VTU
CSE

DATA STRUCTURES LAB MANUAL

COMPILED FOR “FSMK”
BY – VASUKI VARDHAN.G

Introduction to GCC C-Compiler

GCC is an alternative to the Turbo C compiler. It provides a variety of features and supports many languages apart from C itself.

When you compile a program , "gcc" checks the source code for errors and creates a binary object file of that code (if no errors exist). It then calls the linker to link your code's object file with other pre-compiled object files residing in libraries. These linked object binaries are saved as your newly compiled program.

Options can be provided to gcc to dictate the way a process is performed. For example, you could tell "gcc" to just create the object file and skip the linking specially when developing large programs or building your own libraries.

Options used in GCC:
The important options commonly used in gcc are-

* -Wall
* -L{directory_name}
* -l{library}
* -o{file_name}

where:
{library} denotes a library file
{file_name} denotes the name of a Unix file
{directory_name} name of the directory

Example: main.c

```
#include<stdio.h>
int main(void)
      {
        printf("FSMK");
        return 0;
      }
```

Compilation

```
gcc main.c
```

Explanation of the options:
-Wall
> This option enables all the warnings in GCC.

-o
> This is to specify the output file name for the executable.
ex: **gcc main.c -o main**

-l
> Tells the linker to search a standard list of directories for the library (i.e,used to link with shared libraries). The linker then uses this file as if it had been specified precisely by name.
ex: **gcc main.c -lccp**

-lm

> To use the library math.h, use the -lm option during compilation.
  ex: **gcc main.c -lm**

-L
> Tells the linker to search standard system directories plus user specified directories.

-g
> Generates additional symbolic debuggging information for use with gdb debugger.

-C
> Produce only the compiled code (without any linking)
  ex: **gcc -C main.c**

-D
> The compiler option -D can be used to define the macro MY_MACRO from command line.
  ex: **gcc -DMY_MACRO main.c**

-fopenmp
> This option is used to enable the different OpenMP directives (#pragma omp). This option along with -static is used to link OpenMP.
  ex: **gcc main.c -fopenmp -o main**

Syntatic differences between Turbo C and GCC:

* The library conio.h is not used in GCC. Hence, getch() and other functions using conio.h do not work.
* The function clrscr() does not work.
* Since GNU/Linux environment always expects a running process to return an exit status when the process is completed, the main() function in C Programs should always return an integer instead of returning void.

Advantages of GCC:

1. GCC is free.
2. Supports multiple languages (C,C++,Java etc)
3. GCC is portable. Runs on almost all platforms.
4. Generates backend code.

Resources
* Please go through the video tutorials on C Programming and GCC developed and released by **Spoken Tutorial Project**, an initiative of National Mission on Education through ICT, Government of India, to promote IT literacy through Open Source Software. Students can go through these video tutorials to get better understanding of the subject. The tutorials can be downloaded from [here](http://files.spoken-tutorial.org/disc-source/c-and-c-plus-plus.zip). More info about the project can be found [here](http://spoken-tutorial.org/)

Introduction To Eclipse C/C++ IDE

Eclipse is a complete GUI Based IDE for C/C++ on both Windows and Linux.This also uses GCC at the backend. A perfect IDE for Newbies.

Step By Step Setup Procedure
1) Download Eclipse from https://eclipse.org/downloads/
2) Extract the .zip/.tgz file to a directory
3) And run the Eclipse executable

Program 1 : polynomial.c
Aim:
Using circular representation for a polynomial, design, develop, and execute a program in C to accept two polynomials, add them, and then print the resulting polynomial

Theory:
This program shows the operation of polynomial with the help of circulat linked list. Circulat linked list is a linked list in which the head element and the tail element's next pointer points to the head element.In the special case of a circular list with only one element, the element's previous and next pointers point to itself, and it is both the head and tail of the list.

Algorithm

1.Start

2.Get the coefficients and powers for the two polynomials to be added.

3.Add the coefficients of the respective powers.

4.Display the added polynomial.

5.Stop.

Source Code:

```
   #include<stdio.h>
   #include<stdlib.h>
   typedef struct abb
   {
     int coeff;
     int expo;
     struct abb* link;

    }node;
    node* get_node()
    {
      return((node*)malloc(sizeof(node)));
    };


    void insert(node*header,node*temp)
    {
      node*p=header;
      node*prev;
      if(header->link==NULL)
       {
         header->link=temp;
         temp->link=header;
       }
      else
       {
        p=header->link;
        while(p->coeff!=header->coeff)
        {
            prev=p;
            p=p->link;
        }
        prev->link=temp;
        temp->link=header;
       }
     }


    void display(node*header)
     {
       node*p=header->link;
       while(p->coeff!=header->coeff)
       {
        printf("%dx^%d ",p->coeff,p->expo);
        p=p->link;
        }
      }


    void cadd(node*header1,node*header2,node*header3,int m,int n)
    {
      node*temp1=header1->link;
      node*temp2=header2->link;
      node*temp;
      int x=0,y=0,i;
      while(x!=m&&y!=n)
      {
        if((temp1->expo)>(temp2->expo))
         {
           temp=get_node();
           temp->expo=temp1->expo;
           temp->coeff=temp1->coeff;
           temp->link=NULL;
           insert(header3,temp);
           temp1=temp1->link;
           x++;
          }
        else if((temp1->expo)<(temp2->expo))
         {
           temp=get_node();
           temp->expo=temp2->expo;
           temp->coeff=temp2->coeff;
           temp->link=NULL;
           insert(header3,temp);
           temp2=temp2->link;
           y++;
          }
        else
        {
           temp=get_node();
           temp->coeff=temp1->coeff+temp2->coeff;
           temp->expo=temp1->expo;
           temp->link=NULL;
           temp1=temp1->link;
           temp2=temp2->link;
           insert(header3,temp);
           x++;y++;
         }
      }
      if(x==m)
       {
        for(i=y+1;i<n;i++)
        {
           temp=get_node();
           temp->expo=temp2->expo;
           temp->coeff=temp2->coeff;
           temp->link=NULL;
           insert(header3,temp);
           temp2=temp2->link;
         }
       }
      if(y==n)
       {
        for(i=x+1;i<m;i++)
         {
            temp=get_node();
            temp->expo=temp2->expo;
            temp->coeff=temp2->coeff;
            temp->link=NULL;
            insert(header3,temp);
            temp1=temp1->link;
         }
        }
     }



  int main()
     {
      node*temp;
      int m,n,i;
      node*header1=get_node();
      header1->link=NULL;
      header1->coeff=header1->expo=-1;
      node*header2=get_node();
      header2->link=NULL;
      header2->coeff=header2->expo=-2;
      node*header3=get_node();
      header3->link=NULL;
      header3->coeff=header3->expo=-3;
      printf("Enter the number of terms of the first polynomial\n");
```

```
        scanf("%d",&m);
        for(i=0;i<m;i++)
         {
           temp=get_node();
           printf("Enter the coeff and expo to be inserted\n");
           scanf("%d%d",&temp->coeff,&temp->expo);
           temp->link=NULL;
           insert(header1,temp);
         }
        printf("Enter the number of terms of the second polynomial\n");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
           temp=get_node();
           printf("Enter the coeff and expo to be inserted\n");
           scanf("%d%d",&temp->coeff,&temp->expo);
           temp->link=NULL;
           insert(header2,temp);
        }
        printf("\nThe first polynomial is");
        display(header1);
        printf("\nThe second polynomial is");
        display(header2);
        printf("\nThe resultant addition polynomial is");
        cadd(header1,header2,header3,m,n);
        display(header3);
        return 0;
       }
```

