University Of Pune

Data Structures Using C & Object Oriented Programming Concepts Using C++

S. Y. B. Sc. (Computer Science)

CS -223

SEMESTER I & II

Name _____

College Name _____

Roll No. _____ Division _____

Academic Year _____

Prepared By

Dr. Shailaja C. Shirwaikar

Head, Dept of Comp Sc, Nowrosjee Wadia College

Reviewed By

Ms. Poonam Ponde

Ms. Jyoti Yadav

Mrs. Shubhangi Page

Mrs. Chitra Nagarkar

Preface

This Lab Book supplements the text books and classroom teaching of Data structures and C++. The intention is to bring uniformity in conducting the lab sessions across various affiliated colleges. The assignments are designed so that the theory concepts in the syllabus are broadly covered. There is scope for improvement and additions and deletions can be carried out as the Lab book is always going to remain in digital form and available on the Department of Computer Science, University of Pune, website. I am indebted to all the reviewers of the book as their valuable suggestions have improved the book contents. We are all indebted to Dr. Vilas Kharat, Chairman, Board of studies in Computer Science for continuous encouragement, support and guidance.

Dr. Shailaja C. Shirwaikar

Member, BOS Computer Science

Table of contents

Introduction

1. About the work book

This workbook is intended to be used by S. Y. B. Sc (Computer Science) students for the Data structures using C Lab course in Semester I and for the Object Oriented Programming Concepts using C++ Lab course in Semester II. Data structures is an important core subject of computer science curriculum, and hands-on laboratory experience is critical to the understanding of theoretical concepts studied as part of this course. Study of any programming language, including C++, is incomplete without hands-on experience of implementing solutions using programming paradigms and verifying them in the lab. This workbook provides rich set of problems covering the basic algorithms as well as numerous computing problems demonstrating the applicability and importance of various data structures and related algorithms. The programming exercises in C++ are designed to demonstrate the applicability of Object Oriented programming concepts to problem solving.

The objectives of this book are

1) Defining clearly the scope of the course
2) Bringing uniformity in the way the course is conducted across different colleges
3) Continuous assessment of the course
4) Bring in variation and variety in the experiments carried out by different students in a batch
5) Providing ready reference for students while working in the lab
6) Catering to the need of slow paced as well as fast paced learners

2. How to use this workbook

The workbook is divided into two sections. Section I is related to Assignment in Data structures and are to be solved using C programming language while Section II relates to assignments to be solved in C++ programming language.

The Data Structures syllabus is divided into eight assignments. Each assignment has problems divided into three sets – A, B and C

Set A is used for implementing the basic algorithms or implementing data structure along with its basic operations. Set A is mandatory.

Set B is used to demonstrate small variations on the implementations carried out in set A to improve its applicability. Depending on the time availability the students should be encouraged to complete set B.

Set C prepares the students for the viva in the subject. Students should spend additional time either at home or in the Lab and solve these problems so that they get a deeper understanding of the subject.

Case studies at the end of the Section form the essence of the lab Book. Students should solve the case studies to become skilled programmers. Each problem given as a case

study can be tackled in multiple ways. Multiple data structures and use of more than one algorithm may be used in solving these problems. At least four case studies, amongst the one star marked or chosen by the instructor should be solved as part of work completion.

Some code snippets are also provided which can be used to generate, read or write data. Some text files are provided as attachments with the digital copy of the lab book which can be used as data files.

The C++ syllabus is also divided into eight assignments. Each assignment has problems divided into three sets – A, B and C

Set A is used for demonstrating the C++ concept and knowing the essential syntax to implement it. Set A is mandatory.

Set B is used to demonstrate small variations on the implementations carried out in set A to improve its applicability. Depending on the time availability the students should be encouraged to complete set B.

Set C prepares the students for the viva in the subject. Students should spend additional time either at home or in the Lab and solve these problems so that they get a deeper understanding of the subject.

Case studies at the end of the Section combine concepts learned in the previous assignments so that varieties of concepts are used to tackle a single problem. Students should solve the case studies to become skilled programmers.

2.1 Instructions to the students

Please read the following instructions carefully and follow them.

1) Students are expected to carry this book every time they come to the lab for computer science practicals.

2) Students should prepare oneself before hand for the Assignment by reading the relevant material.

3) Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However student should spend additional hours in Lab and at home to cover as many problems as possible given in this work book.

4) Students will be assessed for each exercise on a scale from 0 to 5

i) Not done	0
ii) Incomplete	1
iii) Late Complete	2
iv) Needs improvement	3
v) Complete	4
vi) Well Done	5

2.2. Instruction to the Instructors

1) Explain the assignment and related concepts in around ten minutes using white board if required or by demonstrating the software.

2) Make available to students digital copies of text files provided with the book as per the requirement of Assignment,

3) Choose appropriate problems to be solved by students. Set A is mandatory. Choose problems from set B depending on time availability. Discuss set C with students and encourage them to solve the problems by spending additional time in lab or at home.

4) Make sure that students follow the instruction as given above.

5) Choose at least 4 case studies to be solved by each student and evaluate them. Different students can be given different case studies so that class as a whole has solved all of them. Encourage students to discuss their solutions with their peers

6) You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.

7) The value should also be entered on assignment completion page of the respective Lab course.

2.3. Instructions to the Lab administrator

You have to ensure appropriate hardware and software is made available to each student.

The operating system and software requirements on server side and also client side are as given below:

1) Server and Client Side - (Operating System) Fedora Core Linux

2) Server side and Client Side - editor and GCC compiler

Assignment Completion Sheet

Lab Course I			
Section I - Data structures using C			
Sr. No	**Assignment Name**	**Marks (out of 5)**	**Signature**
1	Sorting Algorithms – Non – Recursive		
2	Sorting Algorithms – Recursive		
3	Searching Algorithms		
4	Stack		
5	Queue		
6	Linked List		
7	Tree		
8	Graph		
9	Case study 1 infix to postfix /		
10	Case Study 2 postfix evaluation /		
11	Case study 3 Polynomial Addition /		
12	Case study 4 DFS / BFS /		
Total (out of 60)			
Total (Out of 10)			

Lab Course I			
Section II – Object Oriented Programming Concepts using C++			
Sr. No	**Assignment Name**	**Marks (out of 5)**	**Signature**
1	Class , Object and methods implementation		
2	Constructor: Copy Constructor, Default Constructor, Parameterized Constructor		
3	Memory Allocation: new and delete operators , dynamic constructor		
4	Inline function, friend function, default argument,		
5	Function Overloading.		
6	Operator overloading.		
7	Inheritance: Single, multiple, multilevel, hierarchy, Constructor and destructor in derived class		
8	File Handling: Updation of files using random access		
9	Case study 1		
10	Case Study 2		
Total (out of 50)			
Total (Out of 10)			

This is to certify that Mr/Ms _____ has successfully completed the course work for Lab Course I and has scored ___ Marks out of 20.

Instructor **Head, Dept. Of Comp. Sc.**

Internal Examiner **External Examiner**

Section I - Data structures using C

Assignment 1: Sorting Techniques (Non recursive)

Sorting techniques considered here are internal sorting techniques. The data to be sorted is in memory usually in an array. It could be an array of integers, characters, strings or of defined structure type. To test a sorting algorithm we require large data set. Data is generated using random (rand()) function. The array of random integers in the range 0 to 99 is generated by using following code:

```
void generate ( int * a , int n)

{  int i;

for (i=0; i<n; i++) a[i]=rand()%100;

}
```

In reality data to be sorted is externally stored in files. One need to read data from files and bring it into memory in an array before sorting and sorted array also need to be written back to an external file.

Suppose the records to be sorted containing name, age and salary of a set of employees, is in a text file “employee.txt” as follows:

Rajiv 43 100000

Prakash 34 29000

Vinay 35 20000

.....

The data is read into memory in an array of structures as follows

Variable declarations & main program	Function for reading from a file	Function for writing to a file

typedef struct{
char name[30];
int age;
int salary;
}RECORD;
RECORD emp[100];
main()
{int n;
n=readFile(emp);
sort(emp,n);
writeFile(emp,n);
}
	int readFile(RECORD *a)
	{int i=0;
	FILE *fp;
	if((fp=fopen("emp.txt","r"))!=NULL)
	while(! feof(fp))
	{
	fscanf(fp,"%s%d%d", a[i].name, &a[i].age, &a[i].salary);
	i++;
	}
	return i-1; // number of records read
	}
		void writeFile(RECORD *a, int n)
		{int i=0;
		FILE *fp;
		if((fp=fopen("sortedemp.txt","w"))!=NULL)
		for(i=0;i<n; i++)
		fprintf(fp,"%s\t%d\t%d\n", a[i].name, a[i].age, a[i].salary);
		}

The sorting algorithms you are to use in this assignment are bubble sort and insertion sort.

