Program 1: Write a menu driven program to implement the following sparse matrices using one-dimensional array:

a) Diagonal Matrix

b) Lower Triangular Matrix

c) Upper Triangular Matrix

d) Symmetric Matrix

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 2: WAP to compute br using recursion where b represents base and r represents power.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 3: WAP to reverse a user entered string using recursion.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 4: Perform the following Stack operations using Linked List implementation.

a) Push

b) Pop

c) Clear

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 5: Perform the following Stack operations using Array implementation using Templates:

a) Push

b) Pop

c) Clear

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 6: Perform the following Queue operations using Circular Array implementation (Use Templates):

a) Enqueue

b) Dequeue

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 7: Create and perform the following operations on Queues using Linked List implementation:

a) Enqueue

b) Dequeue

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 8: WAP to add two large integers using stack.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 9: WAP to evaluate postfix expression using stack.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 10: Implementation Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list and concatenate two linked lists (include a function and also overload operator +).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 11: Implement Doubly Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 12: Implement Circular Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 13: WAP to add two polynomials using linked list representation.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 14: Write a menu driven program to implement the following operations in an ordered linked list:

a) Insertion

b) Deletion

c) Merging

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 15: Write a menu driven program to implement the following operations in a Binary Search Tree:

a) Insertion

b) Deletion by copying

c) Deletion by Merging

d) Search a no. Merging

e) Search a no. in BST

f) Display its preorder, postorder and inorder traversals

g) Display its level-by-level traversals

h) Count the non-leaf nodes and leaf nodes

i) Display height of tree

j) Create a mirror image of tree

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 16: Write a menu driven program to implement the following sorting and searching algorithms:

a) Bubble Sort

b) Insertion

c) Selection

d) Merge Sort

e) Quick Sort

f) Linear Search

g) Binary Search

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 1: Write a menu driven program to implement the following sparse matrices using one-dimensional array:

a) Diagonal Matrix

b) Lower Triangular Matrix

c) Upper Triangular Matrix

d) Symmetric Matrix

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<iomanip>

using namespace std;

class dia

{

int \*A;

int size;

public:

dia(int x)

{

A=new int[x];

size=x;

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

A[i]=0;

}

void set(int i, int j, int element)

{

if(i<0 || j<0 || i>size || j>size)

{

cout<<"\nOut of bounds!";

}

else

{

if( i==j)

A[i-1]=element;

}

}

void chk(int i, int j)

{

if(i<0 || j<0 || i>size || j>size)

{

cout<<"\nOut of bounds!";

}

else

{

if( i==j)

cout<<"\nValue= "<<A[i-1];

else

cout<<"Value= 0";

}

}

void disp()

{

int i=0, j=0;

while(i!=size)

{

j=0;

while(j!=size)

{

if(i==j)

cout<<A[i]<<" ";

else

cout<<"0 ";

j++;

}

cout<<endl;

i++;

}

}

};

class tri\_dia

{

int \*A;

int size, mat\_size;

public:

tri\_dia(int x)

{

A=new int[ (3\*x)-2];

size=(3\*x)-2;

mat\_size=x;

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

A[i]=0;

}

void set(int i, int j, int element)

{

if(i<0 || j<0 || i>mat\_size || j>mat\_size)

{

cout<<"\nOut of bounds!";

}

else

{

if(i==j+1)

A[j-1]=element;

else if( i==j)

A[ (mat\_size-1)+(i-1)]=element;

else if(i==j-1)

A[ ((2\*mat\_size)-1)+(i-1)]=element;

}

}

void chk(int i, int j)

{

if(i<0 || j<0 || i>mat\_size || j>mat\_size)

{

cout<<"\nOut of bounds!";

}

else

{

if(i==j+1)

cout<<"\nValue= "<<A[j-1];

else if( i==j)

cout<<"\nValue= "<<A[mat\_size-1+i-1];

else if(i==j-1)

cout<<"\nValue= "<<A[2\*mat\_size-1+i-1];

else

cout<<"\nValue= 0";

}

}

void disp()

{

int i=1, j=1;

while(i<=mat\_size)

{

j=1;

while(j<=mat\_size)

{

if(i==j+1)

cout<<A[j-1]<<" ";

else if( i==j)

cout<<A[ (mat\_size-1)+(i-1) ]<<" ";

else if(i==j-1)

cout<<A[ (2\*mat\_size-1)+(i-1) ]<<" ";

else

cout<<"0 ";

j++;

}

cout<<endl;

i++;

}

}

};

class low\_tri

{

int \*A;

int size, mat\_size;

public:

low\_tri(int x)

{

A=new int[ (x\*(x+1))/2];

size=(x\*(x+1))/2;

mat\_size=x;

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

A[i]=0;

}

void set(int i, int j, int element)

{

if(i<0 || j<0 || i>mat\_size || j>mat\_size)

{

cout<<"\nOut of bounds!";

}

else

{

if(i>=j)

A[ (i\*(i-1))/2+ (j-1)]=element;

}

}

void chk(int i, int j)

{

if(i<0 || j<0 || i>mat\_size || j>mat\_size)

{

cout<<"\nOut of bounds!";

}

else

{

if(i>=j)

cout<<"\nValue= "<<A[ (i\*(i-1))/2+ (j-1)];

else

cout<<"\nValue= 0";

}

}

void disp()

{

int i=1, j=1;

while(i<=mat\_size)

{

j=1;

while(j<=mat\_size)

