	Listen to and comprehend information, instructions, and viewpoints of others
	10.2.2	Deliver effective oral presentations to technical and non- technical audiences
	10.3.1	Create engineering-standard figures, reports and drawings to complement writing and presentations
	10.3.2	Use a variety of media effectively to convey a message in a document or a presentation
PO 11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.	11.1.1	Describe various economic and financial costs/benefits of an engineering activity
	11.1.2	Analyze different forms of financial statements to evaluate the financial status of an engineering project
	11.2.1	Analyze and select the most appropriate proposal based on economic and financial considerations.

	11.3.1	Identify the tasks required to complete an engineering activity, and the resources required to complete the tasks.
	11.3.2	Use project management tools to schedule an engineering project so it is completed on time and on budget.
PO 12: Life-long learning: Recognise the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.	12.1.1	Describe the rationale for requirement for continuing professional development
	12.1.2	Identify deficiencies or gaps in knowledge and demonstrate an ability to source information to close this gap
	12.2.1	Identify historic points of technological advance in engineering that required practitioners to seek education in order to stay current
	12.2.2	Recognize the need and be able to clearly explain why it is vitally important to keep current regarding new developments in your field
	12.3.1	Source and comprehend technical literature and other credible sources of information
	12.3.2	Analyze sourced technical and popular information for feasibility, viability, sustainability, etc.
PSO1: Core Engineering Skills: Acquire basic concepts of Data Structures, Databases, Operating Systems, Computer Network, Theory of Computation, Advanced Programming and Software Engineering.	PSO1.1.1	Possess the concepts of Data Structure and Database Management System
	PSO1.1.2	Possess the concepts of core engineering subjects including Operating System, Computer Networks and Software Engineering.
	PSO1.1.3	Apply basic programming skills to solve real world problems
PSO2: Standard Software Engineering practices: Demonstrate an ability to design, develop, test, debug, deploy, analyse, troubleshoot, maintain, and secure mobile applications and software solutions for automation applications.	PSO2.1.1	Apply fundamental software engineering concepts to solve real world problem
	PSO2.1.2	Possess conceptual knowledge for designing, analysing and testing a software
	PSO2.1.3	Estimate and evaluate the cost related to a Software
PSO3: Project Endeavours: Provide a platform for students to develop new and innovative projects as per current industry needs.	PSO3.1.1	Recognise the need and feasibility of project and apply standard practices for software project development
	PSO3.1.2	Identify the functional and nonfunctional requirement of current industry trends.
	PSO3.1.3	Recognise the challenges of changing trends and career opportunities as per current industry needs.

Swami Keshvanand Institute of Technology, Management &Gramothan, Jaipur

Programme: B.Tech. (Information Technology)
Semester: III
Course Name (Course Code): OBJECT ORIENTED PROGRAMMING (3IT4-06)

Course Outcomes

After completion of this course, students will be able to –

3IT4-06.1	***Describe*** the Object Oriented Programming paradigm with the concept of objects and classes.
3IT4-06.2	***Explain*** the memory management techniques using constructors, destructors and pointers..
3IT4-06.3	***Classify*** and ***demonstrate*** the various Inheritance techniques.
3IT4-06.4	***Understand*** how to ***apply*** polymorphism techniques on the object oriented problem.
3IT4-06.5	***Summarize*** the exception handling mechanism, file handling techniques and ***Use*** of generic programming in Object oriented programming

Name of Faculty:Ms.Shalini Singhal
(Signature)

Name of Faculty: Dolly Mittal
(Signature)

Verified by Course Coordinator

Signature
(Name: ……..)

Verified by Verification and Validation Committee, DPAQIC

Signature

(Name: ……..)

16

Swami Keshvanand Institute of Technology, Management &Gramothan, Jaipur

COURSE: Object Oriented Programming(3IT4-06)

Course Outcomes		Bloom's Level	PO Indicators	PSO Indicators
Upon successful completion of this course, students should be able to:				
3IT4-06.1	***Describe*** the Object Oriented Programming paradigm with the concept of objects and classes	1	1.3.1,1.4.1, 5.1.1, 5.2.1	1.1.3
3IT4-06.2	***Explain*** the memory management techniques using constructors, destructors and pointers.	2	1.3.1, 1.4.1, 5.1.1, 5.2.1	1.1.3,2.1.2
3IT4-06.3	***Classify*** and ***demonstrate*** the various Inheritance techniques.	3,4	1.3.1, 1.4.1, 2.1.1,2.1.2,2.1.3, 2.2.1, 2.2.2, 5.1.1, 5.2.1	1.1.3,2.1.2
3IT4-06.4	**Understand** how to ***apply*** polymorphism techniques on the object oriented problem.	3	1.3.1,1.4.1, 2.1.1,2.1.2,2.1.3, 2.2.1, 2.2.2, 5.1.1, 5.2.1	1.1.3,2.1.2
3IT4-06.5	***Summarize*** the exception handling mechanism, file handling techniques and ***Use*** of generic programming in Object oriented programming	5,3	1.3.1,1.4.1, 2.1.1,2.1.2,2.1.3, 2.2.1, 2.2.2, 5.1.1, 5.2.1	1.1.3,2.1.2

Swami Keshvanand Institute of Technology, Management &Gramothan, Jaipur

CO-PO/PSO Mapping: Formulation and Justification

The CO-PO/PSO mapping is based on the correlation of course outcome (CO) with Program Outcome Indicators. These indicators are the breakup statements of broad Program Outcome statements.

The correlation is calculated as the number of correlated indicators of a PO/PSO mapped with CO divided by total indicators of a PO/PSO. The calculated value represents the correlation level between a CO & PO/PSO. Detailed formulation and mathematical representation can be seen below in equation 1:

Input: CO_i: *The i^{th} course outcome of the course*

PO_j: *The j^{th} Program Outcome*

Ij_k: *The k^{th} indicator of the j^{th} Program Outcome*

$\langle (I_{jk}, CO_i)$: *level of CO-PO mapping*

$$= 1, \text{ if, } 0 < \langle < 0.33$$

$$= 2, \text{ if, } 0.33 \geq \langle < 0.66$$

$$= 3, \text{ if, } 0.66 \geq \langle < 1$$

$$\alpha\left(I_{jk}, CO_i\right) = \frac{count(\lambda(I_{jk}, CO_i))}{count\left(I_k, PO_j\right)}$$

□: *Degree of correlation*

Swami Keshvanand Institute of Technology, Management &Gramothan, Jaipur

CO-PO/PSO Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO 1	PSO 2	PSO 3
CO1	2				2								2		
CO2	2				2								2	2	
CO3	2	2			2								2	2	
CO4	2	2			2								2	2	
CO5	2	2			2								2	2	

Name of Faculty:-Ms.Shalini Singhal

(Signature)

Name of Faculty:-Ms.Dolly Mittal

(Signature)

Verified by Course Coordinator

Signature
(Name: ……..)

Verified by Verification and Validation Committee, DPAQIC

Signature
(Name: ……....)

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Teaching-Learning Methodology

1. Topics will be covered essentially through plenty of examples. Lecture classes are conducted as lecture-cum-tutorial classes.
2. It is a course that aims to develop programming skills. It is therefore "practical" in orientation. Plenty of exercises of various kinds will be done by the students both inside and outside the class-room.
3. The teacher is not depending on a single or a set of two or three text books. He has chosen his materials from diverse sources.
4. Use of Power Point Presentations, hand written course notes, video lectures will be done to communicate with students.
5. Various programming assignments will be given to students.
6. Live demonstration of execution of programs will be shown online by screen sharing.

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

RTU Papers (Previous Years)

3E1139

Roll No. _____ Total No of Pages: 3

3E1139

B. Tech. III - Sem. (Main / Back) Exam., Dec. 2019
PCC Computer Science & Engineering
3CS4-06 Object Oriented Programming
Common For CS, IT

Time: 3 Hours **Maximum Marks: 120**

Instructions to Candidates:

Attempt all ten questions from Part A, five questions out of seven questions from Part B and four questions out of five from Part C.

Schematic diagrams must be shown wherever necessary. Any data you feel missing may suitably be assumed and stated clearly. Units of quantities used /calculated must be stated clearly.

Use of following supporting material is permitted during examination. (Mentioned in form No. 205)

1. NIL 2. NIL

PART – A

(Answer should be given up to 25 words only) **[10×2=20]**

All questions are compulsory

Q.1 What is dynamic binding? How it is useful in OOP?

Q.2 List a few areas of application of OOP technology.

Q.3 What is dynamic initialization of a variable? Give example of C++.

Q.4 What is the difference between following two statements:

(a) Char * const i;

(b) Char const * i;

Swami Keshvanand Institute of Technology, Management &
Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Q.5 List some properties of constructor.

Q.6 Explain Importance of destructors.

Q.7 What is operator overloading?

Q.8 What is abstract class?

Q.9 Explain generic programming and its implementation in C++.

Q.10 What is exception and how can we handle it.

PART – B

(Analytical/Problem solving questions) **[5×8=40]**

Attempt any five questions

Q.1 Write a function using reference variables to swap the values of two integers.

Q.2 Define a class in C++ to represent a Bank account. Take Assumptions for Data & Member functions.

Q.3 Explain constructor overloading with suitable example.

Q.4 Derive a class from a base class which is having a pure virtual function. Write C++ code.

Q.5 Write a Program to copy contents of one text file to other.

Q.6 Write a program to show the application of multiple catch statements.

Q.7 What is friend function? Write code to explain it.

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

PART – C

(Descriptive/Analytical/Problem Solving/Design Questions) **[4×15=60]**

Attempt any four questions

Q.1 What is Multiple Inheritance? How Ambiguities can be resolved in Multiple Inheritance. Show by suitable code.

Q.2 Define a class to create, update & manage online shopping list of a customer. Take your own Assumptions for Data & Member Functions.

Q.3 Write a C++ Program to overload '+' operator for string concatenation.

Q.4 With the help of suitable code explain Runtime Polymorphism.

Q.5 Prepare a Swap Function Containing template arguments to swap the content of two variables of type "int" , "char" & "float" .

Swami Keshvanand Institute of Technology, Management &
Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

3E1139

Roll No.

Total No of Pages: 4

3E1139

B. Tech. III - Sem. (Main) Exam., Dec. - 2018
PCC Computer Science & Engineering
3CS4 – 06 Object Oriented Programming
CS, IT

Time: 3 Hours **Maximum Marks: 120**

Instructions to Candidates:

Attempt all ten questions from Part A, selecting five questions from Part B and four questions from Part C.

Schematic diagrams must be shown wherever necessary. Any data you feel missing may suitably be assumed and stated clearly. Units of quantities used /calculated must be stated clearly.

Use of following supporting material is permitted during examination. (Mentioned in form No. 205)

1. NIL

2. NIL

PART – A

(Answer should be given up to 25 words only) **[10×2=20]**

All questions are compulsory

Q.1 What is copy constructor? Explain with suitable example.

Q.2 What is the purpose of using overloading operators?

Q.3 What is the output of the program

```
#include<iostream.h>
void main()
{
 int n=1;
 cout<<endl<<"The numbers are;"<<endl;
  do
  {
    cout<<n<<"\t";
    n++;
  } while (n<=100);
  cout<<endl;
}
```

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Q.4 Name the operators which cannot overload. What is the significance of scope resolution operators (::)?

Q.5 What are the differences between class and structure in terms of memory allocation?

Q.6 What are the differences between member function of a class and friend function?

Q.7 What is Run-Time Error, Logical Error and Syntax Error?

Q.8 Write a C++ program to implement pure virtual function.

Q.9 What is the role of public, private and protected access specifiers within the class?

Q.10 Write a C++ program using function template to swap two values.

PART – B

(Analytical/Problem solving questions) **[5×8=40]**

Attempt any five questions

Q.1 Write short notes on the following: [4×2=8]

(a) Dynamic Binding

(b) Abstract Class

(c) New & delete

(d) Function Overriding

Q.2 Design a class having the constructor and destructor functions that should display the number of objects being created or destroyed of class type. [8]

Q.3 Write a template function that returns the average of all the elements of an array. The arguments to the function should be the array name and the size of the array (type int). In main (), exercise the function with arrays of type int, long, double, and char. [8]

Q.4 Create a time class that includes integer member values for hours, minutes and seconds. Make a member function get_time() that gets a time value from the user, and a function put_time() that displays a time in 12:59:59 format. Add error checking to the get_time() function to minimize user mistakes. This function should request hours, minutes and seconds separately, and should check that the range should be in between 0 and 23, and minutes and seconds between 0 and 59. [8]

Swami Keshvanand Institute of Technology, Management &
Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Q.5 Write an inline function, factorial (int x), which returns the factorial of value x. Test the function by reading values from the keyboard. [8]

Q.6 Use for loops to construct a program that displays a pyramid on the screen. The pyramid should look like this [8]

1

121

12321

1234321

123454321

Q.7 What do you mean by Programming Paradigm? Describe the concept of message passing in Object Oriented Programming. [8]

PART – C

(Descriptive/Analytical/Problem Solving/Design Questions) **[4×15=60]**

Attempt any four questions

Q.1 Write a C++ program which will implement '= =' binary operators. The overload class should contain one parameterized constructor to initialize default value to the member variable, and a display function to display value of member variables.

Q.2 Write C++ Program for the following statements.

(a) What do you mean by inheritance? Implement multilevel inheritance with suitable example.

(b) Write a C++ Program which will implement static function, friend function and member function of a class.

Q.3 Create a Class named "student" that contains roll_number, stu_name and course_name, father_name, DOB as data member and Input_student and display_student as member functions. Create A derived class named "exam" from the class named "student" with publicly inherited mode. The derived class contains members as mark1, mark2, mark3 as marks of three subjects and input_marks and display_results as member functions. Create an array of object of the "exam" class and display the result of 10 students.

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Q.4 Handle the following exceptions to input two integers from keyboard to perform division operation.

(a) A try block to throw an exception when a wrong type of data is entered.

(b) When division by zero occurs.

Write appropriate catch block to handle above exceptions which are thrown from "thrown block".

Q.5 Write a C++ program using file handling to perform the following operations.

(a) A file named 'data.txt' contains number from 0 to 100. Open the 'data.txt' file in read mode.

(b) Read data from 'data.txt' file.

(c) If the number is odd, open a new file named 'odd.txt' and write the odd number into 'odd.txt' file.

(d) If the number is even, open a new file named 'even.txt' and write the even number into 'even.txt' file.

(e) Display the data of all the files.

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

3E1654

Roll No. _____ [Total No. of Pages : 3]

3E1654

B.Tech. III Semester (Main/Back) Examination, Dec.- 2016
Computer Sc. & Engg.
3CS5A Object Oriented Programming
EE,EX,CS,IT

Time : 3 Hours **Maximum Marks : 80**
Min. Passing Marks : 26

Instructions to Candidates:

*Attempt any **five** questions, selecting one question from **each unit**. All questions carry equal marks. (Schematic diagrams must be shown wherever necessary. Any data you feel missing suitably be assumed and stated clearly. Units of quantities used/calculated must be stated clearly.*

Unit - I

1. a) What are the difference between homogeneous and heterogeneous data type? What are the features of structure? **(8)**

   b) Write any program to pass the structure to a function. Explain each and every step in detail. **(8)**

OR

1. Create a structure to specify data of customers, in a bank. The data to be stored is: Account number, Name, Balance in account. Assume maximum of 20 customers in the bank.

   a) Write a function to print the account number and name of each customer with balance below Rs. 100.

   b) If a customer request for withdrawal or deposit, it is given in the form :

   Acct. no, amount, code (1 for deposit, 0 for withdrawal) write a program to give a message, "The balance is insufficent for specified withdrawal". **(16)**

Unit - II

2. a) How do structures in C and C++ differ? **(8)**

   b) Explain container class and proxy classes in detail. **(8)**

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

OR

2. a) Describe the mechanism of accessing data members and member functions in the following case :

   i) Inside the main program.

   ii) Inside the member function of the same class. **(10)**

   b) What are object? How are they created? **(6)**

Unit - III

3. a) What is operator overloading? Why is it necessary to overload an operator? **(8)**

   b) Differentiate unary and binary operators. **(8)**

OR

3. a) What is an operator function? Describe the syntax of an operator function. **(8)**

   b) When is a friend function compulsory? Give an example with detail. **(8)**

Unit - IV

4. a) What is inheritance? What are the different types of inheritance? Give an example of each. **(10)**

   b) Explain the concept of base class and derived class. **(6)**

OR

4. a) When do we make a abstract class? Explain in detail. **(8)**

   b) Describe how an object of a class that contains object of other classes created? **(8)**

Unit - V

5. a) What is a virtual base class? Explain. **(8)**

   b) What are the difference between error and exception? Explain the keywords used in exception handling. **(8)**

OR

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

5. Write short note on : (any **two**)

i) Templates

ii) Pointer to classes

iii) Multiple inheritance.

(8×2=16)

+ + + +

30

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Mid Term Papers

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur
First Mid Term Examination, 2020-21

B.Tech./ Semester –III | **Branch: IT**
Subject: OOP | **Subject Code : 3IT4-06**
Time: 105 Minutes(90+15) | **Maximum Marks : 50**

Read the following instructions:

- ✓ Attempt all questions.
- ✓ Each question is of 10 marks.

1. Explain Programming Paradigms with relevant examples.
2. What is 'this' pointer? Write a C++ program to illustrate use of 'this' pointer.
3. Explain friend function and its characteristics in detail. Write a C++ program to calculate simple interest by friend function.
4. Write a program to overload * (multiplication) operator in C++.
5. Write a C++ program to compute area of rectangle and square using function overloading.

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur
First Mid Term Examination, 2020-21

B.Tech./ Semester –III | **Branch: IT**
Subject: OOP | **Subject Code : 3IT4-06**
Time: 105 Minutes(90+15) | **Maximum Marks : 50**

Read the following instructions:

- ✓ Attempt all questions.
- ✓ Each question is of 10 marks.

1. Explain characteristics of OOP with reference to C++.
2. Explain static member function and static data members. Write a program in C++ which displays the number of objects created and destroyed.
3. Explain operator overloading in detail. List the operators that cannot be overloaded. How many arguments are required to overload unary and binary operators by member function and friend function? What are the restrictions and limitations to operator overloading?
4. What are Constant data member and member function in C++? Explain with suitable example.
5. Write a C++ program to calculate of a mean of any 4 numbers by using friend class.

32

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur
Extra Mid Term Examination, 2020-21

B.Tech./ Semester –III | **Branch: IT**
Subject: OOP | **Subject Code : 3IT4-06**
Time: 105 Minutes(90+15) | **Maximum Marks : 50**

Read the following instructions:

- ✓ Attempt all questions.
- ✓ Each question is of 10 marks.

1. Differentiate between following with suitable examples.
   a) C++ class and C++ structures
   b) C++ Structures and C structures.

2. Explain the concept of friend function. Write a program to swap private data member of two different classes.

3. Explain the following with suitable C++ programs as example.
   a) New and delete operators.
   b) Destructor

4. Write a program to overload % (Modulus) Operator in C++.

5. Write a C++ Program to find factorial of a number by using constructors in C++.

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur
First Mid Term Examination, 2020-21

B.Tech./ Semester –III **Branch: IT**
Subject: OOP **Subject Code : 3IT4-06**
Time: 105 Minutes(90+15) **Maximum Marks : 50**

Read the following instructions:

- ✓ Attempt all questions.
- ✓ Each question is of 10 marks.

1. Write short note on:-
   a) Difference between C and C++ Programming.
   b) Access Specifiers public and private .

2. Explain following with suitable C++ programs as example.
   i) Function overloading
   ii) Constructors and its types.

3. Write a Program to overload binary minus Operator in C++.

4. What is inline Function in C++? Explain with appropriate example.

5. Write a C++ program to add the complex numbers by using constructors in C++.

Swami Keshvanand Institute of Technology, Management &
Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur

II Mid Term Examination, 2020-21

B.Tech./ Semester –III | **Branch: IT**

Subject: OOP | **Subject Code : 3IT4-06**

Time: 105 Minutes(90+15) | **Maximum Marks : 50**

Read the following instructions:

- ✓ Attempt all questions.
- ✓ Each question is of 10 marks.

1. What is inheritance? What are the different types of inheritance? Explain each with proper example.
2. Write a C++ program to create a function template for swapping two variables of type "int", "char", and "double".
3. Explain run time polymorphism with suitable programming example.
4. Write a C++ program to show the application of multiple catch statement.
5. Write a C++ program to count number of vowels and consonants present in a file.