Set A

a) Sort a random array of n integers (accept the value of n from user) in ascending order by using bubble sort algorithm.

b) Sort a random array of n integers (accept the value of n from user) in ascending order by using insertion sort algorithm.

Set B

a) Read the data from the file “employee.txt” and sort on age using insertion sort / bubble sort.

b) Read the data from the file “employee.txt” and sort on names in alphabetical order (use strcmp) using bubble sort / insertion sort.

Set C

a) What modification is required to insertion sort to sort the integers in descending order?

b) What modifications are required to bubble sort to sort the integers in descending order?

c) What modifications are required to bubble sort to count the number of swaps?

d) What modifications are required to insertion sort to count the number of key comparisons?

e) What modifications are required to improve bubble sort to stop further passes if the file is already sorted that is when there are no more swaps?

f) Compare the system time taken by insertion sort and bubble sort by using 'time' command on a random file of size 10000 or more.

$ time ./a.out

g) What modifications are required to output the array contents after every pass of the sorting algorithm?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 2: Sorting Techniques (Recursive)

Recursive sorting techniques to be used in this assignment are Merge sort and Quick sort. Both use divide and conquer strategy.

In Merge sort, data is divided into two parts, each part is sorted by using the same merge sort technique and the sorted files are then merged using a Merge procedure.

In Quick sort, data is partitioned into two parts in such a way that all elements in first part are less than or equal to elements in second part. Both the parts are then sorted using the same Quick sort technique.

Set A

a) Sort a random array of n integers (accept the value of n from user) in ascending order by using a recursive Merge sort algorithm.

b) Sort a random array of n integers (accept the value of n from user) in ascending order by using recursive Quicksort algorithm.

Set B

a) Read the data from the ‘employee.txt’ file and sort on age using Merge sort/ Quick sort and write the sorted data to another file 'sortedemponage.txt'.

b) Read the data from the file and sort on names in alphabetical order (use strcmp) using Mege sort/ Quick sort and write the sorted data to another file 'sortedemponname.txt'.

Set C

a) What modifications are required to choose the pivot element randomly instead of choosing the first element as pivot element while partitioning in Quick sort algorithm?

b) Compare the system time taken by Merge sort and bubble sort by using time command on a random array of integers of size 10000 or more.

c) What modification is required to Merge sort to sort the integers in descending order?

d) In 'employee.txt' there are records with same name but different age and salary values. What are the relative positions when the data is sorted on name using Merge sort and what happens in case of quick sort?

e) Sort a random array of integers of large size and store the sorted file. Compare the system time taken by Quicksort on a random file of large size and the sorted file of same size. Repeat the same for Merge sort. Does sorted file give best time?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 3: Searching Techniques

The commonly used searching methods used are linear search and Binary search

Linear search is very simple technique to be used on any file while Binary search requires the file to be sorted.

Set A

a) Create a random array of n integers. Accept a value x from user and use linear search algorithm to check whether the number is present in the array or not and output the position if the number is present.

b) Create a random array of n integers. Sort the array using bubble sort. Accept a value x from user and use binary search algorithm to check whether the number is present in array or not and output the position if the number is present.

Set B

a) Read the data from file 'cities.txt' containing names of 100 cities and their STD codes. Accept a name of the city from user and use linear search algorithm to check whether the name is present in the file and output the STD code, otherwise output “city not in the list”.

b) Read the data from file ‘sortedcities.txt’ containing names of 100 cities and their STD codes. Accept a name of the city from user and use binary search algorithm to check whether the name is present in the file and output the STD code, otherwise output “city not in the list”.

Set C

a) If the file contains multiple occurrences of a given element, linear search will give the position of the first occurrence, what modifications are required to get the last occurrence?

b) If the file contains multiple occurrences of a given element, will binary search output the position of first occurrence or last occurrence?

c) Which is best case search when searching using linear search and when using binary search?

d) What modifications are required to linear search and binary search algorithm to count the number of comparisons?

e) What modifications are required to binary search so that it returns the position where x can be inserted in the sorted array to retain the sorted order?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 4: Stack

Stack is an ordered set of elements in which insertion and deletion are from one end of the stack called the top of the stack. It is a Last in First Out structure.

1. Static Implementation

A stack is implemented statically by using an array of size MAX to hold stack elements and an integer top storing the position of top of the stack. A stack is single entity that is a structure made up of both the array and the top. The stack elements can be integers, characters, strings or user defined types

The operations to be performed on a stack are

init(S) – Create an empty stack by initializing top to -1 indicating the stack is empty

push(S, x) – adding an element x to the stack S

pop (S) – deletes and returns the top element from the stack S

Peek(S) - returns the top element from the stack S without deleting the element from the stack

isEmpty(S) – Check if the stack is empty

isFull(S) – check if the stack is full which happens when top equals MAX -1

2. Dynamic Implementation

A stack is implemented dynamically by using a Linked list where each node in the linked list has two parts, the data element and the pointer to the next element of the stack. Stack is a single entity i.e. a pointer pointing to the top node in the linked list. The stack elements can be integers, characters, strings or user defined types. There is no restriction on how big the stack can grow.

The operations to be performed on a stack are

init(S) – Create an empty stack S using linked list and by initializing S to NULL indicating the stack is empty

push(S, x) – Adding an element x to the stack S requires creation of node containing x and putting it in front of the top node pointed by S. This changes the top node S and the function should return the changed value of S. The function call will be as follows

S=push(S,x);

pop (S) – deletes the top node from the stack S so that next element becomes the top. Since the top node S is changed function should return the changed value of S. The function call will be as follows

S=pop(S);

peek(S) - returns the data element in the top node of the stack S.

isEmpty(S) – Check if the stack is empty which is equivalent to checking if S==NULL

Set A

a) Implement a stack library (ststack.h) of integers using a static implementation of the stack and implementing the above six operations. Write a driver program that includes stack library and calls different stack operations.

b) Implement a stack library (dystack.h) of integers using a dynamic (linked list) implementation of the stack and implementing the above five operations. Write a driver program that includes stack library and calls different stack operations.

Set B

a) Write a function that reverses a string of characters. The function should use a stack library (cststack.h) of stack of characters using a static implementation of the stack.

b) Write a function that checks whether a string of characters is palindrome or not. The function should use a stack library (cststack.h) of stack of characters using a static implementation of the stack.

Set C

a) Assuming one already has a stack library with above six operations available, how to implement the operation that deletes the bottom (not the top) element of the stack using available stack operations?

b) In dynamic implementation of stack, How to modify pop operation so that it also returns the popped element as an argument of the pop function?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 5: Queue

Queue is an ordered set of elements in which insertions are from the rear and deletions are from the front. It is a First in First Out structure.

1. Static Implementation

A Queue is implemented statically by using an array of size MAX to hold stack elements and two integers – front and rear. The 'front' stores the position of the current front element and 'rear' stores the position of the current rear element of the queue. A queue is a single entity that is a structure made up of the array, rear and front. The Queue elements can be integers, characters, strings or user defined types

The operations to be performed on a Queue are

init (Q) – Create an empty queue by initializing both front and rear to -1 indicating the queue is empty

add (Q, x) – adding an element x to the rear end of the queue Q

delete (Q) – deletes the element from the front of the queue Q

peek (Q) - returns the front element from the queue without deleting the element from the Queue

isEmpty (Q) – check if the queue is empty, that is, when rear equals front

isFull (Q) – check if the Queue is full which happens when rear equals MAX -1

2. Dynamic Implementation

A Queue is implemented dynamically by using a Linked list where each node in the linked list has two parts, the data element and the pointer to the next element of the queue. Since Queue should be a single entity, we need to use only one external pointer while here we need two one for rear and one to the front. To avoid this we use a circular linked list and Queue pointer is pointing to the rear of the queue. Front can be easily accessed as it is next to rear. The Queue elements can be integers, characters, strings or user defined types. There is no restriction on how big the Queue can grow.

