TPCT's
College of Engineering, Osmanabad

Laboratory Manual

DATA STRUCTURE USING C

For

Second Year Students

Manual Prepared by

Mr. T.K.Takbhate

Author COE, Osmanabad

TPCT's

College of Engineering
Solapur Road, Osmanabad
Department of Computer Science and Engineering

<u>Vision of the Department</u>:

To develop computer engineers with necessary analytical ability and human values who can creatively design, implement a wide spectrum of computer systems for welfare of the society.

<u>Mission of the Department:</u>

1. Preparing graduates to work on multidisciplinary platforms associated with their professional position both independently and in a team environment.
2. Preparing graduates for higher education and research in computer science and engineering enabling them to develop systems for society development.

College of Engineering

Technical Document

This technical document is a series of Laboratory manuals of Computer Science and Engineering Department and is a certified document of College of Engineering, Osmanabad. The care has been taken to make the document error-free. But still if any error is found. Kindly bring it to the notice of subject teacher and HOD.

Recommended by,

HOD

Approved by,

Principal

Copies:

1. Departmental Library
2. Laboratory
3. HOD
4. Principal

FOREWORD

It is my great pleasure to present this laboratory manual for Second year engineering students for the subject of Data Structures (Using C).

As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly what has been tried is to answer through this manual.

As you may be aware that MGM has already been awarded with ISO 9001:2000 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO 9 001:2000 Certification.

Faculty members are also advised that covering these aspects in initial stage itself, will greatly relieve them in future as much of the load will be taken care by the enthusiasm energies of the students once they are conceptually clear.

H.O.D.

CSE Dept.

LABORATORY MANUAL CONTENTS

This manual is intended for the Second year students of Computer Science and engineering in the subject of Data Structures (Using C). This manual typically contains practical/lab sessions related Data Structures implemented in C covering various aspects related the subject to enhanced understanding.
Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.
Good Luck for your Enjoyable Laboratory Sessions

Mr. T.K.Takbhate

SUBJECT INDEX

Lab No.	Index	Week Involved
1	Perform arithmetic operation on two dimensional array using function. 1. Write a C program to implement Bubble sort,Insertion Sort and Selection sort. 2. Write a C program to implement Heap sort. 3. Write a C program to implement construct binary tree and binary tree traversal.	1
2	Write a C program to maintain records of students using structure and union.	2
3	Write a C program to swap 2 values by using call by value and call by reference.	3
4	Write a C program to implement stack using dynamic array.	4
5	Design, develop, and execute a program in C to convert a given valid parenthesized infix arithmetic expression to postfix expression and then to print both the expressions. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide).	5
6	Design, develop, and execute a program in C to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are + (add), - (subtract), * (multiply) and / (divide).	6
7	Design, develop, and execute a program in C to simulate the working of a queue of integers using an array. Provide the following operations: a. Insert b. Delete c. Display	7
8	Design, develop, and execute a program in C to implement a singly linked list/doubly linked list where each node consist of integers.the program should support the following operations: A) Create a singly / doubly linked list by adding each node at the front B) Insert a new node to the left of the node whose key value is read as an input. C) Delete the node of a given data if it is found,otherwise display appropriate message D) Display the contents of the list.	8

9	Using circular representation for a polunomial , Design, develop, and execute a program in C toaccept two polynomial ,add them and then print the resulting polynomial.	9
10	Write a C program to implement Graph Traversal Techniques(DFS and BFS)	10
11	Write a C program to implement Linear and Binary search.	11
12	Write a C program to implement Bubble sort,Insertion Sort and Selection sort.	12
13	Write a C program to implement Heap sort.	13
14	Write a C program to implement construct binary tree and binary tree traversal.	14
	Conduction of Viva-Voce Examinations	

Appendix - A

Appendix – B

1. Do’s and Don’ts in the laboratory

- Make entry in the Log Book as soon as you enter the Laboratory.
- All the students should sit according to their roll numbers starting from their left to right.
- All the students are supposed to enter the terminal number in the log book.
- Do not change the terminal on which you are working.
- All the students are expected to get at least the algorithm of the program/concept to be implemented.
- Strictly observe the instructions given by the teacher/Lab Instructor.

2. Pre-lab (Introduction to Data structure)

Data Structure is a way of collecting and organising data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have some data which has, player's name "Virat" and age 26. Here "Virat" is of String data type and 26 is of integer data type.

We can organize this data as a record like Player record, which will have both player's name and age in it. Now we can collect and store player's records in a file or database as a data structure. For example: "Dhoni" 30, "Gambhir" 31, "Sehwag" 33

If you are aware of Object Oriented programming concepts, then a class also does the same thing, it collects different type of data under one single entity. The only difference being, data structures provides for techniques to access and manipulate data efficiently.

In simple language, Data Structures are structures programmed to store ordered data, so that various operations can be performed on it easily. It represents the knowledge of data to be organized in memory. It should be designed and implemented in such a way that it reduces the complexity and increases the efficiency.

3. Lab Experiments:

Experiment No:-1

Aim:- Perform arithmetic operation on two dimensional array using function.

Objective:- Array operation techniques, accessing arrays in expressions, performing array math operations, and using variables in expressions..

Theory:-

2-dimensional arrays provide most of this capability. Like a 1D array, a 2D array is a collection of data cells, all of the same type, which can be given a single name. However, a 2D array is organized as a matrix with a number of rows and columns.

Code:

```
#include <stdio.h>
#include<conio.h>
int main( )
{
   int n, i;
   float num[100], sum = 0.0, average;

   printf("Enter the numbers of elements: ");
   scanf("%d", &n);

   while (n > 100 || n <= 0)
   {
      printf("Error! number should in range of (1 to 100).\n");
      printf("Enter the number again: ");
      scanf("%d", &n);
   }

   for(i = 0; i < n; ++i)
   {
      printf("%d. Enter number: ", i+1);
```

```
        scanf("%f", &num[i]);
        sum += num[i];
    }

    average = sum / n;
    printf("Average = %.2f", average);

    return 0;
}
```

Output:

```
Enter the numbers of elements: 6
1. Enter number: 45.3
2. Enter number: 67.5
3. Enter number: -45.6
4. Enter number: 20.34
5. Enter number: 33
6. Enter number: 45.6
Average = 27.69
```

Experiment No:-2

Aim:- Write a C program to maintain records of students using structure and union.

Objective:-

Theory:-

Code:

```
#include <stdio.h>
#include<conio.h>
struct student
{
   char  name[10];
   int roll;
   float marks;
} s;
int main( )
{
   int i;
   clrscr( );
   printf("Enter information of students:\n");
   printf("The  roll number=");
   scanf("%d",&s.roll);
   printf("Enter name: ");
   scanf("%s",&s.name);
   printf("Enter marks: ");
   scanf("%f",&s.marks);
   printf("\nDisplay information is\n");
   printf("The  roll number=%d,s.roll");
   printf("The  name is=%s, s.name");
   printf("The  marks are=%f, s.marks ");
   getch( );

   }
```

```
Output:
Enter information of students:
The  roll number= 1
Enter name: tushar
Enter marks: 100
Display information is
The  roll number= 1
The  name is= tushar
The  marks are= 100
```

```
/* W.A.P. in c to maintain record of student using  union*/
#include <stdio.h>
#include<conio.h>
union student
{
   char  name[10];
   int roll;
   float marks;
} s;
int main( )
{
   int i;
   clrscr( );
   printf("Enter information of students:\n");
   printf("The  roll number=");
   scanf("%d",&s.roll);
   printf("Enter name: ");
   scanf("%s",&s.name);
   printf("Enter marks: ");
   scanf("%f",&s.marks);
   printf("\nDisplay information is\n");
   printf("The  roll number=%d,s.roll");
   printf("The  name is=%s, s.name");
   printf("The  marks are=%f, s.marks ");
```

```
    getch( );

    }
```

Output:

```
Enter information of students:
The  roll number= 1
Enter name: tushar
Enter marks: 100
Display information is
The  roll number= 1
The  name is= tushar
The  marks are= 100
```

Experiment No:-3

Aim:- Write a C program to swap 2 values by using call by value and call by reference.