35

SWAMI KESHVANAND INSTITUTE OF TECHNOLOGY MANAGEMENT AND GRAMOTHAN, RAMNAGARIA, JAGATPURA, JAIPUR - 302017

Branch : Information Technology **II B.Tech. III SEM 2020-21**

Award List

SUB OOP **1st Mid Term Test** **MM.** 30

SR. NO.	EXAM ROLL NO.	STUDENT'S NAME	MARKS OBTAINED		
			Mid Term Marks 24	Assignment Marks 06	Total 30
1	19ESKIT001	AASHISH JAIN	19	06	25
2	19ESKIT002	AAYUSH JAIN	18	05	23
3	19ESKIT003	ABDUL RAHMAN	20	00	20
4	19ESKIT004	ADITYA BAGHELA	21	06	27
5	19ESKIT005	AKSHAJ AGARWAL	19	05	24
6	19ESKIT007	AMAN JAIN	19	00	19
7	19ESKIT008	AMAN PORWAL	19	05	24
8	19ESKIT009	ANKIT SAINI	18	05	23
9	19ESKIT010	ANOSH FIELD	19	06	25
10	19ESKIT011	ANUJ SHARMA	19	06	25
11	19ESKIT012	ANURAG KASHYAP	19	05	24
12	19ESKIT013	ANUSH UPADHYAY	20	06	26
13	19ESKIT014	APOORVA KHANDELWAL	22	06	28
14	19ESKIT015	ARCHIKA	18	05	23
15	19ESKIT016	ARIHANT JAIN	18	06	24
16	19ESKIT017	ARPITA DUBEY	20	06	26
17	19ESKIT018	ARYAN SHARMA	19	06	25
18	19ESKIT019	ATISHAY HARSOLA	20	06	26
19	19ESKIT020	ATISHAY JAIN	17	05	22
20	19ESKIT021	BHAVYA MATHUR	18	05	23
21	19ESKIT022	DARSHAN PARSOLIYA	18	00	18
22	19ESKIT023	DEVKINANDAN SHARMA	19	06	25
23	19ESKIT024	DHRUV RAJ NARUKA	19	00	19
24	19ESKIT025	DIVYANSH SHARMA	16	05	21
25	19ESKIT026	GAURAV JANGID	17	00	17
26	19ESKIT027	HARSH JAIN	20	06	26
27	19ESKIT028	HARSH SHARMA	21	05	26
28	19ESKIT029	HARSHIT RAJAN	20	06	26
29	19ESKIT030	HARSHIT SUKHWAL	19	05	24

Name of Examiner Dolly Mittal

Head of Deptt.
SKIT/Section IT-A
28/10/20

Signature of Examiner

(36)

SWAMI KESHVANAND INSTITUTE OF TECHNOLOGY MANAGEMENT AND GRAMOTHAN, RAMNAGARIA, JAGATPURA, JAIPUR - 302017

Branch : Information Technology **II B.Tech. III SEM 2020-21**

Award List

SUB OOP **2nd Mid Term Test** **MM.** 30

SR. NO.	EXAM ROLL NO.	STUDENT'S NAME	MARKS OBTAINED		
			Mid Term Marks 24	Assignment Marks 06	Total 30
1	19ESKIT001	AASHISH JAIN	18	06	24
2	19ESKIT002	AAYUSH JAIN	18	06	24
3	19ESKIT003	ABDUL RAHMAN	16	NS	16
4	19ESKIT004	ADITYA BAGHELA	19	05	24
5	19ESKIT005	AKSHAJ AGARWAL	18	05	23
6	19ESKIT007	AMAN JAIN	13	NS	13
7	19ESKIT008	AMAN PORWAL	21	05	26
8	19ESKIT009	ANKIT SAINI	14	06	20
9	19ESKIT010	ANOSH FIELD	18	06	24
10	19ESKIT011	ANUJ SHARMA	18	NS	18
11	19ESKIT012	ANURAG KASHYAP	16	06	22
12	19ESKIT013	ANUSH UPADHYAY	16	05	21
13	19ESKIT014	APOORVA KHANDELWAL	20	06	26
14	19ESKIT015	ARCHIKA	22	06	28
15	19ESKIT016	ARIHANT JAIN	16	05	21
16	19ESKIT017	ARPITA DUBEY	20	06	26
17	19ESKIT018	ARYAN SHARMA	19	NS	19
18	19ESKIT019	ATISHAY HARSOLA	22	06	28
19	19ESKIT020	ATISHAY JAIN	19	05	24
20	19ESKIT021	BHAVYA MATHUR	14	NS	14
21	19ESKIT022	DARSHAN PARSOLIYA	13	NS	13
22	19ESKIT023	DEVKINANDAN SHARMA	16	06	22
23	19ESKIT024	DHRUV RAJ NARUKA	18	06	24
24	19ESKIT025	DIVYANSH SHARMA	10	NS	10
25	19ESKIT026	GAURAV JANGID	20	NS	26
26	19ESKIT027	HARSH JAIN	18	06	24
27	19ESKIT028	HARSH SHARMA	16	06	22
28	19ESKIT029	HARSHIT RAJAN	19	06	25
29	19ESKIT030	HARSHIT SUKHWAL	18	05	23

Name of Examiner Dolly Mittal

Head of Deptt.
SKIT/Section IT-A

Signature of Examiner

35

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Assignments

SWAMI KESHVANAND INSTITUTE OF TECHNOLOGY MANAGEMENT & GRAMOTHAN, JAIPUR

Session 2020-21

Assignment - I

B. Tech (IT), III Sem

Branch: IIT **Subject:** OOP (3IT4-06)

Last Date of submission: 21-September-2020 (Monday)

PART-A

Q.1 Write a program in C++ to overload binary minus operator '-' by friend function.
Q.2 What is operator function? Describe the syntax of operator function.
Q.3 Explain dynamic initialization of variables and objects.
Q.4 What is 'this' pointer? Write a C++ program to illustrate use of 'this' pointer.
Q.5 What is a copy constructor? Explain with example.
Q.6 What will be the output of the following C++ code?

```
1   #include <iostream>
2   using namespace std;
3   int fun(int m= 5, int n =
10) 4 {
5     int c;
6     c = m + n;
7     return c;
8   }
9   int main()
10  {
11    cout << fun();
12    return 0;
13  }
```

Q.7 Differentiate between
a) C++ class and C++ structures
b) C++ Structures and C structures.
Q.8 Write the output of the following C++ code with proper explanation?

```
1   #include <iostream>
2   using namespace std;
3   int main()
4   {
5     int a = 9, b = 20;
6     int &aref = a;
7     aref = b;
8     a++;
9     cout << "The value of a is " << aref;
10    return 0;
11  }
```

Q.9 What is pointer to objects. Write a program to implement it.
Q.10 Write a C++ program to swap two numbers using pass by reference.

PART-B

Q.11 Explain characteristics of OOP with reference to C++.

Q.12 Explain friend function and its characteristics in detail. Write a C++ program to calculate simple interest by friend function.

Q.13 Explain the following with suitable C++ programs as example.
a) new and delete operators.
b) Destructor

37

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Q.14 Write a C++ programs to overload '==' by member function taking two data members of integer type.

Q.15 Define a class in C++ to represent Bank account. The data to be stored is account no, name, balance in account. Assume maximum of 10 customers in bank.

a) Write a member function to print name and account number of the customers with balance below Rs. 100.
b) Write member functions for withdrawal and deposit
c) Write other member functions as required.

OR

Q.15 Define a class in C++ to create, update and manage online shopping list of customers. Take assumptions for data and member functions.

PART-C

Q.16 Explain static member function and static data members. Write a program in C++ which displays the number of objects created and destroyed.

Q.17 Explain inline functions in detail. Differentiate between user defined functions and inline functions. Write a C++ program, in which the class has an integer data member, an inline member function max () which calculates maximum of two objects values by passing and returning object, and prints it. Also, there must be external inline member functions to get and print the data.

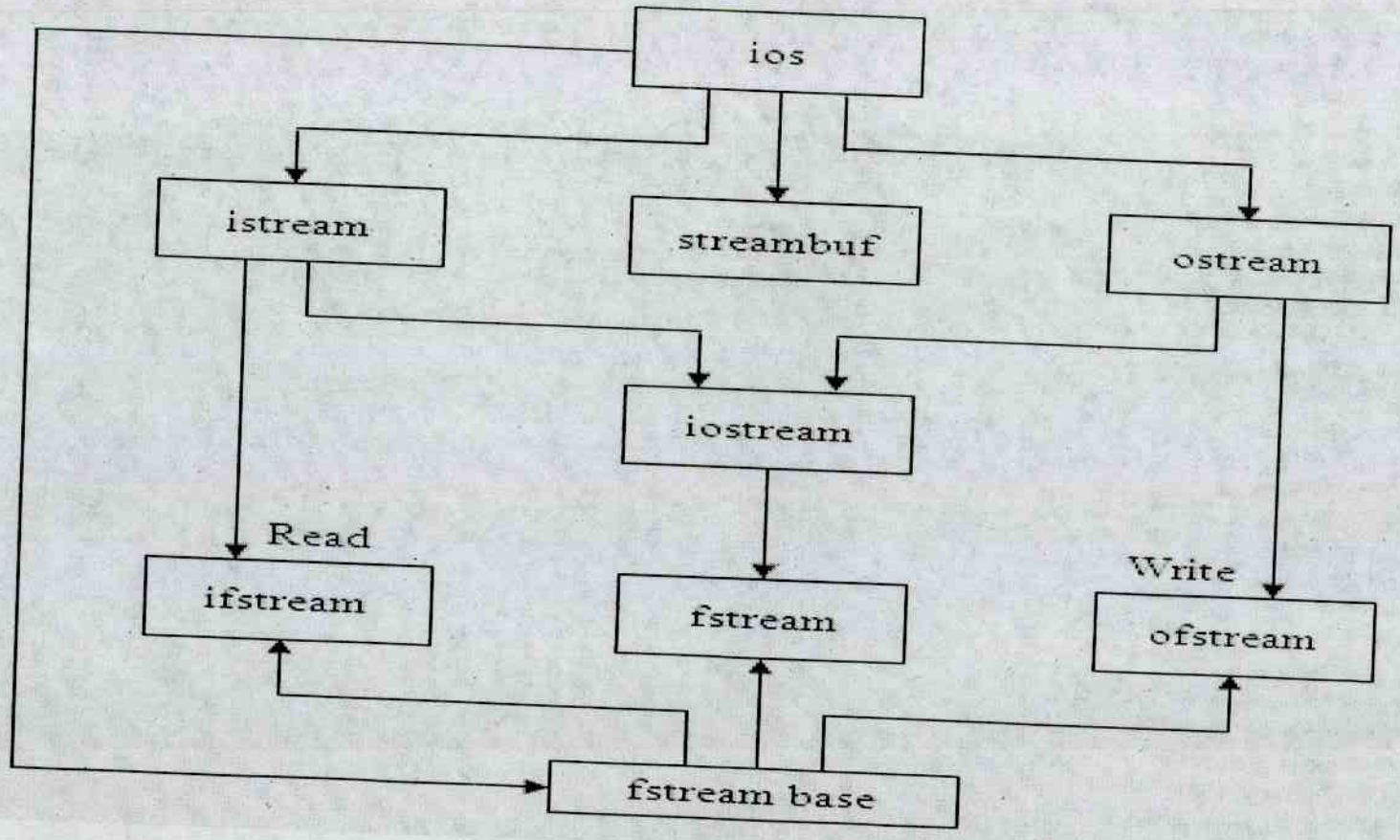
Q.18 Explain operator overloading in detail. List the operators that cannot be overloaded. How many arguments are required to overload unary and binary operators by member function and friend function? What are the restrictions and limitations to operator overloading?
Write a C++ program to overload += operator for a class named complex having two objects to represent two complex numbers.

Q.19 Write short notes on following with short C++ programs as example.
i) Function overloading
ii) Constructors and its types.

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

COURSE NOTES

landling


ios
istream
streambuf
ostream
iostream
Read
ifstream
fstream
Write
ofstream
fstream base

Unit -1

Topics:

1. structures in C
2. Accessing members of structures using structure variables
3. Array in structure
4. Array of srtucture
5. Passing structures to functions,
6. Pointer of structure
7. Pointer to structures
8. Structures as user defined data types.

Structure in c

- Structure is a user defined data type ,is a collection of items of different types.
- It is a method for packing data of different types.
- The main use of structures is to conveniently treat such collection as a unit.
- Keyword 'struct' is used to declare structure.
- The variables which are declared inside the structure are called as 'members of structure'.

Syntax:

```
struct structure_nm
{
<data-type> element 1;
<data-type> element 2;
- - - - - - - - - - -
- - - - - - - - - - -
<data-type> element n;
}struct_var;
```

Example-

```
struct book_bank
{
char title[20];
    char author[15];
```

```
    int pages;
float price;
};
```

Here book_bank is the name of structure and is called the **structure tag.**

The tag name such as book_bank can be used to declare structure variable of its type.

Accessing member of structure using structure variable

Structure members can be accessed using member operator '.' . It is also called as 'dot operator' or 'period operator'.

```
struct book_bank
{
char title[20];
    char author[15];
    int pages;
float price;
}book1,book2;
```

Or

```
Struct book_bank  book1,book2;
```

It declares book1, book2 as structure variables.

```
main()
{
book1.pages=250;
book1.price=750;
strcpy(Book1.author,"bala");
strcpy(book1.title,"OOPs");
}
```

We can use scanf

```
scanf("%s",book1.title);
scanf("%s",book1.author);
scanf("%d",&book1.pages);
scanf("%f",&book1.price);
```

Array in Structure:

- We can use array as a structure member.
- To access that array with structure variable we use the syntax:

  Structure variable.array name[index];

Syntex:

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

struct result
{
int rno, mrks[5];
```

```
char nm;
}res;

void main()
{
int i,total;
clrscr();
total = 0;
printf("\n\t Enter Roll Number : ");
scanf("%d",&res.rno);
printf("\n\t Enter Marks of 3 Subjects : ");
for(i=0;i<3;i++)
{
scanf("%d",&res.mrks[i]);
total = total + res.mrks[i];
}
printf("\n\n\t Roll Number : %d",res.rno);
printf("\n\n\t Marks are :");
for(i=0;i<3;i++)
{
printf(" %d",res.mrks[i]);
}
printf("\n\n\t Total is : %d",total);
getch();
}
```

Array of Structure:

we can also define an array of any structure for multiple instance of structure.

Syntax:

```
struct structure_nm
```

{
<data-type> element 1;
<data-type> element 2;
- - - - - - - - - - -
- - - - - - - - - - -
<data-type> element n;
}struct_var[size];

For accessing member of structure with array :
Syntax:
Var[index].member;

Example:

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

struct emp_info
{
int emp_id;
char nm[50];
}emp[2];

void main()
{
int i;
clrscr();
for(i=0;i<2;i++)
{
printf("\n\n\t Enter Employee ID : ");
```

```
scanf("%d",&emp[i].emp_id);
printf("\n\n\t Employee Name : ");
scanf("%s",emp[i].nm);
}
for(i=0;i<2;i++)
{
printf("\n\t Employee ID : %d",emp[i].emp_id);
printf("\n\t Employee Name : %s",emp[i].nm);
}
getch();
}
```

Passing structure to a function:

- A structure variable can also be a argument of function.
- A structure can be passed to function by two ways

1.Pass by value

2.pass by reference

Prototype of function

Return type func_name(structure_name);

Program:

```
#include <stdio.h>

struct struct_type {
```

```
int a, b;
 char ch;
} ;


void f1(struct struct_type parm)
{
 printf("%d", parm.a);
}


int main(void)
{
 struct struct_type arg;

 arg.a = 1000;

 f1(arg);

 return 0;
}
```

in this prog function f1 is using structure as argument,and structure is pass by value.so the changes done in function does not reflect in main

Pointer and structure:

A pointer variable can declare for structure by using syntax

struct structure_ name *variable name;

to access the member of structure by ponter variable we use

-> Operator in the place of (.)

Syntax:

Pointer variable->member;

Example:

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

struct somestruct{
int i;
};

main(){
struct somestruct *ssp, s_item;

s_item.i = 1;
ssp = &s_item;
ssp->i += 2;

}
```

Pointer to structure:

Structure cannot contain an instance of its own type, it can can contain a pointer to another structure of its own type, or itself. This is because a pointer to a structure is not itself a structure, but merely a variable that holds the address of a structure.

A pointer to a structure type variable is declared by a statement

struct Structure_name *pointer name;

Structures as user defined data types.:

typedef can also be used with structures.Using structure declaration by keyword typedef we can use structure by as user define data type

```
typedef struct gun
{
char name[50];
int magazinesize;
float calibre;
} agun;
```

here typedef creates a new type agun which is of type struct gun and can be initialised as

```
agun arnies={"Uzi",30,7};
```

when we using typedef, the tag of structure is optional like

```
typedef struct
{
char name[50];
int magazinesize;
float calibre;
} agun;
```

This type declaration in also work.

Unit - 2

Topics:

1. Introduction to programming paradigms-
   A) Process oriented
   B) Object oriented
2. Concept of object, class
3. Objects as variables of class data type
4. Difference in structures and class in terms of access to members
5. private and public members of a class(data & function members.)
6. Characteristics of OOP
   A) Data hiding
   B) Encapsulation
   C) Data security.
7. Basics of C++:
   A) Structure of C++ programs
   B) Introduction to defining member functions within and outside a class
   C) Declaring class

D) Creating objects

E) Constructors & destructor functions

F) Dangers of returning reference to a private data member,

G) Constant objects and members function

H) Friend functions and classes

I) *this* pointer

8. Static class members

20

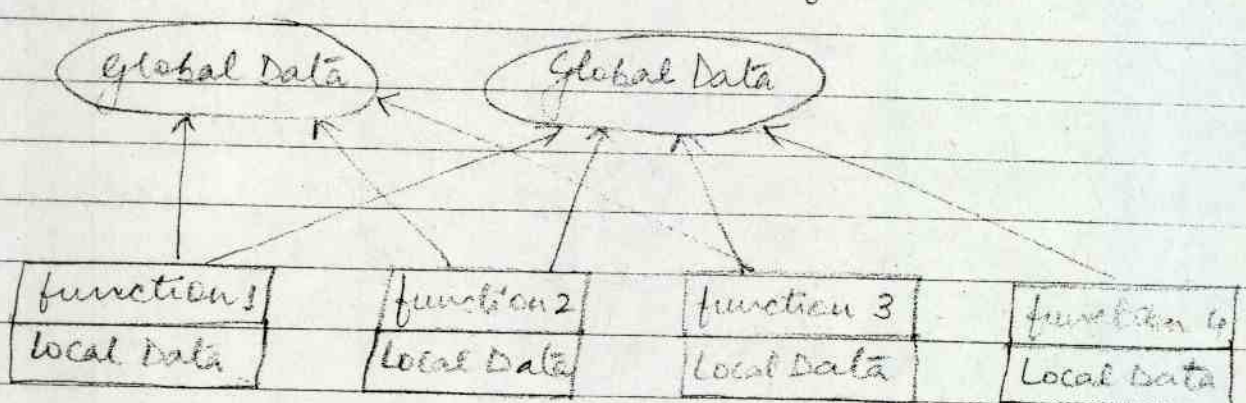
Why do we need Object-Oriented Programming?
(Programming approach)
OOP paradigm was developed because limitations were discovered in earlier approaches to programming.

Procedure-Oriented Programming / Procedural Languages.
Conventional programming languages like Pascal, Fortran, C, BASIC, Cobol etc are the procedural languages.
In procedural programming paradigm, a program is a list of instructions, which tells the computer to do something eg, get some input, add the numbers, divide by 5, display the output. Emphasis is on algorithm (way of solving a problem)
When programs become larger, they are divided into smaller units called functions (as in C) (also referred as subroutines, subprograms or procedures in other languages) where each function carries out a specific task.

Problems with Procedural Programming-
1. Data undervalued - The emphasis is on doing things, action. The subdivision of a program into functions continues this emphasis. Functions are also action-oriented. What happens to the data in this paradigm? The data is given second-class status here, when data is, after all, the reason for a program's existence.
Secondly, in a multifunction program, many important data items are defined as global and are accessible to all the functions without restriction. It has reduced data security and integrity, as any function can change any data without any check.

...dly, in this programming paradigm, datatypes are used and ...erked upon by many functions. If we make any change ...o a datatype, eg if we add new data items, all the ...unctions that access this data will also have to be ...odified so that they can also access these new items.

The Procedural Paradigm



Relationship to the real world. – Procedural programming does not model real world very well as it is just concerned about the procedure (or doing things)
For instance, a vehicle is an object, which is capable of moving in real world, however this paradigm would just think of moving part and not the vehicle that it's a car, bike, truck or cycle.

New data types – Computer languages have several built-in ata types like integers, floating-point, characters etc, but we want to invent our own datatype. eg. if we ...ant to work with complex nos., or 2-D co-ordinates or ...ates — quantities, the built in data types don't handle

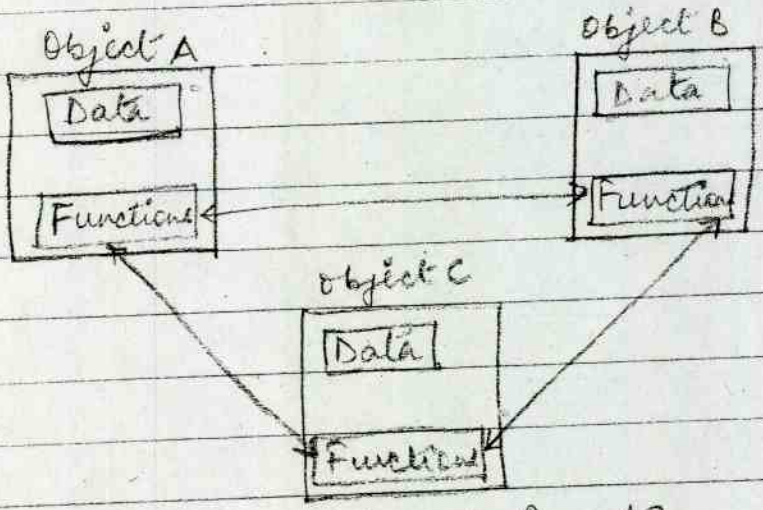
sily. Using procedural languages you can't bundle together both x and y coordinates into a single variable called 'point', and then add and subtract values of this type.

Object-Oriented Programming Paradigm-

OOP emphasizes on the data rather than the algorithm. It treats data as critical and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it and protect it from accidental modifications from outside functions, as the data is hidden.

It combines into a single unit both data and the functions that operate on that data. This unit is called as an Object. Data & functions are said to be encapsulated into a single entity.

C++ program, consists of objects, which communicate with each other by calling one another's member functions.



Object-Oriented Paradigm

tures of OOP are:

Emphasis is on data rather than procedure.

Programs are divided into objects

Functions that operate on the data of an object are tied together.

Data is hidden and cannot be directly accessed by external functions.

Constructing reusable software components and easily extendable libraries.

Modeling the real world problem as close as possible to the user's perspective.

asic elements / concepts of Object-Oriented Languages:—

Objects

Classes

Data Abstraction & Encapsulation

Inheritance

Polymorphism

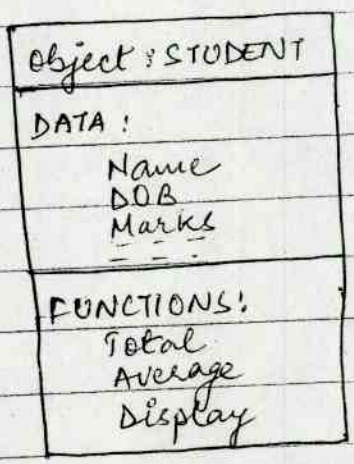
Dynamic Binding

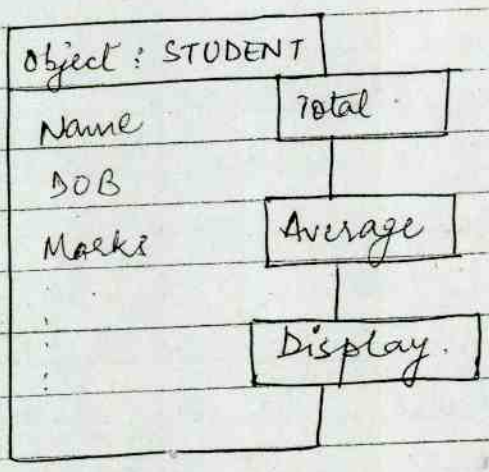
Message Passing.

ects - Objects are the basic run-time entities in an ject-oriented system. They may represent a person, a place, thing, a bank account or any item which a computer ust deal.

jects may correspond to real-world entities such as student, mployees, bank accounts, inventory items, etc or computer ardware like a keyboard, mouse, video etc and software

components like stacks, queues, tree, etc. They may be concrete like a bank account, a car or conceptual like linked list, time, angles etc.
Every object take up space in the memory and has an associated address like a structure in C.
Every object has some **attributes** (or characteristics) called as DATA and **behaviour** called as functions (or operations). Eg if take object account, its attributes are acc no., acctype, balance etc & operations can be deposit, withdraw, enquire etc.
When a program is executed, the objects interact by sending messages to one another. For eg. if 'customer' and 'account' are two objects in a program, then the customer object may send a message to the account object requesting for the bank balance.
Each object contains data and code (or functions) to manipulate the data. Objects interact with each other through these functions. They need not know the details of each other's data or code.





Representation of an object.

ss – A class is a collection of objects of similar
pe. eg mango, orange, apple are members of class fruit.
e entire set of data & code of an object can be made
user-defined data type with the help of a template
lled class. Objects are variables of type class.
ce a class has been defined, we can create any
umber of objects belonging to that class. [as 'int' is a
ata type in C & C++ & we can declare many variables of
ype int in the program.]