The operations to be performed on a Queue

init (Q) – Create an empty queue as a circular linked list by initializing S to NULL indicating that the queue is empty

add (Q, x) – Adding an element x to the queue Q requires creation of node containing x and putting it next to the rear and rear points to the newly added element. This changes the rear pointer Q and the function should return the changed value of Q. The function call will be as follows

Q=add(Q, x);

delete (Q) – deletes the front node from the queue Q which is actually next element to the

rear pointer Q. However if queue contains only one element, (Q→next == Q) then deleting this single element can be achieved by making empty Q (Q =NULL). Since the rear pointer Q is changed in this case, function should return the changed value of Q. The function call will be as follows

Q=delete(Q);

peek (Q) - returns the data element in the front (Q->next) node of the Queue Q.

isEmpty (Q) – Check if the Queue is empty which is equivalent to checking if Q==NULL

Set A

a) Implement a queue library (stqueue.h) of integers using a static implementation of the queue and implementing the above six operations. Write a driver program that includes queue library and calls different queue operations.

b) Implement a queue library (dyqueue.h) of integers using a dynamic (circular linked list) implementation of the queue and implementing the above five operations. Write a driver program that includes queue library and calls different queue operations.

Set B

a) A statically implemented queue may become full even if the initial positions in the array are unoccupied. To avoid this situation, a wrap around can be attempted and initial positions reused. The array can be treated as if it were circular. This can be implemented making use of mod function. Implement a queue library (cstqueue.h) of integers using a static implementation of the circular queue and implementing the above six operations.

b) A doubly ended queue allows additions and deletions from both the ends that is front and rear. Initially additions from the front will not be possible. To avoid this situation, the array can be treated as if it were circular. Implement a queue library (dstqueue.h) of integers using a static implementation of the circular queue and implementing the seven operations init(Q), isempty(Q), isFull(Q), addFront(Q,x), deleteFront(Q), addRear(Q,x) and deleteRear(Q) .

Set C

a) Write a create(Q, N) procedure which returns a dynamically implemented queue (circular linked list) containing N elements 1, 2, ... N in ascending order.

b) What should be the procedure to split a single queue into N queues so that queue principle of FIFO is not violated?

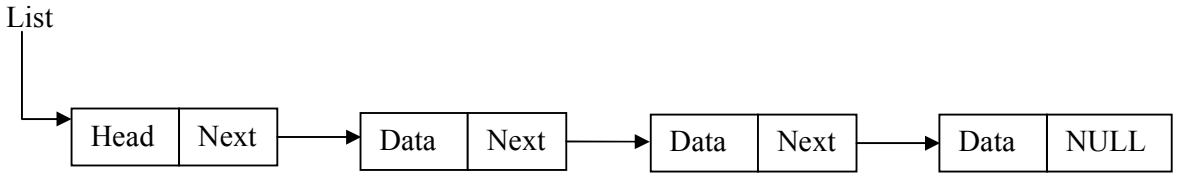
Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 6: Linked List

An abstract data type List is an ordered set of elements where insertions and deletions are possible from any position. Implementing List statically using an array to store elements is costly as insertions and deletions require moving of array elements. List is efficiently implemented dynamically using Linked list.

The linked list is a series of nodes where each node contains the data element and a link to the node containing the next element. The data element can be integer, character or user defined type. A list is a single entity which is a pointer to the first node of the linked list. A dummy node is used as header of the list so that it is not affected by insertions or deletions.



The operations to be performed on a linked list are

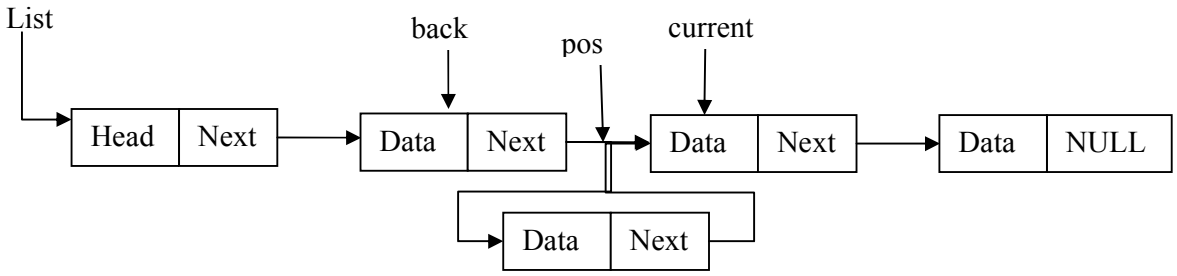
insert (L, x, pos) – inserts the data element x by creating the node containing the data element x and inserting it in position pos. The links are appropriately changed. If pos equals 1, the node is inserted in first position immediately after the header. If pos is greater than the nodes present in the list, the node is added at the end of the list.

search (L, x) – searches for the data element x and returns the pointer to the node containing x if x is present or returns NULL.

delete (L, x) – deletes the node containing the data element x by appropriately modifying the links.

display (L) – displays all the data elements in the list

In a singly linked list there is only a link to the next element. For insertion as well as deletion one need to traverse with two pointers back and current.



In a Doubly linked list there is link to the next element as well as a link to the previous element. For insertion one needs only the pointer to current element and same is true for deletion.

Set A

a) Implement a list library (singlylist.h) for a singly linked list with the above four operations. Write a menu driven driver program to call the operations.

b) Implement a list library (doublylist.h) for a doubly linked list with the above four operations. Write a menu driven driver program to call the operations.

Set B

a) There are lists where insertion should ensure the ordering of data elements. Since the elements are in ascending order the search can terminate once equal or greater element is found. Implement a singly linked list of ordered integers(ascending/descending) with insert, search and display operations.

b) There are lists where new elements are always appended at the end of the list. The list can be implemented as a circular list with the external pointer pointing to the last element of the list. Implement singly linked circular list of integers with append and display operations. The operation append(L, n), appends to the end of the list, n integers either accepted from user or randomly generated.

Set C

a) How to divide a singly linked list into two almost equal size lists?

b) The union operation of two disjoint sets takes two disjoint sets S1 and S2 and returns a disjoint set S consisting of all the elements of S1 and S2 and the original sets S1 and s2 are destroyed by the union operation. How to implement union in O(1) time using a suitable list data structure for representing a set?

c) What is the method to reverse a singly linked list in just one traversal?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 7: Tree

Tree is a recursive data structure. A Binary tree consists of a root and two disjoint binary trees called left and right trees. In Binary search tree every element is distinct and elements in the left subtree are smaller than the root and root is smaller than elements in right subtree.

The operations on binary search tree are

init (T) – creates an empty Binary search tree by initializing T to NULL

insert (T, x) – inserts the value x in the proper position in the Binary search tree

search (T, x) – searches if the value x is present in the search tree

inOrder (T) – displays the node using inorder traversal of binary search tree

postOrder (T) – displays the node using postorder traversal of binary search tree

preOrder (T) – displays the node using preorder traversal of binary search tree

Set A

a) Implement a Binary search tree library (btree.h) with above six operations. Write a menu driven driver program to call the above functions

Set B

a) Write a C program which uses Binary search tree library and implements two more functions.

Create(T, n) – inserts n nodes in the Binary search tree where the values are either accepted from user or randomly generated (use insert)

Count(T) – returns the number of nodes in the tree

Set C

a) How Count(T) function can be modified instead to count leaf nodes in the tree?

b) Write a delete(T, x) which deletes the node containing data element x if and only if it is a leaf node.

c) What is the strategy to implement delete operation in case of a non leaf node?

d) How to obtain a mirror image of a binary search tree?

e) How to find the minimum element in a Binary search tree? How to find the maximum element in a Binary search tree?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 8: Graph

A graph consists of a set of vertices and a set of edges. The two main ways of representing graphs are adjacency matrix representation and adjacency list representation.

In adjacency matrix representation of a Graph with n vertices and e edges, a two dimensional nxn array , say a , is used , with the property that a[i,j] equals 1 if there is an edge from i to j and a[i,j] equals 0 if there is no edge from i to j.

In adjacency list representation of a graph with n vertices and e edges, there are n linked lists, one list for each vertex in the graph.

The usual operations on graph are:

Indegree(i) – returns the indegree (the number of edges ending on) of the i^{th} vertex

Outdegree(i) – returns the outdegree(the number of edges moving out) of the i^{th} vertex)

displayAdjMatrix – displays the adjacency matrix for the graph

Set A

a) Write a C program that accepts the vertices and edges for a graph and stores it as an adjacency matrix. Implement functions to print indegree, outdegree and to display the adjacency matrix.

b) Write a C program that accepts the vertices and edges for a graph and stores it as an adjacency list. Implement functions to print outdegree of any vertex i.