Objective:-

A function is a group of statements that together perform a task. Every Objective-C program has one C function, which is main(), and all of the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is so each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

Theory:-

Defining a Method

The general form of a method definition in Objective-C programming language is as follows −

```
- (return_type) method_name:( argumentType1 )argumentName1
joiningArgument2:( argumentType2 )argumentName2 ...
joiningArgumentn:( argumentTypen )argumentNamen {
  body of the function
}
```

A method definition in Objective-C programming language consists of a method header and a method body. Here are all the parts of a method −

Return Type − A method may return a value. The return_type is the data type of the value the function returns. Some methods perform the desired operations without returning a value. In this case, the return_type is the keyword void.

Method Name − This is the actual name of the method. The method name and the parameter list together constitute the method signature.

Arguments − A argument is like a placeholder. When a function is invoked, you pass a value to the argument. This value is referred to as actual parameter ***or*** argument. The parameter list refers to the type, order, and number of the arguments of a method. Arguments are optional; that is, a method may contain no argument**.**

Code:

```
/* W.A.P. in c to swap 2 values by using call by value */
#include<stdio.h>
// function prototype, also called function declaration
void swap(int a, int b);

int main()
{
   int m = 22, n = 44;
   // calling swap function by value
   printf(" values before swap  m = %d \nand n = %d", m, n);
   swap(m, n);
}

void swap(int a, int b)
{
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
   printf(" \nvalues after swap m = %d\n and n = %d", a, b);
}
```

OUTPUT:

```
values before swap m = 22
and n = 44
values after swap m = 44
and n = 22
```

```
/* W.A.P. in c to swap 2 values by using call by reference*/
#include<stdio.h>
// function prototype, also called function declaration
void swap(int *a, int *b);

int main()
{
   int m = 22, n = 44;
   //  calling swap function by reference
   printf("values before swap m = %d \n and n = %d",m,n);
   swap(&m, &n);
}

void swap(int *a, int *b)
{
   int tmp;
   tmp = *a;
   *a = *b;
   *b = tmp;
   printf("\n values after swap a = %d \nand b = %d", *a, *b);
}
```

OUTPUT:

```
values before swap m = 22
and n = 44
values after swap a = 44
and b = 22
```

Experiment No:-4

Aim:- Write a C program to implement stack using dynamic array.

Objective:-
Student should be able to develop a program from the logic of stack

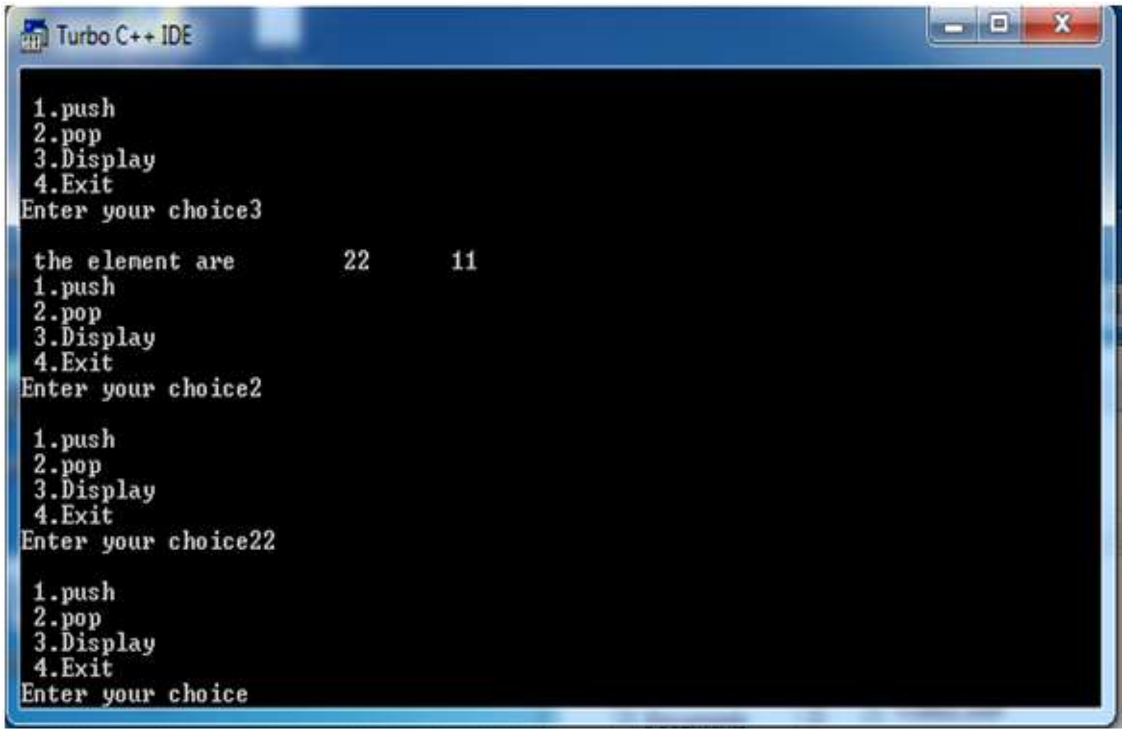
Theory:-
A stack is an ordered collection of homogeneous data element where the insertion and deletion operation take place at one end only. An n -element array can be used to implement a stack with capacity n. The stack top is indicated by the index top and it points to the element that was entered into the stack.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define maxsize 4
void push();
int pop();
void display();
int stack[maxsize];
int top=-1;
void main()
{
int choice;
char c;
clrscr();
do
{
printf("\n1.push");
printf("\n2.pop");
printf("\n3.display");
printf("\n4.exit");
scanf("%d",&choice);
switch(choice)
{
case 1:push();
break;
case 2:printf("\ndeleted element is%d",pop());
break;
case 3: display();
break;
case 4:printf("\nexit");
exit(0);
break;
}
}
while(choice!=4);
}
void push()
{
int item;
if(top==maxsize-1)
{
printf("\n stack is full");
getch();
return;
}
else
{
printf("\n enter the element to be inserted");
scanf("%d",&item);
top=top+1;
stack[top]=item;
}
}
int pop()
{
int item;
if(top==-1)
{
printf("\n stack is empty");
getch();
return;
```

```
}
else
{
item=stack[top];
top=top=-1;
}
return(item);
}
void display()
{
int i;
if(top==-1)
{
printf("\n stack is empty");
getch();
return;
}
else
{
printf("\n the element are:");
for(i=top;i>=0;i--)
{
printf("\t%d",stack[i]);
getch();
}
}
}
```

OUTPUT:



Turbo C++ IDE

```
 1.push
 2.pop
 3.Display
 4.Exit
Enter your choice3

 the element are         22       11
 1.push
 2.pop
 3.Display
 4.Exit
Enter your choice2

 1.push
 2.pop
 3.Display
 4.Exit
Enter your choice22

 1.push
 2.pop
 3.Display
 4.Exit
Enter your choice
```

Experiment No:-5

Aim:- Design, develop, and execute a program in C to convert a given valid parenthesized infix arithmetic expression to postfix expression and then to print both the expressions. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide).

Objective:-

Student should be able to develop a program from the logic of infix expression to prefix and postfix conversion.

Theory:-

Infix notation: X + Y

Operators are written in-between their operands. This is the usual way we write expressions. An expression such as A * (B + C) / D is usually taken to mean something like: "First add B and C together, then multiply the result by A, then divide by D to give the final answer."

Infix notation needs extra information to make the order of evaluation of the operators clear: rules built into the language about operator precedence and associativity, and brackets ( ) to allow users to override these rules. For example, the usual rules for associativity say that we perform operations from left to right, so the multiplication by A is assumed to come before the division by D. Similarly, the usual rules for precedence say that we perform multiplication and division before we perform addition and subtraction. (see CS2121 lecture).

Postfix notation (also known as "Reverse Polish notation"): X Y +

Operators are written after their operands. The infix expression given above is equivalent to A B C + * D /

The order of evaluation of operators is always left-to-right, and brackets cannot be used to change this order. Because the "+" is to the left of the "*" in the example above, the addition must be performed before the multiplication.

Operators act on values immediately to the left of them. For example, the "+" above uses the "B" and "C". We can add (totally unnecessary) brackets to make this explicit:

((A (B C +) *) D /)

Thus, the "*" uses the two values immediately preceding: "A", and the result of the addition. Similarly, the "/" uses the result of the multiplication and the "D".

Prefix notation (also known as "Polish notation"): + X Y

Operators are written before their operands. The expressions given above are equivalent to / * A + B C D

As for Postfix, operators are evaluated left-to-right and brackets are superfluous. Operators act on the two nearest values on the right. I have again added (totally unnecessary) brackets to make this clear:

(/ (* A (+ B C)) D)

Although Prefix "operators are evaluated left-to-right", they use values to their right, and if these values themselves involve computations then this changes the order that the operators have to be evaluated in. In the example above, although the division is the first operator on the left, it acts on the result of the multiplication, and so the multiplication has to happen before the division (and similarly the addition has to happen before the multiplication).

Because Postfix operators use values to their left, any values involving computations will already have been calculated as we go left-to-right, and so the order of evaluation of the operators is not disrupted in the same way as in Prefix expressions.

Code:

```
#include<stdio.h>
 #include <conio.h>
typedef                                                                enum
{lparen,rparen,plus,minus,times,divide,mod,eos,operand}precedence;
precedence token;
int stack[10];
int top=-1;
 char expr[100];
 precedence get_token(char*symbol,int*n)
 {
         *symbol = expr[*n];
*n=*n+1;
switch (*symbol)
 {
 case '(' : return(lparen);break;
 case ')' : return(rparen);break;
 case '+' : return(plus);break;
case '-' : return(minus);break;
 case '*' : return(times);break;
 case '/' : return(divide);break;
 case '%' : return(mod);break;
 case '\0' : return(eos);break;
 default : return(operand);break;
 }
 }
 void push (int a)
{
 if(top>=9)
{
 printf("Stack overflow \n");
 return;
 }
 Else
 {
          top=top+1;
 stack[top]=a;
 }
 }
 int pop()
 {
int b;
 if(top==-1)
{
 printf("Stack is empty \n");
 return 0;
 }
 Else
 {
 b=(stack[top]);
 top=top-1;
 return b;
 }
 }
 int eval()
 {
 char symbol;
int op1,op2;
int n=0;
 token=get_token(&symbol,&n);
          while(token!=eos)
{
 if(token==operand)
 {
 push (symbol-'0');
 }
 else
{
```

```
op2=pop();
op1=pop();
 switch(token)
 {
 case plus : push(op1+op2);break;
 case minus : push(op1-op2);break;
 case times : push(op1*op2);break;
 case divide : push(op1/op2);break;
 case mod : push(op1%op2);break;
 }
 token=get_token(&symbol,&n);
                      }
 return pop();
 }
 }
 int main()
 {
 int ans;
 printf("Enter the postfix expression \n");
gets(expr);
ans=eval();
printf("The answer is %d",ans);
return 0;
 }
```

Experiment No:-6

Aim:- Design, develop, and execute a program in C to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are + (add), - (subtract), * (multiply) and / (divide).

Objective:-

Student should be able to develop the program for evaluation of the postfix expression.
This is for only application infix expression to postfix conversion.

Theory:-

As discussed in Infix To Postfix Conversion Using Stack, the compiler finds it convenient to evaluate an expression in its postfix form. The virtues of postfix form include elimination of parentheses which signify priority of evaluation and the elimination of the need to observe rules of hierarchy, precedence and associativity during evaluation of the expression.

As Postfix expression is without parenthesis and can be evaluated as two operands and an operator at a time, this becomes easier for the compiler and the computer to handle.

Evaluation rule of a Postfix Expression states:

While reading the expression from left to right, push the element in the stack if it is an operand.

Pop the two operands from the stack, if the element is an operator and then evaluate it.

Push back the result of the evaluation. Repeat it till the end of the expression.

Code:

```
#include<stdio.h>
int stack[20];
int top = -1;
void push(int x)
{
        stack[++top] = x;
}
int pop()
{
        return stack[top--];
```

```
}
int main()
{
	char exp[20];
	char *e;
	int n1,n2,n3,num;
	printf("Enter the expression :: ");
	scanf("%s",exp);
	e = exp;
	while(*e != '\0')
	{
		if(isdigit(*e))
		{
			num = *e - 48;
			push(num);
		}
		else
		{
			n1 = pop();
			n2 = pop();
			switch(*e)
			{
				case '+':
				{
					n3 = n1 + n2;
					     break;
				}
				case '-':
				{
					n3 = n2 - n1;
					break;
				}
				case '*':
				{
					n3 = n1 * n2;
					break;
				}
				case '/':
				{
					n3 = n2 / n1;
```

```
                    break;
                }
            }
            push(n3);
        }
        e++;
    }
    printf("\nThe result of expression %s  =  %d\n\n",exp,pop());
    return 0;
}
```

OUTPUT:

```
Enter the expression :: 245+*

The result of expression 245+*  =  18
```

Experiment No:-7

Aim:- Design, develop, and execute a program in C to simulate the working of a queue of integers using an array. Provide the following operations: a. Insert b. Delete c. Display

Objective:-

Student should be able to develop the program for queue using arrays.

Theory:-

A Queue is an ordered collection of homogeneous data elements where the insertion and deletion operation take place at two extreme ends.

```
Function insert()
Check for the overflow condition of the Queue
If(REAR = N) then  Print “Queue is full”
If not overflow , increment the value of rear
REAR = REAR+1
Get the element to be inserted into the queue from the user Q[REAR] = ITEM
Assign it as the last value ,queue[rear]

Function delete()
Check for the underflow( empty) condition of the queue
If not empty ,Output the element to be deleted from the queue
Increment the value of front.

Function display()
Display the elements of  the queue
```

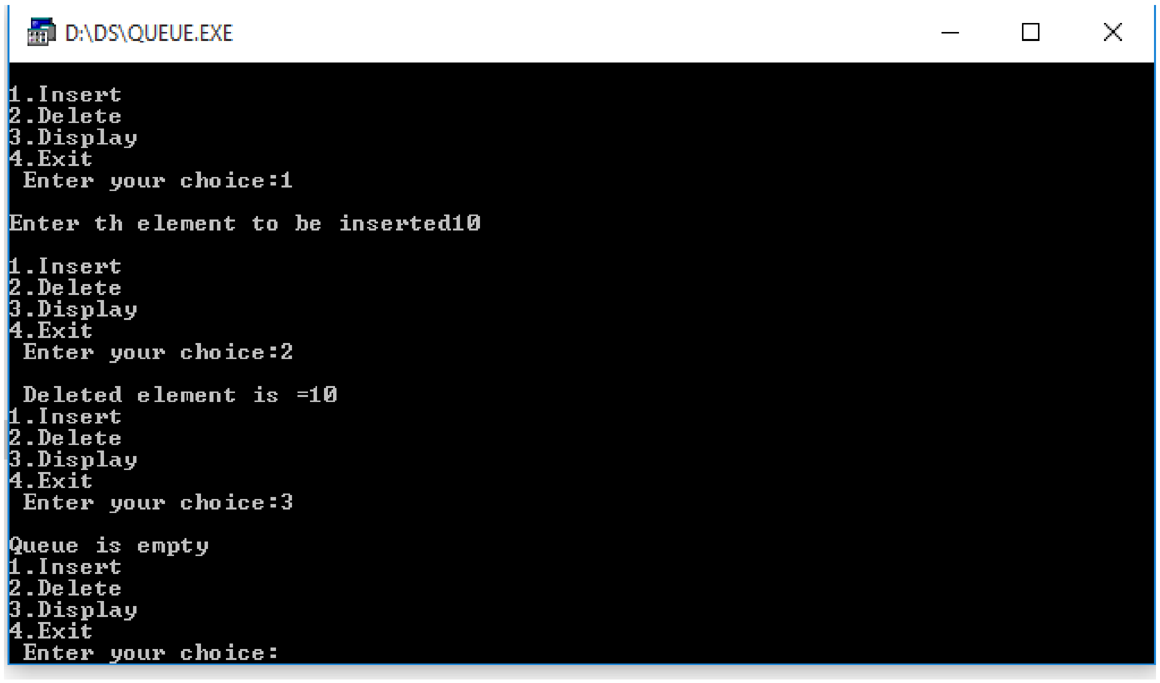
Code:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define maxsize 5

int cq[maxsize];
int front=-1,rear=0;
int choice;
char ch;
void main()
{
```

```
clrscr();
do
{
printf("\n1.Insert");
printf("\n2.Delete");
printf("\n3.Display");
printf("\n4.Exit");
printf("\n Enter your choice");
scanf("%d",&choice);
switch(choice)
{
case 1: cqinsert();
        break;
case 2: cqdelete();
        break;
case 3: cqdisplay();
        break;
case 4: exit(0);
        break;
}
}
while(choice|=4);

}
 cqinsert()
{
int num;
if(front==(rear+1)%maxsize)
{
printf("\n Queue is full");
return;
}
else
{
printf("\nEnter th element to be inserted");
scanf("%d",&num);
if(front==-1)
front=rear=0;
else
rear=(rear+1)%maxsize;
cq[rear]=num;
}
}

int cqdelete()
{
int num;
if(front==-1)
{
printf("\nQueue is empty");
return;
}
else
{
num=cq[front];
printf("\n Deleted element is =%d",cq[front]);
if(front==rear)
front=rear=-1;
else
front=(front+1)%maxsize;
}
return(num);
}
cqdisplay()
{
int i;
if(front==-1)
{
printf("\nQueue is empty");
return;
}
else
{
printf("\nThe element of the queue");
for(i=front;i<rear;i++)
{
printf("\t%d",cq[i]);
}
}
```

```
if(front>rear)
{
for(i=front;i<maxsize;i++)
{
printf("%d",cq[i]);
}
for(i=0;i<=rear;i++)
{
printf("%d",cq[i]);
}
}
printf("\n");
}
```