```
class account
{
private:
    char name[20];
    int accno;          // data members
    float balance;
public:
    Deposit();
    Withdraw();         // members functions
    Enquire();
};
account savings_acc;
account current_acc;
account Fd_acc;
```

Class template.

Objects of class account

capsulation & Data Abstraction: The wrapping up of data
d functions into a single unit (called class) is known
encapsulation. The data cannot be accessed directly.
you want to read data item in an object, you call a
mber function in the object. It will read the item and

given the value to you. This insulation of the data from direct access by the program is called data-hiding. The data is hidden, so it is safe from accidental alteration.

While encapsulating data & functions, abstraction is implement ie, only relevant details are exposed and rest are hidden

Encapsulation is a way to implement data abstraction. For eg., in a company, different departments have their own data. One dept. cannot access data of other dept. directly. Rather a request is made for the required data which is then handed over by the members of the requested dept. Thus, we can say that dept. data & dept. employees are encapsulated into a single entity: department.

Abstraction refers to the act of representing essential features without including the background details or explanations.

eg. In a switchboard, we only switch it on or off, without knowing whatts happening inside, how the wiring is done, how the current is passing etc. This is abstraction.

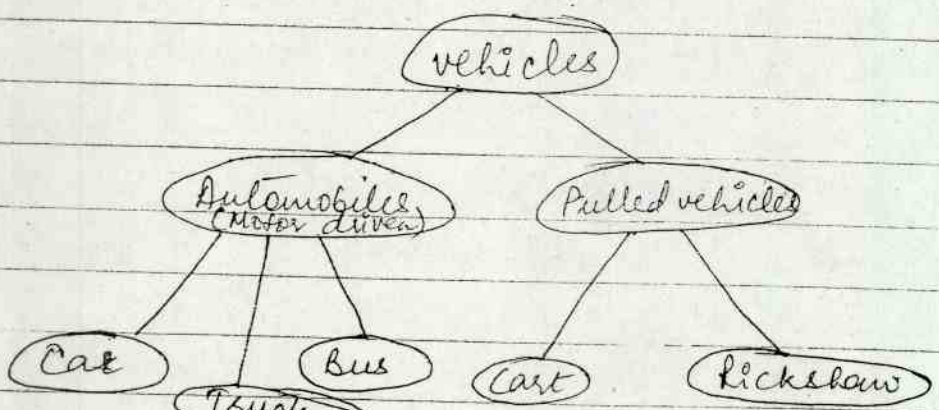
Classes use the concept of abstraction, and are also known as Abstract Data Types (ADT).

Inheritance - It is the capability of one class of things to inherit / acquire capabilities from another class.

A class of vehicles can be divided into automobiles (Motor driven) and Pulled vehicles.

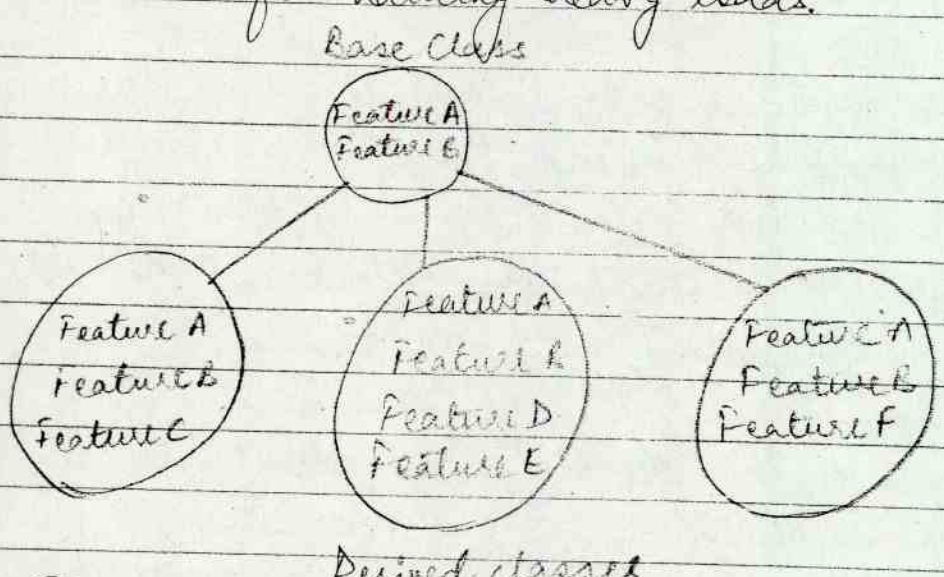
The class whose properties are inherited, is called base class (super class) and the class that inherits these

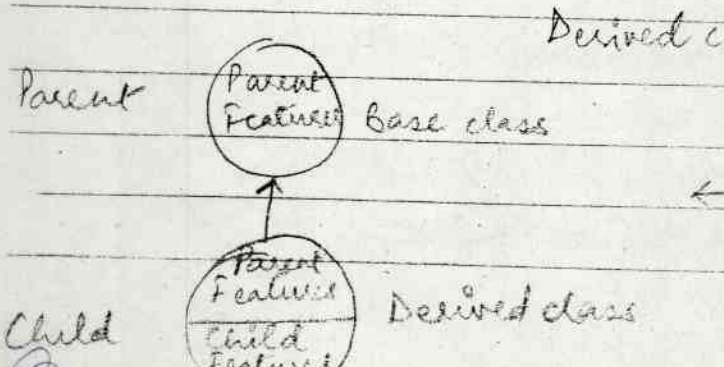
operties is called Derived class (Sub class)

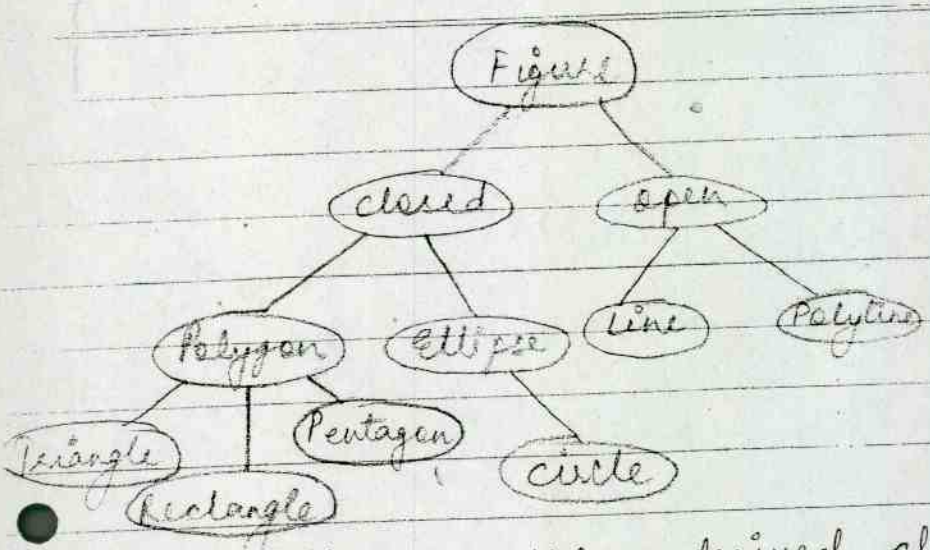


Car, Bus are subclasses of Automobiles which in turn is a subclass of vehicles.

Derived class inherits the features of its base class but each derived class also has its own particular characteristics. Eg car & bus both are motor driven but bus has seats for many people whereas car has few seats truck is used for hauling heavy loads.



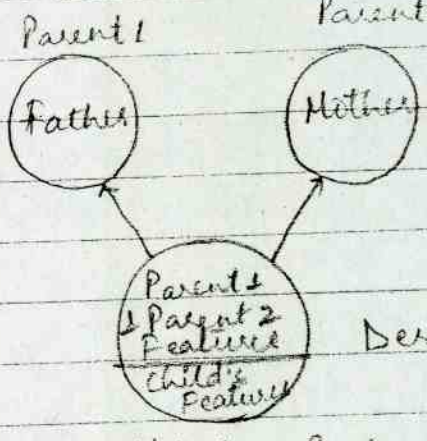




Class hierarchy

Single Inheritance → when derived class inherits properties from only one base class.

Multiple Inheritance → The derived class inherits the features of more than one base class.



Multiple Inheritance

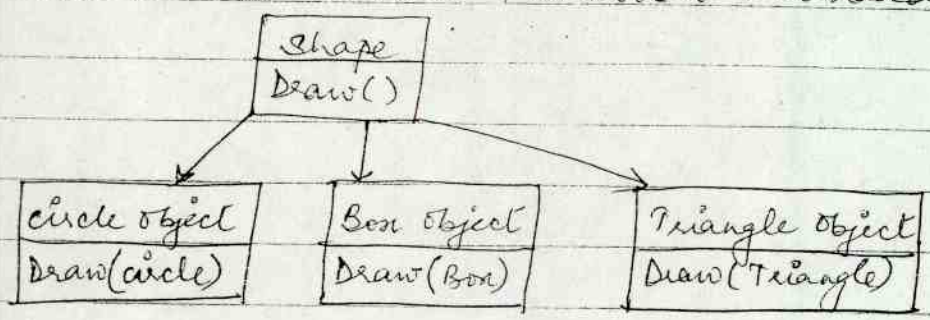
Benefits of inheritance:

Reusability – A programmer can take an existing class, and without modifying it, add additional features and capabilities to it.

The inherited code does not have to be rewritten. Thus such reusable codes increase reliability, decrease maintenance cost, reduce development time and more concentration can be given to the new portion of the software.

morphism - It means the ability to take more than form. The meaning of an operation varies with content e behaviours depends on the type of data used in the ration. Eg. if '+' operator is used on nos., it'll give m sum and if used on 2 strings, it'll concatenate 2 2 strings. This process of making an operator to exhibit ferent behaviours in different instances is known as erator overloading.

milarly a single function name can be used to form different tasks, depending on the types of arguments used to it. This is called Function Overloading.



namic Binding (Late Binding) - Binding refers to the inking of a function call to the code to be executed response to that call.

ynamic Binding means that the code associated with function call is not known until its call at run-time. e selection of the appropriate function is done dynamically run time.

nsidering the function Draw, by inheritance, every object ll have this function. Its algorithm is different for each ect, so, the draw function will be redefined in each ass. At run-time, the code matching the object under rent reference will be called

Message Passing - Objects communicate with one another by sending & receiving message/information much the same way as in conventional languages. a function is invoked on a piece of data.

A message for an object is interpreted as a request for the execution of a function. A suitable function is invoked soon after receiving the message & the desired results are general message passing involves specifying the name of the object the name of the function(message) and the information to be sent.

Eg. student.marks(rollno);

Object Message Information

C++ - C++ is an object-oriented programming languag developed by Bjarne Stroustrup at AT&T Bell laboratories, USA the early 1980's.

C++ is an extension of C with a major addition of the class construct feature.

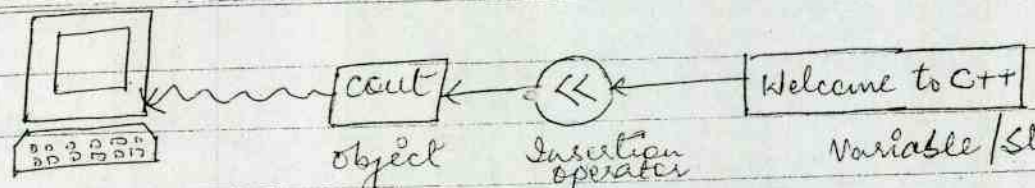
Stroustrup initially called it 'C with classes'. Later the name was changed to C++, ++ comes from the Increment operator ++ of C, hence C++ is an incremented version of C. C++ is a superset of C. Except a few differences, all C programs are also C++ programs.

Simple C++ program

```
#include <iostream.h>
void main()
{
    cout << "Welcome to C++";      //C++ statement
}
```

gram features

e execution begins from the fx^n main(). There should be actly one main fx^n. Default return type of main is int.
e body of a fx^n is surrounded by braces.
++ is a free-form language ie C++ compiler ignores white space most completely.
++ statements terminate with semicolons.
eprocessor Directives: The prepocessor directive # include lls the compiler to insert another file into your source de before compiling. iostream is a header file which contains claration for the identifier cout and the operator <<
mments: Comments start with a double slash symbol (//) and terminate at the end of the line. // is basically a single line comment.
/* */ comments are also valid and are multiline comments.
Output using cout: cout << "Welcome to C++"; causes the string in quotation marks to be displayed on the screen. The identifier cout is actually a predefined object that represents the standard output stream in C++. Here the standard output stream represents the screen.
The operator << is called 'insertion' or 'put to' operator It directs the contents of the variables on its right to the object on its left.

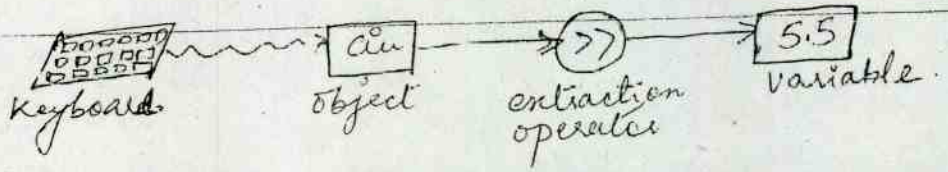


<< → is the bitwise left-shift operator & it can also be used to direct output. This is known as operator overload that is an operator can perform different activities, depending on the context.

Prog. for average of 2 nos.

```
#include <iostream.h>
void main()
{
  float n1, n2, sum, avg;
  cout << " Enter 2 nos";
  cin >> n1;          // Reading nos from keyboard.
  cin >> n2;
  sum = n1+n2;
  avg = sum/2;
  cout << " Sum = " << sum << " \n";
  cout << " Average = " << avg << " \n";
}
```

→ Input with cin: cin >> n1 causes the program to wait for the user to type in a no. & the no. entered is placed in variable n1. The identifier cin is a predefined object in C++ that that corresponds to the standard input stream. Here it is the keyboard. >> is known as 'extraction' or 'get from' opr. It extracts the value from the keyboard and assigns ~~the~~ it to the variable on its right.



or >>

cading of I/O operators - Multiple use of << in one statement called cascading.

cout << "Sum = " << sum << "\n"; → first sends the tring "Sum = " to cout and then sends the value of sum. he last 2 statements can be combined as:

cout << "Sum = " << sum << "\n" << "Average = "
<< avg << "\n";

cout << "Sum = " << sum << " , " << "Average = "
<< avg << "\n";

↳ o/p → Sum = 14 , Average = 7.

opr can also be cascaded as:

cin >> n1 >> n2; → The values are signed from left to right ie. if 2 values 5 & 10 are tered 5 will be assigned to n1 & 10 to n2.

cape Sequences → \n , \t , \v , \b → are escape equences as the backslash causes an escape from the rmal way characters are interpreted.

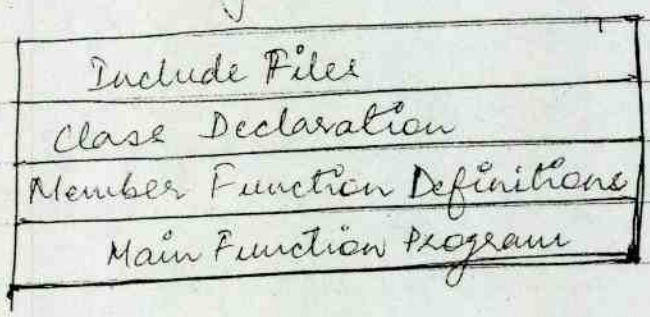
E]: cout and << operator knows how to treat an integer nd a string differently. If we send them a string, they int it as text. If we send them an integer, they print it a no. (whereas in C, using printf(), the type of variable so has to be defined using format specifiers like %d, %f etc).

E]: Unlike C, in C++, we can define variables throughout program, not just at the beginning.

formatt free input & output are special features of C++.

```
#include <iostream.h>
void main()
{ int ftemp;
  cout << "Enter temperature in 'F";
  cin >> ftemp;
  int ctemp = (ftemp - 32) * 5/9;
  cout << "Equivalent in Celcius is: " << ctemp << "\n";
}
```

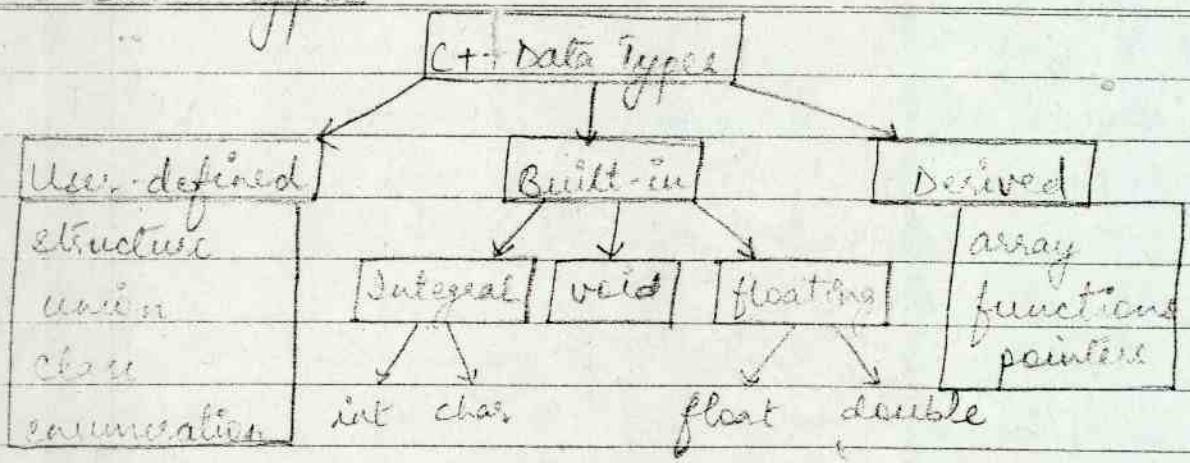
Structure of a C++ Program



Tokens - Smallest individual units in a program are known as tokens.

- →Keywords → C++ reserved words, around 63 keywords.
- →Identifiers → Names of variables, functions, arrays, classes etc created by the programmer.
  Rules: * Only alphabets, digits & underscores are permitted.
  * It cannot start with a digit.
  * Uppercase & lowercase letters are distinct.
  * Keyword cannot be used as a variable name
  * C++ places no limit on the length of the variable name.
- →Constants
- →Strings
- →Operators

ic Data Types



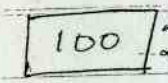
mbolic constants:

const qualifier – In C++ we can use const in a constant expression. eg. const int size = 10;
char name[size]; → This is illegal in C.
#define will also create constants.

riable Declaration – C++ allows the declaration of a variable anywhere in the scope, unlike C.

ference Variables – A reference variable provides an alias (alternative name) for a previously defined variable.
If a variable 'sum' is made a reference variable of variable 'total', then 'sum's 'total' can be used interchangeably
It's created as: datatype & ref-var = variablename
eg. int total = 100;
int & sum = total;
both the variables refer to the same data object in the memory.

cout << sum; } both will print value 100.
cout << total



one location, 2 names.

total = total + 10;

↳ will change the value of both sum & total to 110

NOTE: A reference variable must be initialized to some variable at the time of declaration.