Set B

a) Write a C program that accepts the graph as an adjacency matrix and converts it to adjacency list representation. Write a function to display the graph in adjacency list form.

b) Write a C program that accepts the graph as an adjacency matrix and checks if the graph is undirected. The matrix for undirected graph is symmetric

Set C

a) What can be concluded about the directed graph if there is no vertex with indegree zero?

b) In the adjacency list representation the dummy head node of each linked list can be used to store the indegree of the vertex when the adjacency list is created. Modify the program so that every time an edge node is added to the list the head node entry is incremented. Write a function to print indegree of vertex i.

c) A graph may not have an edge from a vertex back to itself(self edges or self loops). Given an adjacency matrix representation of a graph, how to know if there are self edges?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Data Structure Case studies

Case studies are to be solved using following steps

Step 1 – Formulation of the problem

Step 2 – Choice of variables and appropriate data structures

Step 3 – Choice of Algorithms

Step 4 – Implementation of solution

Step 5 – Validation

1. Banks often record transactions on an account, in order of the times of the transactions, but many people like to receive their bank statements with cheques listed in order by cheque number. People usually write(use) cheques in order by cheque number, and merchants usually cash them with reasonable dispatch. Thus few cheque numbers are usually out of order. Use an appropriate sorting algorithm for converting time of transaction ordering to cheque number ordering. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

2. Given a data set consisting of n integers, a five point summary is to be produced consisting of Minimum, Maximum, Median, 1st and 3rd Quantile. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

3. A spell checker is a program that looks at a document and compares each word in the document to words stored in a dictionary. If it finds words in the dictionary, it moves on to the next word, If it does not find the word, it reports the user about the misspelled(possibly) word. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

*4. A postfix expression of the form ab+cd-*ab/ is to be evaluated after accepting the values of a, b, c and d. The value should be accepted only once and the same value is to be used for repeated occurrence of same symbol in the expression. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

5. An Infix expression of the form a(b+c)*((d-a)/b) need to be converted to postfix form using usual precedences of operators. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

6. Suppose that we are selling the services of a machine. Each user pays a fixed amount per use. However the time needed by each user is different. We wish to maximize the returns from this machine under the assumption that the machine is not to be kept idle unless no user is available. Whenever the machine becomes available, the user with the smallest time requirement is selected. When a new user requests the machine, he has to wait if there are pending requests. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

7. Suppose that we are selling the services of a machine. Each user uses the machine for a fixed amount of time. However the people are ready to pay different amounts for the service. We wish to maximize the returns from this machine under the assumption that the machine is not to be kept idle unless no user is available. Whenever the machine becomes available, the user with the highest paying amount is selected. When a new user requests the machine, he has to wait if there are pending requests. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

8. A factory has many machines and many jobs that require processing on some of the machines. Each job has a job card which contains all the attributes of a job including the machines on which it need to be processed and the processing time. For simplicity let us assume that there are ten machines numbered 0,1, 2...9 , each job is processed for fixed amount of time and the job card number is coded in such a way that the machine requirements can be accessed. For example if job card number is 5438, it means job requires processing on machines 5, 4, 3 and 8. The job can be processed simultaneously on these machines. However each machine can process jobs one at a time and takes up the new job when it becomes idle. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

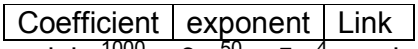
0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

9. A file manager identifies each file using an inode-no. A directory is also a file identified by an inode-no. Directory is a list of files each represented by an inode-no. The list has the header containing the inode-no of the parent that is the directory itself. A separate table indexed by inode-no maintains attributes of files including the name of the file. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

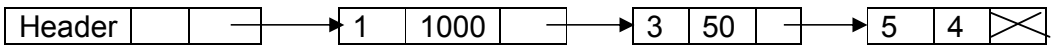
Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

*10. Polynomials with varying degree can be represented using linked list. A term in a polynomial is represented by a node with structure.



For example a polynomial $x^{1000} + 3x^{50} + 5x^{4}$ can be represented as



The number of nodes required to represent the polynomial is same as the number of terms in the polynomial with an additional node for the header. The nodes are maintained in the descending order of exponents. One need to accept, display polynomial and also perform other operation on polynomials such as addition. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

11. The Josephus problem is the following game. N people, numbered 1 to N are sitting in a circle. Starting at person 1, a hot potato is passed, After M passes, the person holding the hot potato is eliminated, the circle closes ranks, and the game continues with the person who was sitting after the eliminated person picking up the hot potato. The last remaining person wins. Thus, If M=0 and N=5, players are eliminated in order, and player 5 wins. If M=1 and N=5, the order of elimination is 2, 4, 1, 5 and the player 3 wins. The Josephus problem needs to be solved for general values of M and N (N > 10000). Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

12. Consider the database of books maintained in a library system When a user wants to check whether a particular book is available, a search operations is called for. If the book is available and is issued to the user, a delete operation can be performed to remove this book from the set of available books. When the user returns the book, it can be inserted back into the set of available books. It is essential that we are able to support the above mentioned operations as efficiently as possible as since these operations are performed quite frequently. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

13. An Address book contains the name and other details of friends. User wants to check for a particular name and get the details such as phone number or email address of his friend. New Addresses are added to the address book. The contact information may also require frequent updation. Certain important contact numbers (a small number) which are frequently required should be accessible very fast. These contacts may be important (favourites) at some point in time but may not continue to remain so and have to be removed. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

*14. In breadth first search (BFS) of a Graph we start at vertex v and mark it as having been visited. All unvisited vertices adjacent from v are visited next. The v is thus completely explored. The visited but at unexplored vertices are taken up next for exploration. Exploration continues until no unexplored vertex is left. If BFS is used on a connected undirected graph G, then all vertices in G get visited and the graph is completely traversed. Thus BFS can be used to check whether graph is connected. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

*15. In depth first search of a Graph we start at vertex v and mark it as having been visited. The exploration of the vertex involves visiting all the adjacent vertices however the exploration of a vertex is suspended as soon as a new vertex is reached and the exploration of a new vertex begins. When the exploration of new vertex is over, the exploration v is resumed. DFS can be best implemented as a recursive function. A topological sort of a directed acyclic graph is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering. For topological ordering DFS is called on the graph and every vertex explored is added onto the front of a linked list that forms the topological order. Formulate the problem and write a C program to solve the problem by using appropriate data structures and algorithms.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Section II – Object Oriented Programming Concepts using C++

Assignment 9: Class, Object and methods implementation

We will illustrate C++ program structure, using a sample program. It will be also used to bring out differences in C++ and C.

C++ is an object Oriented programming language and thus classes, objects and methods form the essence of the program structure.

C++ is a modular programming language thus program is divided into several files which can be separately compiled to form the object file. A program is divided into header files, with .h as extension containing class and method declarations and code files with .cpp extension containing method implementations and driver function

The following programs is greeting employees and employee being a separate entity, we define a class for the same

Header File employee.h	Code file employee.cpp	Code file greet.cpp

using namespace std;
class employee{
int id;
int salary;
static int number;
public:
employee();
void setSalary(int);
void display();
};
	#include "employee.h"
	#include <iostream>
	using namespace std;
	employee::employee()
	{ id = number++;
	}
	void employee::setSalary(int val)
	{ salary=val;
	}
	void employee::display()
	{
	cout << "\nIdentification no : " << id ;
	cout << "\nSalary : " << salary << "\n";
	}
	int employee::number=1;
		#include <iostream>
		#include "employee.h"
		using namespace std;
		int main()
		{
		char command;
		employee e, e1;
		cout << "Welcome to the Program\n";
		do{
		cout << "Enter 'g' to greet\n";
		cout << "Enter 'q' to quit \n";
		cout << "Enter command : ";
		cin >> command;
		e1.setSalary(50);
		switch(command){
		case 'g': cout << "Hello\n"; e1.display(); break;
		default: e.display(); break;
		}
		} while (command !='q');
		return 0;
		}

1. Each of the above code snippets start with include pre-processor directive. By including iostream, the input output library, we are able to use predefined stream object ‘cout’ which is useful for generating output. The user defined class Employee is available in main.cpp and employee.cpp by using include pre-processor directive

2. The next line is

using namespace std;

The names of user defined types and variable names used in various library files, form the namespace of a program. C++ allows grouping of such variable names into namespaces

identified by the group name. Thus if there is duplication of name leading (user defined and library) to conflict, it can be resolved using the groupname along with the variable name.