{

if(i>=j)

cout<<A[ (i\*(i-1))/2+ (j-1)]<<" ";

else

cout<<"0 ";

j++;

}

cout<<endl;

i++;

}

}

};

class up\_tri

{

int \*A;

int size, mat\_size;

public:

up\_tri(int x)

{

A=new int[ (x\*(x+1))/2];

size=(x\*(x+1))/2;

mat\_size=x;

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

A[i]=0;

}

void set(int j, int i, int element)

{

if(i<0 || j<0 || i>mat\_size || j>mat\_size)

{

cout<<"\nOut of bounds!";

}

else

{

if(i>=j)

A[ (i\*(i-1))/2+ (j-1)]=element;

}

}

void chk(int j, int i)

{

if(i<0 || j<0 || i>mat\_size || j>mat\_size)

{

cout<<"\nOut of bounds!";

}

else

{

if(i>=j)

cout<<"\nValue= "<<A[ (i\*(i-1))/2+ (j-1)];

else

cout<<"\nValue= 0";

}

}

void disp()

{

int i=1, j=1;

while(j<=mat\_size)

{

i=1;

while(i<=mat\_size)

{

if(i>=j)

cout<<A[ (i\*(i-1))/2+ (j-1)]<<" ";

else

cout<<"0 ";

i++;

}

cout<<endl;

j++;

}

}

};

class sym

{

int \*A;

int size, mat\_size;

public:

sym(int x)

{

A=new int[ (x\*(x+1))/2];

size=(x\*(x+1))/2;

mat\_size=x;

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

A[i]=0;

}

void set(int i, int j, int element)

{

if(i<0 || j<0 || i>mat\_size || j>mat\_size)

{

cout<<"\nOut of bounds!";

}

else

{

if(i>=j)

A[ (i\*(i-1))/2+ (j-1)]=element;

}

}

void chk(int i, int j)

{

if(i<0 || j<0 || i>mat\_size || j>mat\_size)

{

cout<<"\nOut of bounds!";

}

else

{

if(i>=j)

cout<<"\nValue= "<<A[ (i\*(i-1))/2+ (j-1)];

else if(i<j)

{

swap(i,j);

cout<<"\nValue= "<<A[ (i\*(i-1))/2+ (j-1)];

}

else

cout<<"\nValue= 0";

}

}

void disp()

{

int i=1, j=1;

while(i<=mat\_size)

{

j=1;

while(j<=mat\_size)

{

if(i>=j)

cout<<setw(2)<<A[ (i\*(i-1))/2+ (j-1)]<<" ";

else if(i<j)

{

swap(i,j);

cout<<setw(2)<<A[ (i\*(i-1))/2+ (j-1)]<<" ";

swap(i,j);

}

else

cout<<setw(2)<<"0";

j++;

}

cout<<endl;

i++;

}

}

};

int main()

{

int num,size,num2,i,j,element;

char ch;

cout<<"\nEnter matrix size: ";

cin>>size;

dia ob(size);

tri\_dia ob1(size);

low\_tri ob2(size);

up\_tri ob3(size);

sym ob4(size);

do

{

system("cls");

cout<<"\n\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout<<"\n1. Diagonal matrix.";

cout<<"\n2. Tri-Diagonal matrix.";

cout<<"\n3. Lower-Diagonal matrix.";

cout<<"\n4. Upper-Diagonal matrix.";

cout<<"\n5. Symmetric matrix.";

cout<<"\n6. Exit.";

cout<<"\nEnter your choice: ";

cin>>num;

switch(num)

{

case 1: do

{

system("cls");

cout<<"\n1. Insert element.";

cout<<"\n2. check any element.";

cout<<"\n3. display matrix.";

cout<<"\nEnter your choice: ";

cin>>num2;

switch(num2)

{

case 1: cout<<"\n\tEnter row(i): ";

cin>>i;

cout<<"\n\tEnter column(j): ";

cin>>j;

cout<<"\n\tEnter element value: ";

cin>>element;

ob.set(i,j,element);

break;

case 2: cout<<"\n\tEnter row(i): ";

cin>>i;

cout<<"\n\tEnter column(j): ";

cin>>j;

ob.chk(i,j);

break;

case 3: ob.disp();

break;

default:cout<<"\nWrong input";

break;

}

cout<<"\n\n\tEnter again(y/n): ";

cin>>ch;

}while(ch=='y');

break;

case 2: do

{

system("cls");

cout<<"\n1. Insert element.";

cout<<"\n2. check any element.";

cout<<"\n3. display matrix.";

cout<<"\nEnter your choice: ";

cin>>num2;

switch(num2)

{

case 1: cout<<"\n\tEnter row(i): ";

cin>>i;

cout<<"\n\tEnter column(j): ";

cin>>j;

cout<<"\n\tEnter element value: ";

cin>>element;

ob1.set(i,j,element);

break;

case 2: cout<<"\n\tEnter row(i): ";

cin>>i;

cout<<"\n\tEnter column(j): ";

cin>>j;

ob1.chk(i,j);

break;

case 3: ob1.disp();

break;

default:cout<<"\n\tWrong input";

break;

}

cout<<"\n\n\tEnter again(y/n): ";

cin>>ch;

}while(ch=='y');

break;

case 3: do

{

system("cls");

cout<<"\n1. Insert element.";

cout<<"\n2. check any element.";

cout<<"\n3. display matrix.";

cout<<"\nEnter your choice: ";

cin>>num2;

switch(num2)