```
int &x = y[100];   // x is an alias of y[100] element
int &n = 100       // invalid as no alias for constant value.
```

```
int x;
int *p = &x;
int &m = *p;   → m refers to x
```

→ Major application of reference variables is in passing arguments to fn.

```
void main()
{ int m=10;
  f(m);
  ---
}
void f(int &x)
{ x = x+10;   // x as well as m is incremented.
}
```

When f(m) is executed it results in int &x = m ∴ x is an alias of m. This is call by reference.

In C it is done using pointers.

NOTE :- references can be created not only for built-in data types but also for users-defined data-types such as structures & classes.

ators in C++ - Unary, Binary, Ternary

Input - Output operators -

Output op^r → << (Insertion op^r)

Input op^r → >> (extraction op^r)

Arithmetic op^rs - + , - , * , / , %

Relational op^rs - < , > , <= , >= == !=

Logical op^rs - && , || , !

Designment op^rs. - =

Shorthand assignment op^r → += -= *= %= /=

Increment/Decrement op^r ++ --

Conditional op^r ?= (Ternary op^r)

Bitwise op^r & | ^ ~ << >>

Special op^rs - sizeof() , comma op^r.

Scope resolution operator ::

In C, the global variable cannot be accessed from within the inner block. C++ resolves this by the op^r ::

:: var name → allows access to the global variable that has been hidden by another variable of the same name in the local scope.

```
#include <iostream.h>
int m = 10;        <-- global m
void main()
{
  int m = 20;      <-- m local to main
  {
    int m = 30;
    cout << "m = " << m << "\n";
    cout << "::m = " << ::m;
  }
}
```

O/P → m = 30

::m = 10

Its application is in the classes to identify the class to which a member function belongs.

Memory management op^{r} – For dynamically allocating & deallocating memory, new and delete op^{r} are used. An object can be created using new & destroyed using delete. Data object will remain in existence until it is explicitly destroyed using delete.

Syntax:

ptr_var = new data_type;

ptr_var is a pointer of type data_type. new allocates memory to hold data object of type data_type and returns the address of the object.

eg. int *p;
p = new int;
*p = 10;

or float *q = new float
*q = 7.5;

Memory can be initialized using new op^{r}, like

int *p = new int(10);
float *q = new float(7.5);

Also, int *p = new int[10]; → It creates memory space for an array of 10 integers.

p[0] will refer to 1st element
p[1] " " " 2nd "
p[2] " " " 3rd "

When a data object is no longer needed, it is destroyed to release the memory space for reuse.

delete ptr_var.

eg. delete p;
delete q;
delete [size] ptr_var; ← free dynamically allocated array.

also, delete [] p; will delete the entire array.

sufficient memory is not available, new returns a
ll pointer.

new is better than malloc() as:
It automatically computes the size of data object, whereas
in malloc sizeof operator is required.
It automatically returns the correct pointer type, no
need to use a type cast.
Possible to initialize the object while creating the memory.
new & delete can be overloaded.

Manipulators - for formatting the data display.
↳ endl & setw
endl is same as newline character '\n'.

Eg cout << "Hello" << endl;
cout << "HRV" << endl;

I/P → Hello
HRV.

I/O — setw (fieldwidth); → The output can be right aligned and fieldwidth can be specified using setw.

Eg. void main()
{
int B=100, A=80, T=1050;
cout << setw(5) << B << endl;
cout << setw(5) << A << endl;
cout << setw(5) << T << endl;
}

O/P →

		1	0	0
			8	0
	1	0	5	0

Type Conversion

1) Implicit conversion: We can mix data types in expressions. C++, like C converts one of them to match with the other, using the rule that the 'smaller' type is converted to the 'wider' type automatically. If one of the operand is int & the other float, int is converted into float.

2) Explicit / Type Casting: Data type can be converted explicitly by the programmer by type casting.

(type_name) expression // C notation

type_name (expression) // C++ notation.

Operator precedence & associativity: Unary operators assume higher precedence, while :: (scope resolution op) having the highest precedence.

Control structures 1) Sequential
2) Branching
3) Looping / Iteration.

Inline functions - Function execution involves the overhead of jumping to the fn, saving registers, pushing arguments into the stack, & returning to the calling fn. When a fn is small, a lot of time may be spent in such overheads. To eliminate the cost of calls to small functions, C++ proposes 'inline fn'. An inline function is a function that is expanded in line when it is invoked. The compiler replaces the fn call with the fn code (similar to macro expansion).
Inline fns are defined in the same way as ordinary fns except

If the keyword 'inline' precedes the fn definition

```
inline int sqr(int num)
{
    return (num * num);
}
void main()
{
    :
    a = sqr(5);      ⟶ a = 5*5;
    :                  expanded to.
}
```

usually, the fns are made inline when they are small enough to be defined in one or two lines.

Inline keyword just sends a request, not a command to the compiler. Compiler may ignore the request if the fn definition is too long & compile the fn as normal fn.

Default arguments - C++ allows to call a fn without specifying all its arguments. In such cases, the fn assigns default value to the parameter which does not have a matching argument in the fn call. Default values are specified when the fn is declared.

```
float amt(float P, int T, float rate = 0.15);
```

It declares default value of 0.15 to the argument rate.

If fn call, `val = amt(5000, 5);` is made, the default value of 0.15 is automatically passed as 3rd argument the arguments specified in fn call override the default values specified in fn declaration.

eg. `amt(5000, 5, 0.12);` here rate value will be 0.12

Only the trailing arguments can have default values & we must add defaults from right to left. We cannot provide a default value in the middle of an argument list.

eg. int mul(int i, int j=5, int k=10); → legal
int mul(int i, int j=5; int k); → illegal.

→ They are useful when some arguments always have the same value. eg. interest rate

```
void Line(char c='-', int i=70);
void main()
{ line();        ⟶ expand to  Line('-', 70);
  Line('!');     ⟶            Line('!', 70);
  Line('*', 40); ⟶            Line('*', 40);
}
```

Classes & Objects

Limitations of C structures - ① C does not allow the struc data type to be treated like built-in types.

```
struct complex
{ float x;
  float y;
};
struct complex c1, c2, c3;
```

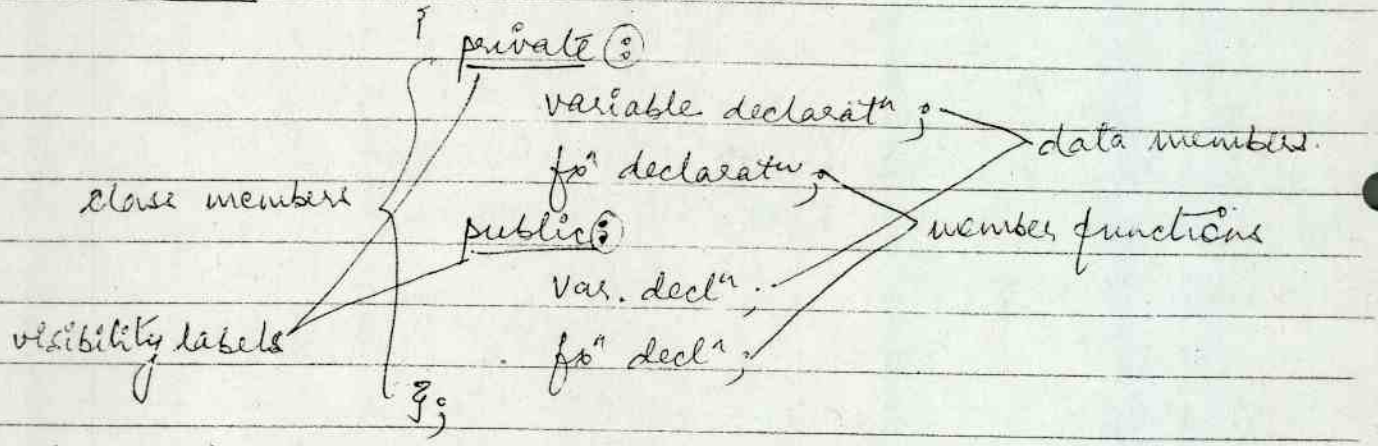
c1, c2, c3 can be assigned values using dot opr but we cannot add 2 complex nos. or subtract like, c3 = c1 + c2;

② They do not permit data hiding. Structure members can be directly accessed by structure variables by any fn

Extension to C structures - In C++, a structure can have variables & functions as members.
So some of its members can be declared private so that they can't be accessed by external fns.
Also keyword struct can be omitted in declaring structure variables. eg. student A; → C++ declarat^n.

Structures & classes are almost similar in C++ with the difference that by default members of a class are private & by default members of a structure are public.

Class - It binds the data & its associated functions together while defining a class, we create new abstract data types.
Syn declaration :- class classname



Private members can be accessed only from within the class.
Only the member fn^s can have access to private data members and private fn^s. By default the members of a class are private.
Public members can be accessed from outside the class.

```
class student
{ int roll_no;
  int cd_id;
public:
  void getdata (int rn, int cd);
  void putdata (void);
};
```

} private by default

student is now a new data type and instances/objects of this class type can be declared.

Creating Objects -

```
student S1;   // memory is created for S1
```

S1 is a variable of type student and is called as object.

```
student S1, S2, S3;
```

also vars/objects can be created in the following way also:

```
class student
{ - -
  - -
  - -
} S1, S2, S3;
```

Accessing Class Members - Private data of a class can be accessed only through the member fns of that class.

A fn can be called as: obj_name . fn_name (actual args);

eg. S1.getdata (60, 1001); → It assigns 60 to roll_no & 1001 to cd_id.

A member fn can be invoked only by using an object.

∴ getdata (60, 1001) is incorrect.

also S1. roll_no = 60 is also illegal as roll_no is declared private & can be accessed only through mem. fn.

variable declared as public can be accessed by the fcts directly.

```
class abc
{ int x;
  int y;
public :
  int z;
};
abc p;
p.x = 5;   ← error, as x is private.
p.z = 10;  ← ok, as z is public
```

defining member fcns: Member fcns can be defined in 2 places:
* outside the class definition
* inside " " "

outside the class def^n - The fcn has an 'identity label' to the fcn header which tells the compiler which class the fcn belongs to.

```
return-type class_name :: fcn_name (argument declarat$^n$)
{
   fcn body
}
```

```
void student :: getdata (int a, float b)
{
  roll_no = a;
  col_id = b;
}
void student :: putdata (void)
{ cout << "Roll no : " << roll_no;
  cout << " College id : " << col_id;
```

Different classes can use the same fn name. The membership label will resolve their scope.
A member fn can call another member fn directly, without using the dot op.

Inside the class def^n

```
class student
{ int roll-no;
  int col-id;
public:
  void getdata(int a, int b);
  void putdata(void)
  { cout << roll-no;
    cout << col-id;
  }
};
```

When a fn is defined inside a class, it is treated as an inline fn. Normally only small fns are defined inside the class def^n.

Making an outside fn inline - only by adding 'inline' in the fn header

```
inline void student :: getdata(int a, int b)
{
  - - -
}
```

...ting of member fn^s — A member fn^n can be called by ...ing its name inside another member fn^n of the same class.

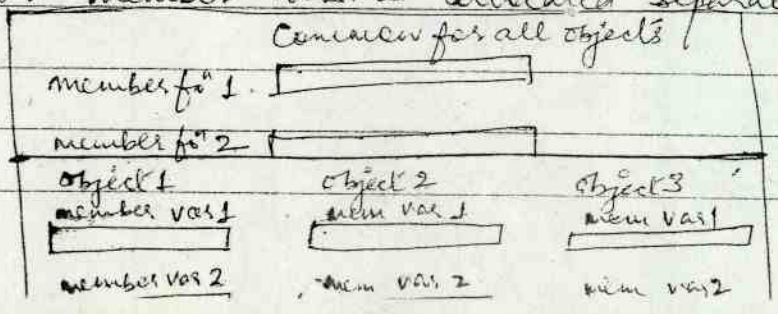
...ivate member fn^s — A private member fn^n can only be ...lled by another member fn^n of its class. Even an object ...annot invoke a private fn^n using dot op^r.

eg. class sample
{
int m;
void read();
public;
void update(void);
void write(void);
};
s1.read() won't work

...owever, void sample::update(void)
{
read();
} will work.

...mory Allocation for Objects — The member fn^s are created ...nd placed in the memory only once when they are defined as a part of class. Since all the objects of the class ...se the same member fn^s, no separate space is allocated ...r member fn^s when objects are created.

Only space for member vars. is allocated separately for each ...ject.



Arrays within a class – Arrays can be used as data members in a class.

```
eg  const int size = 10;
    class arr
    {  int a[size];
    public:
       void setval();
       void display();
    };
```

array a[7] can be used in the member fns like any other variable, and can perform various operations on it

Array of objects – An array of variables that are of type class are called arrays of objects.

```
class employee
{  char name[30];
   float age;
public:
   void getdata();
   void putdata();
};
employee manager[3]; ← array of object manager
```

manager[0], manager[1] & manager[2] are the 3 objects.

...ts as arguments:

...ke any other data type, an object can be passed as an ...gument to a fn in 2 ways:

Pass-by-value – A copy of the entire object is passed to the fn and any modifications made to the object inside the fn is not reflected in the object used to call the fn.

pass-by-reference – an address of the object is passed to the fn and any changes made to the object inside the fn reflected in the actual object. This method is more ...ficient, since only the address of the object is passed & ...t a copy of the entire object.

```
#include <iostream.h>
class time
{ int hours;
  int min;
public:
  void gettime (int h, int m)
  { hours = h; min = m; }
  void puttime ( )
  { cout << hours << " hours &" << min << "minutes" << "\n";
  void sum ( time, time);
};
void time:: sum ( time t1, time t2)
{   min = t1.min + t2.min;
    hours = min/60;
    min = min%60;
    hours = hours + t1.hours + t2.hours;
}
void main( )
{ time T1, T2, T3;
  T1.gettime (2,45);
  T3.sum (T1, T2);   // T3 = T1 + T2
  cout << " T1= "; T1.puttime ( );
  cout << " T2= "; T2.puttime( );
  cout << " T3= "; T3.puttime( );
```

T3.sum(T1,T2) invokes the fn sum() by the object T3, with objects T1 & T2 as arguments, it can directly access the hours & min variables of T3, but the members of T1 & T2 can be accessed only by using dot operator (like t1.hours & t1.min)
∴ inside the fn sum hours & min refer to T3.

Friend Functions & Friend Classes

Private members cannot be accessed from outside the class ie a non-member fn cannot access private data of a class, but sometimes there may be a situation where we would like 2 ~~fn~~ classes to share a particular fn. eg. 2 classes manager and technicians have been defined & a fn income_tax() is to operate on objects of both these classes, ∴ this fn can be made friendly with both the classes, thereby allowing this friend fn to have access to the private data of these classes.

To make an outside fn friendly to a class, it has to be declared as a 'friend' of the class only, as

```
class abc
{
  public:
    friend void xyz();   // declaration.
};
```

* The fn declaration should be preceded by the keyword friend
* The fn is defined elsewhere like a normal fn & does not use the keyword friend or :: op^r.
* a friend fn can be declared in either public or private sectio

fn can be declared as a friend in any no. of classes
It can be invoked like a normal fn, without using the object of that class.
Unlike class member fns, it cannot access the class members directly & has to use the object & the dot op with each member name.
It can be declared either in public or private part of the class.
Friendship is not mutual by default, ie. if B is declared as friend of A, this does not give A the right to access the private members of class B.
Usually it has the objects as arguments.

⇒ → A fn made friend in another class.

```
class sample
{
  int a;
  int b;
public:
  void setval()
  { a=25; b=40; }
  ~~float~~ friend float avg(sample s);
};
float avg(sample s)
{
  return((s.a+s.b)/2.0);
}
void main()
{
  sample X;
  X.setval();
  cout<<"Mean value = "<<avg(X);
}
```

friend fn accesses class vars a & b using dot op

Members fn of one class can be friend fns of another class. This is done using :: opr.

```
class X
{ - -
  int fun1();  // member fn of X
 - -
}
class Y
{ - -
  friend int X :: fun1();   // fun1() of X is friend of Y.
 - -
}
```

→ A fn friendly to two classes

```
# include <iostream.h>
class ABC;  → forward declaration
class XYZ
{ int x;
  public:
    void setval(int i)
    { x=i; }
    friend void max(XYZ, ABC);
};
class ABC
{ int a;
  public:
    void setval(int i){a=i;}
    friend void max(XYZ, ABC);
};
void max(XYZ m, ABC n)
{ if(m.x > n.a)
    cout << m.x;
  else
    cout >> n.a;
}
void main()
{ ABC abc;
  abc.setval(10);
  XYZ xyz;
  xyz.setval(20);
  max(xyz, abc);
}
```

Friend Classes – We can also declare all the member fns of one class as the friend fns of another class. In such cases, the class is called a friend class.

```
class z
{ - -
  - -
  friend class x; //class x can access private
                  //members of class z
}
```

class x is friend class of class z but it does not mean class z is friend class of class x.

Eg.
```
class alpha
{ private:
    int data1;
  public:
    setval
    alpha.() { data1 = 99; }
    friend class beta;
};
class beta
{
  public:
    void f1(alpha a) { cout << "data1" << a.data1;
    void f2(alpha a) { cout <<"\t data1" << a.data1;
};
void main()
{ alpha a;
  beta b;
  a.setval();
  b.f1(a);
  b.f2(a);
}
```

In class alpha the entire class beta is a friend, so now all the member fns of beta can access the private data of alpha.

rss by reference -

Swapping private data of classes:

```
#include <iostream.h>
class c2;
class c1
{ int val1;
  public:
    void indata (int a) {val1=a;}
    void display() {cout << val1 << "\n";
    friend void exchange(c1 &, c2 &);
};
class c2
{ int val2;
  public:
    void indata (int a) {val2=a;}
    void display() {cout << val2 << "\n";
    friend void exchange(c1 &, c2 &);
};
void exchange (c1 &x, c2 &y)
{ int temp = x.val1;      } Directly
  x.val1 = y.val2;        } modify
  y.val2 = temp;          } values of
}                           val1 & val2
void main()
{ c1 cl1;
  c2 cl2;
  cl1.indata(100);
  cl2.indata(200);
  cout<< "Values before exchange:";
  cl1.display();
  cl2.display();
  exchange (cl1, cl2);
  cout << "values after exchange";
  cl1.display();
  cl2.display();
}
```

objects x & y are aliases of cl1 & cl2 resp.

Returning Objects

fn can also return an Object

```
class complex
{
  float x;
  float y;
public:
  void input (float real, float imag)
  { x = real; y = imag; }
  friend complex sum (complex, complex);
  void show ( ){ cout << x << " +i" << y;}
};
complex sum (complex C1, complex C2)
{ complex C3;
  C3.x = C1.x + C2.x
  C3.y = C1.y + C2.y;
  return (C3);
}
void main()
{
  complex A, B, C;
  A.input (3.1, 5.6);
  B.input (5, 10);
  C = sum (A, B);        // C = A+B
  cout << "A= "; A.show();
  cout << "B = "; B.show();
  Cout << "C = "; C.show();
}
```

program adds 2 complex nos A & B to produce third complex no. C

Static Data Members -

The members of a class may be qualified as static. There may be static data members & static member functions

static data members are similar to C static vars.

* It is initialized to zero when the first object of its class is created.
* Only one copy of this data member is created for the entire class and is shared by all the objects of that class.
* It is visible only within the class, but its lifetime (the time for which it remains in the memory) is the entire program.

⇒ Static data members are normally used to maintain values common to the entire class. For eg, a class may have a static data member keeping track of the occurences of all the objects.

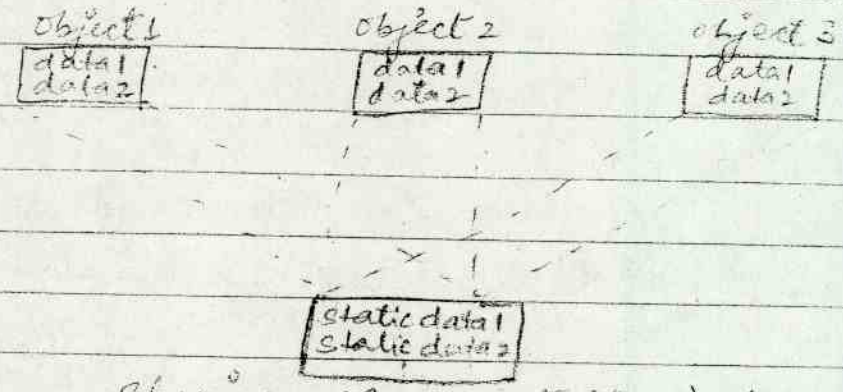
Two things are needed for making a data member static : (i) declaration within the class definition
(ii) definition outside the class definition.

```
class X
{  static int count;   // declaration
   :
};
int X :: count;        // definition.
```

type & scope of count

definition is necessary because the static data members are stored separately rather than as a part of an object. Since they are associated

th the class itself rather than any object, hey are also known as class variables.



Sharing of a static Data member

utomatic data members (of each object) are stored in istinct memory locations, whereas static data embers (of all objects) are stored in the same ocations.

E ① When an object is created, memory is not located to its static members, because this would ause multiple copies of the static data members ppear in every object.

While defining a static data member, some initial value can be assigned to it eg,

int item :: count = 10;

```
class ABC
{
  static int count;
  int no;
  public:
    void set(int a)
    {
      no = a;
      count++;
    }
```

```
void show()                    "count :"
{ cout << " No. of calls made to set :" << count << "\n";
}
}; int ABC :: count;        // definition
void main()
{ ABC ob1, ob2, ob3;
  ob1.show();
  ob2.show();
  ob3.show();
  ob1.set(100);
  ob2.set(200);
  ob3.set(300);
  ob1.show();
  ob2.show();
  ob3.show();
}
```

```
o/p →  count : 0
ob1.show();   count : 0
ob1.set(100); count : 0
ob1.show();   count : 3
ob2.set(200); count : 3
ob2.show()    count : 3
ob2.set(250);
ob3.set(300);
ob1.show();
ob2.show();
ob3.show();
```

→ count : 0
count : 1
count : 2
count : 4

Access rules for static data members

Accessibility of private static data mem is same as that of normal private members.

public static data members can be accessed by prefixing class name & scope resolution operator.