OUTPUT:

```
compilation command: cc polynomial.c/g++ -c polynomial.c --> g++ -o polynomial.o polynomial
## output command: ./a.out

Enter the number of terms of the first polynomial

4

Enter the coeff and expo to be inserted

4 4

Enter the coeff and expo to be inserted

3 3

Enter the coeff and expo to be inserted

2 2

Enter the coeff and expo to be inserted
```

```
1 1

Enter the number of terms of the second polynomial

3

Enter the coeff and expo to be inserted

6 5

Enter the coeff and expo to be inserted

4 3

Enter the coeff and expo to be inserted

3 1

The first polynomial is 4x^4 3x^3 2x^2 1x^1

The second polynomial is 6x^5 4x^3 3x^1

The resultant addition polynomial is 6x^5 4x^4 7x^3 2x^2 4x^1 */
```

PROGRAM 2 : infix.c

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

THEORY:
Infix Expression : Any expression in the standard form like "2*3-4/5" is an Infix(Inorder) expression.

Postfix Expression : The Postfix(Postorder) form of the above expression is "23*45/-".

Infix to Postfix Conversion : In normal algebra we use the infix notation like a+b*c. The corresponding postfix notation is abc*+. The algorithm for the conversion is as follows :

* Scan the Infix string from left to right.
* Initialise an empty stack.
* If the scannned character is an operand, add it to the Postfix string. If the scanned character is an operator and if the stack is empty Push the character to stack.
* If the scanned character is an Operand and the stack is not empty, compare the precedence of the character with the element on top of the stack (topStack). If topStack has higher precedence over the scanned character Pop the stack else Push the scanned character to stack. Repeat this step as long as stack is not empty and topStack has precedence over the character.
  Repeat this step till all the characters are scanned.
* (After all characters are scanned, we have to add any character that the stack may have to the Postfix string.) If stack is not empty add topStack to Postfix string and Pop the stack. Repeat this step as long as stack is not empty.
* Return the Postfix string.

ALGORITHM:

1. Start the program

2. Scan the Infix string from left to right.

3. Initialise an empty stack.

4. If the scannned character is an operand, add it to the Postfix string. If the scanned character is an operator and if the stack is empty Push the character to stack.
if the scanned character is an Operand and the stack is not empty,compare the precedence of the character with the element on top of thestack (topStack). If topStack has higher precedence over the scannedcharacter Pop the stack else Push the scanned character to stack.Repeat this step as long as stack is not empty and topStack has precedence over the character.Repeat this step till all the characters are scanned.

5. (After all characters are scanned, we have to add any character that the stack may have to the Postfix string.)If stach is not empty add top stack to prefix string and pop the stack.Repeat this step as long as stack is not empty.

6. Return the postfix string.

7. Terminate the program.

CODE: post.cpp

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

    int top=-1,length,iinf=0,ipostf=0;
    int precd(char);
    char pop();
    void inf_to_postf(char[],char[]);
    void push(char);
    char infix[20],postf[20],stk[20],sym;

    int main()
    {

    printf("\nConversion of infix to postfix expression");
    printf("\nEnter the infix expression to be converted:");
    gets(infix);
    inf_to_postf(infix,postf);
    printf("\nEntered  infix expression is:%s ",infix);
    printf("\nConverted  infix to postfix expression is : %s \n",postf);
    }
     void inf_to_postf(char infix[],char postf[]) //function for conversion of infix to postfix
    {
    char temp;
    push('#');
    while(infix[iinf]!='\0')
    {
      sym=infix[iinf];
      switch(sym)
    {

       case '(':push(sym);
              break;

       case ')':temp=pop();
              while(sym!='(')
              {
               postf[ipostf++]=temp;
               temp=pop();
              }
               break;

       case '+':
       case '-':
       case '*':
       case '/':
               while(precd(stk[top])>=precd(sym))
               {
                    temp=pop();
                     postf[ipostf++]=temp;
                   }
                 push(sym);
                   break;
        default: postf[ipostf++]=sym;
    }
    iinf++;
    }
    while(top>0)
    {

     temp=pop();
     postf[ipostf++]=temp;
    }
    }
    void push(char x)
    {
    stk[++top]=x;
    }
    char pop()
    {
    int x;
    x=stk[top];
    top--;
    return x;
    }

int precd(char x)
{
 int p;
switch(x)
{

 case '+':
        case '-': p=1;
                  break;
        case '*':
        case '/': p=2;break;
        case '(':
        case ')':p=0;break;
        case'#': p=-1;break;
}
return p;
}
```

Output:

```
Conversion of infix to postfix expression

Enter the infix expression to be converted:a+b-c*d/e
Entered  infix expression is:a+b-c*d/e

Converted  infix to postfix expression is : ab+cd*e/-
```

PROGRAM 3 : postfix.c

Name of the Program:
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.

Theory:
A stack is a particular kind of abstract data type or collection in which the principal operations on the collection are the addition of an entity to the collection, known as push and removal of an entity, known as pop. The relation between the push and pop operations is such that the stack is a Last-In-First-Out (LIFO) data structure. In a LIFO data structure, the last element added to the structure must be the first one to be removed. This is equivalent to the requirement that, considered as a linear data structure, or more abstractly a sequential collection, the push and pop operations occur only at one end of the structure, referred to as the top of the stack. Often a peek or top operation is also implemented, returning the value of the top element without removing it.

All the operators in the expression are pushed in the stack and then popped out when an operand is obtained.

Algorithm:
```
1. Read the postfix expression as a string.
2. Scan the expression character by character till the end. Repeat the following operations
   a.  If it is an operand push it onto the stack.
   b.  If it is an operator
1. Pop out two operands from stack
2. Apply the operator onto the popped operands.
3. Store the result back on to the stack.
4. On reaching the end of expression pop out the contents of the stack and
display as the result.
```

Code:
postfixEval.cpp

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#define MAXSIZE 30
int s[MAXSIZE];
int top=-1;
int isdig(char);
int main()
{
        char symbol,postfix[30];
        int a,b,res,i;
        void push(int);
        int pop();
        int op(int, int, char);
```

```
	printf("Postfix expression\n");
	scanf("%s",postfix);
	for(i=0;i<strlen(postfix);i++)
	{
		symbol=postfix[i];
		if(isdig(symbol))
			push(symbol-'0');
		else
		{
			a = pop();
			b = pop();
			res = op(b,a,symbol);
			push(res);
		}
	}
	printf("The result of the expression is = ");
	printf("%d\n",pop());
	return 0;
}
int pop()
{
	if(top!=-1)
		return s[top--];
	else
	{
		printf("Stack underflow\n");
		return 0;
	}
}
void push(int item)
{
	if(top!= MAXSIZE-1)
		s[++top]=item;
	else
		printf("\nStak Overflow\n");
}
int op(int op1,int op2,char symbol)
{
	switch(symbol)
	{
		case '+': return op1 + op2;
		case '-': return op1 - op2;
		case '*': return op1 * op2;
		case '/': return op1 / op2;

	}
}
int isdig(char symbol1)
{
	return (symbol1>='0' && symbol1<='9');
}
```