{

case 1: cout<<"\n\tEnter row(i): ";

cin>>i;

cout<<"\n\tEnter column(j): ";

cin>>j;

cout<<"\n\tEnter element value: ";

cin>>element;

ob2.set(i,j,element);

break;

case 2: cout<<"\n\tEnter row(i): ";

cin>>i;

cout<<"\n\tEnter column(j): ";

cin>>j;

ob2.chk(i,j);

break;

case 3: ob2.disp();

break;

default:cout<<"\n\tWrong input";

break;

}

cout<<"\n\n\tEnter again(y/n): ";

cin>>ch;

}while(ch=='y');

break;

case 4: do

{

system("cls");

cout<<"\n1. Insert element.";

cout<<"\n2. check any element.";

cout<<"\n3. display matrix.";

cout<<"\nEnter your choice: ";

cin>>num2;

switch(num2)

{

case 1: cout<<"\n\tEnter row(i): ";

cin>>i;

cout<<"\n\tEnter column(j): ";

cin>>j;

cout<<"\n\tEnter element value: ";

cin>>element;

ob3.set(i,j,element);

break;

case 2: cout<<"\n\tEnter row(i): ";

cin>>i;

cout<<"\n\tEnter column(j): ";

cin>>j;

ob3.chk(i,j);

break;

case 3: ob3.disp();

break;

default:cout<<"\tWrong input";

break;

}

cout<<"\n\tEnter again(y/n): ";

cin>>ch;

}while(ch=='y');

break;

case 5: do

{

system("cls");

cout<<"\n1. Insert element.";

cout<<"\n2. check any element.";

cout<<"\n3. display matrix.";

cout<<"\nEnter your choice: ";

cin>>num2;

switch(num2)

{

case 1: cout<<"\nEnter only lower triangular matrix elements:-\n";

cout<<"\n\tEnter row(i): ";

cin>>i;

cout<<"\n\tEnter column(j): ";

cin>>j;

cout<<"\n\tEnter element value: ";

cin>>element;

ob4.set(i,j,element);

break;

case 2: cout<<"\n\tEnter row(i): ";

cin>>i;

cout<<"\n\tEnter column(j): ";

cin>>j;

ob4.chk(i,j);

break;

case 3: ob4.disp();

break;

default:cout<<"\n\tWrong input";

break;

}

cout<<"\n\n\tEnter again(y/n): ";

cin>>ch;

}while(ch=='y');

break;

case 6: cout<<"\nEXITING...";

exit(100);

break;

default:cout<<"\nWrong Input!!!";

break;

}

cout<<"\n\nChoose different matrix(y/n): ";

cin>>ch;

}while(ch=='y'|| ch=='Y');

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 2: WAP to compute br using recursion where b represents base and r represents power.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

using namespace std;

int calc(int b, int r)

{

if(r==1)

return b;

else

return b\*calc(b,r-1);

}

int main()

{

int b,r;

int tvalue;

cout<<"\nEnter any number to calculate it's power: ";

cin>>b;

cout<<"\nEnter power value: ";

cin>>r;

tvalue=calc(b,r);

cout<<b<<"^"<<r<<" = "<<tvalue;

return 0;

}

Program 3: WAP to reverse a user entered string using recursion.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<conio.h>

using namespace std;

void rev\_string(char A[], int i)

{

if(A[i]!='\0')

{

rev\_string(A,i+1);

cout<<A[i];

}

}

void rev(char A[], int i, int size)

{

if(i<=(size/2))

{

swap(A[i],A[size-i]);

rev(A,i+1,size);

}

}

int main()

{

int size=0;

char A[100];

cout<<"\nEnter any string: ";

cin.getline(A,100);

while(A[size]!='\0'){

size++;

}

cout<<"\nReverse string: ";

rev(A,0,size-1);

cout<<A;

getch();

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 4: Perform the following Stack operations using Linked List implementation.

a) Push

b) Pop

c) Clear

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<stdlib.h>

#include<process.h>

using namespace std;

struct node

{

int data;

node\*next;

}\*top,\*ptr,\*save,\*rear,\*ptr1;

node\*create(int n)

{

ptr1=new node;

ptr1->data=n;

ptr1->next=NULL;

return ptr1;

}

void push(node\* np)

{

node\* t;

if(top==NULL){

top=np;

}

else

{

t=top;

np->next=top;

top=np;

}

}

void pop()

{

node\*t;

if(top==NULL)

{

cout<<"underflow ";

}

else

{

t=top;

top=top->next;

}

delete t;

}

void display(node \* np)

{

while(np!=NULL)

{

cout<<np->data<<endl;

np=np->next;

}

}

int main()

{

int choice;

char chii;

int ch;

char choo;

int inf;

top=NULL;

char ch1;

do{

cout<<"Enter your choice"<<endl;

cout<<"1.Push"<<endl<<"2.Pop"<<endl<<"3.Display"<<endl<<"4.Exit"<<endl;

cin>>ch;

switch(ch)

{

case 1: cout<<"Enter the inf of new node ";

cin>>inf;

ptr=create(inf);

if(ptr!=NULL)

{

cout<<endl<<"node created successfully ";

}

else

cout<<"Error creating node "<<endl;

push(ptr);

break;

case 2: pop();

break;

case 3: cout<<"The link list is "<<endl;

display(top);

break;

case 4:exit(0);

default: cout<<"wrong choice";

break;

}

cout<<endl<<"do you want to continue ";

cin>>ch1;

}while((ch1=='y')||(ch1=='Y'));