```
class test
{ public:
    static int a;
  private:
    static int b;
};
void main()
{ test :: a = 50; test myobj;
  myobj.a = 50;
```

72

59

tatic member fn's — Static fn's can access only
tatic ~~mem~~ data members declared in the same class.
non-static data are unavailable to these fn's.
it can be called using the class name :

class_name :: fn_name ;

Constructors & Destructors — Objects are initialized
hen a new fn is called but one of the main aims
of C++ is to create user-defined data types such as
lass that behave similar to built-in data types ie.
e should be able to initialize a class type var (object)
hen it is declared same as ordinary var.

Eg. int m = 20;

ondly, when a variable of built-in data type goes out
scope, compiler automatically destroys it.
Can we initialize and destroy the var. of user-defined
ata types (class) also ? Answer is Yes.

We can initialize the object by using constructor &
stroy it using destructer.

structer is a special member fn which is used to
itialize the objects data members with the values
t the time of its creation.

structor is a special member fn which destroys the objects
hen they are no longer required.

These are special because their name should be the
ame as the class name.

Constructor is invoked whenever an object of its
lass is created.

```
class user
{
  public:
    user(void);   // constructor declared
    ...
};
user :: user(void)   // constructor defined
{ m=0; n=0;
}
user u1;
```

→ object is created and is also initialized automatically.

→ Here, instead of using constructor, if a normal member fn is defined for zero initialization, this fn would have to be invoked for each of the objects separately.

Properties of a constructor -

1. A constructor name must be same as the classname
2. They should be declared in the public section.
3. They are invoked automatically when the objects are created.
4. They do not return any value (not even void)
5. Constructors are member fns so they can be overloaded and also can have default arguments.

Types of constructors -

1. Parameterized constructors
2. default "
3. default argument "
4. Copy constructors
5. dynamic "

Default constructor

constructor without any parameters is called default constructor. If no user-defined constructor is provided, then the compiler supplies a default constructor.
∴ for class A, A a invokes the default constructor of the compiler to create Object a.

Default constructor for a class A is A::A(). → This may also contain a body.

Once we define a constructor in a class, we must also define the default constructor. This will/may not do anything, but is defined just to satisfy the compiler.

```
class sample
{ public:
    sample()
    { cout << "Object is initialized" << endl;
    }
};
void main()
{ sample X, Y, Z;      → Here the default constructor
  getch();               sample is automatically executed
}                        & 3 times it displays the
                         o/p → Object is initialized, coz.
                         3 objects X, Y, & Z are created &
```

for each ~~object~~ it is executed automatically in the sequence.

Parameterized constructor

A constructor where parameters are passed is called parameterized constructor.

```
class user
{ int m,n;
  public:
    user (int x, int y);
};
user :: user (int x, int y)
{ m=x; n=y;
}
```

When parameterized constructors are used, it is necessary to pass the appropriate arguments when objects are created.

This can be done in 2 ways:

(1.) By calling the constructor explicitly

```
user u1 = user (0, 10);
```

↓ object u1 is created & 0 & 10 are passed to it

2.) By calling the constructor implicitly.

```
user u1 (0,10);
```

Eg.

```
class user
{ int m,n;
  public:
    user (int, int);
    void display()
    { cout << "m=" << m << "\n" << " n=" << n; }
};
```

```
user :: user (int x, int y)
{ m = x ; n = y; }
void main()
{
    user u1 (0, 10);
    user u2 = user (5, 20);
    cout << "Object 1 :";
    u1. display();
    cout << "Object 2 :";
    u2. display();
}
```

eg can also be defined as inline fns.

```
class user
{ int m, n;
  public;
    user (int x, int y)
    { m = x ; n = y;
    }
  -  -
  -  -
};
```

TE: The parameters of a constructor can be of any type except that of the class to which it belongs.

Eg.
```
class A
{ -  -
  public:
    A(A);
};
```
is illegal.

Multiple constructors

It is possible to have more than one constructor in a class. This is known as "overloading".

```
class dist
{ int m, cm;
public:
  dist() { }                    // default constructor
  dist (int met, int cen)
  { m = met;
    cm = cen;
  }
  dist (int met)
  { m = met;
    cout << "Enter the distance in cm: ";
    cin >> cm;
  }
  void display()
  { cout << "The distance is: ";
    cout << m << " and " << cm;
  }
};
void main()
{ int dm, dcm;
  dist X = dist (3,8);
  X. display();
  cin >> dm >> dcm;
  dist Y (dm, dcm);
  Y. display();
  dist Z (6);
  Z. display();
}
```

Default Argument Constructor

Constructor are fine & so we can use the default guments with the constructors also. The assignment ust be for the trailing arguments, ie, the order of ssignment should be from the right-most var. to the ft-most var.

3. Time(int hh, int mm, int ss=20);

For the statement, Time(2,3,30); the object will be itialized with hh=2, mm=3 and ss=30. (default alue 20 is overridden).

Time X(2,3) → hh=2, mm=3 & ss=20

Copy constructor - It initializes one object by another bject. An argument for a copy constructor is a reference an object of the same class.

yntax: (when defined inside the class)

```
class-name (classname & var)
{ body of the constructor
}
```

t Sample(sample & obj) be declared in the class ample, then to initialize the objects:

1. sample X=Y; } // define object X & at the same time
2. sample X(Y); } initialize it to the values of Y.

Here object Y is assigned to object X member by member.

TE: A constructor can accept a reference to its own class . an argument, but it cannot take parameters of the ame class type.

If we write X=Y and X & Y both are objects, then this statement will assign the values of Y to X, member-by-member.

```
class code
{ int id;
public:
   code() {}
   code (int a) { id = a; }
   code( code & x)
   { id = x.id;
   }
   void display()
   { cout << id;
   }
};
void main()
{ code A(100);
  code B(A);
  code C=A;
   D = A;
   cout << " id of A:";
   A.display();
   cout << "id of B:"; B.display();
   cout << "id of C:"; C.display();
   cout << "id of D:"; D.display();
}
```

O/P :— id of A: 100
id of B: 100
id of C: 100
id of D: 100

namic constructors (Dynamic initialization of objects)

he constructors which allocate memory at runtime e called dynamic constructors. They use new operator, s this purpose.

llocation of memory to objects at the time of their nstruction is known as dynamic construction of objects.

```
Class String
 char *name;
  int len;
public:
    String()                    // 1st constructor
    { len = 0;
      name = new char [len + 1];
    }

    String( char *s)            // 2nd constructor
    { len = strlen(s);
      name = new char [len + 1];
      strcpy (name , s);
    }

    void display()
    { cout << name << "\n";
      void join (String &a, String &b);
    };
void string :: join (String &a , string &b)
{  len = a. len + b. len;
   delete name ;
   name = new char [len+1];
   strcpy ( name , a.name);
   strcat ( name , b.name);
}
```

```
void main()
{
String name1("Welcome"), name2("To"), name3("C++"), S1, S2;
S1.join(name1, name2);
S2.join(S1, name3);
name1.display();
name2.display();
name3.display();
S1.display();
S2.display();
}
```

O/P →
Welcome
To
C++
Welcome To
Welcome To C++

Destructor - When the variables of built-in data types are not needed they are automatically destroyed by the compiler. Similarly, we can destroy the objects of the class. To do this a member fn, destructor whose name is the same as the class name but preceded by a tilde (~). Destructor is automatically called whenever an object of the class to which it belongs goes out of existence. It releases the memory occupied by the object in the hea

* It does not take any argument & does not return any va
* If object is defined inside the block (local) then destructor is called when the block gets over & if object is globally defined then constructor is called when program terminates.
* It cannot be overloaded.
* When the closing brace of a scope is encountered, the destructors for each object in the scope are called. Objects are destroyed in the reverse order of creation.

```
#include <iostream.h>
int count=0;
class alpha
{ public:
   alpha ()
   { count ++;
     cout<<" No. of object created "<< count <<"\n";
   }
   ~ alpha ()
   { cout << "No. of object destroyed " <<count;
     count --;
   }
};
void main()
{ ~~cout~~ alpha A1, A2, A3, A4;
   {
    cout << "Enter block 1";
    alpha A5;
   }
  { cout<<" Enter block 2";
    alpha A6;
  }
}
```

O/P →

```
No. of object created 1
   "    "     "    2
   "    "     "    3
   "    "     "    4
Enter block 1
No. of object created 5
No. of object destroyed 5
Enter block 2
```

```
No. of object de.. 5
No. of object des.. 4
   "    "    "   3
   "    "    "   2
   "    "    "   1
```

Unit -3

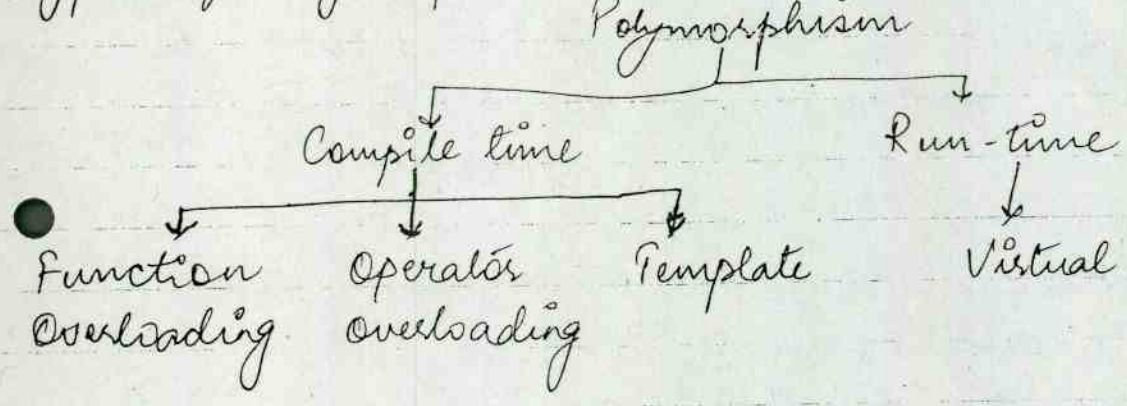
Topics:

1. Operator overloading
   - A) Fundamentals
   - B) Restrictions
   - C) Unary operators
   - D) Binary operators
   - E) Operator functions as class members v/s as friend functions
   - F) Overloading stream function
2. Converting between types

Polymorphism:

'Poly' means many and 'morphism' means forms, ∴ Polymorphism means many forms.

Types of Polymorphism



Compile time polymorphism — It is also called early binding or static binding because an object is bound to its function call at compile time.

Function overloading — Using the same function name to create functions that perform different tasks.
The compiler determines which function is used on the basis of fn's name, the number and type of parameter passed to a fn. This is known to the compiler at the compilation time

- The return type of a function is not considered for fn overloading.
- The parameter list associated with the fn function is called the function's signature.

Two fns having same no. & same type of parameter in the same order are said to have the same signatures.
Fns with same signatures and same name are not allowed in C++. even if they return different types of values.

Same signature

```
add (int a, float b)
add (int x, float y)
```

Different signature

```
add (int a, char b)
add (char b, int a)
```

```
int add (int a, float b)
float add (int x, float y)
```

```
add (int a, float b, int c)
add (int x, float y)
```

```
// Declarations
int add (int a, int b);                    // prototype 1
double add (double x, double y);           //    "      2
double add (int p; double q);              //    "      3
int add (int a, int b, int c);             //    "      4.

// function calls
cout << add (5,10);                        // uses prototype 1.
cout << add (15, 10.0);                    //  "      "      3
cout << add (12.5, 7.5);                   //  "      "      2
cout << add (5,10,15);                     //  "      "      4.
```

Rules for best matching –

A fn call first matches the prototype having the same no. and type of arguments and then calls the appropriate fn.

* best match must be unique

If a best match is not found, it applies built in conversions to find a match. This is done through promotion of the actual arguments to reconcile them with the formal arguments. Eg. int ex (float, float)

int x = ex (3,7);

Here fn^ call does not match with the prototype, so 'int' parameter is promoted to float and the fn^ is called
→ If the conversion is possible to have multiple matches, then compiler generates error.

eg if 2 fn^s : long sq(long n)
double sq(double x) are declared

A fn^ call sq(10) will cause an error as int argument (10) can be converted to either long or double, thereby creating an ambiguous situation as to which version of sq() fn^ should be used.

```
#include <iostream.h>
class volume
{
public:
  int vol(int);
  double vol(double, int);
  long vol(long, int, int);
};
int volume::vol(int s)    // cube
{ return(s*s*s); }
double volume::vol(double r, int h)    // cylinder
{ return(3.14*r*r*h); }
long volume::vol(long l, int b, int h)
{ return(l*b*h); }
void main()
{
  volume v;
  cout << v.vol(10);
  cout << v.vol(2.5,8);
  cout << v.vol(100L, 75, 15);
}
```

Operator overloading: It is a technique to define two or more distinct meanings for the existing operators. Every language supports this to some extent, as '+' opr can add 2 integers as well as 2 float nos. in C. '*' can be used for multiplication as well as value at address opr in pointers.
This overloading capacity is extended to user-defined data types such as class in C++, ie. operators which are used with built-in data types, can also be used with user-defined data types.

eg.
```
class sample
{
...
};
sample A, B, C;
C = A * B;
```

Here, * opr is used to multiply 2 objects of sample class.

Rules for operator overloading.

Semantics of an opr can be extended, but its syntax can't be changed, such as the no. of operands, precedence and associativity.
When an opr is overloaded, its original meaning is not lost. If + is overloaded to concatenate 2 strings, can still be used to add 2 integers.
While overloading, we cannot change the binary opr to unary opr & vice versa.
Only existing operator symbols can be overloaded. New operators like ∝, β, ** cannot be defined.
Opr overloading can be accomplished by member fr or friend fr.

Operators can be overloaded only in the context of users-defined data types & not built-in data types ie we cannot redefine + for int or any other built-in types- ie int operator + (int i, int j) is not allowed.

7. The operators :: . .* ?: # cannot be overloaded.

8. The overloaded opr must have at least 1 operand of users-define type.

Syntax:

returntype operator op to be overloaded (parameters);
keyword

keyword 'operator' gives information to the compiler that overloading of operator is to be carried out.

eg. int operator > (sample, sample);

Steps for overloading:

1. Create a class ie datatype to be used in overloading
2. Declare the operator fn in public part of the class
   ↳ member fn / friend fn.
3. Define the operator fn;

Invoking fns - Overloaded opr fns can be invoked by expressions such as:

op x or x op (for unary oprs)

and x op y (for binary oprs)

For unary oprs: In member fn, x op or op x is considered as x.operator op() & in friend fn it's considered as operator op(x)

For binary oprs: In member fn, x op y is interpreted as x.operator op(y) & in friend fn it's considered as operator op(x,y

Overloading unary operators

When unary op^rs are overloaded by member fo^n, it takes no formal arguments, whereas when they are overloaded by friend functions they take a single argument.

Overloading unary minus op^r – a minus op^r takes just one operand when used as a unary op^r. Unary minus when applied to an object should change the sign of each of its data items.

```
class point
{ int x, y;
public:
    point()
    { x = 0; y = 0; }
   void inputcoord();
   void display();
    void operator -();
};
void point :: inputcoord()
{ cout << "enter x & y coord";
  cin >> x >> y;
}
void point :: display()
{ cout << "Point is:";
  cout << x << " " << y << endl;
}void point :: operator -()
 { x = -x;
   y = -y;
 }
```

```
void main()
{
  point pt1;
  pt1.inputcoord();
  cout << "Pt1: ";
  pt1.display();
  -pt1;        // pt1.operator-();
  cout << "Pt1:";
  pt1.display();
}
```

NOTE: pt2 = -pt1; will not work 'coz the fn operator -() does not return any value. It can work if the fn is modified to return an object.

- can be overloaded using a friend fn also as:

```
friend void operator -(point &pt);
void operator -(point pt)
{ pt.x = -pt.x;
  pt.y = -pt.y;
}
```

The argument is passed by reference. If passed by value, it'll not work as the changes made in the operator fn will not reflect in the called object. If we want to pass by value, then to reflect the changes, the operator fn should return the value

```
friend point operator -(point pt);
point operator -(point pt)
{ point temp;
  temp.x = -pt.x;
  temp.y = -pt.y;
  return temp;
}
```

Overloading ++ pre/post increment operator -

```
class Inc
{ int val;
  public:
    Inc()
    { val=0; }
    int Getval()
    { return val; }
    void operator ++()
    { val = val + 1;
    }
};
void main()
{ Inc i1, i2;
  cout << " I1 = " << i1.Getval();
  cout << " I2 = " << i2.Getval();
  ++i1;    // equivalent to i1.operator++();
  i2++;    //          "          i2.operator++();
  cout << " I1 = " << i1.Getval();
  cout << " I2 = " << i2.Getval();
}
```

Here if we try to use `i1 = i2++;` will give an error, 'coz return type of operator++ is defined as void type.

It can be done by changing the return type as:

```
Inc operator ++()
{ Inc temp;
  temp.val = val + 1;
  return temp;
}
```

Overloading binary operators – In the overloading of binary operator, as a rule, the left hand operand is used to invoke the operator fn and the right-hand operand is passed as an argument to the operator function.

```
class complex
{ float r, i;
public:
  complex()
  { r = i = 0.0; }
void getdata()
{ cout << "Enter real part";
  cin >> r;
  cout << "enter imaginary part";
  cin >> i;
} void display()
  { cout << r << "+i" << i << endl; }
  complex operator + (complex );
};
complex complex :: operator + (complex c)
{ complex temp;
   temp.r = r + c.r;
   temp.i = i + c.i;
   return temp;
  }
void main()
{ complex c1, c2, c3;
  c1.getdata(); c2.getdata();
  c3 = c1+c2;   //equivalent to c3=c1.operator + (c2);
  c3.display();
}
```

the argument on the left side of the operator (C1 in this case) takes the responsibility of invoking the function and the ~~arg~~ argument on the right side of the operator (C2 in this case) is passed as the actual argument ~~passed~~ to the overloaded fn.

C3 = C1 + C2 is equivalent to C3 = C1.operator+(C2);

Hence, in the operator+() fn, data members of C1 are accessed directly, like r and i, and of C2 (passed as an argument) are accessed using the dot operator, like C.r & C.i.

Overloading binary operators using Friends

Friend fn may be used in place of member fns, friend fn requires 2 arguments to be explicitly passed, while a member fn requires only one.

The above prog. can be modified as;

Change the member fn prototype as:

```
friend complex operator+(complex, complex);
```

Redefine the operator fn as:

```
complex operator+(complex C1, complex C2)
{ complex temp;
  temp.r = C1.r + C2.r;
  temp.i = C1.i + C2.i;
  return temp;
}
```

Here C3 = C1 + C2 is equivalent to: C3 = operator+(C1, C2);

E: Friend fns offer the flexibility of writing an expression a combination of operands of user defined and in-built data types.

4, C3 = C1 + 2; → It is made up of object C1 and a primitive type. This will work for a member fn but if we write C3 = 2 + C1, it won't work. Because the left hand operand should be an object of the same class as it is responsible for invoking the member fn. However friend fn allows both approaches as an object need not be used to invoke a friend fn and only should be passed as an argument.

Overloading stream operators (<< and >>) using friend fn
The classes istream and ostream are defined in the header file iostream.h
istream class uses predefined stream cin to read data from standard input device. The extraction op^r >> is used for this.
ostream class uses cout to display data on standard o/p device the insertion op^r << is used for this.
→ Similar to the built-in variables, user-defined objects can also be read or displayed using >> and << operators.
Syntax for overloading << and >> op^rs:

1) friend istream & operator >> (istream &, class_type &obj)
{ }

2) friend ostream & operator << (ostream &, class_type &obj)
{ }

In both the cases, a reference to an object of the class type is taken as the second argument.

```
class complex
  private:
     float r, i;
    complex()
    { r = i = 0.0; }
    friend istream & operator >> (istream & In , complex &C);
    friend ostream & operator << (ostream &out , complex &c);
    friend complex operator + (complex c1 , complex c2);
};
istream & operator >> (istream & In , Complex &C)
{ cout << "Real part ?";
   In >> C.r;
   cout << " Imag Part ?";
   In >> C.i;
   return In;
}
ostream & operator << (ostream & Out , Complex &C)
{ out << C.r << "+i" << C.i << endl;
   return Out;
}
complex operator + (complex c1, complex c2)
{ complex c;
   c.r = c1.r + c2.r;
   c.i = c1.i + c2.i;
   return c;
}
void main()
  complex c1, c2, c3 ;
  cin >> c1 ; cin >> c2;
  c3 = c1 + c2 ; cout << "Sum = " ;
```

(63)

In the above program, cin >> c1 & cin >> c2 read the user-defined class's (complex) object c1 & c2 in the same way as built-in data type variables using >> opr. Also the sum of the complex nos. c1 & c2 is stored in c3 and displayed by cout << c3; similar to any built-in data items using << opr.

Operators that cannot be overloaded using friend function

= assignment opr
() Fn call opr
[] subscripting opr
→ Class member access.

Overloading + operator for concatenation of Strings

Strings can be defined as class objects which can be then manipulated like the built-in types.
The relational operators such as <, >, ==, etc can be overloaded to operate on strings.