Steps for obtaining output:
Steps for checking output-

* Locate the folder in which the file is present in the terminal.
* Execute the command "gcc the <filename.cpp>"
* Execute ethe command "./a.out"

Output:
**

```
*Postfix expression
* 123*+4-
* The result of the expression is = 3
```

PROGRAM 4 : queue.c
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:
1.Insert
2.Delete
3.Display

Theory:
Queue is a perticular kind of collection in which the entities in collection are kept in order and the principal operations on the collection are the addition of the entities to the rear terminal position and removal of entities from the front terminal position. Thus queue works in First In First Out(FIFO) order.

Algorithm:
1. Start
2. To insert an element to the queue first check if queue is full. If it is full display a message saying 'queue overflow', else insert the element from rear end of the queue.
3. To delete an element from the queue check if there are elements in queue which has to be deleted. If there is no elements in the queue print a message saying 'queue underflow', else delete the element from front end of the queue.
4. Display the contents of the queue.
5. Stop.

Code:

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

   void add(int,int[]);              //declare functions required to perform neccessary operations.
   void del(int[]);
   void display(int[]);
   int q[10];
   int front=-1,rear=-1;

   void main()
   {
    int ch,el;

    // provide options to select specific operation.
    do{

                printf("\n\t\tMENU:\n\t\t1.Add an element\n\t\t2.Delete an element\n\t\t3.Display
Queue\n\t\tAnother to Exit\n");
                scanf("%d",&ch);
                switch(ch)
              {
                    case 1:        printf("\nEnter the element to be added\n");
                    scanf("%d",&el);
                    add(el,q);
                    break;
                    case 2: del(q);
                    break;
                    case 3: display(q);
                    break;
                    default:exit(0);
              }
      }while(ch);
    }

   // add fuction performs insertion of an element to the queue.

   void add(int a,int q[10])
   {
    if(rear<10)
              q[++rear]=a;
    else
              printf("\nQueue Overflow");
   }

   //del function performs deletion of an element from the queue.

   void del(int q[10])
   {

    if(front<rear)
       {
              printf("\nElement deleted is %d",q[++front]);
          q[front]=NULL;
       }
    else
              printf("\nQueue underflow");
    }

   // display function displays the elements in the queue
   void display(int q[10])
   {

   int temp=front;
   if(front==rear)
   printf("queue is empty\n");
   else
    {
     printf("QUEUE: ");
     while(temp<rear)
              printf("%d ",q[++temp]);
    }
   }
```

OUTPUT

compilation command:cc queue.c

Output command: ./a.out

```
                MENU:
                1.Add an element
                2.Delete an element
                3.Display Queue
                Another to Exit
1

Enter the element to be added
3

                MENU:
                1.Add an element
                2.Delete an element
                3.Display Queue
                Another to Exit
3
QUEUE: 3
                MENU:
                1.Add an element
                2.Delete an element
                3.Display Queue
                Another to Exit
2

Element deleted is 3
                MENU:
                1.Add an element
                2.Delete an element
                3.Display Queue
                Another to Exit
3
queue is empty

                MENU:
                1.Add an element
                2.Delete an element
                3.Display Queue
                Another to Exit
```

PROGRAM – 5 : employee.cpp

AIM: Design, develop, and execute a program in C++ based on the following requirements:
in EMPLOYEE class is to contain the following data members and member functions:

Data members: Employee_Number (an integer), Employee_Name (a string of characters),Basic_Salary (an integer) , All_Allowances(an integer), IT (an integer), Net_Salary (an integer).

Member functions: to read the data of an employee,to calculate Net_Salary and to print the values of all the data members.
(All_Allowances = 123% of Basic; Income Tax (IT) = 30% of the gross salary (= basic_Salary _ All_Allowance), Net_Salary = Basic_Salary + All_Allowances � IT)

THEORY:
CREATE an employee class and insert the data members and member function. That is,

Data members: Employee_Number (an integer), Employee_Name (a string of characters),Basic_Salary (an integer) , All_Allowances(an integer), IT (an integer), Net_Salary (an integer).

Member functions: to read the data of an employee,to calculate Net_Salary and to print the values of all the data members.

(All_Allowances = 123% of Basic; Income Tax (IT) = 30% of the gross salary (= basic_Salary _ All_Allowance), Net_Salary = Basic_Salary + All_Allowances � IT)

ALGORITHM:
1. start
2. create the class employee.
3. create the functions getdata,display,cal_sal and getnum.
4. end

CODE: emp.cpp
```
#include<iostream>
using namespace std;

class employee
{
   char name[15];
   int id;
   float basic,sal,netsal,da,it;

public:
         void getdata();
         void display();
         float cal_sal();
         int getnum();
};
```

```
void employee::getdata()
{
   cout<<" ID : ";
   cin>>id;

   cout<<" Name : ";
   cin>>name;

   cout<<" Basic Salary : ";
   cin>>basic;
}

void employee::display()
{


   cout.width(5);

   cout<<"id:"<<id;
   cout.width(15);
   cout<<"name:"<<name;
   cout.width(10);
   cout<<"basic:"<<basic;
   cout.width(10);
   cout<<"it:"<<it;
   cout.width(10);
   cout<<"da:"<<da;
   cout.width(10);
   cout<<"netsal:"<<netsal;

}

int employee::getnum()
{
   return id;
}

float employee::cal_sal()
{
   da = 1.23 * basic;
   sal = da + basic;
   it = 0.30 * sal;
   netsal = sal - it;
}


int main()
{
   short int i,j,n;
   employee emp[20];

   cout<<"Enter the number of Employees : ";
```

```
        cin>>n;

        for(i=0;i<n;i++)
        {
                emp[i].getdata();
                emp[i].cal_sal();
                emp[i].display();
    }
    return 0;
    }
```

Output:

```
 Enter the number of Employees : 1

 ID : 1

 Name : ram

 Basic Salary : 10000

  id:1
  name:ram
  basic:10000
  it:6690
  da:12300
  netsal:15610
```

PROGRAM – 6: string.cpp

Name of the Program:
Design, develop, and execute a program in C++ to create a class called STRING and implement the following operations. Display the results after every operation by overloading the operator <<
i) STRING s1 = "VTU"
ii) STRING s2 = "BELGAUM"
iii) STIRNG s3 = s1 + s2; (Use copy constructor)

Theory:
This program using the the basic fundamentals of C++.
Using this program we overload the '**+**' operator.
This program uses the concept of copy constructor and friend ostream function.
The copy constructor is a special constructor in the C++ programming language for creating a new object as a copy of an existing object.
The friend ostream dunction helps us to overload '**<<**' operator as it isnt part of the class.

Algorithm:
1. Declare an object called '**STRING**'
2. Declare a character array(string).
3. Declare the default constructor intiliazing the string to NULL.
4. Declare a parameterized constructor passing a string to the constructor and initialize the string of the object to the string passed in the constructor
5. Declare a copy constructor copying the contents of existing object to the new object.
6. Using friend ostream function overload '**<<**' operator.
7. Create a funtion '**operator +()**' passing the object as ththe parameter to it and overload the **+** operator in it.
8. In the main function declare three objects of class STRINGre
9. Intialize the object 1 string to **VTU**.
10. Intialize the object 2 string to **BELGAUM**
11. Intialize the object 3 by adding ob1+obj2(using the + operator).
12. Display all the strings.

Code:
stringConcat.cpp

```
#include<iostream>
#include<string.h>
using namespace std;
class STRING
{
        char  str[100];
        public:
        STRING()
        {
                strcpy(str," ");
        }
        STRING(char *s1)
        {
                strcpy(str,s1);
        }
```

```
            STRING(STRING& s)
            {
                        strcpy(str,s.str);
            }
            STRING operator +(STRING );
            friend ostream& operator <<(ostream& c,STRING s1)
            {
                        c<<s1.str;
                        cout<<endl;
                        return c;
            }
            };
   int main()
   {
            STRING s1,s2,s3;
            s1=(char *)"VTU";
            s2=(char *)"BELGAUM";
            s3=s1+s2;
            cout<<"String s1 is = "<<s1;
            cout<<"\n";
            cout<<"String s2 is = "<<s2;
            cout<<"\n";
            cout<<"The concatenated string is ";
            cout<<s3<<"\n";
            return 0;
   }
   STRING STRING::operator +(STRING s)
   {
            strcat(str,s.str);
         return (*this);
   }
```
Output:
Steps for checking output-

* Locate the folder in which the file is present in the terminal.
* Execute the command "g++ the <filename.cpp>"
* Execute ethe command "./a.out"

PROGRAM 7 : stack.cpp

Design, develop, and execute a program in C++ to create a class called STACK using an array of integers and to implement the following operations by overloading the operators + and - :
a) S1=S1 + element; where S1 is an object of the class STACK and element is an integer to be pushed on to top of the stack.
b) S1=S1- ; where S1 is an object of the class STACK and - operator pops off the top element.
Handle the STACK Empty and STACK Full conditions. Also display the contents of the stack after each operation, by overloading the operator <<

Theory:
A stack is a type of data structure

Algorithm:
1.Start
2.Check if the stack is full.The condition in which the stack is full is denoted by 'top=N-1', where N is the size of the stack(this known as stack overflow)
3.If stack is not full, insert the element at the top of the stack.

Code:
```
#include <iostream>
#include <stdlib.h>
#define size 20
using namespace std;
class stack
{
 int a[size],top,ms;
 public:
  stack()
  {
          top=-1; cout<<"MAX:";cin>>ms;
 }

 stack operator +(int ele)
 {
   if(top>ms-2)
   cout<<"Stack Overflow\n";
   else
       {
            top++;
            a[top]=ele;
        }

       return (*this);
 }

  stack operator --(int)
  {
    if(top==-1)
```

```
            cout<<"Stack Underflow";
        else
        cout<<"\nElement deleted is "<<a[top--];
        return (*this);
     }

     friend ostream& operator <<(ostream& c,stack s)
     {
        if(s.top==-1)
              cout<<"Stack Empty";
        else
        for(int j=s.top;j>=0;j--)
                     cout<<" "<<s.a[j];
        return c;
     }
    }s;
    int main()
    {
      int ch,ele;
      while(1){
      cout<<"\nEnter your choice \n1:Enter an element\n2:Delete an element\n3:Display\nAnyother
to Exit";
      cin>>ch;
      switch(ch)
      {
          case 1:cout<<"Enter the element:\n";cin>>ele;s=s+ele;break;
          case 2:s=s--;break;
          case 3:cout<<s;break;
          default:exit(0);
       }
      }
    return 0;
}
```

OUTPUT

```
MAX: 20

a.Enter your choice

1:Enter an element

2:Delete an element

3:Display

Anyother to Exit 1

Enter the element:

 12
```

```
b.Enter your choice

1:Enter an element

2:Delete an element

3:Display

Anyother to Exit 1

Enter the element:

 12

c.Enter your choice

1:Enter an element

2:Delete an element

3:Display

Anyother to Exit 1

Enter the element:

34

d.Enter your choice

1:Enter an element

2:Delete an element

3:Display

Anyother to Exit 2

Element deleted is 34


e.Enter your choice

1:Enter an element

2:Delete an element

3:Display

Anyother to Exit 3

 12
```

PROGRAM 8 : list.cpp

Name of the Program:
Design, develop, and execute a program in C++ to create a class called LIST (linked list) with member functions to insert an element at the front of the list as well as to delete an element from the front of the list. Demonstrate all the functions after creating a list object.

Theory:
In computer science, a linked list is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a datum and a reference (in other words, a link) to the next node in the sequence; more complex variants add additional links. This structure allows for efficient insertion or removal of elements from any position in the sequence.

A linked list whose nodes contain two fields: an integer value and a link to the next node. The last node is linked to a terminator used to signify the end of the list.

![singly linked list](singlylist.png)

Linked lists are among the simplest and most common data structures. They can be used to implement several other common abstract data types, including lists (the abstract data type), stacks, queues, associative arrays, and S-expressions, though it is not uncommon to implement the other data structures directly without using a list as the basis of implementation.

The principal benefit of a linked list over a conventional array is that the list elements can easily be inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk. Linked lists allow insertion and removal of nodes at any point in the list, and can do so with a constant number of operations if the link previous to the link being added or removed is maintained during list traversal.

On the other hand, simple linked lists by themselves do not allow random access to the data, or any form of efficient indexing. Thus, many basic operations - such as obtaining the last node of the list (assuming that the last node is not maintained as separate node reference in the list structure), or finding a node that contains a given datum, or locating the place where a new node should be inserted - may require scanning most or all of the list elements.

Algorithm:
1. Create a class by name "list" having data members info and a link.
2. Member methods are insfrnt(), delfrnt() and display().
3. insfrnt() method adds a new node to the front of the list and assigns the data to it.
4. delfrnt() checks if the list is empty if not then the first node from the list is deleted.
5. display() displays the whole list by traversing through the list.
6. A class list is created with data members info,link and methods insfrnt(),delfrnt() and display().

Code:
list.cpp