OUTPUT:



```
1.Insert
2.Delete
3.Display
4.Exit
 Enter your choice:1

Enter th element to be inserted10

1.Insert
2.Delete
3.Display
4.Exit
 Enter your choice:2

 Deleted element is =10
1.Insert
2.Delete
3.Display
4.Exit
 Enter your choice:3

Queue is empty
1.Insert
2.Delete
3.Display
4.Exit
 Enter your choice:
```

Experiment No:-8

Aim Design, develop, and execute a program in C to implement a singly linked list/doubly linked list where each node consist of integers.the program should support the following operations:

A) Create a singly / doubly linked list by adding each node at the front
B) Insert a new node to the left of the node whose key value is read as an input.
C) Delete the node of a given data if it is found,otherwise display appropriate message
D) Display the contents of the list.

Objective:-

Student should be able to develop the program for linked list creation and operations on it

Theory :-

Linked list is a chain of structures or records called nodes. Each node has at least two members, one of which points to the next item or node in the list! These are defined as Single Linked Lists because they only point to the next item, and not the previous. Those that do point to both are called Doubly Linked Lists. Unlike arrays there is no upper limit on the amount of memory reserved. A singly linked list is the simplest of linked lists. Each node of a singly linked list has data elements and a single link (pointer) that points to the next node of the list (or) NULL if it is the last node of the list. Addition/ Deletion of a node to the singly linked list involve the creation/ deletion of the node and adjusting the node pointers accordingly. Some operations include

Traversing a list

Insertion of a node into a list.

Deletion of a node from the list.

Searching for an element in a list.

Linked list can be developed using the user defined data types. i.e Structure struct list

```
{
int   data;   struct
list * next;
}
```

Code:

```
#include <stdio.h>
#include <stdlib.h>
 struct node
{
    struct node *prev;
    int n;
    struct node *next;
}*h,*temp,*temp1,*temp2,*temp4;
 void insert1();
void insert2();
void insert3();
void traversebeg();
void traverseend(int);
void sort();
void search();
void update();
void delete();
 int count = 0;
 void main()
{
    int ch;
     h = NULL;
    temp = temp1 = NULL;
     printf("\n 1 - Insert at beginning");
    printf("\n 2 - Insert at end");
    printf("\n 3 - Insert at position i");
    printf("\n 4 - Delete at i");
    printf("\n 5 - Display from beginning");
    printf("\n 6 - Display from end");
    printf("\n 7 - Search for element");
    printf("\n 8 - Sort the list");
    printf("\n 9 - Update an element");
    printf("\n 10 - Exit");
     while (1)
```

```
{
   printf("\n Enter choice : ");
   scanf("%d", &ch);
   switch (ch)
   {
   case 1:
      insert1();
      break;
   case 2:
      insert2();
      break;
   case 3:
      insert3();
      break;
   case 4:
      delete();
      break;
   case 5:
      traversebeg();
      break;
   case 6:
      temp2 = h;
      if (temp2 == NULL)
         printf("\n Error : List empty to display ");
      else
      {
         printf("\n Reverse order of linked list is : ");
         traverseend(temp2->n);
      }
      break;
   case 7:
      search();
      break;
   case 8:
      sort();
```

```
            break;
        case 9:
            update();
            break;
        case 10:
            exit(0);
        default:
            printf("\n Wrong choice menu");
        }
    }
}
 /* TO create an empty node */
void create()
{
    int data;
     temp =(struct node *)malloc(1*sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\n Enter value to node : ");
    scanf("%d", &data);
    temp->n = data;
    count++;
}
 /*  TO insert at beginning */
void insert1()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp->next = h;
        h->prev = temp;
        h = temp;
    }
}
 /* To insert at end */
void insert2()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp1->next = temp;
        temp->prev = temp1;
        temp1 = temp;
    }
}
 /* To insert at any position */
void insert3()
{
    int pos, i = 2;
     printf("\n Enter position to be inserted : ");
    scanf("%d", &pos);
    temp2 = h;
     if ((pos < 1) || (pos >= count + 1))
    {
        printf("\n Position out of range to insert");
        return;
    }
    if ((h == NULL) && (pos != 1))
    {
        printf("\n Empty list cannot insert other than 1st position");
        return;
    }
    if ((h == NULL) && (pos == 1))
    {
        create();
        h = temp;
        temp1 = h;
        return;
    }
    else
    {
        while (i < pos)
        {
            temp2 = temp2->next;
            i++;
        }
        create();
        temp->prev = temp2;
        temp->next = temp2->next;
        temp2->next->prev = temp;
        temp2->next = temp;
    }
}
 /* To delete an element */
void delete()
{
    int i = 1, pos;
     printf("\n Enter position to be deleted : ");
    scanf("%d", &pos);
    temp2 = h;
     if ((pos < 1) || (pos >= count + 1))
    {
        printf("\n Error : Position out of range to delete");
        return;
    }
    if (h == NULL)
    {
        printf("\n Error : Empty list no elements to delete");
        return;
    }
    else
    {
        while (i < pos)
        {
            temp2 = temp2->next;
            i++;
        }
        if (i == 1)
        {
            if (temp2->next == NULL)
            {
                printf("Node deleted from list");
                free(temp2);
                temp2 = h = NULL;
                return;
            }
        }
        if (temp2->next == NULL)
        {
            temp2->prev->next = NULL;
            free(temp2);
            printf("Node deleted from list");
            return;
        }
        temp2->next->prev = temp2->prev;
        if (i != 1)
            temp2->prev->next = temp2->next;     /* Might not need this statement if i
== 1 check */
```

```
        if (i == 1)
            h = temp2->next;
        printf("\n Node deleted");
        free(temp2);
    }
    count--;
}
 /* Traverse from beginning */
void traversebeg()
{
    temp2 = h;
     if (temp2 == NULL)
    {
        printf("List empty to display \n");
        return;
    }
    printf("\n Linked list elements from begining : ");

    while (temp2->next != NULL)
    {
        printf(" %d ", temp2->n);
        temp2 = temp2->next;
    }
    printf(" %d ", temp2->n);
}
 /* To traverse from end recursively */
void traverseend(int i)
{
    if (temp2 != NULL)
    {
        i = temp2->n;
        temp2 = temp2->next;
        traverseend(i);
        printf(" %d ", i);
    }
```

```
}
 /* To search for an element in the list */
void search()
{
    int data, count = 0;
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("\n Error : List empty to search for data");
        return;
    }
    printf("\n Enter value to search : ");
    scanf("%d", &data);
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {
            printf("\n Data found in %d position",count + 1);
            return;
        }
        else
             temp2 = temp2->next;
            count++;
    }
    printf("\n Error : %d not found in list", data);
}
 /* To update a node value in the list */
void update()
{
    int data, data1;
     printf("\n Enter node data to be updated : ");
    scanf("%d", &data);
    printf("\n Enter new data : ");
    scanf("%d", &data1);
    temp2 = h;
    if (temp2 == NULL)
    {
        printf("\n Error : List empty no node to update");
        return;
    }
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {
             temp2->n = data1;
            traversebeg();
            return;
        }
        else
            temp2 = temp2->next;
    }
     printf("\n Error : %d not found in list to update", data);
}
 /* To sort the linked list */
void sort()
{
    int i, j, x;
     temp2 = h;
    temp4 = h;
     if (temp2 == NULL)
    {
        printf("\n List empty to sort");
        return;
    }
     for (temp2 = h; temp2 != NULL; temp2 = temp2->next)
    {
        for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)
        {
            if (temp2->n > temp4->n)
            {
                x = temp2->n;
                temp2->n = temp4->n;
                temp4->n = x;
            }
        }
    }
    traversebeg();
}
```