```
#include <iostream.h>
#include <string.h>
class string
{
  char str[50];
 public:
  string()
  { strcpy (str, "10");
  }
  string ( char *s)
  { strcpy ( str, s);
  }
```

```
void echo()
{ cout << str << endl;
}

string operator + (string s)
{ string temp;
  strcpy (temp.str , str);
  strcat (temp.str, s.str);
  return temp;
 }
};
void main()
{ string s1 ("Welcome to");
  string s2 ("string");
  string s3;
  s3 = s1 + s2;
  s1. echo();
  s2. echo();
  s3. echo();
}
```

overloading comparision operators

```
#include <iostream.h>
enum boolean {TRUE, FALSE};
class Index
{ int value;
  public:
    Index()
    { value = 0;}
```

```
Index (int val)
{
  value = val;
}

int getIndex()
{ return value;
}

boolean operator < (Index idx)
{ return (value < idx.value ? TRUE : FALSE);
}
};
void main()
{ Index idx1 = 5;
  Index idx2 = 10;
  cout << " Index1 = " << idx1.getIndex();
  cout << " Index2 = " << idx2.GetIndex();
  if (idx1 < idx2)
    cout << "Index1 is less than Index2";
  else
    cout << "Index1 is not less than Index2";
}
```


86

Unit 4

Topics:

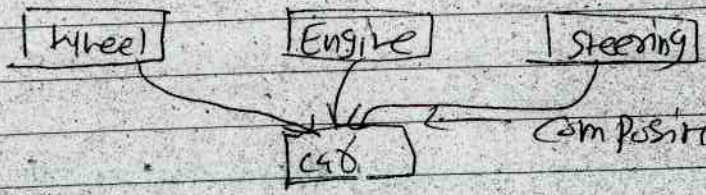
Inheritance

A) **Base classes and derived classes**
B) **Protected members**
C) **Relationship between base class and derived classes**
D) **Constructors and destructors in derived classes**
E) **Public, private and protected inheritance**
F) **Relationship among objects in an inheritance hierarchy**
G) **Aabstract classes**

87

(Generalization to specialization)

Inheritance

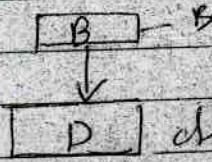


<Derived> is a type of <base> : inheritance
<new> consist of <old> ; composition

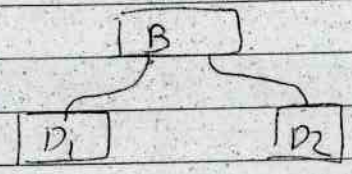
Inheritance is a concept of designing new classes using the properties of already exiting class. The new class thus created contains some or all of the properties of the older class. The new class thus created is known as derived class and older one is known as base class.

Types of Inheritance

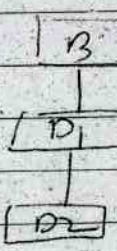
1. single Inheritance



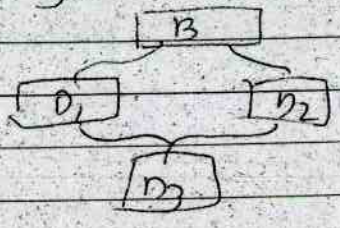
3) Heirarchical Inheritance

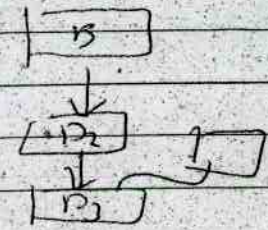


4) Multilevel Inheritance



5) Hybrid Inheritance





Declaration of derived class :-

```
class Base
{
___
___
```

class Derived : <visibility mode> <Base class>,
{
    ——
    ——
    ——
} exclusive feature of derived
};

Visibility modes:-

Private :- members declared under the private mode cannot be inherited.

Public :- members declared under this mode can be inherited

Protected :- This is a specilized mode introduced for inheritance only. This visibility mode acts as such a private for the class which can be inherited.

Derived class

class Base
{

(be inherited ... नहीं होगा)

→ Public → protected

[List]

Public :
    → private → Private
    → protected → protected
    → public → Public
};

class Base
{
    int a;
protected :
    int b;
public :
    int c;
    int d;
};

class Derived : public Base
{
    int x;
protected :
    int y;
public :
    int z;
    int f;
};
main ()
{
    Derived D;

D.b = 10; // Invalid : Protected is not accessible

b.c = 10; ✓

D.x = 10; // Invalid : private in derived

D.y = 10; // Invalid : not accessible

b.z = 10; ✓

D.i = 10; // derived member is given precedence

D.Base::i = 10;

D.fun(); // Derived

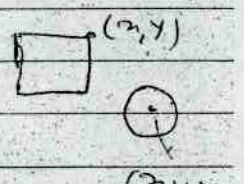
D.Base::fun(); // Base

}

Ex.

```
class employee
{
    ==
    ==
};

class manager : public employee
{
    ==
    ==
};

class worker : public employee
{
    ==
    ==
};

class ceo : public manager
{
    ==
    ==
};
```

Design a class for MS-paint

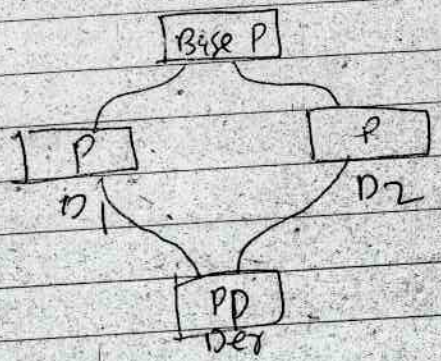



```
class shape
{
    int x,y;
    int bordercolor, borderstyle;
    int fillcolor, fillstyle;
    int area, perimeter;
    public:
        void Getco-ordinate();
        void BorderDetails();
        void GetFillDetails();
};

class Rectangle : public shape
{
```

public:
    void Draw();
    void calcArea();
};

class circle: Public shape
{
    int radius;
public:

virtual base classes :-



Der D;
D.P;

class Base
{
    ==
public:

D.D1::P; ✗
D.D2::P; ✗
since P is a member of Base class

class D1: Virtual Public Base
{
    ==
}

class D2: Virtual Public Base
{
    ==
}
Interchangable

class Der: Public D1, Public D2
{
    : public D1, D2
    ==
}

If any property of ultimate base class is transferred through various routes; to reach upto an ultimate derived class, an ambigious situation arises due to multiple copies of same property.
The problem can be resolved by making the intermediatry base classes as virtual. In that case the compiler will take necessary care to transfer only one copy of the property to the ultimate derived class.

total size = Base + Derived

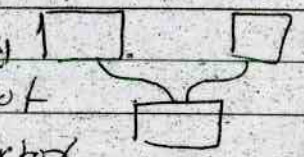
Constructors in inheritance :-

```
class Base
  {   int d;
      Protected:
        int a,b;
      public:
        int M;
  }; fun() { d=10; }

class Derived: Public Base
  {
      int x;
      public:
        int y;
  };

main()
  {
    ---
    Derived D;
    cout << sizeof(D);      10 bytes
(If int v) then cout << sizeof(D);   12 bytes
    D.fun();
```

Base class as independent object

Whenever any object of the derived class is declared, first of all a complete object of the base class is created, followed by object declaration creation of derived class. Later on both the objects gets merged up ~~to be~~ to be identified as a single object of derived class. The same sequence is followed by constructors of the respective class.

That means when an object of derived class is declared, first of all the constructors for the base class is invoked followed by constructor ~~invoking~~ invocation of derived class.



In case of multilevel inheritance the order of invocation of constructor is according to the level of inheritance.

```
class Derive : Public B1, B2
```

Whereas in case of multiple inheritance the order is same as the order in which they have listed while inheriting (i.e. B1 then B2 and then Derived)

constructor before any other base class constructor.

class Derived : Public B1, virtual Public B2
(Each virtual B2 ka constructor invoke hoga)

```
class Base
{
  Public:
    Base()
    {
      cout << "Def constructor";
    }
    Base(int n)
    {
      cout << "Single Arg. const:";
    }
    Base(int n, int m)
    {
      cout << "Two arg const.";
    }
};
class Derived : Public Base
{
  Derived()
  {
    cout << "Def const Der";
  }
  Derived(int n)
  {
    cout << "one arg. cons. of Der";
  }
  Derived(int n, int m)
  {
    cout << "Two arg Cons of
          Derived";
  }
};
main()
{
  Derived D;
  Derived D1=5;
  Derived D(5,10);
}
```

Base Default
Der : Default
Base : Default
Der : single
Base - Default
Derive_Two

```
Derived(int n): Base(n)
  {
   cout << "one arg. const. of der";
  }
  Derived(int n, int m): Base(n,m)
    {
     cout << "Two arg. cons. of derived";
    }
};
```

The responsibility of invoking the required base class constructor lies with the derived class constructor.
By default only the default constructor is invoked for the base class.

```
class sample
  {
   const int n=10;          // they can't be initialised before
     int M;                 // constructor2 data member
     int &r = M;   X        // reference variable
class sample                (the declaration of classes)
  {
   const int n;
     int M;
     int &r;
```

```
   sample(int a):
     {
      =
     }
};
```

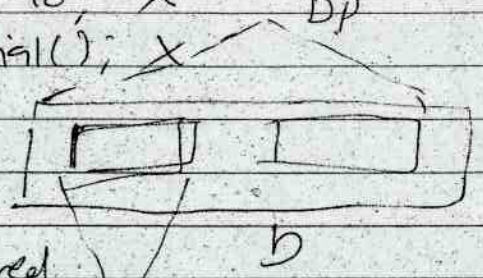
reference variable and constant data members would be initialised in the constructor's header otherwise it would be reported by the compiler as error.

Base class pointers :-

```
class Base
  {
   ==
   public:
     int m, n;
     fun();
};
class Derived : public Base
  {
   ==
   public:
     int q;
     trial();
};
```

Sample1 S1;
Sample2 *sp;
sp = &S1; X
due to

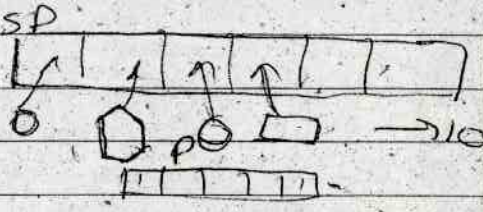
```
main()
{
    Base *BP;
    Derived D, *DP;
    BP = &D;
    DP = &D;
    DP->m = 10;
    DP->n = 10;
    DP->a = 10;
    DP->fun();
    DP->trial();
    BP->m = 10;
    BP->n = 10;
    BP->fun();
    BP->a = 10;      X
    BP->trial();     X
```



A base class pointer is full compatiable with its respective derived class object that means a base class pointer can store the address of object of any of its derived class, in the

// due to scaling factor Base class of pointer only Base class

Although Now this base class pointer can refer to only those members which has been inherited from the base class.



* The compatibility of base class pointer helps to treat the base class as the universal type for array of all of its derived class.

```
void main()
{
    cout << "Enter total number of shapes
            to designs: ";
    int n;
    cin >> n;
    Shape **sp = new Shape *[n];
    int *p = new int[n];
    for(int i=0; i<n; i++)
    {
        cout << "\n choices are:";
        cout << "\n1. Circle";
        cout << "\n2. Rectangle";
        cout << "\n3 Polygon";
        cout << "\n Enter your choice:";
```

// Array of Pointer to base class

P[i]= ch;
switch(ch)
{
case 1 : SP[i]= new circle; break;
case 2 : sp[i] = new rectangle; break;
case 3 : SP[i]= new Polygon;
}
}
for (i=0 ; i<n; i++)
{
switch (P[i])
{
case 1 : ((circle*)SP[i])->Draw(); break;
case 2 : ((Rectangle*)SP[i])->Draw(); break;
case 3 : ((Polygon*)SP[i])->Draw();
}
}
}

~~class shape~~

virtual functions :

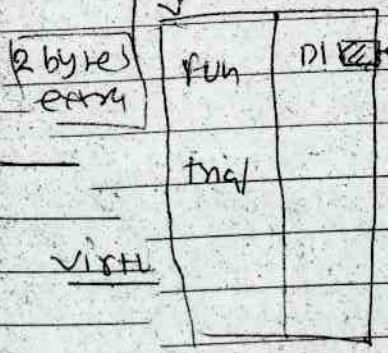
class shape
{
=
Public:
virtual void Draw();

class circle : public shape
{
=
Public:
void Draw()
{
=
}
};
class Rectangle : public shape
{
=
}
for (i=0 ; i<n; i++)
{
SP[i]->Draw(); // All ph ch
Base class
class shape
{
=
virtual void Draw() = 0; // Pure virtual fn
};
class circle : Public shape
{
=
Public:
void Draw()

Virtual table & Virtual Pointer :-

```
class Base
{
    int a;
public:
    virtual void fun()
    {
    }
    int virt();
};
main()
{
    Base B;
    cout << size of (B);   // 4 byte
}
```

virtual table [VTBL]



2 bytes are allocated to that pointer which point to the virtual table

As soon as any virtual fn is declared inside the class the CH compiler automatically creates a virtual

information about the fn belonging to the base class and details of those derived classes, which have overridden the base class methods.

Along with that a pointer is associated with the size of base class, responsible for pointing the virtual table. The virtual table also contains pointers to derived class definition responsible for overriding. Known as virtual Pointer

Unit -5

Topics:

1. **Multiple inheritance**
2. **Virtual Function**

   A)**Binding**

   B)**Virtual Destructor**

3. **Pointers to classes and class members**
4. **Virtual base classes**
5. **Templates**
6. **Exception handling.**

Multiple inheritance:

A class inherits members from more than one class ,this concept is called multiple inheritance. This is done by simply separating the different base classes with commas in the derived class declaration.

For example, if we had a specific class to print on screen (COutput) and we wanted our classes CRectangle and CTriangle to also inherit its members in addition to those of CPolygon we could write:

class CRectangle: public CPolygon, public COutput;

class CTriangle: public CPolygon, public COutput;

Example;

```
/ multiple inheritance
#include <iostream>
using namespace std;

class CPolygon {
 protected:
  int width, height;
 public:
  void set_values (int a, int b)
    { width=a; height=b;}
};
```

```
class COutput {
 public:
  void output (int i);
 };

void COutput::output (int i) {
 cout << i << endl;
 }

class CRectangle: public CPolygon, public COutput {
 public:
  int area ()
    { return (width * height); }
 };

class CTriangle: public CPolygon, public COutput {
 public:
  int area ()
    { return (width * height / 2); }
 };

int main () {
 CRectangle rect;
```

```
CTriangle trgl;
rect.set_values (4,5);
trgl.set_values (4,5);
rect.output (rect.area());
trgl.output (trgl.area());
return 0;
}
```

This may introduce naming conflicts in child class if at least two of its superclasses define properties with the same name.

To resolve naming conflictsa we use the scope resolution operator with the function name in child class.

Static binding:

When(By default), C++ matches a function call with the correct function definition at compile time. This is called static binding.

Dynamic binding: When C++ matches a function call with the correct function definition at run time, this is called dynamic binding.

How can C++ achieve dynamic binding and static buinding

When you have a pointer to an object, the object may actually be of a class that is derived from the class of the pointer (e.g., a Vehicle* that is actually pointing to a Car object; this is called "polymorphism"). Thus there are two types: the (static) type of the pointer (Vehicle, in this case), and the (dynamic) type of the pointed-to object (Car, in this case).

Static typing means that the legality of a member function invocation is checked at the earliest possible moment: by the compiler at compile time. The compiler uses the static type of the pointer to determine whether the member function invocation is legal. If the type of the pointer can handle the member function, certainly the pointed-to object can handle it as well. E.g., if Vehicle has a certain member

function, certainly Caralso has that member function since Car is a kind-of Vehicle.

Dynamic binding means that the address of the code in a member function invocation is determined at the last possible moment: based on the dynamic type of the object at run time. It is called "dynamic binding" because the binding to the code that actually gets called is accomplished dynamically (at run time). Dynamic binding is a result of virtual functions.

Virtual Function:Virtual function is a

- A member function of a class
- Declared with virtual keyword
- Usually has a different functionality in the derived class
- A function call is resolved at run-time

Non-virtual member functions are resolved statically. That is, the member function is selected statically (at compile-time) based on the type of the pointer (or reference) to the object.

In contrast, virtual member functions are resolved dynamically (at run-time). That is, the member function is selected dynamically (at run-time) based on the type of the object, not the type of the pointer/reference to that object. This is called "dynamic binding.

```
class Window // Base class for C++ virtual function example
{
public:
  virtual void Create() // virtual function for C++ virtual function example
  {
    cout <<"Base class Window"<<endl;
  }
};
```

```
class CommandButton : public Window
{
  public:
    void Create()
    {
      cout<<"Derived class Command Button - Overridden
C++ virtual function"<<endl;
    }
};

void main()
{
  Window *x, *y;

  x = new Window();
  x->Create();

  y = new CommandButton();
  y->Create();
}
```

The output of the above program will be,

Base class Window
Derived class Command Button

If the function had not been declared virtual, then the base class function would have been called all the times. Because,the function address would have been statically bound during compile time. But now, as the function is declared virtual it is a candidate for run-time linking and the derived class function is being invoked

Pointers to base class

One of the key features of derived classes is that a pointer to a derived class is type-compatible with a pointer to its base class(polymorphism).

```
// pointers to base class
#include <iostream>
using namespace std;

class CPolygon {
 protected:
  int width, height;
 public:
  void set_values (int a, int b)
    { width=a; height=b; }
};

class CRectangle: public CPolygon {
 public:
  int area ()
    { return (width * height); }
};

class CTriangle: public CPolygon {
 public:
  int area ()
```

```
    { return (width * height / 2); }
};

int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
  return 0;
}
```

Output:

20

10

In function main, we create two pointers that point to objects of class CPolygon (ppoly1 and ppoly2). Then we assign references torect and trgl to these pointers, and because both are objects of classes derived from CPolygon, both are valid assignment operations.

Abstract classes: An abstract class is a class that is designed to be specifically used as a base class

we cannot use an abstract class as a parameter type, a function return type, or the type of an explicit conversion, nor can you declare an object of an abstract class, however,we can declare pointers and references to an abstract class.

```
abstract class Class name {

 attributes:

  methods:



 }
```

In the the abstract class', the properties are only specified but not fully defined. The full definition including the semantics of the properties must be provided by derived classes.

Pure virtual function:when any function inn class declair like

Virtual function()=0;

The function is called pure virtual function.

Abstract Base class:the class contaion at least one pure virtual function is called abstract base class

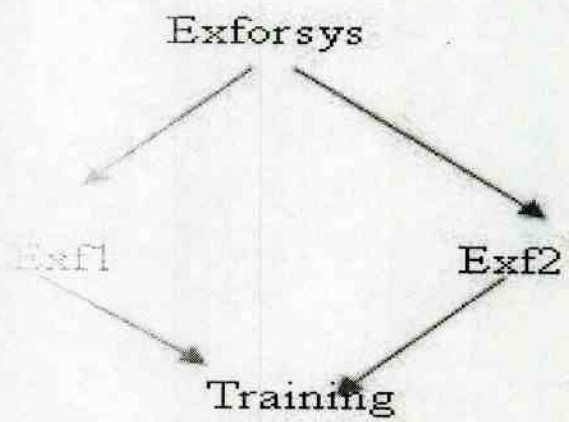
Example:

```
// abstract base class

#include <iostream>

using namespace std;


class CPolygon {
```

```
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
  };

class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };

class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };

int main () {
  CRectangle rect;
  CTriangle trgl;
```

```
CPolygon * ppoly1 = &rect;
CPolygon * ppoly2 = &trgl;
ppoly1->set_values (4,5);
ppoly2->set_values (4,5);
cout << ppoly1->area() << endl;
cout << ppoly2->area() << endl;
return 0;
}
```

Virtual Base Class



In the above example, there are two derived classes Exf1 and Exf2 from the base class Exforsys. As shown in the above diagram, the Training class is derived from both of the derived classes Exf1 and Exf2. In this enviourment , if a user has a member function in the class Training where the user wants to access the data or member functions of the class Exforsys it would result in error if it is performed like :

```
class Exforsys
{
    protected:
    int x;
};

class Exf1:public Exforsys
{ }

class Exf2:public Exforsys
{ };

class Training:public Exf1,public Exf2
{
public:
int example()
{
return x;
}
};
```

The above program results a compile time error as the member function example() of class Training tries to access member data x of class Exforsys. This results in an

error because the derived classes Exf1 and Exf2 (derived from base class Exforsys) create copies of Exforsys called subobjects.
This means that each of the subobjects have Exforsys member data and member functions and each have one copy of member data x. When the member function of the class Training tries to access member data x, confusion arises as to which of the two copies it must access since it derived from both derived classes, resulting in a compile time error.
At this scenario, Virtual base class is used. Both of the derived classes Exf1 and Exf2 are created as virtual base classes, meaning they should share a common subobject in their base class.