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 5: Perform the following Stack operations using Array implementation using Templates:

a) Push

b) Pop

c) Clear

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

int t;

template<class T>

class Mystack

{

T \*st;

int tos;

int size;

public:

Mystack()

{

tos=-1;

st=new T[10];

}

Mystack(int n)

{

tos=-1;

t=n;

size=n;

st=new T[n];

}

void set(int i)

{

t=i;

size=i;

st=new T[i];

}

void push(T item);

T pop();

void show();

};

template<class T>

void Mystack<T>::push(T item)

{

if(tos>size-1)

{

cout<<"stack overflow"<<endl;

return;

}

else

{

st[++tos]=item;

}

}

template<class T>

T Mystack<T>::pop()

{

if(tos<0)

{

cout<<"stack underflow "<<endl;

::t=-1;

}

else

{

return st[tos--];

}

}

template <class T>

void Mystack<T>::show()

{

if(tos<0)

{

return;

}

else

{

for(int i=tos;i>=0;i--)

cout<<st[i]<<endl;

}

}

int main()

{

float f;

int i;

char c;

char ch1;

int n,ch,ch2,item;

float item1;

char item2;

cout<<"Enter the type of array "<<endl<<"1.int"<<endl<<"2.float"<<endl<<"3.character"<<endl;

cin>>ch2;

cout<<"enter the size of stack"<<endl;

cin>>n;

Mystack<int> ob1(n);

Mystack<float> ob2(n);

Mystack<char> ob3(n);

switch(ch2)

{

case 1: for(int i=0;i<n;i++)

{

cout<<"Enter element to pushed"<<endl;

cin>>item;

ob1.push(item);

}

cout<<"The stack is "<<endl;

ob1.show();

break;

case 2: for(int i=0;i<n;i++)

{

cout<<"Enter element to pushed"<<endl;

cin>>item1;

ob2.push(item1);

}

cout<<"The stack is "<<endl;

ob2.show();

break;

case 3: for(int i=0;i<n;i++)

{

cout<<"Enter element to pushed"<<endl;

cin>>item2;

ob3.push(item2);

}

cout<<"The stack is "<<endl;

ob3.show();

break;

default: cout<<"Enter genuine choice";

}

do{

cout<<"do you want to delete any element"<<"1.yes"<<"2.NO"<<endl;

cin>>ch;

switch(ch)

{

case 1: if(ch2==1)

{

ob1.pop();

cout<<"The new stack is "<<endl;

ob1.show();

}

else if(ch2==2)

{

ob2.pop();

cout<<"The new stack is "<<endl;

ob2.show();

}

else if(ch2==3)

{

ob3.pop();

cout<<"the new stack is "<<endl;

ob3.show();

}

break;

case 2: cout<<"dont want to enter : exit";

exit(0);

break;

default: cout<<"wrong choice"<<endl;

}

if(::t<0)

break;

else

{

cout<<"do you want to continue"<<endl;

cin>>ch1;

}

}while(ch1=='Y'||ch1=='y');

getch();

return 0;

}

Program 6: Perform the following Queue operations using Circular Array implementation (Use Templates):

a) Enqueue

b) Dequeue

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <iostream>

#include <conio.h>

using namespace std;

class Cqueue

{

int \* carray;

int size;

int front;

int rear;

public:

Cqueue(int n)

{

size = n;

carray = new int [size];

front = -1;

rear = -1;

}

bool isempty()

{

if (front == -1 && rear == -1)

return true;

else return false;

}

void enqueue(int value)

{

if((rear+1)% size == front)

cout<<"queue is full"<<endl;

else

if(front == -1)

front = 0;

rear = (rear+1)%size;

carray[rear] = value;

}

void dequeue()

{

if (isempty())

cout<<"queue is empty"<<endl;

else if (front == rear)

front = rear = -1;

else

front = (front+1)% size;

}

void showfront()

{

if(isempty())

cout<<"queue is empty"<<endl;

else

cout<<carray[front]<<endl;

}

void display()

{

cout<<endl;

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

cout<<carray[i]<< endl;

}

};

int main()

{

int size1 , element;

char ch1;

int choice, item;

cout<<"enter size of queue" <<endl;

cin>>size1;

Cqueue a (size1);

do{

cout<<"Enter your choice"<<endl;

cout<<"1.Enqueue"<<endl<<"2.dequeue"<<endl<<"3.display"<<endl;

cin>>choice;

switch(choice)

{

case 1: cout<<"Enter the element to be enqueued"<<endl;

cin>>item;

a.enqueue(item);

break;

case 2: a.dequeue();

break;

case 3: a.display();

break;

default: cout<<"Wrong choice";

}

cout<<"do you want to continue"<<endl;

cin>>ch1;

}while(ch1=='y'||ch1=='Y');

getch();

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 7: Create and perform the following operations on Queues using Linked List implementation:

a) Enqueue

b) Dequeue

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <iostream>

using namespace std;

template <class T>

class node

{

public:

T info;

node \*next;

node(T a,node \*p=0)

{

info=a;

next=p;

}

~node()

{

delete next;

}

};

template <class T>

class queue

{

node <T> \*f;

node <T> \*r;

public:

queue()

{

f=r=0;

}

~queue()

{

delete f;

delete r;

}

void Enqueue(T ele);

T DeQueue();

void display();

};

template <class T> void queue<T>::Enqueue(T ele)

{

if(f==0 && r==0)

{

r=new node<T>(ele);

f=r;

}

else

{

r->next=new node<T>(ele);

r=r->next;

}

}

template <class T> T queue<T>::DeQueue()