3. In C++ class is used to define user defined data types. Several object instances of class can be used by defining them as we define other data types. Here employee class has both data members and function members.

4. Access qualifiers allow data members and functions to be hidden, if necessary from external use. Data members of employee class as defined above are private (by default) while function members are public.

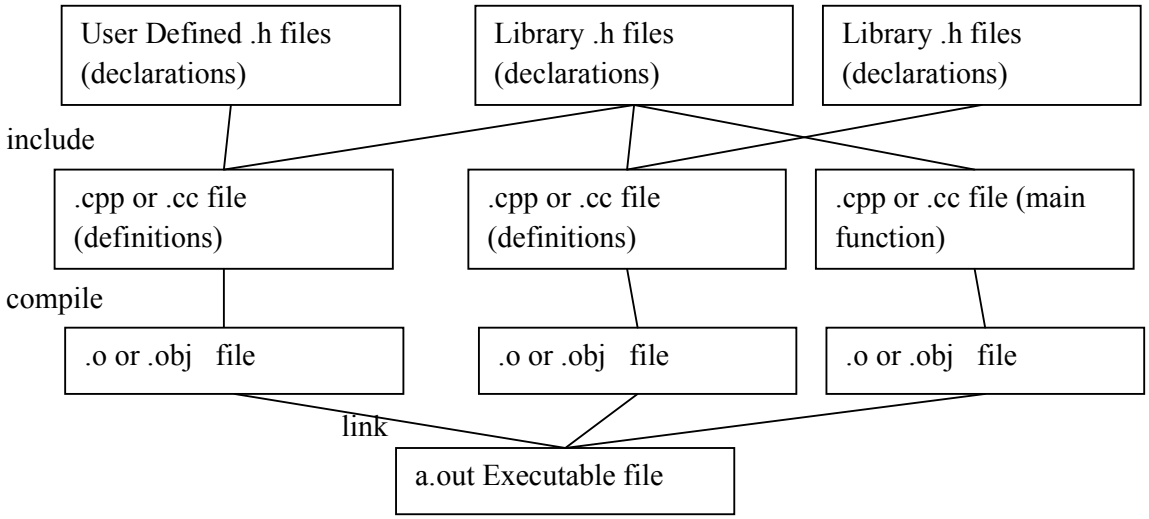
5. The variables e and e1 defined in main function are instances of class employee. Each object instance will have its own set of variables however we sometimes need to have variables which are common to all object instances. In C++ such variables need to be declared as static. In employee class number is one such variable. Static variables have to be initialized outside of the class definition. It can be inside a class declaration or outside. The value of the static variable is available throughout. The static variable number of employee class is initialized in employee.cpp by using the statement

int employee::number=1;

6. The constructor is having the same name as that of the class. Here the constructor increments the static variable.

7. Only the member functions can have access to the private data members and private functions. However, the public members can be accessed from outside the class. If the data members of a class are private, one need to have public function members to get or set the values of these data members. Here setSalary function sets the value of the salary variable. The display function uses input output library object cout to output the data members.

8. The main function uses both cin and cout stream library objects to perform input and output.



The general program structure is as above. The compilation and linking can be carried out using g++ compiler

$ g++ employee.cpp employee.o
$ g++ main.o employee.o trial
$ ./trial
	$ g++ employee.cpp employee.o
	$ g++ main.o employee.o
	$ ./a.out

Set A

a) Write the definition for a class called 'time' that has hours, minutes & seconds as integer data members. The class has the following member functions:

void settime(int, int, int) to set the specified values of hours, minutes and seconds in object

void showtime() to display contents of time object

time add(time) add the corresponding values of hours, minutes and seconds (<60) in time object argument to current time object and make appropriate conversions and return time

time diff(time) subtract values of hours, minutes and seconds in time object argument from current time object after making appropriate conversions and return time difference

Write a main program to illustrate the use of above class.

b) Write the definition for a class called "cuboidSolid" that has length, breadth, height and mass has float data members. The class has the following member functions.

Setters and getters for each of the data members

float getVolume() that returns the volume of the metal

float getSurfaceArea() that returns the surface area

float getDensity() that returns the density

Write a main program to illustrate the use of above class.

Set B

a) Define a class account with following specifications

private data members

account number – automatically generated six digit account number, first two digit are used for bank code (assume the value 82) and next four digits for account number

account type – it can be one of the following type { saving, current, fixed, recurring}

amount – long integer for the balance amount

Owner Name – name of the owner

Public data members for

Setting and getting account type, initial amount and Owner Name

Display account information

Write a main program to accept information from user and open at least five accounts and display their information.

b) Implement a class 'RomanNumeral' which stores the RomanNumeral as a string. Write a member function getDecimal which returns the decimal value of the Roman numeral. The decimal values of roman numerals are as follows M 1000, D 500, C 100, L 50, X 10, V 5 and I 1. Write the program and create the roman numerals such as VIII, MCXIV , MCDLXVI and print their decimal values.

c) Write the definition for a class called Rectangle that has floating point data members

length and width. The class has the following member functions:

void setlength(float) to set the length of data member

void setwidth(float) to set the width of data member

float perimeter() to calculate and return the perimeter of the rectangle

float area() to calculate and return the area of the rectangle

void show() to display the length and width of the rectangle

Write main function to create two rectangle objects and display each rectangle and its area and perimeter.

Set C

a) What purpose is served by using const keyword with a member function? Can it be used with a static member function?

b) How to write a function for employee class which can tell us whether any instance of employee has been created or not?

c) Having declared two variables t1 and t2 of type time, what happens if we write the following statement? Justify.

t1=t2;

d) An Accessor function of a class is a member function that only accesses but does not modify the data members of the class. How to declare function showtime() or getVolume() to safeguard them from modifying?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 10: Constructor: Copy, Default & Parameterized Constructor

A constructor is special member function that creates an object. It is special because its name is the same as the class name. Constructors are declared as public member functions and do not have return types.

Constructors are invoked explicitly using new operator as in

cuboidSolid cubemetal = new cuboidSolid(5, 5, 5, 520);

Constructors are invoked implicitly when objects are declared as in

CuboidSolid cubemetal(12, 3, 4, 233);

There can be multiple ways of creating an object so there are different constructor types.

Default Constructor -It does not accept any parameters. Purpose of this, is only to create object. It initializes data members to predefined default values.

Parameterized Constructor - It accepts any number of formal parameters and uses these parameters to initialize the objects.

Copy Constructor - It creates exact copy of an object. It takes a reference to an object of the same class as itself as argument. It copies data from one object to other by copying every member of an object with the member of object passed as argument. An object passed to copy constructor is of const type as constructor is not supposed to change the contents of the object to be copied.

```
cuboidSolid (const cuboidSolid &  cubemetal)

{  length =cubemetal.length;

 ……

}
```

Set A

a) Define a class 'Fraction' having integer data members numerator and denominator. Define parameterized and default constructors (default values 0 and 1). Parameterized constructor should store the fraction in reduced form after dividing both numerator and denominator by gcd(greatest common divisor). Write a private function member to compute gcd of two integers.

Write four member functions for addition, subtraction. multiplication and division of fraction objects. Each function will have two fraction objects as arguments. Write the main function to illustrate the use of the class.

Set B

a) Design a class Book with the data members to hold title, number of authors, ISBN number, price and number of copies. The title and ISBN number are pointer to characters Define parameterized and default constructors(number of authors and number of copies

equal to 1). Write getters and setters for each of the member functions. Write the copy constructor.

Write a member function to check the validity of the ISBN number; it is a unique number assigned to a book which can be a 10 digit or 13 digit number. For 10 digit ISBN number, the sum of all the 10 digits, multiplied by its integer weight, descending from 10 to 1, or ascending from 1 to 10, is a multiple of 11.

$10x_1+9x_2+8x_3+7x_4+6x_5+5x_6+4x_7+3x_8+2x_9+1x_{10} \equiv 0 \bmod 11$

For 13 digit ISBN number, the last digit is a check digit which must range from 0 to 9 and sum of all the thirteen digits multiplied by weights alternating between 1 and 3 is a multiple of 10

$x_1+3x_2+x_3+3x_4+x_5+3x_6+x_7+3x_8+x_9+3x_{10}+x_{11}+3x_{12}+x_{13} \equiv 0 \bmod 10$

Call the function in the constructor before initializing the ISBN number.