```
#include <iostream>
#include <malloc.h>
#include <stdlib.h>

using namespace std;

class list
{
    int info;
    list *link;
    public:
            void insfrnt();
            int delfrnt();
            void display();
}*header;

int main()
{
    int a;
    list obj;
    while(1)
    {
        cout<<"\nEnter your choice\n1-Insert at front\n2-Delete at front\n3-Display\nAnyother to
exit\n";
        cin>>a;
       switch(a)
        {
            case 1:obj.insfrnt();break;
            case 2:obj.delfrnt();break;
            case 3:obj.display();break;
            default:exit(0);
        }
    }
    return 0;
}

void list::insfrnt()
{
    list *temp;
    int x;
    temp=new list;
    temp->link=NULL;
    cout<<"Enter the value of info:";
    cin>>x;
    temp->info=x;
    if(header==NULL)
        header=temp;
    else
    {
        temp->link=header;
        header=temp;
    }
```

```
   }

   int list::delfrnt()
   {

       if(header==NULL)
           cout<<"Empty list";
       else
       {
           list *temp;
           temp=header;
           cout<<temp->info;
           header=header->link;
           free(temp);
       }
   }

   void list::display()
   {
       list *temp;
       if(header==NULL)
           cout<<"Empty list";
       else
       {
           for(temp=header;temp!=NULL;temp=temp->link)
               cout<<temp->info<<" ";
           cout<<"\n";
       }

   }
```

Steps for obtaining output:
Steps for checking output-

* Locate the folder in which the file is present in the terminal.
* Execute the command "gcc the <filename.cpp>"
* Execute ethe command "./a.out"

Output:

```
<pre>Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:1
Enter the element to be inserted:12
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:1
Enter the element to be inserted:23
Enter the choice of operation:
```

```
1.Insert
2.Delete
3.Display
4.Exit:1
Enter the element to be inserted:34
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:1
Enter the element to be inserted:45
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:3
The elements of the list are...
45->34->23->12->NULL
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:2
The deleted element = 45
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:2
The deleted element = 34
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:3
The elements of the list are...
23->12->NULL
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:2
The deleted element = 23
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:3
The elements of the list are...
12->NULL
Enter the choice of operation:
1.Insert
```

```
2.Delete
3.Display
4.Exit:2
The deleted element = 12
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:3
The list is empty
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:2
The list is empty
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:4
Enter the choice of operation:
1.Insert
2.Delete
3.Display
4.Exit:4</pre>
```

PROGRAM 9 : sparse.c

Aim:
Design, develop, and execute a program in C to read a sparse matrix of integer values and to search the sparse matrix for an element specified by the user. Print the result of the search appropriately. Use the triple <row, column, value> to represent an element in the sparse matrix.

Theory:
A sparse matrix is a matrix populated primarily with zeros (Stoer & Bulirsch 2002, p. 619) as elements of the table. By contrast, if a larger number of elements differ from zero, then it is common to refer to the matrix as a dense matrix. The fraction of zero elements (non-zero elements) in a matrix is called the sparsity (density).

Algorithm:
1. Start.
2. Insert elements into the matrix in such a way that it is populated with more number of zeors.
3. Accept an element as key element.
4. Check the sparse matrix for the specified key element. If the key element is present, then print a message for successful search, else print that the search unsuccessful or element is not found in the sparse matrix.
5. Stop.

Code:

```
#include<stdio.h>
#define MAXSIZE 101

typedef struct
{
 int row;
  int col;
 int value;
}sparse;

sparse sp[MAXSIZE];

main()
{
 int m,n,a,i,key,f=0;
 printf("enter the order of matrix\n");
 scanf("%d%d",&m,&n);
 printf("enter the non-zero terms in sparse\n");
 scanf("%d",&a);
 printf("enter the row coloumn value\n");
 for(i=0;i<a;i++)
  {
 scanf("%d%d%d",&sp[i].row,&sp[i].col,&sp[i].value);
  }

 sp[0].row=m;
```

```
     sp[0].col=n;
     sp[0].value=a;

     printf("row\tcoloumn\tvalue\n");
     printf("%d\t%d\t%d\n",m,n,a);

     for(i=0;i<a;i++)
     printf("%d\t%d\t%d\n",sp[i].row,sp[i].col,sp[i].value);
     printf("enter the key element\n");
     scanf("%d",&key);

     for(i=1;i<=a;i++)
     {
     if(sp[i].value==key)
       f=1;
     }


     if(f==1)
     {

     printf("search successful");
     }
    else
    printf("search unsuccessful");
    }
```

OUTPUT

```
compilation command:cc sparse.c
output command: ./a.out
enter the order of matrix
2 2

enter the non-zero terms in sparse
3

enter the row coloumn value
0 1 1
1 0 2
1 1 3

row     coloumn         value
2       2       3
2       2       3
1       0       2
1       1       3

enter the key element
3

search successful
```

Program 10: heap.c
AIM:
Design, develop, and execute a program in C to create a max heap of integers by accepting one element at a time and by inserting it immediately in to the heap. Use the array representation for the heap. Display the array at the end of insertion phase.

THEORY:
A heap is a specialized tree-based data structure that satisfies the heap property: If A is a parent node of B then the key of node A is ordered with respect to the key of node B with the same ordering applying across the heap. Either the keys of parent nodes are always greater than or equal to those of the children and the highest key is in the root node (this kind of heap is called max heap) or the keys of parent nodes are less than or equal to those of the children and the lowest key is in the root node (min heap).

Heaps are crucial in several efficient graph algorithms such as Dijkstra's algorithm, and in the sorting algorithm heapsort.
A min-max heap is a complete binary tree containing alternating min and max levels

Example of Min-max heap
: If it is not empty, each element has a data member called key. The root is always present at min level. Let x be any node in a min-max heap. If x is on a min (max) level then the element in x has the minimum (maximum) key from among all elements in the subtree with root x. A node on a min (max) level is called a min (max) node.

ALGORITHM:

Max-Heapify(A, i)

// Input: A: an array where the left and right children of i root heaps (but i may not), i: an array index

// Output: A modified so that i roots a heap

//Running Time: O(log n) where n = heap-size[A] - i

```
1. l ← Left(i)
2. r ← Right(i)
3. if l ≤ heap-size[A] and A[l] > A[i]
4. largest ← l
5. else largest ← i
6. if r ≤ heap-size[A] and A[r] < A[largest]
7. largest ← r
8. if largest = i
9. exchange A[i] and A[largest]
10. Max-Heapify(A, largest)
```

Max-Heap-Insert(A, key)

//Input: A: an array representing a heap, key: a key to insert

//Output: A modified to include key

```
//Running Time: O(log n) where n =heap-size[A]

1 .heap-size[A] ←heap-size[A] + 1
2 .A[heap-size[A]] ← -∞
3 .Heap-Increase-Key(A[heap-size[A]], key)
```

CODE: heap.c

```
#include <stdio.h>
#include <stdlib.h>
#define max 20

void ins(int);
void heap(int);
void disp(int);
int f=0,t=1,h[max];

int main()
{
 int n;
 printf("Enter the size of array: ");
 scanf("%d",&n);
 ins(n);
 heap(n);
 disp(n);
 return 0;
 }


//ins function inserts the accepted input to the heap array.

void ins(int n)
{
 int i=0;
 for(;i<n;i++)
 {
     printf("Enter the element: ");
     scanf("%d",&h[i]);
 }
}

// disp funtion is used display the contents of the array after insertion.