Output:-

```
1 - Insert at beginning
2 - Insert at end
3 - Insert at position i
4 - Delete at i
5 - Display from beginning
6 - Display from end
7 - Search for element
8 - Sort the list
9 - Update an element
10 - Exit
Enter choice : 1

Enter value to node : 10

Enter choice : 2

Enter value to node : 50

Enter choice : 4

Enter position to be deleted : 1

Node deleted
Enter choice : 1
Enter value to node : 34

Enter choice : 3

Enter position to be inserted : 2

Enter value to node : 13

Enter choice : 4

Enter position to be deleted : 4

Error : Position out of range to delete
Enter choice : 1

Enter value to node : 15

Enter choice : 1

Enter value to node : 67

Enter choice : 3

Enter position to be inserted : 2

Enter value to node : 34

Enter choice : 4

Enter position to be deleted : 3

Node deleted
Enter choice : 7
Enter value to search : 15

Error : 15 not found in list
Enter choice : 8

Linked list elements from begining :  13  34  34  50  67
Enter choice : 9

Enter node data to be updated : 45

Enter new data : 89

Error : 45 not found in list to update
Enter choice : 9

Enter node data to be updated : 50

Enter new data : 90
Enter choice : 5

Linked list elements from begining :  13  34  34  90  67
Enter choice : 6

Reverse order of linked list is :  67  90  34  34  13
Enter choice : 7

Enter value to search : 90

Data found in 4 position
Enter choice : 8

Linked list elements from begining :  13  34  34  67  90
Enter choice : 7

Enter value to search : 90
```

```
Data found in 5 position
Enter choice : 9

Enter node data to be updated : 34

Enter new data : 56

Linked list elements from begining :  13  56  34  67  90
Enter choice : 10
```

Experiment No:-9

Aim:- Using circular representation for a polunomial , Design, develop, and execute a program in C toaccept two polynomial ,add them and then print the resulting polynomial.

Objective:-

A polynomial is an expression that contains more than two terms. A term is made up of coefficient and exponent.

Theory:-

A polynomial may be represented using array or structure. A structure may be defined such that it contains two parts – one is the coefficient and second is the corresponding exponent. The structure definition may be given as shown below:

```
Struct polynomial
{
        int coefficient;
        int exponent;
};
```

Code:

```
        #include <stdio.h>
        #include<conio.h>
        // A utility function to return maximum of two integers
        int max(int m, int n)
        {
        return (m > n)? m: n;
        }
        // A[ ] represents coefficients of first polynomial
        // B[ ] represents coefficients of second polynomial
        // m and n are sizes of A[ ] and B[ ] respectively
        int   *add(int A[ ], int B[ ], int m, int n)
        {
             int size = max(m, n);
             int *sum = new int[size];
             // Initialize the porduct polynomial
             for (int i = 0; i<m; i++)
             sum[i] = A[i];
             // Take ever term of first polynomial
```

```
      for (int i=0; i<n; i++)
      sum[i] += B[i];
      return sum;
}
 // A utility function to print a polynomial
void printPoly(int poly[ ], int n)
{
    for (int i=0; i<n; i++)
    {
       printf("print", poly[i]);
       if (i != 0)
        printf("print", x^, i );
       if (i != n-1)
       printf("print", + );
    }
}
 // Driver program to test above functions
int main()
{
    // The following array represents polynomial 5 + 10x^2 + 6x^3
    int A[ ] = {5, 0, 10, 6};
     // The following array represents polynomial 1 + 2x + 4x^2
    int B[ ] = {1, 2, 4};
    int m = sizeof(A)/sizeof(A[0]);
    int n = sizeof(B)/sizeof(B[0]);
   printf("First polynomial is \n");
    printPoly(A, m);
    printf("\nSecond polynomial is \n");
    printPoly(B, n);
    int *sum = add(A, B, m, n);
    int size = max(m, n);

    printf("\nsumuct polynomial is \n");
    printPoly(sum, size);
     return 0;
```

```
getch( );
}
```

Output:

```
First polynomial is
5 + 0x^1 + 10x^2 + 6x^3
Second polynomial is
1 + 2x^1 + 4x^2
Sum polynomial is
6 + 2x^1 + 14x^2 + 6x^3
```

Experiment No:-10

Aim:- Write a C program to implement Graph Traversal Techniques(DFS and BFS)

Objective:-

Student should be able to develop a program from the logic of graph traversal techniques using base of stack and queue.

Theory:-

Depth First Search (DFS) Program in C

Most of graph problems involve traversal of a graph. Traversal of a graph means visiting each node and visiting exactly once. There are two types of traversal in graphs i.e. Depth First Search (DFS) and Breadth First Search (BFS).

Breadth First Search (BFS) Program in C

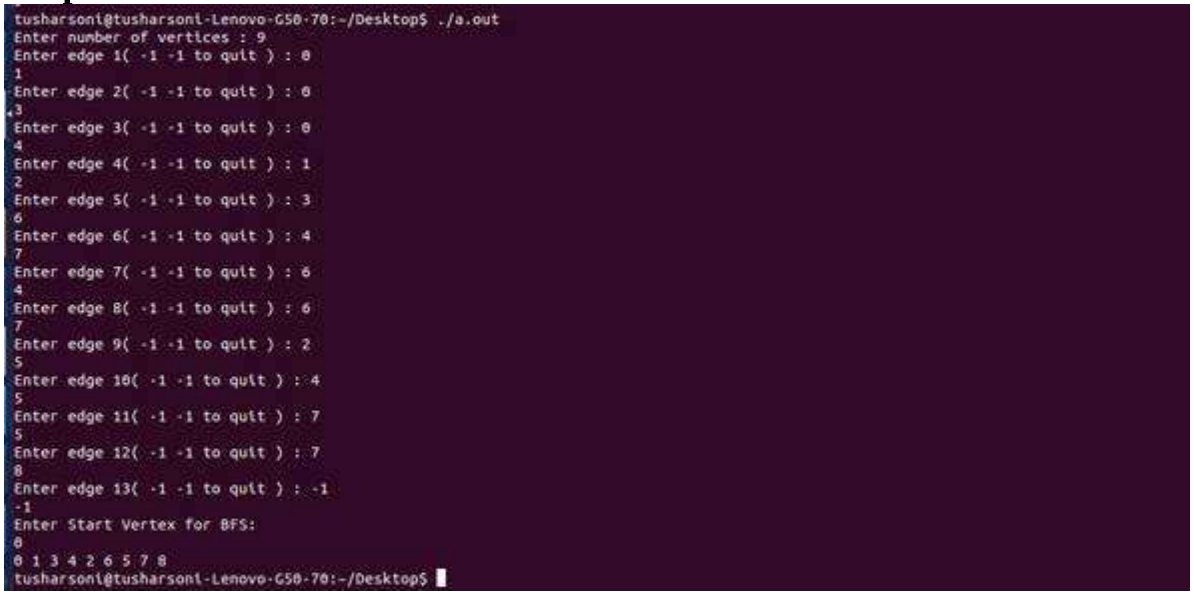
It is like tree. Traversal can start from any vertex, say Vi . Vi is visited and then all vertices adjacent to Vi are traversed recursively using DFS. Since, a graph can have cycles. We must avoid revisiting a node. To do this, when we visit a vertex V, we mark it visited. A node that has already been marked as visited should not be selected for traversal. Marking of visited vertices can be done with the help of a global array visited[ ]. Array visited[ ] is initialized to false (0).