Set C

a) Write the copy constructor which apart from memberwise copying outputs the message that the constructor is called. What are the situations when the copy constructor is automatically invoked?

b) Why copy constructor accepts reference to an object and not the object itself? What happens if we do otherwise?

c) A copy constructor is always implicitly defined for a class by the compiler. When it needs to be explicitly defined for the class?

d) A Mutator is a member function that modifies the data members of the class. A class can have more than one mutators. Justify.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 11: Memory Allocation: new and delete operators, dynamic constructor

C++ provides two operators new and delete for dynamic memory usage. The new operator allocates memory from free store while the delete operator returns the allotted memory to free store.

For built in types we use them as follows

```
int * p, *arr ;
p =new int;
arr = new int[30];
delete p;
delete [] arr; // we need not specify the size
```

For user defined types it is similar

```
fraction *single, *vector;
single = new fraction;
vector = new fraction[30];
```

Set A

a) Implement a class vector which contains integers in sorted order. The size of the vector varies so the memory should be dynamically allocated. It should have three data members vectorarray – a pointer to integer, maxsize – the maximum allocated size to take care of insertions and size – actual size. Write member function to get and set value at a particular position in vector, to insert values in vector to keep it in sorted order, print the vector. Implement the copy constructor. Write member functions to form union and intersection of vectors.

Set B

a) Implement a class 'sequence' which contains a sequence of strings. The size of each string varies so also the number of strings in a sequence hence memory should be dynamically allocated. The constructor should accept the number of strings and each of the strings to be added to the sequence. Write member function to append a new string in the sequence, print the sequence, and search a string in the sequence. Implement the copy constructor. Write a function reverse that reverses every string in the sequence. Write a friend function intersection that takes two sequences and returns a sequence containing common strings.

Set C

a) When does a destructor need to be explicitly defined for a class? Write destructor for vector class.

b) Define parameterized constructor for vector class to initialize maxsize and size. Use this to create a vector object dynamically.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 12: Inline function, friend function, default argument,

Inline function

To save the execution time that gets lost in the overheads of calling a function, the compiler can be instructed to insert the code wherever function is called.

An **inline** function is a function whose code is expanded in line at the point at which it is

invoked, rather than being called.

There are two ways to create an **inline** function. The first is to use the **inline** qualifier.

inline int f()

{

.....

}

There is another way to create an inline function. This is accomplished by defining the code of a member function inside a class declaration (.h file).

Friend function

If a member of a particular class is private or protected, functions outside the class cannot access the non-public members of the class. This is the primary objective of encapsulation. However, at times, it becomes a problem for the programmer. In order to access the nonpublic members of a class, C++ provides the friend keyword.

Any non-member function may be declared a **friend** by a class, in which case the function may directly access the private member attributes and methods of the class objects.

Default arguments

In C++, you can give a parameter a default value that is automatically used when no

argument corresponding to that parameter is specified in a call to a function.

For example the constructor for the class fraction can be declared with default arguments

fraction (int num=0, int denom=1);

and the definition will be

fraction(int num , int denom)

{

numerator=num/gcd(num, denom);

denominator = denom/gcd(num, denom);

}

Function can be called with zero, one or two arguments. Missing arguments must be the trailing arguments and get the default values. Note that if one argument is missing when the function is called it is assumed to be last argument

Set A

a) Implement a class date with three integer data members day, month and year. Write inline function to compress date into a single integer, to count the difference in number of days between the date and the one passed as argument. The date can be an unsigned integer (16 bits) in the following manner year: bits 15-9, month: bits 8-5 and day bits 4-0.

b) Implement a class Message having two data members a pointer to character and an integer storing length of the string. Implement a class Key having two data members a character array of size 30 and an integer storing actual length of the string. The restrictions on Key are that the length should be less than 30 but minimum size 8 and having atleast one digit and one upper case character in it. Write a friend function to encrypt a Message using key (use some encryption algorithm)

c) Implement a class TalkingInterface with following private member functions. Define a member function greet which displays a greeting 'n' number of times.

greet(); //displays "Hi" once

greet("Namaste"); //displays "Namaste" once

greet("hello", 10); //displays hello 10 times

Define a member function tables

Tables() ;// displays table from 1 to 10

Tables(2); // displays table of 2 from 1 to 10

Tables(2, 3) // displays table of 2 from 3 to 10

Tables(2, 3, 6) // displays table of 2 from 3 to 6

Declare TalkingInterface as a friend class of Person or Employee class and use these functions.

Set B

a) Implement a class Time12 which stores time in 12 hr format (hh, mm, ss, am/pm). It has four data members, three integers for hour, minutes and seconds and a character array of size 2 storing am/pm. Define constructor with default arguments (midnight). Define accessors and mutators as inline functions. Implement a second class Time24 which stores time in 24 hr format (hh, mm, ss). It has three data members, three integers for hours, minutes and seconds. Define accessors and mutators as inline functions. Write friend functions to compare time in 12 hr format to time in 24 hr format, to convert time in 12 hr format to 24 hr format and vice versa.

Set C

a) What purpose is served by defining a class as friend? Is friendship between classes mutual?

b) If one argument is missing when the function with default arguments is called, which argument will get the default value?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 13: Function Overloading.

Function overloading is a kind of polymorphism in which there can be several functions with same name but different set of parameters, all functions conceptually carrying out the same task but there is variation depending on the arguments. The definition of functions should differ from each other by type and /or number of arguments in the argument list. One commonly found overloaded function in a Class is the constructor itself as there can be more than one possible ways of constructing an object.

Set A

a) Implement a class 'printdata' with three member functions all with the same name 'print'
   void print(int) - outputs value - <int>, that is, value followed by the value of the integer
   void print (int, int) – outputs value – [<int>, <int>], that is, value followed by the two integers separated by comma in square brackets.
   void print(char *) – outputs value –"char*", that is, value followed by the string in double quotes.
   Write a main function that uses the above class and its member functions.
b) Implement a class 'maxdata' with two member functions both with the same name 'maximum'
   int maximum (int, int) – returns the maximum between the two integer arguments
   int maximum (int *) – returns the maximum integer in the array of integers
   Write a main function that uses the above class and its member functions.

Set B

a) Implement a class 'invertdata' with three member functions all with the same name 'invert'

int invert (int) - returns the inverted integer – invert(5438) will return 8345

char * invert (char *) – returns the reversed string – reverse("comp") will return "pmoc"

void invert(int *) – will reverse the array order – An array [5, 7, 12, 4] will be inverted to

[4, 12, 7, 5]

Set C

a) Can we have two functions with same name and set of arguments but different return data types?

b) Can we have two functions with same name with two different data types, for one function the argument is int for other it is user defined type 'Number' which is basically int but renamed 'Number' using typedef?

c) Can we have two functions void f(int x) and void f(int & x), that is, one with integer argument and other with reference to integer argument?

d) Can we have two functions given as follows?

void value(float x) { cout << “float ” << x << endl; }

void value(double x) { cout << “ double” << x << endl;

What will be printed for the call

value(10) ;

value (10.1);

e) Can we have two functions with same name, one having integer as argument while other enumerated data type as argument?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 14: Operator overloading.

Operator overloading gives special meaning to usual operators like +, *, < etc. for user defined types similar to the one they have with built in data types. There are few operators in C++ that cannot be overloaded such as ternary operator ?:, sizeof, scope resolution operator :: and membership operators . and .* .

Operator overloading is carried out by writing member functions using the operator keyword called operator functions.

The general syntax of an operator function is

return-type operator operatorsymbol (parameter list)

{

}

Here return-type is commonly the name of the class itself as the operations would commonly return object of that class type.