void disp(int n)
{
  printf("Displaying elements in heap \n \n");
  int i=0;
  for(;i<n;i++)
   {
     printf("%d ",h[i]);
   }
 printf("\n\n");
```

```
     }

    // heap fuction is used to insert the accepted input to the heap.

    void heap(int n)
    {
     int i,v,k=0,j=0,he;
     for(i=n/2;i>=0;i--)
     {
        j=i;
        k=j-1;
        v=h[k];
        he=f;
        while(he==f && (2*k)<=n)
        {
            j=2*k+1;
            if(j<n)
            {
               if(h[j]<h[j+1])
                   j++;
            }
            if(v>=h[j])
               he=t;
            else
            {
               h[k]=h[j];
               k=j;
            }
         }
        h[k]=v;
      }
     }
```

OUTPUT:
Compilation command : cc heap.c
Output command: ./a.out

```
Enter the size of array: 7

Enter the element: 10

Enter the element: 2

Enter the element: 5

Enter the element: 15

Enter the element: 30

Enter the element: 1

Enter the element: 40

40 30 10 15 2 1 5
```

Program 11 : doublelinkedlist.c

Write a C program to support the following opearations on a doubly linked list where each node consists of integers.
a. Create a 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.

THEORY:
A doubly linked list is a linked data structure that consists of a set of data records, each having two special
link fields that contain references to the previous and to the next record in the sequence .It can be viewed as
two singly-linked lists formed from the same data items, in two opposi orders.
linked
opposite
A doubly-linked list whose nodes contain three fields: an integer value (data), the link to the next node (rlink),
linked
,
and the link to the previous node (llink). The two links allow walking along the list in either direction with
.
equal ease. Compared to a singly-linked list, modifying a doubly-linked list usually requires changing more
linked
linked
pointers, but is sometimes simpler because there is no need to keep track of the address of the previous node.

ALGORITHM:

Algorithm for adding each node at the front

```
1. create a new node using malloc function; .it returns address of the node to temp.
2. temp->info=info;
3. temp->llink=NULL
4. temp->rlink=NULL;
5. If first=NULL then first=temp .
6. temp-.>rlink=first
7. first->llink=temp; first=temp;
```

Algorithm for inserting a node to the left of the node

```
1. Create a new node using malloc function. It returns address of the node to temp.
2. temp->info=info
3. temp->llink=NULL
4. temp->rlink=NULL
5. Get the left node key value from user
6. if first= NULL print doubly linked list is empty.
7. if lvalue=first->info, call the function insert_front
```

```
8. start from the first node and traverse the node until the key is found; store that node
address in cur
9.temp->llink=cur->llink 35
10. (temp->llink)->rlink=temp
11. cur->llink=temp;
12. temp->rlink=cur
```

Algorithm for delete a node

```
1. set temp=first
2. if first=NULL print doubly linked list is empty.
3. Get node to be deleted from the user
4. if date=first ->info then first=temp->rlink and free the temp node, then first->llink=NULL.
5. start from the first node and traverse until delete key is found; then temp=temp->rlink
6. print the deleted node
7. (temp->rlink)->llink=temp->llink
8. (temp->llink)->rlink=temp->rlink
```

Algorithm for display

```
1. set temp=first
2. if first=NULL print list is empty
3. while(temp!=NULL) print temp->info and then temp=temp->rlink
```

CODE:

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

struct abb
{
    int info;
    struct abb *p, *n;
 };
typedef struct abb *node;

node header=NULL;

node getnode();
void ins();
void insl();
void del();
void disp();

int main()
{
     int ch;
     while(1)
     {
             printf("\nChoices:");
             printf("\n\t1-Insert");
             printf("\n\t2-Insert left");
```

```
                printf("\n\t3-Delete node");
                printf("\n\t4-Display");
                printf("\n\t5-Exit");
                printf("\nEnter your choice: ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: ins();
                                break;
                        case 2: insl();
                                break;
                        case 3: del();
                                break;
                        case 4: disp();
                                break;
                        default:return 0;
                }
        }
}

node getnode()
{
        node x;
        x=(node) malloc(sizeof(struct abb));
        return x;
}

void ins()
{
        node temp;
        int x;
        temp=getnode();
        printf("\nEnter the element to be inserted: ");
        scanf("%d",&x);
        temp->info=x;
        temp->p=NULL;
        temp->n=NULL;
        if(header==NULL)
                header=temp;
        else
        {
                temp->n=header;
                header->p=temp;
                header=temp;
        }
}

// function insl performs insertion to the left of the left of the node.
void insl()
{
        node temp,ele;
        int x, y;
```

```
    if(header==NULL)
    {
        printf("\nEmpty list\n");
        return;
    }
    printf("\nEnter the element to be inserted: ");
    scanf("%d",&y);
    printf("\nTo left of which element should %d be inserted? Enter: ",y);
    scanf("%d",&x);
    temp=getnode();
    ele=header;
    temp->info=y;
    temp->p=NULL;
    temp->n=NULL;
    if(header->info==x)
    {
        temp->n=header;
        header->p=temp;
        header=temp;
    }
    else
    {
        while(ele!=NULL)
        {
            if(ele->info==x)
                break;
            ele=ele->n;
        }
        if(ele!=NULL)
        {
            temp->p=ele->p;
            (ele->p)->n=temp;
            temp->n=ele;
            ele->p=temp;
        }
        else
            printf("\nNo element found");

    }
}

// del function is used to delete the node
void del()
{
    node temp;
    int x;
    temp=header;
    if(header==NULL)
    {
        printf("\nNo element deleted.\n");
        return;
    }
    printf("Enter the node to be deleted: ");
    scanf("%d",&x);
    if(header->info==x)
    {
        temp=header;
        header=header->n;
        header->p=NULL;
        free(temp);
    }
    else
    {
        while(temp!=NULL)
        {
            if(temp->info==x)
                break;
            temp=temp->n;
        }
        if(temp!=NULL)
        {
            if(temp->n!=NULL)
            {
                (temp->n)->p=temp->p;
                (temp->p)->n=temp->n;
                free(temp);
            }
            else
            {
                (temp->p)->n=NULL;
                free(temp);
            }

        }
        else
        {
            printf("Element not found");
        }
    }
}

// disp fuction displays the content of the list.
void disp()
{
    node temp;
    if(header==NULL)
        printf("Absent");
    else
    {
        for(temp=header;temp!=NULL;temp=temp->n)
            printf("%d ",temp->info);
    }
}
```

OUTPUT:
compilation command:cc doublelinkedlist.c
output command: ./a.out

```
Choices:
1-Insert
2-Insert left
3-Delete node
4-Display
5-Exit
Enter your choice: 1

Enter the element to be inserted: 10

Enter your choice: 1

Enter the element to be inserted: 20

Enter your choice: 2

Enter the element to be inserted: 11

To left of which element should 11 be inserted? Enter: 10

Enter your choice: 2

Enter the element to be inserted: 21

To left of which element should 21 be inserted? Enter: 20

Enter your choice: 4
21 20 11 10

Enter your choice: 2

Enter the element to be inserted: 19

To left of which element should 19 be inserted? Enter: 11

Enter your choice: 4
21 20 19 11 10

Enter your choice: 3
Enter the node to be deleted: 19

Enter your choice: 3
Enter the node to be deleted: 21

Enter your choice: 3
Enter the node to be deleted: 10

Enter your choice: 4
20 11
```

PROGRAM 12 : date.cpp

Name of the Program:
Write a C++ program to create a class called DATE. Accept two valid dates of the
form dd/mm/yy. Implement the following operations by overloading the operators + and -.
After each operation ,display the result by overloading the << operator.
1. no_of_days=d1-d2, where d1 and d2 are date objects,
d1>=d2 and no_of_days is an integer
2. d2=d1+no_of_days, where d1 is a DATE object and
no_of_dys is an integer.