Code:

```
#include<stdio.h>
#include<stdlib.h>
 #define MAX 100
 #define initial 1
#define waiting 2
#define visited 3
int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);
 int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();
int main()
{
    create_graph();
    BF_Traversal();
    return 0;
}
void BF_Traversal()
{
    int v;
    for(v=0; v<n; v++)
        state[v] = initial;
    printf("Enter Start Vertex for BFS: \n");
    scanf("%d", &v);
    BFS(v);
}
void BFS(int v)
{
    int i;
    insert_queue(v);
    state[v] = waiting;
    while(!isEmpty_queue())
    {
        v = delete_queue( );
        printf("%d ",v);
        state[v] = visited;
        for(i=0; i<n; i++)
        {
            if(adj[v][i] == 1 && state[i] == initial)
            {
                insert_queue(i);
                state[i] = waiting;
            }
        }
    }
    printf("\n");
```

```
}
void insert_queue(int vertex)
{
    if(rear == MAX-1)
        printf("Queue Overflow\n");
    else
    {
        if(front == -1)
            front = 0;
        rear = rear+1;
        queue[rear] = vertex ;
    }
}
int isEmpty_queue()
{
    if(front == -1 || front > rear)
        return 1;
    else
        return 0;
}
int delete_queue()
{
    int delete_item;
    if(front == -1 || front > rear)
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    delete_item = queue[front];
    front = front+1;
    return delete_item;
}
void create_graph()
{
    int count,max_edge,origin,destin;
    printf("Enter number of vertices : ");
    scanf("%d",&n);
    max_edge = n*(n-1);
    for(count=1; count<=max_edge; count++)
    {
        printf("Enter edge %d( -1 -1 to quit ) : ",count);
        scanf("%d %d",&origin,&destin);
        if((origin == -1) && (destin == -1))
            break;
        if(origin>=n || destin>=n || origin<0 || destin<0)
        {
            printf("Invalid edge!\n");
            count--;
        }
        else
        {
            adj[origin][destin] = 1;
        }
    }
}
```

Output:

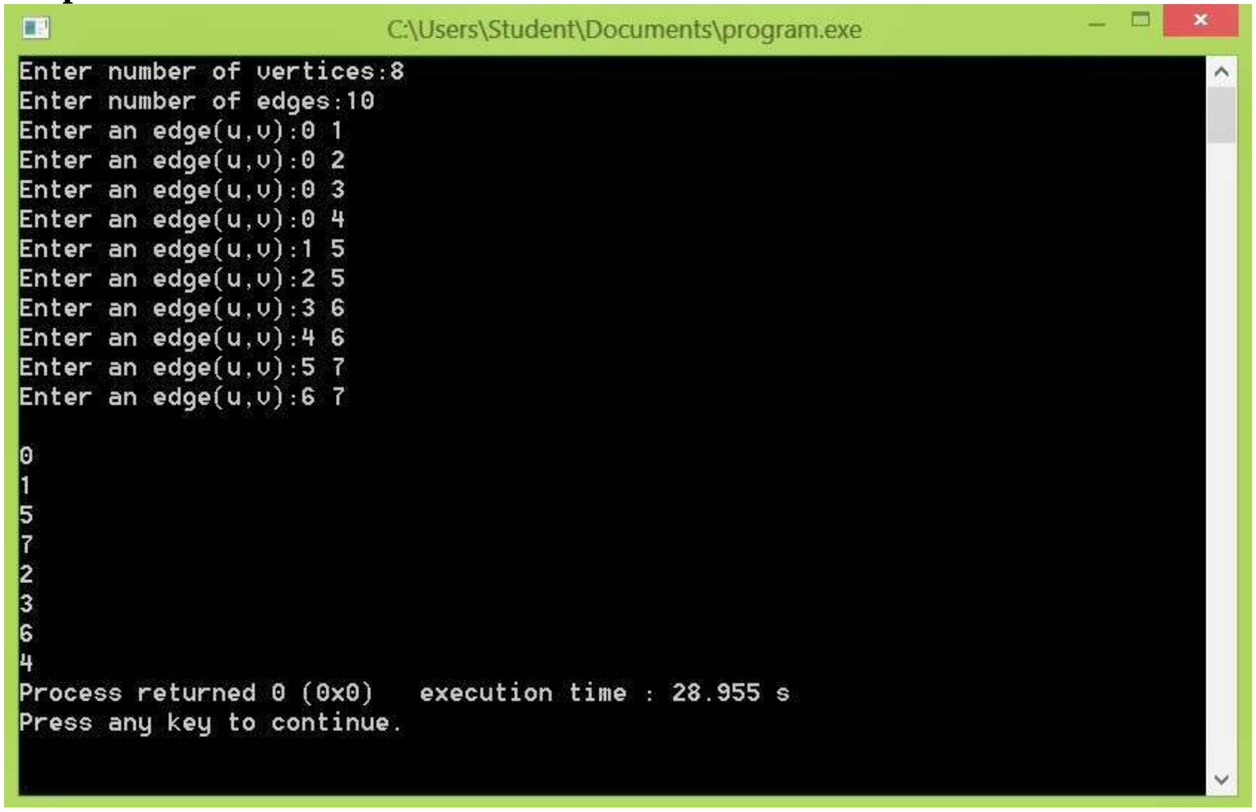


```
tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$ ./a.out
Enter number of vertices : 9
Enter edge 1( -1 -1 to quit ) : 0
1
Enter edge 2( -1 -1 to quit ) : 0
3
Enter edge 3( -1 -1 to quit ) : 0
4
Enter edge 4( -1 -1 to quit ) : 1
2
Enter edge 5( -1 -1 to quit ) : 3
6
Enter edge 6( -1 -1 to quit ) : 4
7
Enter edge 7( -1 -1 to quit ) : 6
4
Enter edge 8( -1 -1 to quit ) : 6
7
Enter edge 9( -1 -1 to quit ) : 2
5
Enter edge 10( -1 -1 to quit ) : 4
5
Enter edge 11( -1 -1 to quit ) : 7
5
Enter edge 12( -1 -1 to quit ) : 7
8
Enter edge 13( -1 -1 to quit ) : -1
-1
Enter Start Vertex for BFS:
0
0 1 3 4 2 6 5 7 8
tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$
```

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

typedef struct node
{
    struct node *next;
    int vertex;
}node;

node *G[20];
//heads of linked list
int visited[20];
int n;
void read_graph();
//create adjacency list
void insert(int,int);
//insert an edge (vi,vj) in te adjacency list
void DFS(int);

void main()
{
    int i;
    read_graph();
    //initialised visited to 0

    for(i=0;i<n;i++)
        visited[i]=0;

    DFS(0);
}

void DFS(int i)
```

```
{
   node *p;

   printf("\n%d",i);
   p=G[i];
   visited[i]=1;
   while(p!=NULL)
   {
     i=p->vertex;

     if(!visited[i])
         DFS(i);
      p=p->next;
   }
}

void read_graph()
{
   int i,vi,vj,no_of_edges;
   printf("Enter number of vertices:");

   scanf("%d",&n);

   //initialise G[] with a null

   for(i=0;i<n;i++)
   {
      G[i]=NULL;
      //read edges and insert them in G[]

      printf("Enter number of edges:");
        scanf("%d",&no_of_edges);

        for(i=0;i<no_of_edges;i++)
      {
            printf("Enter an edge(u,v):");
            scanf("%d%d",&vi,&vj);
            insert(vi,vj);
        }
    }
}

void insert(int vi,int vj)
{
    node *p,*q;

    //acquire memory for the new node
    q=(node*)malloc(sizeof(node));
    q->vertex=vj;
    q->next=NULL;

    //insert the node in the linked list number vi
    if(G[vi]==NULL)
        G[vi]=q;
    else
    {
        //go to end of the linked list
        p=G[vi];

        while(p->next!=NULL)
            p=p->next;
        p->next=q;
    }
}
```

Output:



C:\Users\Student\Documents\program.exe

```
Enter number of vertices:8
Enter number of edges:10
Enter an edge(u,v):0 1
Enter an edge(u,v):0 2
Enter an edge(u,v):0 3
Enter an edge(u,v):0 4
Enter an edge(u,v):1 5
Enter an edge(u,v):2 5
Enter an edge(u,v):3 6
Enter an edge(u,v):4 6
Enter an edge(u,v):5 7
Enter an edge(u,v):6 7