```
class Exforsys

protected:
int x;



class Exf1:virtual public Exforsys


class Exf2:virtual public Exforsys


class Training:public Exf1,public Exf2

public:
void example()

cout x;
```

In the above example, both Exf1 and Exf2 are created as Virtual base classes by using the keyword virtual. This enables them to share a common subobject of their base class Exforsys. This results in only one copy that the member function example() of Class Training can access the member data x.

Templates:

1.Tamplate is the enpowered programmer to work with generic data type.

2.It allow functions and classes to operate with generic types,by work on many different data types without being rewritten for each one.

3. There are two kinds of templates:

A) *function templates*

B) *class templates*.

Function templates:

1. Function templates are special functions that can operate with generic types.

2. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating code for each type.

Suppose you write a function printData: We write function like:

```
void printData(int value)
{
std::cout<<"The value is "<<value<<std::endl;
}
```

If we later decide and also want to print double values, or std::string values, then we have to overload the function

```
void printData(double value)
```


111

```
{
std::cout<<"The value is "<<value<<std::endl;
}
void printData(std::string value)
{
std::cout<<"The value is "<<value<<std::endl;
}
```

The actual code written for the function is identical in each case; it is just the type of the variable value that changes, yet we have to duplicate the function for each distinct type,here we have to use function templates.

Defining Templates

A Template Definition starts with the keyword template, followed by a list of Template Parameters:

The format for declaring function templates with type parameters is:

template<class identifier>function_declaration;

To create a template function that returns the greater one of two objects:

```
template <class myType>
myType GetMax (myType a, myType b) {
return (a>b?a:b);    }
```

Here we have created a template function with myType as its template parameter

Template parameter represents a type that has not yet been specified, but that can be used in the template function as if it were a regular type.

To use this function template we use the following format for the function call:

function_name<type>(parameters);

Like:

int x,y;

GetMax <int> (x,y);

When the compiler encounters this call to a template function, it uses the template to automatically generate a function replacing each appearance of myType by the type passed as the actual template parameter (int in this case) and then calls it. This process is automatically performed by the compiler and is invisible to the programmer.

Example:

```
// function template
#include <iostream>
using namespace std;

template <class T>
T GetMax (T a, T b) {
  T result;
  result = (a>b)? a : b;
  return (result);
}
```

```
int main () {
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax<int>(i,j);
  n=GetMax<long>(l,m);
  cout<< k << endl;
  cout << n << endl;
  return 0;
}
```

Class templates:

1. A class template provides a specification for generating classes based on parameters.
2. Class templates are commonly used to implementcontainers.
3. A class template is instantiated by passing a given set of types to it as template arguments

Like

```
template <class T>
class mypair {
  T values [2];
  public:
    mypair (T first, T second)
    {
      values[0]=first; values[1]=second;
```

```
};
```

Here class mypair is created with T type template parameter,it serves to store two elements of any valid type. For example, if we wanted to declare an object of this class to store two integer values of type int with the values 115 and 36 we would write:

```
mypair<int> myobject (115, 36);
```

```
// class templates
#include <iostream>
using namespace std;

template <class T>
class mypair {
  T a, b;
  public:
    mypair (T first, T second)
      {a=first; b=second;}
    T getmax ();
};

template <class T>
T mypair<T>::getmax ()
```

```
{
  T retval;
  retval = a>b? a : b;
  return retval;
}

int main () {
  mycalc <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

Exceptions Handeling:

Exception:

- Exceptions are run-time anomalies, such as division by zero
- An exception is a situation in which a program has an unexpected circumstance that the section of code containing the problem is not explicitly designed to handle.

C++ language provides built-in support for raising and handling exceptions. With C++ exception handling.

The exception handling mechanism is made up of the following elements

- try blocks
- catch blocks

throw expressions

to react to exceptional circumstances (like runtime errors) in our program by transferring control to special functions called *handlers*

To catch exceptions we must place a portion of code under exception inspection. This is done by enclosing that portion of code in a *try block*. When an exceptional circumstance arises within that block, an exception is thrown that transfers the control to the exception handler. If no exception is thrown, the code continues normally and all handlers are ignored.

An exception is thrown by using the throw keyword from inside the try block. Exception handlers are declared with the keywordcatch, which must be placed immediately after the try block

```
// Example
#include <iostream>
using namespace std;

int main () {
 try
 {
  throw(25) ;
 }
 catch ( int e)
 {
  cout << "An exception occurred. Exception Nr. " << e << endl;
```

```
  }
  return 0;
}
```

The code under exception handling is enclosed in a try block.

The exception handler is declared with the catch keyword. As you can see, it follows immediately the closing brace of the try block. The catch format is similar to a regular function that always has at least one parameter. The type of this parameter is very important, since the type of the argument passed by the throw expression is checked against it, and only in the case they match, the exception is caught.

We can use multiple catch expressions, each one with a different parameter type. Only the handler that matches its type with the argument specified in the throw statement is executed.

```
try {
  // code here
}
catch (int param) { cout << "int exception"; }
catch (char param) { cout << "char exception"; }
catch (...) { cout << "default exception"; }
```

In this case the last handler would catch any exception thrown with any parameter that is neither an int nor a char.

After an exception has been handled the program execution resumes after the try-catch block, not after the throw statement

We can also nest try-catch blocks within try blocks.

Like

```
try {
  try {
    // code here
  }
  catch (int n) {
    throw;
  }
}
catch (...) {
  cout << "Exception occurred";
}
```

SKIT, Jaipur
DEPARTMENT OF INFORMATION TECHNOLOGY

ACADEMIC SESSION 2021- 2022 / ODD (III) SEMESTER

Tutorial Sheets

Subject Code/Name: 3IT4-06/Object Oriented Programming

Year / Sem : II /III

UNIT I

S. No.	Question	EMD Analysis
1	**Difference between Class and structure?** Class is the ADT whereas structure is UDT. Class needs access specifier such as private, public & private whereas structure Members can be accessed by public by default & don't need any access specifiers. Class is oops where structure is borrowed from traditional structured [pop] concept.	M
2	**What are the basic concepts of OOS?** Objects. Classes. Data abstraction and Encapsulation. Inheritance. Polymorphism. Dynamic binding. Message passing	E
3	**What are objects? How are they created?** Objects are basic run-time entities in an object-oriented programming system. The class variables are known as objects. Objects are created by using the syntax: classname obj1,obj2,…,objn; (or) during definition of the class: class classname { ------- }obj1,obj2,…,objn	E
4	**Write a C++ program to demonstrate the concept of array of objects.** #include <iostream> class MyClass { int x; public: void setX(int i) { x = i; } int getX() { return x; }	M

	};
	
	void main()
	{
	MyClassobs[4];
	int i;
	
	for(i=0; i< 4; i++)
	obs[i].setX(i);
	
	for(i=0; i< 4; i++)
	cout<< "obs[" <<i<< "].getX(): " <<obs[i].getX() << "\n";
	
	getch();
	}	

5	**List out the benefits & applications of oops.** • Can create new programs faster because we can reuse code • Easier to create new data types • Easier memory management • Programs should be less bug-prone, as it uses a stricter syntax and type checking. • `Data hiding', the usage of data by one program part while other program parts cannot access the data Will whiten your teeth**22.** **Application of oops.** • Client server computing • Simulation such as flight simulations. • Object-oriented database applications. • Artificial intelligence and expert system • Computer aided design and manufacturing systems.	M
6	Explain Characteristics of OOP with example	D
7	Explain how function can be defined inside and outside class with example	D
8	Write a C++ Program to find roots of Quadratic Equation using class and access specifiers.	D
9	Write a C++ Program to create bank class and operation on bank account. (Assume functions as required)	D
10	Differentiate between private, public and protected access specifiers	D

UNIT II

S. No.	Question	EMD Analysis
1	**What are the advantages of Default Arguments?** The function assigns a default value to the parameter which does not have a matching argument in the function call. They are useful in situations where some arguments always have the same value. e.g., float amt [float P, float n, float r = 0.15];	E
2	**List some of the special properties of constructor function.** • They should be declared in the public section. • They are invoked automatically when the objects are created. • They do not have return types, not even void and cannot return values. • Constructors cannot be virtual. Like other C++ functions, they can have default arguments	M
3	**What do you mean by friend functions?** C++ allows some common functions to be made friendly with any number of classes, thereby allowing the function to have access to the private data of thse classes. Such a function need not be a member of any of these classes. Such common functions are called friend functions.	E
4	**What is 'this' pointer?** This pointer is a pointer accessible only within the nonstatic member functions of a class, struct, or union type. It points to the object for which the member function is called. Static member functions don't have a this pointer.	E
5	**What are new and delete operators?** new and delete operators are provided by C++ for runtime memory management. They are used for dynamic allocation and freeing of memory while a program is running. The new operator allocates memory and returns a pointer to the start of it. The delete operator frees memory previously allocated using new. The general form of using them is : p_var = new type; delete p_var;	E

6	**What is a Destructor?**
	A destructor is used to destroy the objects that have been created by a constructor. It is a special member function whose name is same as the class and is preceded by a tilde '~' symbol. When an object goes out from object creation, automatically destructor will be executed.
	Example: class File { public:
	~File(); //destructor declaration
	}; File::~File()
	close(); // destructor definition
		M

	}	

7	**Give an example for a Copy Constructor**
	#include<iostream>
	#include<conio.h> using namespace std; class Example {
	// Variable Declaration int a,b;
	public:
	
	//Constructor with Argument
	Example(int x,int y) {
	// Assign Values In Constructor a=x;
	b=y;
	cout<<"\nIm Constructor";
	}
	void Display() {
	cout<<"\nValues :"<<a<<"\t"<<b;
	}
	};
	int main() {
	Example Object(10,20);
	//Copy Constructor
	Example Object2=Object;
	// Constructor invoked. Object.Display(); Object2.Display();
	// Wait For Output Screen getch();
	return 0;
		M

8	Explain function overloading? Write a C++ program to find sum of two numbers with different data types using function overloading.	D
9	Write a C++ program to sort an array using dynamic memory allocation	D
10	Write a C++ program to add two complex numbers using friend function	D

UNIT III

S. No.	Question	EMD Analysis
1	**What is the difference between base class and derived class?** The biggest difference between the base class and the derived class is that the derived class contains the data members of both the base and its own data members. The other difference is based on the visibility modes of the data members.	E
2	**Mention the types of inheritance.** 1.Single inheritance. 2. Multiple inheritance. 3. Hierarchical inheritance. 4. Multilevel inheritance. 5. Hybrid inheritance.	M
3	**What do you mean by pure virtual functions?** A pure virtual function is a function declared in a base class that has no definition relative to the base class. In such cases, the compiler requires each derived class to either define the function or redeclare it as a pure virtual function. A class containing pure virtual functions cannot be used to declare any objects of its own.	E
4	**What is a virtual base class?** When a class is declared as virtual c++ takes care to see that only copy of that class is inherited, regardless of how many inheritance paths exist between the virtual base class and a derived class.	E
5	**What are abstract classes?** Classes containing at least one pure virtual function become abstract classes. Classes inheriting abstract classes must redefine the pure virtual functions; otherwise, the derived classes also will become abstract. Abstract classes cannot be instantiated	E

6	**Write a C++ program to implement multiple inheritance**
	#include<iostream>
	using namespace std;
	
	class A
	{
	public:
	A() { cout<< "A's constructor called" <<endl; }
	};
	
	class B
	{
	public:
	B() { cout<< "B's constructor called" <<endl; }
		M

	};
	
	class C: public B, public A // Note the order
	{
	public:
	C() { cout<< "C's constructor called" <<endl; }
	};
	
	int main()
	{
	C c;
	return 0;
	}	

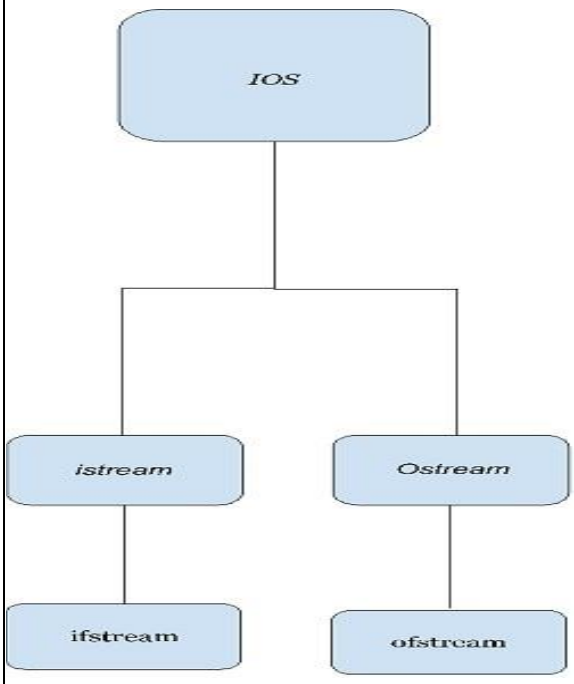
7	Explain diamond problem with help of a diagram. Write a C++ program to implement solution for diamond problem.	D
8	Explain pure virtual function with help of C++ program	D
9	Write a C++ program to implement run time polymorphism using virtual function	D
10	Write a C++ program to implement multi-level inheritance	D

UNIT IV

S. No.	Question	EMD Analysis
1	**What is the need for Overloading an operator?** To define a new relation task to an operator, we must specify what it means in relation to the class to which the operator is applied. This is done with the help of a special function called operator function. It allows the developer to program using notation closer to the target domain and allow user types to look like types built into the language. The ability to tell the compiler how to perform a certain operation when its corresponding operator is used on one or more variables.	E
2	**What are the C++ operators that cannot be overloaded?** "." Member access or dot operator "? : " Ternary or conditional operator "::" Scope resolution operator ".*" Pointer to member operator "sizeof" The object size operator "typeid" Object type operator	M
3	**Define dynamic binding.** Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run-time.	E
4	**Define Polymorphism.** Polymorphism is another important oops concept. Polymorphism means the ability to take more than one form. For example, an operation may exhibit different behavior in different instances. Behavior depends upon the types of data used in the operation.	E
5	**What are abstract classes?** Classes containing at least one pure virtual function become abstract classes. Classes inheriting abstract classes must redefine the pure virtual functions; otherwise, the derived classes also will become abstract. Abstract classes cannot be instantiated	M
6	**What is a static data member?** When a data member is declared as static , only one copy of the data is maintained for all objects of the class. Static data members are not part of objects of a given class type. As a result, the declaration of a static data member is not considered a definition.	M
7	**What is constant member function?** The const member functions are the functions which are declared as constant in the program. The object called by these functions cannot be modified. It is recommended to use const keyword so that	M

	accidental changes to object are avoided. A const member function can be called by any type of object	
8	Write a C++ program to add two complex numbers using operator overloading	D
9	Write a C++ program tocount number of objects created and destroyed in a program.	D
10	Write a C++ program to add two distances (feet, inch) using operator overloading with friend function.	D

UNIT V

S. No.	Question	EMD Analysis
1	**What are stream classes in C++ ?** *Stream classes* in C++ are used to input and output operations on files and io devices. These classes have specific features and to handle input and output of the program. The iostream.h library holds all the stream classes in the C++ programming language. 	E
2	**How is an exception handled in C++?** Exceptions are run time anomalies or unusual conditions that a program may encounter while executing. 1. Find the problem (Hit the exception) 2. Inform that an error has occurred.(Throw the exception) 3. Receive the error information. (Catch the exception) 4. Take corrective actions.(Handle the exception)	E

3	**What is function template? Explain**
	Like class template, we can also define function templates that could be used to create a family of functions with different argument types.
	The general format of a function template is:
	template<class T>
	returntypefunctionname (arguments of type T)
	{
	// ……
	//…….Body of function with type T wherever appropriate
	//……..
	}
		M

4	**Give the syntax of exception handling mechanism.**
	The syntax of exception handling mechanism is as follows:
	try
	{
	---------- throw exception
	---------- 	
	}
	catch(type arguments)
	{
	---------- 	
	---------- 	
	}
		E

5	**What is class template?**
	Templates allows to define generic classes. It is a simple process to create a generic class using a template with an anonymous type.
	The general format of class template is:
	template<class T>10
	
	classclassname
	{// ……class member specification.
	//……. with anonymous type T wherever appropriate
	//……,…..
	}
		M

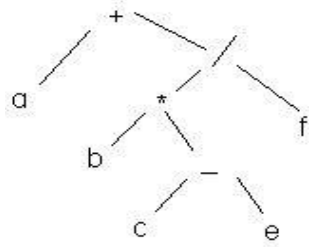
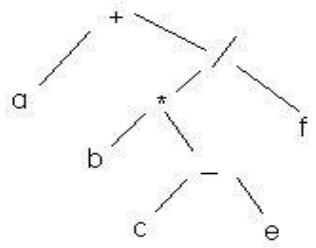
6	**What are 3 basic keywords of exception handling mechanism?** C++ exception handling mechanism is basically built upon three keywords try throw catch	E
7	**What happens when a raised exception is not caught by catch block?** If the type of object thrown matches the arg type in the catch statement, then catch block is executed for handling the exception. If they do not match the program is aborted with the help of abort() function which is invoked by default. When no exception is detected and thrown, the control goes to the statement immediately after the catch block. That is the catch block is skipped.	M

8	Write a C++ program tocopy content of one file to another file.
	
	`#include <iostream>`
	`#include <fstream>`
	`using namespace std;`
	
	`int main()`
	`{`
	`  string line;`
	`  //For writing text file`
	`  //Creating ofstream &ifstream class object`
	`  ifstreamini_file {"original.txt"};`
	`  ofstream out_file {"copy.txt"};`
	
	`  if(ini_file&&out_file){`
	
	`  while(getline(ini_file,line)){`
	`  out_file<< line << "\n";`
	`  }`
		D

	<pre>cout<< "Copy Finished \n";
	
	} else {
	//Something went wrong
	printf("Cannot read File");
	}
	
	//Closing file
	ini_file.close();
	out_file.close();
	
	return 0;
	}</pre>	

9	Write a C++ program to count number of characters, words and lines in a file.	D
10	Write a C++ program to add two distances (feet, inch) using operator overloading with friend function.	D

15	**What is meant by pivot node?** The node to be inserted travel down the appropriate branch track along the way of the deepest level node on the branch that has a balance factor of +1 or -1 is called pivot node.	E
16	**What is the length of the path in a tree?** The length of the path is the number of edges on the path. In a tree there is exactly one path form the root to each node.	E
17	**Define expression trees?** aves of an expression tree are operands such as constants or variable names and the other nodes contain operators.	E
18	**What is a threaded binary tree?** A threaded binary treemay be defined as follows: "A binary tree is *threaded* by making all right child pointers that would normally be null point to the inorder successor of the node, and all left child pointers that would normally be null point to the inorder predecessor of the node	E
19	**What is meant by binary search tree?** Binary Search tree is a binary tree in which each internal node *x* stores an element such that the element stored in the left sub tree of *x* are less than or equal to *x* and elements stored in the right sub tree of *x* are greater than or equal to *x*.	E
20	**Write the advantages of threaded binary tree.** The difference between a binary tree and the threaded binary tree is that in the binary trees the nodes are null if there is no child associated with it and so there is no way to traverse back. But in a threaded binary tree we have threads associated with the nodes i.e they either are linked to the predecessor or successor in the in order traversal of the nodes. This helps us to traverse further or backward in the in ordertraversal fashion. There can be two types of threaded binary tree :- 1) Single Threaded: - i.e. nodes are threaded either towards its in order predecessor or successor. 2) Doublethreaded:-i.e.nodesarethreadedtowardsboththe in order predecessor and successor.	E
21	**What is the various representation of a binary tree?** Tree Representation Array representation Linked list representation	E
22	**List the application of tree.** (i) Electrical Circuit ii) Folder structure a. Binary tree is used in data processing. b. File index schemes c. Hierarchical database management system	E
23	**Define binary tree and give the binary tree node structure.**	E

		
24	**What are the different ways of representing a Binary Tree?** ● Linear Representation using Arrays. ● Linked Representation using Pointers.	M
25	**Give the pre & postfix form of the expression (a + ((b*(c-e))/f).** 	M
26	**Define a heap. How can it be used to represent a priority queue?** A priority queue is a different kind of queue, in which the next element to be removed is defined by (possibly) some other criterion. The most common way to implement a priority queue is to use a different kind of binary tree, called a heap. A heap avoids the long paths that can arise with binary search trees.	M
27	**What is binary heap?** It is a complete binary tree of height h has between 2^{hand} 2^{h+1}-1 node. The value of the root node is higher than their child nodes	M
28	**Define Strictly binary tree?** If every nonleaf node in a binary tree has nonempty left and right subtrees ,the tree is termed as a strictly binary tree.	M
29	**Define complete binary tree?** A complete binary tree of depth d is the strictly binary tree all of whose are at level d.	M
30	**What is an almost complete binary tree?** A binary tree of depth d is an almost complete binary tree if : _ Each leaf in the tree is either at level d or at level d-1 _ For any node nd in the tree with a right descendant at level d,allthe left descendants of nd that are leaves are at level d.	M
31	**Define AVL Tree.** A AVL tree is a binary search tree except that for every node in the tree,the height of the left and right subtrees can differ by atmost 1.	M

1	Define Tree. Explain the tree traversals with algorithms and examples.	D
2	Construct an expression tree for the expression (a + b * c) +((d * e + 1) * g). Give the outputs when you apply preorder, inorder and postorder traversals.	D
3	Explain binary search tree ADT in detail.	D
4	Explain AVL tree ADT in detail.	D
5	Explain b tree and B+ tree ADT in detail.	D
6	Explain Heap tree ADT in detail.	D
7	Explain threaded binary tree ADT in detail.	D