{

if(f==0 && r==0)

{

cout<<"\nEmpty Stack\n";

return -99999;

}

else

{

node <T> \*tmp=f;

f=f->next;

T d=tmp->info;

delete tmp;

return d;

}

}

template <class T> void queue<T>::display()

{

if(f==0 && r==0)

{

cout<<"\nEmpty Stack\n";

}

else

{

for(node<T> \*p=f;p!=0;p=p->next)

cout<<p->info<<"\n";

}

}

int main()

{

int ch,n;

queue <int> q;

do{

cout << "\nQueue Using Linked List\n\n 1.Enqueue\n 2.Dequeue\n 3.Display\n 4.Exit\nEnter Your Choice=";

cin>>ch;

switch(ch)

{

case 1: cout<<"\nEnter a Number to be Enqueued=";

cin>>n;

q.Enqueue(n);break;

case 2: n=q.DeQueue();

if(n!=-99999)

cout<<"\nElement Removed="<<n;break;

case 3: cout<<"\nElements in the Queue are:-\n";

q.display();break;

case 4: break;

default: cout<<"\nInvalid Input.\n";

}

}while(ch!=4);

return 0;

}

Program 8: WAP to add two large integers using stack.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

using namespace std;

template<class t>

class stacks1

{

int n;

int top;

t \*nodes;

public:

stacks1(int t1)

{

n=t1;

top=-1;

nodes=new t[n];

}

void push(int j)

{

if(top<n)

{

top =top+1;

nodes[top]=j;

}

else

{

cout<<" stack is full overflow! "<<endl;

}

}

t pop()

{

if(top<0)

{

return 0;

}

else

return(nodes[top--]);

}

~stacks1()

{

delete[]nodes;

}

int isempty()

{

if(top==-1)

{

return 1;

}

else return 0;

}

void display()

{

for(int i=top;i>=0;--i)

cout<<""<<nodes[i];

}

};

int main()

{

int i2=1;

do{

int c1,c2;

cout<<"\n\t\t\tSTACK OPERATION \n\n";

cout<<" Enter two large No to be added\n"<<endl;

string str1,str2;

cout<<" Enter the No 1 \n";

cin>>str1;

cout<< "\nEnter the No 2\n";

cin>>str2;

c1=str1.size();

c2=str2.size();

stacks1<int> stk1(c1);

stacks1<int> stk2(c2);

stacks1<int> stk3(c1+c2);

int b=0;

while(b!=c1)

{

stk1.push(str1[b]-'0') ;++b;

}

b=0;

while(b!=c2)

{

stk2.push(str2[b]-'0') ;

b++;

}

int k=c1, l=c2, m, c=0;

if(c1>=c2)

{

while(k>=0)

{

m=c+stk1.pop()+stk2.pop();

c=0;

if(m>=10)

{

c=1;

m=m%10;

}

stk3.push(m);

--k;

}

}

c=0;

if(c2>c1)

{

while(l>=0)

{

m=c+stk1.pop()+stk2.pop();

c=0;

if(m>=10)

{

c=1;

m=m%10;

}

stk3.push(m);

--l;

}

}

cout<<"\nBoth no added \n\n";

cout<<"\nThe result is \n\n"<<endl;

cout<<"\t";

stk3.display();

cout<<endl;

cout<<"\t\n Do you want to add again : Press Y \n";

char ch2;

cin>>ch2;

if(ch2=='y' || ch2=='Y')

continue;

else

break;

}while(i2);

return 0;

}

Program 9: WAP to evaluate postfix expression using stack.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <iostream>

#include <stdlib.h>

#include <math.h>

#include <ctype.h>

#include <conio.h>

using namespace std;

const int MAX = 50 ;

class postfix

{

private :

int stack[MAX] ;

int top, nn ;

char \*s ;

public :

postfix( ) ;

void setexpr ( char \*str ) ;

void push ( int item ) ;

int pop( ) ;

void calculate( ) ;

void show( ) ;

};

postfix :: postfix()

{

top = -1 ;

}

void postfix :: setexpr ( char \*str )

{

s = str ;

}

void postfix :: push ( int item )

{

if ( top == (MAX - 1 ))

cout<<"Stack is full" ;

else

{

top++ ;

stack[top] = item ;

}

}

int postfix :: pop( )

{

if ( top == -1 )

{

cout << endl << "Stack is empty" ;

return NULL ;

}

int data = stack[top] ;

top-- ;

return data ;

}

void postfix :: calculate( )

{

int n1, n2, n3 ;

while ( \*s )

{

if ( \*s == ' ' || \*s == '\t' )

{

s++ ;

continue ;

}

if ( isdigit ( \*s ) )

{

nn = \*s - '0' ;

push ( nn ) ;

}

else

{

n1 = pop( ) ;

n2 = pop( ) ;

switch ( \*s )

{

case '+' :

n3 = n2 + n1 ;

break ;

case '-' :

n3 = n2 - n1 ;

break ;

case '/' :

n3 = n2 / n1 ;

break ;

case '\*' :

n3 = n2 \* n1 ;

break ;

case '%' :

n3 = n2 % n1 ;

break ;

case '$' :

n3 = pow ( n2 , n1 ) ;

break ;

default :

cout << "Unknown operator" ;

exit ( 1 ) ;

}

push ( n3 ) ;

}

s++ ;

}

}

void postfix :: show( )

{

nn = pop ( ) ;

cout << "Result is: " << nn ;

}

int main()