Theory:
In operator overloading, we can give special meanings to operators, when they are used with user-defined
classes. For example, to overload the + operator for your class, you would provide a member-function
named operator+ on your class.
The following set of operators is commonly overloaded for user-defined classes:

* = (assignment operator)
* .+ - * (binary arithmetic operators)
* += -= *= (compound assignment operators)
* == != (comparison operators)

Code:
date.cpp

```
#include<iostream>
using namespace std;
class date
{
    int dd,mm,yy;
    int a[13];
    long double dateno;
    void getno();
    public:
    date()
    {
        a[1]=a[3]=a[5]=a[7]=a[8]=a[10]=a[12]=31;
        a[4]=a[6]=a[9]=a[11]=30;
        a[2]=28;
        dd=mm=yy=1;
    }
    void getdate();
    friend ostream& operator<<(ostream&,date&);
    long double operator-(date&);
    date operator+(long double);
};
void date::getdate()
{
```

```
         cout<<"\n Enter date";
         cout<<"\n\t day(dd):";
         cin>>dd;
         cout<<"\n\t\ month(mm):";
         cin>>mm;
         cout<<"\n\t year(yy):";
         cin>>yy;
         while((yy%4!=0&&dd>a[mm])||(yy%4==0 && mm==2 && dd>29) ||dd<=0 || mm>12 ||
mm<=0 )
         {
                 cout<<"\n Invlid entry";
                 getdate();
         }
         getno();
    }
    void date::getno()
    {
         int m=1;
         dateno=(long double)yy*365+yy/4;
         if(yy%4>0)
         dateno++;
         while(m!=mm)
         {
                 dateno+=a[m];
                 if(yy%4==0 && m==2)
                 dateno++;
                 m++;
         }
         dateno+=dd;
    }
    ostream& operator<<(ostream &out,date &d1)
     {
          out<<d1.dd<<"/"<<d1.mm<<"/"<<d1.yy;
          return out;
     }
     long double date::operator-(date &b)
     {
         return(dateno-b.dateno);
     }

    date date::operator+(long double b)
    {
         for(int i=1;i<=b;i++)
         {
                 dd++;
                 if(dd>a[mm])
                 {
                         mm++;
                         dd=1;
                 }
                 if(mm>12)
                 {
```

```
                yy++;
                mm=1;
            }
            if(yy%4==0)
                a[2]=29;
        }
        return *this;
    }
    int main()
    {
        date d1,d2,d3;
        d1.getdate();
        cout<<"\n\td1="<<d1;
        d2.getdate();
        cout<<"\n\td2="<<d2;
        long double diff, days;
        diff=d1-d2;
        cout<<"\n\n The difference between the two dates = ";
        cout<<diff<<endl;
        cout<<"\n\n Enter the no. of days to be added to the date "<<d1<<" :";
        cin>>days;
        d3=d1+days;
        cout<<"\n New date is..."<<d3<<endl;
        return 0;
    }
```

Steps for obtaining output:
Steps for checking output-

* Locate the folder in which the file is present in the terminal.
* Execute the command "g++ the <filename.cpp>"
* Execute ethe command "./a.out"

Output:

```
 Enter date
        day(dd):29

        month(mm):02

        year(yy):2012

        d1=29/2/2012
 Enter date
        day(dd):31

        month(mm):01

        year(yy):2012

        d2=31/1/2012
```

```
The difference between the two dates = 29


Enter the no. of days to be added to the date 29/2/2012 :29

New date is...29/3/2012
```

PROGRAM 13 : octal.cpp

AIM:
Write a C++ program to create a class called OCTAL which has the characteristics of an octal number. Implement the following operations by writing an appropriate constructor and an overloaded operator +.(i) OCTAL h = x; where x is an integer.(ii) int y = h + k; where h is an OCTAL object and k is an integerDisplay the OCTAL result by overloading the operator << . Also display the values of h and y.

THEORY:
A constructor (sometimes shortened to ctor) in a class is a special type of subroutine called to create an object. It prepares the new object for use, often accepting arguments that the constructor uses to set member variables required.

ALGORITHM:

1. START
2. Accept an input say x.
3. Enter the value of x in decimal notation,find the remainder of x on dividing by 8
4. Corresponding value of x in octal notation is showed.
5. Enter the value of k in decimal notation
6. The value of h+k in decimal notation is shown.
7. The value of h+k in octal notation is shown.

CODE: octal.cpp
```
#include <iostream>
#include <math.h>
using namespace std;
class octal
{
int o;
public:
    octal(int);
    int dectooct(int x);
    int octtodec(int x);
    friend ostream &operator<<(ostream &print,octal);
    int operator +(int);
};

octal::octal(int x)
{
    o=dectooct(x);
}

int octal::dectooct(int x)
{
int i=0,sum=0,rem;
while(x!=0)
{
    rem=x%8;
    sum=sum+rem*pow(10,i);
```

```
        i++;
        x=x/8;
    }
    return sum;
    }
    int octal::octtodec(int x)
    {
    int i=0,sum=0,rem;
    while(x!=0)
    {
        rem=x%10;
        sum=sum+rem*pow(8,i);
        i++;
        x=x/10;
    }
    return sum;
    }
    ostream &operator<<(ostream &print,octal x)
    {
    print<<x.o;
    return print;
    }
    int octal::operator+(int x)
    {
        return octtodec(o)+x;
    }

    int main()
    {
    int x,y,k,yOct;
    cout<<endl<<"Enter the value of x in decimal notation:";
    cin>>x;
    octal h=x;
    cout<<"Corresponding value of x in octal notation: h = "<<h;
    cout<<endl<<"Enter the value of k in decimal notation:";
    cin>>k;
    cout<<"The value of k="<<k<<endl;
    y=h+k;
    cout<<"The value of h+k in decimal notation,y="<<y<<endl;

    yOct = h.dectooct(y);
    cout<<"The value of h+k in octal notation,y="<<yOct<<endl;
    return 0;
    }
```

OUTPUT:

```
Enter the value of x in decimal notation:100

Corresponding value of x in octal notation: h = 144

Enter the value of k in decimal notation:5
```

The value of k=5

The value of h+k in decimal notation,y=105

The value of h+k in octal notation,y=151

Program 14: bintree.cpp

AIM: Design, develop, and execute a program in C++ to create a class called BIN_TREE that represents a Binary Tree, with member functions to perform inorder, preorder and postorder traversals. Create a BIN_TREE object and demonstrate the traversals.*/

THEORY:
Each of the objects in a binary tree contains two pointers, typically called left and right. In addition to these pointers, of course, the nodes can contain other types of data.A node that points to another node is said to be the parent of that node, and the node it points to is called a child.