The argument list will depend on whether the operator is unary or binary and whether the function is a member function or friend function. For example for a unary operator, member function will have no arguments as the class object itself is the object on which operator operates. For a binary operator, a non member function will have two arguments while a member function has one argument, the other implicitly being the class object itself.

```
fraction operator + ( fraction f)

{ fraction temp;

 temp.numerator = numerator*f.denominator +denominator*f.numerator

 temp. denominator = denominator*f.denominator;

 temp.numerator = temp.numerator/gcd(temp.numerator, temp.denominator);

 temp.denominator = temp.denominator/ gcd(temp.numerator,temp.denominator);

 return temp;

}
```

Overloading increment and decrement operators

The increment operator ++ has two forms: pre-increment (++u) and post-increment(u++). To distinguish between pre and post increment operator overloading, we use a dummy parameter of type int in the function heading of the post-increment operator function. Decrement operator can be overloaded similarly.

```
fraction operator++()
```

```
{
numerator+=denominator;// add one to the fraction, that changes the numerator
return *this; // return the incremented value
}
fraction operator ++(int)
{
fraction temp=*this; // copy the value before increment
numerator+=denominator; // add one to the fraction, that changes the numerator
return temp;// return the old value of the object
}
```

Overloading insertion and extraction operators

Overloading insertion(<<) operator and extraction (>>) operator is a must when you want to input and output objects using stream class library in the similar fashion as built in data types. For a class fraction we should be able to do the following

```
fraction f1, f2(2,3);
cin >> f1;
cout << f2;
```

Since the first operand is iostream object, the operator function will be friend function and the declarations will be

```
friend ostream& operator<< (ostream &out, fraction &fract)
{
out << fract.numerator <<"/" << fract.denominator ;
return out;
}
friend istream& operator >> (istream &in, fraction &fract)
{
 in >> fract.numerator ;
 in >> fract.denominator;
return in;
```

```
}
```

Overloading the Assignment operator =

Assignment operator does member wise copying and built in assignment operator function works well except with the classes having pointer data members. In such cases one must explicitly overload assignment operator.

The vector class has dynamic data member hence the assignment operator need to be explicitly overloaded

```
const vector & operator = (const vector & rightvector)

{ if (this != rightvector)

{  delete [] vectorarray;

   maxsize= rightvector.maxsize;

  size = rightvector.size;

  vectorarray= new int [maxsize];

  for(int i=0; i< size; i++)

  vectorarray[i]=rightvector.vectorarray[i];

}

return *this;

}
```

Overloading the Subscript operator []

The function to overload the operator [] for a class must be the member of the class. Furthermore, because an array can be declared as constant or nonconstant we need to overload the operator [] to handle both cases.

```
int & operator[](int index)

{

 assert( 0<= index && index <size);

 return vectorarray[index];

}

const int & operator[](int index) const

{

 assert( 0<= index && index <size);
```

```
 return vectorarray[index];

}
```

Set A

a) Define a class named Complex for representing complex numbers. A complex number has the general form a + ib, where a- the real part, b - the imaginary part are both real numbers and i^2=-1. Define parameterized and default constructor. Overload +, - and * operators with usual meaning.

b) Define a class named Clock with three integer data members for hours, minutes and seconds. Define parameterized and default constructors. Overload increment and decrement operators appropriately. Overload extraction and insertion operators.

Set B

a) Define a class Message with two data members one character pointer and an integer storing length. Overload operator binary + to represent concatenation of messages, [] to return a character at a specific position and = to copy one Message object to another.

b) A Matrix has rows and columns which decide the number of elements in the matrix. We will implement a matrix class that can handle integer matrices of different dimensions. We can overload addition, subtraction and multiplication operator to carry out usual matrix addition, subtraction and multiplication.

Set C

a) Why return type in case of assignment operator function is a constant reference?

b) Is it possible to overload new and delete operators?

c) Why are << and >> operators overloaded as friend functions?

d) Which operator should be overloaded to convert the Message to uppercase?

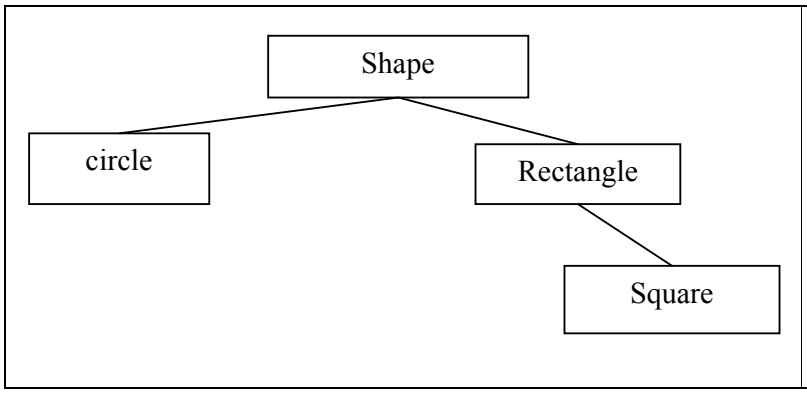
Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 15: Inheritance

Inheritance is the process of creating new classes from an existing class. The existing class is known as **base class** and the newly created class is called a **derived class**. The derived classes inherit the properties of base classes.

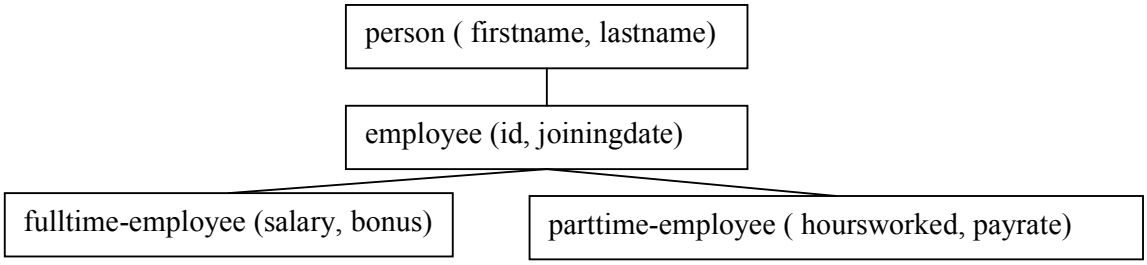
If the derived class inherits from a single parent, the inheritance is said to be single inheritance. Multiple inheritance is the process of creating a new class from more than one base classes. Inheritance can be viewed as a hierarchical structure where base class is at root shown with its derived classes at different levels




```
class shape
{……
};
class circle: public shape
{…….
};
class rectangle : public shape
{…….
};
Class square : public rectangle
{……
};
```

Set A

a) Given the following inheritance hierarchy, implement each of the classes.



The person class has parametrized constructors and getters and setters. The employee id is auto generated from the last allotted value stored in employee class. Write constructors for derived and getters and setters. Define member function computepay for employee and override appropriately in derived class. Overload extraction and insertion operators in base class and override them in the derived classes. Write main program to illustrate use of the classes

b) Implement the following class hierarchy.

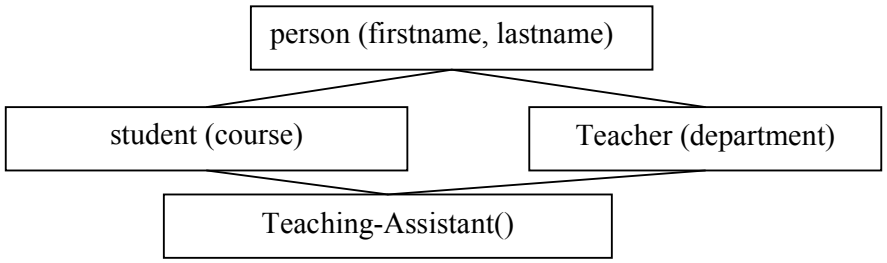
Student: id, name,
StudentExam (derived from Student): Marks of n subjects (n can be variable)
StudentResult (derived from StudentExam): percentage, grade

Define a parameterized constructor for each class and appropriate functions to accept

and display details. Create n objects of the StudentResult class and display the marklist using suitable manipulators.

Set B

a) Implement the following class hierarchy



Define constructors and appropriate functions to accept and display details. Write a program to accept details of 'n' TA's and display the details.

b) A book(ISBN) and CD(data capacity) are both types of media(id, title) objects. A person buys 10 media items, each of which can be either book or CD. Display the list of all books and CD's bought. Define the classes and appropriate member functions to accept and display data. Use pointers and concepts of polymorphism (virtual functions).

Set C

a) When is it mandatory for a derived class to define a constructor?

b) What is the purpose of a pure virtual function?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Assignment 16: File handling

To perform input and output from files, C++ provides the fstream library. It defines several classes including ifstream and ofstream.

We connect stream object to the physical file by creating a stream object by specifying the name of the file as argument.

```
#include <fstream>

ofstream outfile("output");

ifstream infile("input");
```

Alternatively the file can be connected by using the open member function with one or more arguments

```
#include <fstream>

ofstream outfile;

ifstream infile;

infile.open("input");

outfile.open("output", ios::  app | ios :: nocreate);

// do not create a new file, append at the end of file
```

The file should be disconnected by using close member function. To access and manipulate contents of the file, get, put, read and write methods are provided.