{

char expr[MAX] ;

cout << "\nEnter postfix expression to be evaluated : " ;

cin.getline ( expr, MAX ) ;

postfix q ;

q.setexpr ( expr ) ;

q.calculate( ) ;

q.show( ) ;

getch();

}

Program 10: Implementation Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list and concatenate two linked lists (include a function and also overload operator +).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;

struct node

{

int info;

struct node \*next;

}\*start;

class single\_llist

{

public:

node\* create\_node(int);

void insert\_begin();

void insert\_pos();

void insert\_last();

void delete\_pos();

void display();

void search(int e);

single\_llist()

{

start = NULL;

}

};

node \*single\_llist::create\_node(int value)

{

struct node \*temp, \*s;

temp = new(struct node);

if (temp == NULL)

{

cout<<"Memory not allocated "<<endl;

return 0;

}

else

{

temp->info = value;

temp->next = NULL;

return temp;

}

}

void single\_llist::insert\_begin()

{

int value;

cout<<"Enter the value to be inserted: ";

cin>>value;

struct node \*temp, \*p;

temp = create\_node(value);

if (start == NULL)

{

start = temp;

start->next = NULL;

}

else

{

p = start;

start = temp;

start->next = p;

}

cout<<"Element Inserted at beginning"<<endl;

}

void single\_llist::insert\_last()

{

int value;

cout<<"Enter the value to be inserted: ";

cin>>value;

struct node \*temp, \*s;

temp = create\_node(value);

s = start;

while (s->next != NULL)

{

s = s->next;

}

temp->next = NULL;

s->next = temp;

cout<<"Element Inserted at last"<<endl;

}

void single\_llist::insert\_pos()

{

int value, pos, counter = 0;

cout<<"Enter the value to be inserted: ";

cin>>value;

struct node \*temp, \*s, \*ptr;

temp = create\_node(value);

cout<<"Enter the postion at which node to be inserted: ";

cin>>pos;

int i;

s = start;

while (s != NULL)

{

s = s->next;

counter++;

}

if (pos == 1)

{

if (start == NULL)

{

start = temp;

start->next = NULL;

}

else

{

ptr = start;

start = temp;

start->next = ptr;

}

}

else if (pos > 1 && pos <= counter)

{

s = start;

for (i = 1; i < pos; i++)

{

ptr = s;

s = s->next;

}

ptr->next = temp;

temp->next = s;

}

else

{

cout<<"Positon out of range"<<endl;

}

}

void single\_llist::delete\_pos()

{

int pos, i, counter = 0;

if (start == NULL)

{

cout<<"List is empty"<<endl;

return;

}

cout<<"Enter the position of value to be deleted: ";

cin>>pos;

struct node \*s, \*ptr;

s = start;

if (pos == 1)

{

start = s->next;

}

else

{

while (s != NULL)

{

s = s->next;

counter++;

}

if (pos > 0 && pos <= counter)

{

s = start;

for (i = 1;i < pos;i++)

{

ptr = s;

s = s->next;

}

ptr->next = s->next;

}

else

{

cout<<"Position out of range"<<endl;

}

free(s);

cout<<"Element Deleted"<<endl;

}

}

void single\_llist::display()

{

struct node \*temp;

if (start == NULL)

{

cout<<"The List is Empty"<<endl;

return;

}

temp = start;

cout<<"Elements of list are: "<<endl;

while (temp != NULL)

{

cout<<temp->info<<"->";

temp = temp->next;

}

cout<<"NULL"<<endl;

}

int main()

{

int choice,e,nodes, element, position, i;

single\_llist sl;

start = NULL;

while (1)

{

cout<<"Enter ur choice"<<endl;

cout<<"1.Insert Node at beginning"<<endl;

cout<<"2.Insert node at last"<<endl;

cout<<"3.Insert node at position"<<endl;

cout<<"4.Delete a Particular Node"<<endl;

cout<<"5.Search for a number"<<endl;

cout<<"6.Reverse the list"<<endl;

cout<<"7.Concatenate 2 linked lists"<<endl;

cout<<"8.Display "<<endl;

cout<<"9.Exit "<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"Inserting Node at Beginning: "<<endl;

sl.insert\_begin();

cout<<endl;

break;

case 2:

cout<<"Inserting Node at Last: "<<endl;

sl.insert\_last();

cout<<endl;

break;

case 3:

cout<<"Inserting Node at a given position:"<<endl;

sl.insert\_pos();

cout<<endl;

break;

case 4:

cout<<"Delete a particular node: "<<endl;

sl.delete\_pos();

break;

case 5:cout<<"enter the element to be search"<<endl;

cin>>e;

case 8:

cout<<"Display elements of link list"<<endl;

sl.display();

cout<<endl;

break;

case 9:

cout<<"Exiting..."<<endl;

exit(1);

break;

default:

cout<<"Wrong choice"<<endl;

}

}

}

rogram 10: Implementation Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list and concatenate two linked lists (include a function and also overload operator +).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;

struct node

{

int info;

struct node \*next;

}\*start;

class single\_llist

{

public:

node\* create\_node(int);

void insert\_begin();

void insert\_pos();

void insert\_last();

void delete\_pos();

void display();

void search(int e);

single\_llist()

{

start = NULL;

}

};

node \*single\_llist::create\_node(int value)

{

struct node \*temp, \*s;

temp = new(struct node);

if (temp == NULL)

{

cout<<"Memory not allocated "<<endl;

return 0;

}

else

{

temp->info = value;

temp->next = NULL;

return temp;