A binary tree must have the following properties: There is exactly one node in the tree which has no parent. This node is called the root of the tree. Every other node in the tree has exactly one parent. Finally, there can be no loops in a binary tree. That is, it is not possible to follow a chain of pointers starting at some node and arriving back at the same node.

A node that has no children is called a leaf. A leaf node can be recognized by the fact that both the left and right pointers in the node are NULL.

Pre-order = outputting the values of a binary tree in the order of the current node, then the left subtree, then the right subtree.

Post-order = outputting the values of a binary tree in the order of the left subtree, then the right subtree, the the current node.

Pre-order:

1. Visit the root.
2. Traverse the left subtree.
3. Traverse the right subtree.

In-order(symmetric):

1. Traverse the left subtree.
2. Visit the root.
3. Traverse the right subtree.

Post-order:

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root.

ALGORITHM:

Preorder: The first type of traversal is pre-order whose code looks like the following:

sub P(TreeNode)

```
  Output(TreeNode.value)
  If LeftPointer(TreeNode) != NULL Then
    P(TreeNode.LeftNode)
  If RightPointer(TreeNode) != NULL Then
    P(TreeNode.RightNode)
end sub
```

This can be summed up as

1. Visit the root node (generally output this)
2. Traverse to left subtree
3. Traverse to right subtree

Inorder: The second(middle) type of traversal is in-order whose code looks like the following:

```
sub P(TreeNode)
  If LeftPointer(TreeNode) != NULL Then
    P(TreeNode.LeftNode)
  Output(TreeNode.value)
  If RightPointer(TreeNode) != NULL Then
    P(TreeNode.RightNode)
end sub
```

This can be summed up as

Traverse to left subtree
Visit root node (generally output this)
Traverse to right subtree

Post-order: The last type of traversal is post-order whose code looks like the following:

```
sub P(TreeNode)
  If LeftPointer(TreeNode) != NULL Then
    P(TreeNode.LeftNode)
  If RightPointer(TreeNode) != NULL Then
    P(TreeNode.RightNode)
  Output(TreeNode.value)
end sub
```

This can be summed up as

Traverse to left subtree
Traverse to right subtree
Visit root node (generally output this)

CODE:bintree.cpp
```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

class NODE
{
   public:
   int info;
   class NODE *left;
   class NODE *right;
};

class bintree
{
  NODE *root;
   public:
         bintree()
         {
            root=NULL;
         }
         void Inorder(NODE* );
         void Preorder(NODE* );
         void Postorder(NODE* );
         NODE* Insert();
         void Display();
};

 /* To traverse a non-empty binary search tree in in-order (symmetric), perform the following
operations recursively at each node:
       1. Traverse the left sub-tree.
       2. Visit the root.
       3. Traverse the right sub-tree. */


void bintree::Inorder(NODE *p)
{
    if(p!=NULL)
    {
          Inorder(p->left);
          cout<<p->info<<" ";
          Inorder(p->right);
    }
}

/*To traverse a non-empty binary search tree in pre-order, perform the following operations
recursively at each node
     1. Visit the root.
 2. Traverse the left sub-tree.
 3. Traverse the right sub-tree. */

void bintree::Preorder(NODE *p)
{
    if(p!=NULL)
    {
        cout<<p->info<<" ";
        Preorder(p->left);
        Preorder(p->right);
    }
}

/*To traverse a non-empty binary search tree in post-order, perform the following operations
recursively at each node:
 1. Traverse the left sub-tree.
 2. Traverse the right sub-tree.
 3. Visit the root. */

void bintree::Postorder(NODE *p)
{
    if(p!=NULL)
    {
        Postorder(p->left);
        Postorder(p->right);
        cout<<p->info<<" ";
    }
}
NODE* bintree::Insert()
{

NODE *newnode;
    newnode=(NODE *)malloc(sizeof(NODE));
    newnode->left=newnode->right=NULL;
    cout<<"\n\n\tEnter an Element to Insert: ";
    cin>>newnode->info;
    if(root==NULL)
        root=newnode;
    else
    {
        NODE *p,*q;
        p=root;
        while(p!=NULL)
        {
            q=p;
            if(newnode->info>p->info)
                p=p->right;
            else
                p=p->left;
        }
        if(newnode->info>q->info)
            q->right=newnode;
```

```
              else
                     q->left=newnode;
        }
}
void bintree::Display()
{
 int x;
 if(root==NULL)
                   cout<<"\n\n\tNo Nodes in the Tree";
     else
         {
          while(1){
          cout<<"\n\tEnter your choice:\n\t1.Preorder\n\t2.Inorder\n\t3.Postorder\n\t4.Exit: ";
          cin>>x;
          switch(x)
          {
                 case 1:
                             cout<<"\n\n\tPreorder Traversal :";
                             Preorder(root);
                             break;
                 case 2:
                             cout<<"\n\n\tInorder Traversal :";
                             Inorder(root);
                             break;
                 case 3:
                             cout<<"\n\n\tPostorder Traversal :";
                             Postorder(root);
                             break;
                 case 4:
                             exit(0);
                 default:cout<<"Enter proper choice:";
          }
     }
     }
}

int main()
{
     int choice;
 bintree obj;

     while(1)
     {
     cout<<"\n\n\n\t1.Insert\n\t2.Display\n\t3.Exit";
             cout<<"\n\tEnter Your Choice: ";
             cin>>choice;
             switch(choice)
             {
                     case 1: obj.Insert(); break;
                     case 2: obj.Display(); break;
                     case 3: exit(0);
                     default: cout<<"\n\n\tEnter Proper Choice:";
```

```
                  }
         }
     return 0;
    }
```

OUTPUT:

```
compilation command: g++ bintree.cpp
output command : ./a.out
1.Insert
2.Display
3.Exit
Enter Your Choice: 1
Enter an Element to Insert: 40

Enter Your Choice: 1
Enter an Element to Insert: 30

Enter Your Choice: 1
Enter an Element to Insert: 15

Enter Your Choice: 1
Enter an Element to Insert: 45

Enter Your Choice: 1
Enter an Element to Insert: 60

Enter Your Choice: 1
Enter an Element to Insert: 52

Enter Your Choice: 1
Enter an Element to Insert: 39

Enter Your Choice: 1
Enter an Element to Insert: 10

Enter Your Choice: 1
Enter an Element to Insert: 2

1.Insert
2.Display
3.Exit
Enter Your Choice: 2

Enter your choice:
1.Preorder
2.Inoreder
3.Postorder
4.Exit: 2

Inorder Traversal :2 10 15 30 39 40 45 52 60
```

```
Enter your choice: 1

Preorder Traversal :40 30 15 10 2 39 45 60 52
Enter your choice: 3

Postorder Traversal :2 10 15 39 30 52 60 45 40
Enter your choice: 4
```