0
1
5
7
2
3
6
4
Process returned 0 (0x0)   execution time : 28.955 s
Press any key to continue.
```

Experiment No:-11

Aim:- Write a C program to implement Linear and Binary search.

Objective:-

Student should be able to develop a program from the logic of searching method.

Theory:-

Linear Search
Binary Search

A linear search scans one item at a time, without jumping to any item .

The worst case complexity is O(n), sometimes known an O(n) search
Time taken to search elements keep increasing as the number of elements are increased.

A binary search however, cut down your search to half as soon as you find middle of a sorted list.

The middle element is looked to check if it is greater than or less than the value to be searched.
Accordingly, search is done to either half of the given list

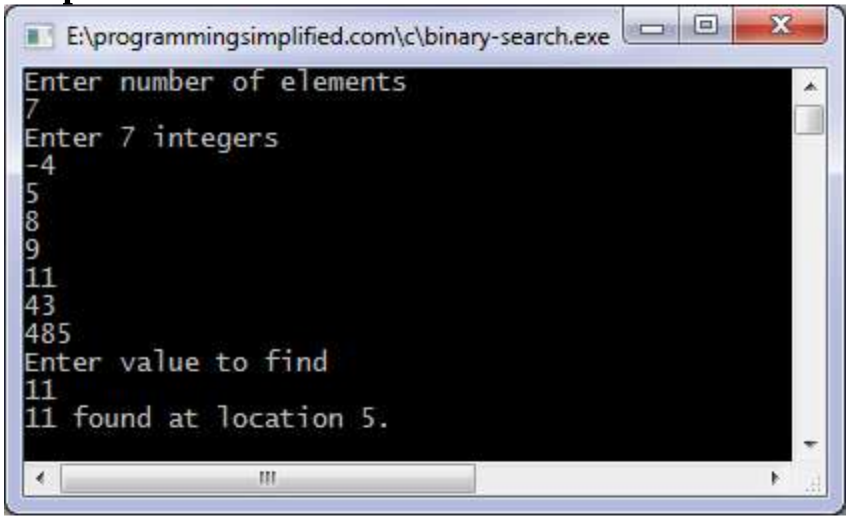
Important Differences

Input data needs to be sorted in Binary Search and not in Linear Search
Linear search does the sequential access whereas Binary search access data randomly.
Time complexity of linear search -O(n) , Binary search has time complexity O(log n).
Linear search performs equality comparisons and Binary search performs ordering comparisons

Code:

```
#include <stdio.h>

int main()
{
  int c, first, last, middle, n, search, array[100];
  printf("Enter number of elements\n");
  scanf("%d",&n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d",&array[c]);

  printf("Enter value to find\n");
  scanf("%d", &search);

  first = 0;
  last = n - 1;
  middle = (first+last)/2;

  while (first <= last) {
    if (array[middle] < search)
      first = middle + 1;
    else if (array[middle] == search) {
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
  if (first > last)
    printf("Not found! %d isn't present in the list.\n", search);

  return 0;
}
```

Output:



```
Enter number of elements
7
Enter 7 integers
-4
5
8
9
11
43
485
Enter value to find
11
11 found at location 5.
```

Experiment No:-12

Aim:- Write a C program to implement Bubble sort,Insertion Sort and Selection sort.

Objective:-

Student should be able to develop a program from the logic of sorting method.

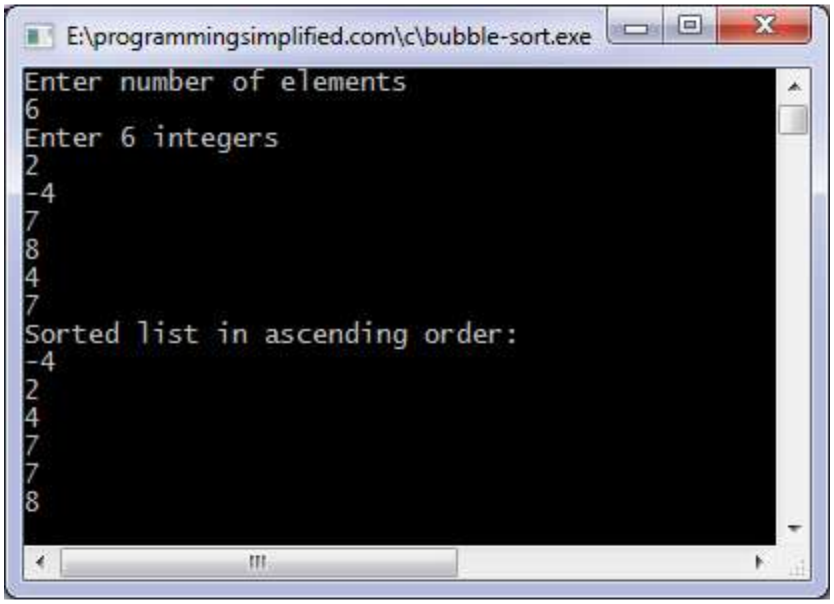
Theory:-

C program for bubble sort: C programming code for bubble sort to sort numbers or arrange them in ascending order. You can modify it to print numbers in descending order.You can also sort strings using Bubble sort, it is less efficient as its average and worst case complexity is high, there are many other fast sorting algorithms like quick-sort, heap-sort, etc. Sorting simplifies problem-solving in computer programming.

Code:

```
/* Bubble sort code */

#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use < */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

  for (c = 0; c < n; c++)
     printf("%d\n", array[c]);

  return 0;
}
```

Output:-



```
Enter number of elements
6
Enter 6 integers
2
-4
7
8
4
7
Sorted list in ascending order:
-4
2
4
7
7
8
```

Experiment No:-13

Aim:- Write a C program to implement Heap sort.

Objective:-

Student should be able to develop a program from the logic of sorting method(insert and delete).

Theory:-

Heapsort is a comparison-based sorting algorithm. Heapsort can be thought of as an improved selection sort: like that algorithm, it divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. The improvement consists of the use of a heap data structure rather than a linear-time search to find the maximum.

Code:

```
#include<stdio.h>
 void create(int []);
void down_adjust(int [],int);
 void main()
{
    int heap[30],n,i,last,temp;
    printf("Enter no. of elements:");
    scanf("%d",&n);
    printf("\nEnter elements:");
    for(i=1;i<=n;i++)
        scanf("%d",&heap[i]);
     //create a heap
    heap[0]=n;
   create(heap);
     //sorting
    while(heap[0] > 1)
    {
        //swap heap[1] and heap[last]
        last=heap[0];
        temp=heap[1];
        heap[1]=heap[last];
        heap[last]=temp;
        heap[0]--;
        down_adjust(heap,1);
    }
     //print sorted data
    printf("\nArray after sorting:\n");
    for(i=1;i<=n;i++)
        printf("%d ",heap[i]);
}
 void create(int heap[])
{
    int i,n;
    n=heap[0]; //no. of elements
    for(i=n/2;i>=1;i--)
        down_adjust(heap,i);
}
 void down_adjust(int heap[],int i)
{
    int j,temp,n,flag=1;
    n=heap[0];
     while(2*i<=n && flag==1)
    {
        j=2*i;    //j points to left child
        if(j+1<=n && heap[j+1] > heap[j])
            j=j+1;
        if(heap[i] > heap[j])
            flag=0;
        else
        {
            temp=heap[i];
            heap[i]=heap[j];
            heap[j]=temp;
            i=j;
        }
    }
}
```

Output

```
Enter no. of elements:5

Enter elements:12 8 46 23 7

Array after sorting:
7 8 12 23 46
```

Experiment No:-14

Aim:- Write a C program to implement construct binary tree and binary tree traversal.

Objective:-

Student should be able to develop a program from the logic of tree traversal techniques.

Theory:-

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot random access a node in a tree. There are three ways which we use to traverse a tree −

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Code:

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

struct node {
   int data;

   struct node *leftChild;
   struct node *rightChild;
};

struct node *root = NULL;

void insert(int data) {
   struct node *tempNode = (struct node*) malloc(sizeof(struct node));
   struct node *current;
   struct node *parent;

   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;
   //if tree is empty
   if(root == NULL) {
      root = tempNode;
   } else {
      current = root;
      parent = NULL;

      while(1) {
         parent = current;

         //go to left of the tree
         if(data < parent->data) {
            current = current->leftChild;

            //insert to the left
            if(current == NULL) {
               parent->leftChild = tempNode;
               return;
            }
         }  //go to right of the tree
         else {
            current = current->rightChild;

            //insert to the right
            if(current == NULL) {
               parent->rightChild = tempNode;
               return;
            }
         }
      }
   }
}

struct node* search(int data) {
   struct node *current = root;
   printf("Visiting elements: ");

   while(current->data != data) {
      if(current != NULL)
         printf("%d ",current->data);

      //go to left tree
      if(current->data > data) {
         current = current->leftChild;
      }
      //else go to right tree
      else {
         current = current->rightChild;
      }

      //not found
      if(current == NULL) {
         return NULL;
      }
   }

   return current;
}

void pre_order_traversal(struct node* root) {
   if(root != NULL) {
      printf("%d ",root->data);
      pre_order_traversal(root->leftChild);
      pre_order_traversal(root->rightChild);
   }
}

void inorder_traversal(struct node* root) {
   if(root != NULL) {
      inorder_traversal(root->leftChild);
      printf("%d ",root->data);
      inorder_traversal(root->rightChild);
   }
}

void post_order_traversal(struct node* root) {
   if(root != NULL) {
      post_order_traversal(root->leftChild);
      post_order_traversal(root->rightChild);
      printf("%d ", root->data);
   }
}

int main() {
   int i;
   int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

   for(i = 0; i < 7; i++)
      insert(array[i]);

   i = 31;
   struct node * temp = search(i);

   if(temp != NULL) {
      printf("[%d] Element found.", temp->data);
      printf("\n");
   }else {
      printf("[ x ] Element not found (%d).\n", i);
   }

   i = 15;
   temp = search(i);

   if(temp != NULL) {
      printf("[%d] Element found.", temp->data);
      printf("\n");
   }else {
      printf("[ x ] Element not found (%d).\n", i);
   }

  printf("\nPreorder traversal: ");
  pre_order_traversal(root);

  printf("\nInorder traversal: ");
  inorder_traversal(root);

  printf("\nPost order traversal: ");
  post_order_traversal(root);

  return 0;
}
```