}

}

void single\_llist::insert\_begin()

{

int value;

cout<<"Enter the value to be inserted: ";

cin>>value;

struct node \*temp, \*p;

temp = create\_node(value);

if (start == NULL)

{

start = temp;

start->next = NULL;

}

else

{

p = start;

start = temp;

start->next = p;

}

cout<<"Element Inserted at beginning"<<endl;

}

void single\_llist::insert\_last()

{

int value;

cout<<"Enter the value to be inserted: ";

cin>>value;

struct node \*temp, \*s;

temp = create\_node(value);

s = start;

while (s->next != NULL)

{

s = s->next;

}

temp->next = NULL;

s->next = temp;

cout<<"Element Inserted at last"<<endl;

}

void single\_llist::insert\_pos()

{

int value, pos, counter = 0;

cout<<"Enter the value to be inserted: ";

cin>>value;

struct node \*temp, \*s, \*ptr;

temp = create\_node(value);

cout<<"Enter the postion at which node to be inserted: ";

cin>>pos;

int i;

s = start;

while (s != NULL)

{

s = s->next;

counter++;

}

if (pos == 1)

{

if (start == NULL)

{

start = temp;

start->next = NULL;

}

else

{

ptr = start;

start = temp;

start->next = ptr;

}

}

else if (pos > 1 && pos <= counter)

{

s = start;

for (i = 1; i < pos; i++)

{

ptr = s;

s = s->next;

}

ptr->next = temp;

temp->next = s;

}

else

{

cout<<"Positon out of range"<<endl;

}

}

void single\_llist::delete\_pos()

{

int pos, i, counter = 0;

if (start == NULL)

{

cout<<"List is empty"<<endl;

return;

}

cout<<"Enter the position of value to be deleted: ";

cin>>pos;

struct node \*s, \*ptr;

s = start;

if (pos == 1)

{

start = s->next;

}

else

{

while (s != NULL)

{

s = s->next;

counter++;

}

if (pos > 0 && pos <= counter)

{

s = start;

for (i = 1;i < pos;i++)

{

ptr = s;

s = s->next;

}

ptr->next = s->next;

}

else

{

cout<<"Position out of range"<<endl;

}

free(s);

cout<<"Element Deleted"<<endl;

}

}

void single\_llist::display()

{

struct node \*temp;

if (start == NULL)

{

cout<<"The List is Empty"<<endl;

return;

}

temp = start;

cout<<"Elements of list are: "<<endl;

while (temp != NULL)

{

cout<<temp->info<<"->";

temp = temp->next;

}

cout<<"NULL"<<endl;

}

int main()

{

int choice,e,nodes, element, position, i;

single\_llist sl;

start = NULL;

while (1)

{

cout<<"Enter ur choice"<<endl;

cout<<"1.Insert Node at beginning"<<endl;

cout<<"2.Insert node at last"<<endl;

cout<<"3.Insert node at position"<<endl;

cout<<"4.Delete a Particular Node"<<endl;

cout<<"5.Search for a number"<<endl;

cout<<"6.Reverse the list"<<endl;

cout<<"7.Concatenate 2 linked lists"<<endl;

cout<<"8.Display "<<endl;

cout<<"9.Exit "<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"Inserting Node at Beginning: "<<endl;

sl.insert\_begin();

cout<<endl;

break;

case 2:

cout<<"Inserting Node at Last: "<<endl;

sl.insert\_last();

cout<<endl;

break;

case 3:

cout<<"Inserting Node at a given position:"<<endl;

sl.insert\_pos();

cout<<endl;

break;

case 4:

cout<<"Delete a particular node: "<<endl;

sl.delete\_pos();

break;

case 5:cout<<"enter the element to be search"<<endl;

cin>>e;

case 8:

cout<<"Display elements of link list"<<endl;

sl.display();

cout<<endl;

break;

case 9:

cout<<"Exiting..."<<endl;

exit(1);

break;

default:

cout<<"Wrong choice"<<endl;

}

}

}

rogram 11: Implement Doubly Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;

struct node

{

int info;

struct node \*next;

struct node \*prev;

}\*start;

class double\_llist

{

public:

void create\_list(int value);

void add\_begin(int value);

void add\_after(int value, int position);

void delete\_element(int value);

void search(int ele);

void display\_dlist();

void count();

void reverse();

double\_llist()

{

start=NULL;

}

};

int main()

{

int choice, element, position;

int ele;

double\_llist dl;

while (1)

{

cout<<endl<<"------------"<<endl;

cout<<endl<<"Operations on Doubly linked list"<<endl;

cout<<endl<<"------------"<<endl;

cout<<"1.Add at end"<<endl;

cout<<"2.Add at begining"<<endl;

cout<<"3.Add after position"<<endl;

cout<<"4.Delete"<<endl;

cout<<"5.Display"<<endl;

cout<<"6.Count"<<endl;

cout<<"7.Reverse"<<endl;

cout<<"8.search"<<endl;

cout<<"9.exit"<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch ( choice )

{

case 1:

cout<<"Enter the element: ";

cin>>element;

dl.create\_list(element);

cout<<endl;

break;

case 2:

cout<<"Enter the element: ";

cin>>element;

dl.add\_begin(element);

cout<<endl;

break;

case 3:

cout<<"Enter the element: ";

cin>>element;

cout<<"Insert Element after postion: ";

cin>>position;

dl.add\_after(element, position);

cout<<endl;

break;

case 4:

if (start == NULL)

{

cout<<"List empty,nothing to delete"<<endl;

break;