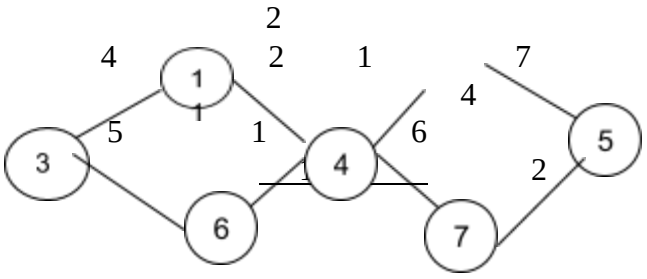
UNIT V

GRAPHS

S. No.	Question	EMD Analysis
1	**Define Graph?** A graph G consist of a nonempty set V which is a set of nodes of the graph, a set E which is the set of edges of the graph, and a mapping from the set for edge E to a set of pairs of elements of V. It can also be represented as G= (V, E).	E
2	**Explain the topological sort.** It is an Ordering of vertices in a directed acyclic graph such that if there is a path from vi to vj, then vj appears after vi in the ordering.	E
3	**Define NP** NP is the class of decision problems for which a given proposed solution for a given input can be checked quickly to see if it is really a solution.	E
4	**Define biconnected graph.** A connected undirected graph is biconnected if there are no vertices whose removal disconnects the rest of the graph.	E
5	**Define shortest path problem?** For a given graph G=(V, E), with weights assigned to the edges of G, we have to find the shortest path (path length is	E

	defined as sum of the weights of the edges) from any given source vertex to all the remaining vertices of G.	
6	**Mention any two decision problems which are NP-Complete.** NP is the class of decision problems for which a given proposed solution for a given input can be checked quickly to see if it is really a solution	E
7	**Define adjacent nodes?** Any two nodes which are connected by an edge in a graph are called adjacent nodes. For E is associated with a pair of nodes∈example, if and edge x (u,v) where u, v V, then we say that the edge x connects the nodes u and v.∈	E
8	**What is a directed graph?** A graph in which every edge is directed is called a directed graph.	E
9	**What is a undirected graph?** A graph in which every edge is undirected is called a directed graph.	E
10	**What is a loop?** An edge of a graph which connects to itself is called a loop or sling.	E
11	**What is a simple graph?** A simple graph is a graph, which has not more than one edge between a pair of nodes than such a graph is called a simplegraph.	E
12	**What is a weighted graph?** A graph in which weights are assigned to every edge is called a weighted graph.	E
13	**Define out degree of a graph?** In a directed graph, for any node v, the number of edges which have v as their initial node is called the out degree of the node v.	E
14	**Define indegree of a graph?** In a directed graph, for any node v, the number of edges which have v as their terminal node is called the indegree of the node v.	E
15	**Define path in a graph?** The path in a graph is the route taken to reach terminal node from a starting node.	E
16	**What is a simple path?** A path in a diagram in which the edges are distinct is called a simple path. It is also called as edge simple.	E
17	**What is a cycle or a circuit?** A path which originates and ends in the same node is called a cycle or circuit.	E
18	**What is an acyclic graph?** A simple diagram which does not have any cycles is called an acyclic graph.	E
19	**What is meant by strongly connected in a graph?** An undirected graph is connected, if there is a path from every vertex to every other vertex. A directed graph with this property is called strongly connected.	E

20	**When is a graph said to be weakly connected?** When a directed graph is not strongly connected but the underlying graph is connected, then the graph is said to be weakly connected.	E
21	**Name the different ways of representing a graph?** a. Adjacency matrix b. Adjacency list	M
22	**What is an undirected acyclic graph?** When every edge in an acyclic graph is undirected, it is called an undirected acyclic graph. It is also called as undirected forest.	M
23	**What are the two traversal strategies used in traversing a graph?** a. Breadth firstsearch b. Depth firstsearch	M
24	**What is a minimum spanning tree?** A minimum spanning tree of an undirected graph G is a tree formed from graph edges that connects all the vertices of G at the lowest total cost.	M
25	**Define topological sort?** A topological sort is an ordering of vertices in a directed acyclic graph, such that if there is a path from $v_i$to v_j appears after $v_i$in the ordering.	M
26	**What is the use of Kruskal's algorithm and who discovered it?** Kruskal's algorithm is one of the greedy techniques to solve the minimum spanning tree problem. It was discovered by Joseph Kruskal when he was a second-year graduate student.	M
27	**What is the use of Dijksra's algorithm?** Dijkstra's algorithm is used to solve the single-source shortest-paths problem: for a given vertex called the source in a weighted connected graph, find the shortest path to all its other vertices. The single-source shortest-paths problem asks for a family of paths, each leading from the source to a different vertex in the graph, though some paths may have edges in common.	M
28	**Prove that the maximum number of edges that a graph with n Vertices is n*(n-1)/2.** Choose a vertex and draw edges from this vertex to the remaining n-1 vertices. Then, from these n-1 vertices, choose a vertex and draw edges to the rest of the n-2 Vertices. Continue this process till it ends with a single Vertex. Hence, the total number of edges added in graph is (n-1)+(n-2)+(n-3)+…+1 =n*(n-1)/2.	M
29	**Define minimum cost spanning tree?** A spanning tree of a connected graph G, is a tree consisting of edges and all the vertices of G. In minimum spanning tree T, for a given graph G, the total weights of the edges of the spanning tree must be minimum compared to all other spanning trees generated from G. -Prim's and Kruskal is the algorithm for finding Minimum Cost Spanning Tree.	M

30	**Define Adjacency in graph.** Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.	E
31	**Define Basic Operations of Graph.** Following are basic primary operations of a Graph • **Add Vertex** − Adds a vertex to the graph. • **Add Edge** − Adds an edge between the two vertices of the graph. • **Display Vertex** − Displays a vertex of the graph.	E
32	**What is Levels in graph?** Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.	E
33	**What is visiting and traversing in graph.** ● Visiting refers to checking the value of a node when control is on the node. ● Traversing means passing through nodes in a specific order.	E
1	Explain the various representation of graph with example in detail?	D
2	Define topological sort? Explain with an example?	D
3	Explain Dijkstra's algorithm with an example?	D
4	Explain Prim's algorithm with an example?	D
5	Explain Krushal's algorithm with an example?	D
6	Write and explain the prim's algorithm and depth first search algorithm.	D
7	For the graph given below, construct Prims algorithm 	D
8	Explain the breadth first search algorithm	D
9	the algorithm to compute lengths of shortest path	D
10	n the depth first search algorithm.	D

UNIT III

SEARCHING, SORTING AND HASHING TECHNIQUES

S. No.	Question	EMD Analysis
1	**Define sorting** Sorting arranges the numerical and alphabetical data present in a list in a specific order or sequence. There are a number of sorting techniques available. The algorithms can be chosen based on the following factors ● Size of the data structure ● Algorithm efficiency Programmer's knowledge of the technique	E
2	**Mention the types of sorting** ● Internal sorting ● External sorting	E
3	**What do you mean by internal and external sorting?** An internal sort is any data sorting process that takes place entirely within the main memory of a computer. This is possible whenever the data to be sorted is small enough to all be held in themain memory. External sorting is a term for a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memoryofacomputingdevice(usuallyRAM)andinsteadthey must reside in the slower external memory (usually a hard drive).	E
4	**How the insertion sort is done with the array?** It sorts a list of elements by inserting each successive element in the previously sorted Sub list. Consider an array to be sorted A[1],A[2],….A[n] a. Pass 1: A[2] is compared with A[1] and placed them in sorted order. b. Pass 2: A[3] is compared with both A[1] and A[2] and inserted at an appropriate place. This makes A[1], A[2],A[3] as a sorted sub array. c. Pass n-1: A[n] is compared with each element in the sub array	E

	A [1], A [2] …A [n-1] and inserted at an appropriate position.	
5	**Define hashing.** Hash function takes an identifier and computes the address of that identifier in the hash table using some function	E
6	**What is the need for hashing?** Hashing is used to perform insertions, deletions and find in constant average time.	E
7	**Define hash function?** Hash function takes an identifier and computes the address of that identifier in the hash table using some function.	E
8	**List out the different types of hashing functions?** The different types of hashing functions are, a. The division method b. The mind square method c. The folding method d. Multiplicative hashing e. Digit analysis	E
9	**What are the problems in hashing?** a. Collision b. Overflow	E
10	**What are the problems in hashing?** When two keys compute in to the same location or address in the hash table through any of the hashing function then it is termed collision.	E
11	**what is insertion sort? How many passes are required for the elements to be sorted ?** one of the simplest sorting algorithms is the insertion sort. Insertion sort consist of N-1 passes . For pass P=1 through N-1 , insertion sort ensures that the elements in positions 0 through P-1 are in sorted order .It makes use of the fact that elements in position 0 through P-1 are already known to be in sorted order .	E

12	**Write the function in C for insertion sort ?**
	void insertionsort(elementtype A[ ] , int N)
	{
	int j, p;
	elementtypetmp;
	for(p=1 ; p <N ;p++)
	{
	tmp = a[p] ;
	for (j=p ; j>0 && a [j -1] >tmp ;j--)
	a [j]=a [j-1] ;
	a [j] = tmp ;
	}}
		E

13	**Who invented shellsort ? define it ?** Shellsort was invented by Donald Shell . It works by comparing element that are distant . The distance between the comparisons decreases as the algorithm runs until the last phase inwhich	E

	adjacent elements are compared . Hence it is referred as diminishing increment sort.	

14	**write the function in c for shellsort?**
	Void Shellsort(Elementtype A[ ],int N)
	{
	int i , j , increment ;
	elementtypetmp ;
	for(elementtype=N / 2;increment > 0;increment / = 2)
	For(i= increment ; i<N ; i ++)
	{
	tmp=A[ ];
	for(j=I; j>=increment; j - =increment)
	if(tmp< A[ ]=A[j – increment];
	A[j]=A[j – increment];
	Else
	Break;
	A[j]=tmp;
	}}
		M

15	**erentiate between merge sort and quick sort?** **Mergesort** **quicksort** 1. Divide and conquer strategy Divide and conquer strategy 2. Partition by position Partition by value	M
16	**Mention some methods for choosing the pivot element in quick sort?** 1. Choosing first element 2. Generate random number 3. Median of three	M
17	**What are the three cases that arise during the left to right scan in quick sort?** 1. I and j cross each other 2. I and j do not cross each other 3. I and j points the same position	M
18	**What is the need of external sorting?** External sorting is required where the input is too large to fit into memory. So external sorting Is necessary where the program is too large	M
19	**What is sorting?** Sorting is the process of arranging the given items in a logical order. Sorting is an example where the analysis can be precisely performed.	M
20	**What is mergesort?** The mergesort algorithm is a classic divide conquer strategy. The problem is divided into two arrays and merged into single array	M
21	**Compare the various hashing techniques.** Technique Load Factor Separate chaining - close to 1 Open Addressing - should not exceed 0.5 Rehashing - reasonable load factor	M

22	**Define collision in hashing.** When two different keys or identifiers compute into the same location or address in the hash table through any of the hashing functions, then it is termed Collision.	M
23	**Define Double Hashing.** Double Hashing is a collision-resolution technique used in open addressing category. In double hashing, we apply a second hash function to x and probe at a distance of hash2 (x), 2hash2 (x)…..... , and so on.	M
24	**What are applications of hashing?** The applications of hashing are, ● Compliers use hash table to keep track of declared variables on source code. ● Hash table is useful for any graph theory problem, where the nodes have real names instead of numbers. ● Hash tables are used in programs that play games. ● Online spelling checkers use hashing.	M
25	**What does internal sorting mean?** Internal sorting is a process of sorting the data in the main memory	M
26	**What are the various factors to be considered in deciding a sorting algorithm?** Factors to be considered in deciding a sorting algorithm are, 1. Programming time 2. Executing time for program 3. Memory or auxiliary space needed for the programsenvironment.	M
27	**How does the bubble sort get its name?** The bubble sort derives its name from the fact that the smallest data item bubbles up to the top of the sorted array.	M
28	**What is the main idea behind the selection sort?** The main idea behind the selection sort is to find the smallest entry among in a(j),a(j+1),……. a(n) and then interchange it with a(j). This process is then repeated for each value of j.	M
29	**Is the heap sort always better than the quick sort?** No, the heap sort does not perform better than the quick sort. Only when array is nearly sorted to begin with the heap sort algorithm gains an advantage. In such a case, the quick deteriorates to its worst performance of O (n2).	M
30	**Name some of the external sorting methods.** Some of the external sorting methods are, 1. Polyphase sorting 2. Oscillation sorting 3. Merge sorting	M
31	**Define radix sort** Radix Sort is aclever and intuitive little sorting algorithm. **Radixsort**isaoncomparativeintegersortingalgorithmthatsorts	M

	data with integer keys by grouping keys by the individual digits which share the same significant position	
32	**Define searching** Searching refers to determining whether an element is present in a given list of elements or not. If the element is present, the search is considered as successful, otherwise it is considered as an unsuccessful search. The choice of a searching technique is based on the following factors a. Order of elements in the list i.e., random or sorted b. Size of the list	M
33	**Mention the types of searching** The types are ● Linear search ● Binary search	M
34	**What is meant by linear search?** **Linear search** or **sequential search** is a method for finding a particular value in a list that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found.	M
35	**What is binary search?** For binary search, the array should be arranged in ascending or descending order. In each step, the algorithm compares the search key value with the middle element of the array. If the key match, then a matching element has been found and its index, or Position, is returned. Otherwise, if the search key is less than the middle element, then the algorithm repeats its action on the sub-array to the left of the middleelementor,ifthesearchkeyisgreater,onthesub-arrayto the right.	M
36	**What are the collision resolution methods?** The following are the collision resolution methods ● Separate chaining ● Open addressing ● Multiple hashing	M
37	**Define separate chaining** It is an open hashing technique. A pointer field is added to each record location, when an overflow occurs; this pointer is set to point to overflow blocks making a linked list. In this method, the table can never overflow, since the linked lists are only extended upon the arrival of new keys.	M
38	**What is open addressing?** Open addressing is also called closed hashing, which is an alternative to resolve the	M

	Collisions with linked lists. In this hashing system, if a collision occurs, alternative cells are tired until an empty cell is found. There are three strategies in open addressing: ● Linear probing ● Quadratic probing ● Double hashing	
39	**What is Rehashing?** If the table is close to full, the search time grows and may become equal to the table size. When the load factor exceeds a certain value (e.g. greater than 0.5) we do Rehashing: Build a second table twice as large as the original and rehash there all the keys of the original table. Rehashing is expensive operation, with running time O(N) However, once done, the new hash tablewill have good performance.	D
40	**What is Extendible Hashing?** Used when the amount of data is too large to fit in main memory and external storage is used. **N** records in total to store, **M** records in one disk block **The problem**: in ordinary hashing several disk blocks may be examined to find an element - a time consuming process. **Extendible hashing**: no more than two blocks are examined.	D
1	Explain the sorting algorithms	D
2	Explain the searching algorithms	D
3	Explain hashing	D
4	Explain open addressing	D
5	Write a C program to sort the elements using bubble sort, insertion sort and radix sort.	D
6	Write a C program to perform searching operations using linear and binary search.	D
7	n in detail about separate chaining.	D
8	Explain Rehashing in detail.	D
9	Explain Extendible hashing in detail.	D

UNIT- I

PART – A

1. Write down the definition of data structures?
2. Give few examples for data structures?
3. Define Algorithm?
4. What are the features of an efficient algorithm?
5. List down any four applications of data structures?
6. What is divide and conquer?
7. State the importance of dynamic programming
8. Define storage structure?
9. Define file structure?
10. What are the four major parts in an iterative process?
11. Write down the algorithm for solving Towers of Hanoi problem?
12. What are the different types of data structures?
13. What do you mean by primitive data structure?
14. What are the three stages of problem solving aspect.
15. Define depth of recursion?
16. What is searching?
17. What is Linear search?
18. Define Space Complexity
19. Define Time Complexity
20. What are asymptotic notations?
21. What is information?
22. Define Recursion?
23. What is a Fibonacci sequence?

PART – B

1. Explain in detail the steps involved in Top down Design. **(16)**

2. Write the verification condition of a program segments with
a. Straight line statements **(4)**
b. Branches **(6)**
c. Loops **(6)**

3. Write short notes on efficiency of an algorithm **(16)**

4. Write short notes on analysis of an algorithm **(16)**

5. (a) Develop an algorithm to compute the sums for the first n terms
S=1+ (1/2) + (1/3) +…. **(8)**

(b) Discuss in detail about the implementation of the algorithm. **(8)**

6. (a) Write an algorithm to reverse the digits of a decimal number. **(8)**

(b) Write an algorithm to compute the Fibonacci series for ‘n’ terms. **(8)**

UNIT-II

PART – A

1. What is an Abstract Data type (ADT)? Explain?
2. What is a Stack?
3. What are the two operations of Stack?
4. Write postfix from of the expression –A+B-C+D?
5. What is a Queue?
6. What is a Priority Queue?
7. What are the different ways to implement list?
8. What are the advantages in the array implementation of list?
9. What is a linked list?
10. Name the two fields of Linked list?
11. What is a doubly linked list?
12. Name the three fields of Doubly Linked list?
13. Define double circularly linked list?
14. What is the need for the header?
15. List three examples that uses linked list?
16. Give some examples for linear data structures?
17. Write postfix from of the expression –A+B-C+D?
18. How do you test for an empty queue?
19. What are the postfix and prefix forms of the expression?
20. Explain the usage of stack in recursive algorithm implementation?
21. Write down the operations that can be done with queue data structure?
22. What is a circular queue?

PART – B

1. Write a program in C to return the position of an element X in a List L. **(16)**

2. (a) State & explain the algorithm to perform Radix Sort. **(8)**

(b) Write a Program in C to create an empty stack and to push an element into it. **(8)**
3. Explain how queues can be implemented using Arrays **(16)**
4. (a) Write a ‘c’ program to multiply two polynomials. **(8)**

(b) Write a ‘c’ program to add two polynomials. **(8)**

5. (a) Write an algorithm to convert infix to postfix expression and explain it with example **(8)**

(b) Write an algorithm to evaluate a postfix expression and explain it with example **(8)**

6. (a) Write an algorithm to check given expression contains balanced parenthesis or not. **(8)**

(b) Write an algorithm for insertion and deletion operation in a circular queue **(8)**

UNIT III

PART – A

1. Define non-linear data structure?
2. Define tree?
3. Define leaf?
4. What is meant by directed tree?
5. What is an ordered tree?
6. What is a Binary tree?
7. What are the applications of binary tree?
8. What is meant by traversing?
9. What are the different types of traversing?
10. What are the two methods of binary tree implementation?
11. Define pre-order traversal?
12. Define post-order traversal?
13. Define in -order traversal?
14. What is the length of the path in a tree?
15. Define expression trees?
16. Define strictly binary tree?
17. Define complete binary tree?
18. What is an almost complete binary tree?
19. Define AVL Tree
20. Define collision resolution

Part B:

1. (a) Construct an expression tree for the expression A+(B-C)*D+(E*F) **(8)**

(b) Write a function to delete the minimum element from a binary heap **(8)**

2. Write a program in C to create an empty binary search tree & search for an Element X in it. **(16)**

3. Explain in detail about Open Addressing **(16)**

4. Explain in detail insertion into AVL Trees **(16)**

5. Write a recursive algorithm for binary tree traversal with an example. **(16)**

6. Write an algorithm for initializing the hash table and insertion in a separate chaining **(16)**

7. State & explain the algorithm to perform Heap sort. Also analyze the time complexity of the algorithm **(16)**

8. Write a C program to perform Merge sort and analyze time complexity of the algorithm. **(16)**

9. State & explain the algorithm to perform Quick sort. Also analyze the time complexity of the algorithm. **(16)**

10. State & explain the algorithm to perform Shell sort. Also analyze the time complexity
of the algorithm. **(16)**

UNIT-IV

PART – A

1. Define Graph?
2. Define adjacent nodes?
3. What is a directed graph?
4. What is an undirected graph?
5. What is a loop?
6. What is a simple graph?
7. What is a weighted graph?
8. Define out degree of a graph?
9. Define indegree of a graph?
10. Define path in a graph?
11. What is a simple path?
12. What is a cycle or a circuit?
13. What is an acyclic graph?
14. What is meant by strongly connected in a graph?
15. When is a graph said to be weakly connected?
16. What is meant by sorting?
17. What are the two main classifications of sorting based on the source of data?
18. What is meant by external sorting?
19. What is meant by internal sorting?
20. What are the various factors to be considered in deciding a sorting algorithm?
21. What is the main idea behind insertion sort?
22. What is the main idea behind selection sort?
23. What is the basic idea of shell sort?
24. What is the other name for shell sort?
25. What is the purpose of quick sort?

PART – B

1. Formulate an algorithm to find the shortest path using Dijkstra's algorithm and explain
with example. **(16)**

2. Explain the minimum spanning tree algorithms with an example. **(16)**

3. (a) Write short notes on Biconnectivity. **(8)**

(b) Write an algorithm for Topological Sort of a graph. **(8)**

4. Write and explain weighted and unweighted shortest path algorithm **(16)**

5. Explain the various applications of Depth First Search. **(16)**

UNIT-V

PART-A

1. Types of automatic list management?
2. What do you meant by Reference count method?
3. What is Garbage collection
4. What is compaction?
5. Give the purpose of list management?
6. What are the disadvantages of reference count method?
7. What are the phases of garbage collection?
8. What do you mean by thrashing?
9. What are the types of pointers?
10. What are methods of implementing add on and tail operations in linked list?
11. Define first fit
12. Define best fit.
13. Define worst fit.
14. What is internal and external fragmentation?
15. What are the types of buddy system?

PART – B

1. Explain the linked list representation of a list with an example. **(16)**

2. Explain reference count method with an example. **(16)**

3. Explain garbage collection with their variations. **(16)**

4. Explain the dynamic memory management with necessary methods. **(16)**

5. Write about operations in linked linear lists. **(16)**

6. Explain the linked list implementation of stack ADT in detail. **(16)**

List of Text and Reference Books

Text Books

1. E. Balagurusamy ,"Object Oriented Programming with C++", Seventh Edition, McGraw Education, 2017.

2. Yashavant Kanetkar, "Let Us C++", Second Edition, BPB Publications, 2020

Reference Books

1. K.R. Venugopal, Rajkumar, T Ravishankar, “Mastering C++”, McGraw Hill Publishing Co. Ltd, 2006.

2. Robert Lafore, "Object Oriented Programming in Turbo C++, First Edition, Galgotia Publications.

3. Herbert Shildt, " C++ : The complete reference", Fourth Edition, McGraw Hill Education, 2017