If we compile and run the above program, it will produce the following result −

Output

```
Visiting elements: 27 35 [31] Element found.
Visiting elements: 27 14 19 [ x ] Element not found (15).

Preorder traversal: 27 14 10 19 35 31 42
Inorder traversal: 10 14 19 27 31 35 42
Post order traversal: 10 19 14 31 42 35 27
```

4. Appendix

5. Quiz on the subject

Q 1 - A complete graph can have

A - n^2 spanning trees

B - $n^{n - 2}$ spanning trees

C - $n^{n + 1}$ spanning trees

D - n^n spanning trees

Answer : B

Explanation

At maximum, a complete graph can have $n^{n - 1}$ spanning trees.

Q 2 - Stack is used for

A - CPU Resource Allocation

B - Breadth First Traversal

C - Recursion

D - None of the above

Answer : C

Explanation

Recursive procedures use stacks to execute the result of last executed procedural call.

Q 3 - Which of the following asymptotic notation is the worst among all?

A - O(n+9378)

B - O(n^3)

C - $n^{O(1)}$

D - $2^{O(n)}$

Answer : B

Explanation

O(n+9378) is n dependent

O(n^3) is cubic

$n^{O(1)}$ is polynomial

$2^{O(n)}$ is exponential

Q 4 - Minimum number of spanning tree in a connected graph is

A - n

B - n^{n-1}

C - 1

D - 0

Answer : C

Explanation

Every connected graph at least has one spanning tree.

Q 5 - Which of the following has search effeciency of O(1) −

A - Tree

B - Heap

C - Hash Table

D - Linked-List

Answer : C

Explanation

A simple hash table has the Ω(1) efficiency.

Q 6 - What will be the running-time of Dijkstra's single source shortest path algorithm, if the graph G(V,E) is stored in form of adjacency list and binary heap is used −

A - O(|V|^2)

B - O(|V| log |V|)

C - O(|E|+|V| log |V|)

D - None of these

Answer : C

Explanation

The runing time will be O(|E|+|V| log |V|) when we use adjacency list and binary heap.

Q 7 - Which of the following is not possible with an array in C programming langauge −

A - Declaration

B - Definition

C - Dynamic Allocation

D - Array of strings

Answer : C

Explanation

Array in C are static and cannot be shrinked or expanded in run-time.

Q 8 - In C programming, when we remove an item from bottom of the stack, then –

A - The stack will fall down.

B - Stack will rearranged items.

C - It will convert to LIFO

D - This operation is not allowed.

Answer : B

Explanation

Stack can only be accessed from top of it.

Q 9 - A pivot element to partition unsorted list is used in

A - Merge Sort

B - Quick Sort

C - Insertion Sort

D - Selection Sort

Answer : B

Explanation

The quick sort partitions an array using pivot element and then calls itself recursively twice to sort the resulting two subarray.

Q 10 - A stable sorting alrithm −

A - does not crash.

B - does not run out of memory.

C - does not change the sequence of appearance of elements.

D - does not exists.

Answer : C

Explanation

A stable sorting algorithm like bubble sort, does not change the sequence of appearance of similar element in the sorted list.

Q 11 - A complete graph can have

A - n^2 spanning trees

B - n^{n-2} spanning trees

C - n^{n+1} spanning trees

D - n^n spanning trees

Answer : B

Explanation

At maximum, a complete graph can have n^{n-1} spanning trees.

Q 12 - Stack is used for

A - CPU Resource Allocation

B - Breadth First Traversal

C - Recursion

D - None of the above

Answer : C

Explanation

Recursive procedures use stacks to execute the result of last executed procedural call.

Q 13 - A stable sorting alrithm −

A - does not crash.

B - does not run out of memory.

C - does not change the sequence of appearance of elements.

D - does not exists.

Answer : C

Explanation

A stable sorting algorithm like bubble sort, does not change the sequence of appearance of similar element in the sorted list.

Q 14. What is a dynamic array?

a) A variable size data structure

b) An array which is created at runtime

c) The memory to the array is allocated at runtime

d) An array which is reallocated everytime whenever new elements have to be added

View Answer

Answer: a
Explanation: It is a varying-size list data structure that allows items to be added or removed, it may use a fixed sized array at the back end.

Q 15. What is an AVL tree?

a) a tree which is balanced and is a height balanced tree

b) a tree which is unbalanced and is a height balanced tree

c) a tree with three children

d) a tree with atmost 3 children

View Answer

Answer: a

Explanation: It is a self balancing tree with height difference atmost 1.

6. Conduction of Viva-Voce Examinations

Define data structure and it’s types
What is the difference between linear and non linear data structure
How to declare the pointer.
How to declare the structure.
What is mean by array
How polynomial is represented using array
How polynomial addition takes place by using array
How polynomial addition takes place by using structure
What is the difference between structure and array
What is the mean by dynamic array
What is the difference between static variable and dynamic variable
What is the difference between static array and dynamic array
How the memory is utilized in structure and union
Define two dimensional array
Define multi dimensional array
What is performance analysis
Define algorithm
What is the criteria for algorithm specification
Define space and time complexity
What is performance measurement
Definition of Stack
What are the different
operations on the stack.
Applications of the stack.
What is dynamic stack
Define PUSH operation
Define POP operation.
What is the drawback of stack
Define multiple stack
Definition of Queue
How to use queue in program
Explain the operations on the Queue
Types of Queue
Application of Queue

Disadvantage of queue
Define circular queue
What is dynamic circular queue
What is the difference between stack and queue
Difference between simple queue and circular queue
How to insert node in the linked list.
Define the Header.
Define BINARY tree
Define BST
What is the difference between binary tree and BST
What is AVL tree
What is heap and it's types
Difference between Min heap and Max heap
Tree representation methods
How to represent binary tree in memory location
Operations on binary tree.
Traversals of the binary tree

7. Evaluation and Marking System

Basic honesty in the evaluation and marking system is absolutely essential and in the process impartial nature of the evaluator is required in the examination system to become popular amongst the students. It is a wrong approach or concept to award the students by way of easy marking to get cheap popularity among the students to which they do not deserve. It is a primary responsibility of the teacher that right students who are really putting up lot of hard work with right kind of intelligence are correctly awarded.

The marking patterns should be justifiable to the students without any ambiguity and teacher should see that `students are faced with unjust circumstances.

The assessment is done according to the directives of the Principal/ Vice Principal/ Dean Academics

DOs and DON' Ts in Laboratory:

1. Do not handle any equipment before reading the instructions/Instruction manuals.

2. Read carefully the program of the equipment before it is switched on whether ratings 230 V/50Hz or 115V/60 Hz. For Indian equipments, the power ratings are normally 230V/50Hz. If you have equipment with 115/60 Hz ratings, do not insert power plug, as our normal supply is 230V/50 Hz, which will damage the equipment.

3. Observe type of sockets of equipment power to avoid mechanical damage

4. Do not forcefully place connectors to avoid the damage

5. Strictly observe the instructions given by the teacher/Lab Instructor

Instruction for Laboratory Teachers::

1. Submission related to whatever lab work has been completed should be done during the next lab session.

2. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students