Set A

a) Implement a class "file" with four data members, a string storing the name of the file and integers storing the number of characters, words and lines in the file. Write the following member functions

The constructor takes only the file name as the argument and opens the file, counts the characters, words and lines and initializes the other data members. If the file does not exist creates a blank file with the name and other data members are initialized to zero. Write getters for all the data members.

Write additional member functions that count number of blank lines, searches for occurrence of word in the file.

Set B

a) The file 'cities.txt' contains names of cities and their STD codes. Define a class 'city' with data members name and STD code. Write a program that declares a city array and reads the data from file cities.txt into the array. Output the list of names and STD codes,

Set C

a) Which functions are used for random access to a file?
b) How would you find out the size of a file in bytes using functions for random access to file?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Case studies in C++

1. A bit vector is a vector with binary elements, that is, each element is either a 0 or a 1. Bit vectors are used in several applications such as resource allocation. Small bit vectors are conveniently represented by unsigned integers. For example, an unsigned char can represent a bit vector of 8 elements. Larger bit vectors can be defined as arrays of such smaller bit vectors. Complete the implementation of the Bitvec class, as defined below. It should allow bit vectors of any size to be created and manipulated using the associated operators.

```
class BitVec {
        unsigned char *vec; // vector of 8*bytes bits
        short bytes; // bytes in the vector
public:
        BitVec (const short dim);
        BitVec (const char* bits);
        BitVec (const BitVec&);
        ~BitVec (void) { delete vec; }
        BitVec& operator = (const BitVec&);
        BitVec& operator &= (const BitVec&);
        BitVec& operator |= (const BitVec&);
        BitVec& operator ^= (const BitVec&);
        BitVec& operator <<= (const short);
        BitVec& operator >>= (const short);
        int operator [] (const short index);
        BitVec operator ~ (void);
        BitVec operator & (const BitVec&);
        BitVec operator | (const BitVec&);
        BitVec operator ^ (const BitVec&);
        BitVec operator << (const short n);
        BitVec operator >> (const short n);
        bool operator == (const BitVec&);
        bool operator != (const BitVec&);
        friendostream& operator << (ostream&, BitVec&);
};
```

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

2. Matrices have wide utility in variety of applications. A Matrix has rows and columns which decide the number of elements in the matrix. We will implement a matrix class that can handle integer matrices of different dimensions. We can overload addition, subtraction and multiplication operator to carry out usual matrix addition and multiplication.

```
class Matrix {
        int  *vec; // storing matrix elements in column major form
        int rows; // number of rows
        int columns;// number of columns
public:
```

```
        Matrix (const int row=1, const int column=1);
        Matrix (const Matrix&); // copy constructor
        ~Matrix (void) { delete vec; }
        Matrix operator + (const Matrix&);
        Matrix& operator += (const Matrix&);
        Matrix operator - (const Matrix&);
        Matrix& operator -= (const Matrix&);
        Matrix operator * (const Matrix&);
        Matrix& operator *= (const Matrix&)'
        friend ostream& operator << (ostream&, Matrix &);
        friend istream & operator >>(istream&, Matrix&);
};
```

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

3. A credit card has information like cardNumber, CVV number, type (VISA/MasterCard etc), ownerName, DOB, Address, balance. A card can be of various types: SILVER, GOLD and PLATINUM. The spending limit and privileges (late fees, payment deadlines) vary according to the card type. SILVER (limit- 35000, late fee – 250, deadline- 15 days), GOLD(limit- 85000, late fee – 150, deadline- 20 days), PLATINUM (limit- 1,50,000, late fee – 100, deadline- 30 days). The credit statement is generated on the 15^{th} of each month. Design classes to represent the above and write a program to perform transactions (Spend, repay) on each card type.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

4. Lists are commonly used to store ordered set of elements having frequent insertions and deletions. Create a class to represent a singly linked list of integers.

```
class SinglyList
{
  Node * head;
  void create();
  void display();
  void insert(int num);
  void delete(int pos);
};
```

The Node class has data members info and next to

```
class Node
{
```

```
    int info;
    Node *next;
};
```

Write a menu driven program to perform operations on the singly linked list. Use concept of friend class.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

5. Roman Numerals are commonly used in numbering chapters and are frequently used in old literature. Implement a class 'RomanNumeral' which stores the RomanNumeral as a string and also its decimal value. Write a private member function to convert Roman number to decimal and vice versa. Overload the parameterized constructor, the argument can be string representing Roman numeral or integer representing its decimal value. Overload the arithmetic operators +, * so that arithmetic operations can be performed on Roman Numbers. Perform the operations on the decimal representation and then convert it to Roman numeral form Overload, the pre and post decrement and increment operators. Overload relational operators <, >, <=, > = and = to compare the Roman numbers.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

6. Implement a class stockObject that captures various characteristics of a stock. The main components of a stock are the stock symbol, stock price, number of shares, opening price, closing price, high price, low price, previous price, and % loss/gain for the day and all this information should be stored in a stock object.

Function members should be provided to perform the following operations

a. Set the stock information
b. Print the stock information
c. Calculate percent gain/loss
d. Display the different prices
e. Natural listing for stock listing is by stock symbol(alphabetical ordering). Overload the relational operators to compare two stock objects by their symbols.
f. Overload the insertion operator << for easy output
g. The data is usually stored in file in the following format
   Symbol openingprice closingprice todayhigh todaylow prevolume numberofshares
   For example sample data is
   MSMT 112.50 115.75 116.50 111.75 113.50 6723823
   CBA 67.50 75.50 78.75 67.50 65.75 378233
   Overload the stream extraction operator for easy input

Write a program that reads the data from the file into an array of stock objects, sorts and prints them.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

7. A complex number c = a + bi consists of two parts, the real part a and the imaginary part bi, where $i^2 = -1$, with i the square root of -1. We can perform arithmetic operations (+, -, *, /, or addition, subtraction, multiplication, and division) on complex numbers:
+: (a1 + b1j) **+** (a2 + b2i) = (a1 + a2) + (b1 + b2)i
-: (a1 + b1j) – (a2 + b2i) = (a1 – a2) + (b1 – b2i)
*: (a1 + b1i) * (a2 + b2i) = (a1a2 – b1b2) + (a1b2 + a2b1) i
/: (a1+b1i) / (a2 + b2i) = (a1 + b1i) * (a2 – b2i) / ($a2^2$ + $b2^2$)
= [(a1a2 + b1b2) + (a2b1 – a1b2)i] / (a22 + b22)

In polar representation a complex number z is represented by two parameters **r** and **Θ**. Parameter **r** is the modulus of complex number and parameter **Θ** is the angle with the positive direction of x-axis, thus z= r(cos θ+isinθ). The partial class declaration is given below.

```
class complex {

private:
double rP; // real part
double iP; // imaginary part
double rad; // radius
double angle ; // argument

public:

complex (double real=0.0, imaginary=0.0); //constructor with default arguments
complex (complex&); // copy constructor
complex  operator + (const complex &);
complex operator – (const complex 7);
complex operator * (const complex 7);
complex operator / (const complex 7);
double getPolarRadius();
double getPolarAngle();
void displayPolar();
….
}
```

Overload insertion and extraction operators. Also overload comparison and assignment operator.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

8. Define a class which represents a file object as shown below and overload operators to perform file operations:

```
class MyFile
{
   fstream file;
   char *filename;
   .....
};
```

Operators :

<< display contents of file

>> write data to file

+ concatenate one file to another

- intersection of two files

< > == compare two files

= copy one file to another

[] display specific character from file

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

A**ttachment List**

1. employee.txt

2. cities.txt

3. dictionary.txt

4. doc1.txt

5. doc2.txt

6. stock.txt

Bibliography

1. Fundamentals of Computer Algorithms – By Ellis Horowitz, Sartaz Sahnj and Sanguthevar Rajasekaran

2.Introduction to algorithms – By Thomas Cormen , Charles Leiserson and Ronald Rivest

3. Data structures using C Lab Book – By Madhuri Ghanekar, A. S. Bachav, Smita Ghorpade, G. S. Marane and Sonali Ghule

4. Data Structures and Algorithm Analysis in C++ - By Mark Allen Weiss

5. Let us C++ - By Yashvant Kanetkar

6. C++ Programming Language – By D. S. Malik

7. Object Oriented Concepts and Programming in C++ Lab Book – By Manisha Suryavanshi, Madhuri Ghanekar, S. G. Lakhdive , Parag Tamhankar and Manisha Jagdale

8. C++ Programming- A practical approach – By Madhusudan Mothe

9. C/C++ Programming manual – By Tim Lin and Saeed Monemi