}

cout<<"Enter the element for deletion: ";

cin>>element;

dl.delete\_element(element);

cout<<endl;

break;

case 5:

dl.display\_dlist();

cout<<endl;

break;

case 6:

dl.count();

break;

case 7:

if (start == NULL)

{

cout<<"List empty,nothing to reverse"<<endl;

break;

}

dl.reverse();

cout<<endl;

break;

case 8:

cout<<"Enter the element to be searched ";

cin>>ele;

dl.search(ele);

break;

case 9:

exit(0);

break;

default:

cout<<"Wrong choice"<<endl;

}

}

return 0;

}

void double\_llist::create\_list(int value)

{

struct node \*s, \*temp;

temp = new(struct node);

temp->info = value;

temp->next = NULL;

if (start == NULL)

{

temp->prev = NULL;

start = temp;

}

else

{

s = start;

while (s->next != NULL)

s = s->next;

s->next = temp;

temp->prev = s;

}

}

void double\_llist::add\_begin(int value)

{

if (start == NULL)

{

cout<<"First Create the list."<<endl;

return;

}

struct node \*temp;

temp = new(struct node);

temp->prev = NULL;

temp->info = value;

temp->next = start;

start->prev = temp;

start = temp;

cout<<"Element Inserted"<<endl;

}

void double\_llist::add\_after(int value, int pos)

{

if (start == NULL)

{

cout<<"First Create the list."<<endl;

return;

}

struct node \*tmp, \*q;

int i;

q = start;

for (i = 0;i < pos - 1;i++)

{

q = q->next;

if (q == NULL)

{

cout<<"There are less than ";

cout<<pos<<" elements."<<endl;

return;

}

}

tmp = new(struct node);

tmp->info = value;

if (q->next == NULL)

{

q->next = tmp;

tmp->next = NULL;

tmp->prev = q;

}

else

{

tmp->next = q->next;

tmp->next->prev = tmp;

q->next = tmp;

tmp->prev = q;

}

cout<<"Element Inserted"<<endl;

}

void double\_llist::delete\_element(int value)

{

struct node \*tmp, \*q;

/\*first element deletion\*/

if (start->info == value)

{

tmp = start;

start = start->next;

start->prev = NULL;

cout<<"Element Deleted"<<endl;

free(tmp);

return;

}

q = start;

while (q->next->next != NULL)

{

/\*Element deleted in between\*/

if (q->next->info == value)

{

tmp = q->next;

q->next = tmp->next;

tmp->next->prev = q;

cout<<"Element Deleted"<<endl;

free(tmp);

return;

}

q = q->next;

}

/\*last element deleted\*/

if (q->next->info == value)

{

tmp = q->next;

free(tmp);

q->next = NULL;

cout<<"Element Deleted"<<endl;

return;

}

cout<<"Element "<<value<<" not found"<<endl;

}

/\*

\* Display elements of Doubly Link List

\*/

void double\_llist::display\_dlist()

{

struct node \*q;

if (start == NULL)

{

cout<<"List empty,nothing to display"<<endl;

return;

}

q = start;

cout<<"The Doubly Link List is :"<<endl;

while (q != NULL)

{

cout<<q->info<<endl;

q = q->next;

}

}

void double\_llist::search(int ele){

int flag=0;

int count=0;

struct node \*t;

t=start;

while(t->next!=NULL)

{

++count;

if(t->info==ele)

{

flag=1;

cout<<"Element found at position "<<count;

break;

}

t=t->next;

}

if(flag==0)

cout<<"Element not found ";

}

/\*

\* Number of elements in Doubly Link List

\*/

void double\_llist::count()

{

struct node \*q = start;

int cnt = 0;

while (q != NULL)

{

q = q->next;

cnt++;

}

cout<<"Number of elements are: "<<cnt<<endl;

}

/\*

\* Reverse Doubly Link List

\*/

void double\_llist::reverse()

{

struct node \*p1, \*p2;

p1 = start;

p2 = p1->next;

p1->next = NULL;

p1->prev = p2;

while (p2 != NULL)

{

p2->prev = p2->next;

p2->next = p1;

p1 = p2;

p2 = p2->prev;

}

start = p1;

cout<<"List Reversed"<<endl;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program 12: Implement Circular Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;

struct node

{

int info;

struct node \*next;

}\*last;

class circular\_llist

{

public:

void reverse();

void search(int ele);

void add\_end(int value);

void add\_begin(int value);

void add\_after(int value, int position);

void delete\_element(int value);

void display\_list();

circular\_llist()

{

last = NULL;

}

};

int main()

{

int choice, element, position,ele;

circular\_llist cl;

do{

cout<<"\nMain Menu";

cout<<"\n1.Add at end";

cout<<"\n2.Add at beginning";

cout<<"\n3.Add after";

cout<<"\n4.Delete";

cout<<"\n5.search";

cout<<"\n6.reverse";

cout<<"\n7.display";

cout<<"\nEnter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"\nEnter the element: ";

cin>>element;

cl.add\_end(element);

cout<<endl;

break;

case 2:

cout<<"\nEnter the element: ";

cin>>element;

cl.add\_begin(element);

cout<<endl;

break;

case 3:

cout<<"\nEnter the element: ";

cin>>element;

cout<<"\nInsert element after position: ";

cin>>position;

cl.add\_after(element, position);

cout<<endl;

break;

case 4:

if (last == NULL)

{

cout<<"\nList is empty, nothing to delete";

break;

}

cout<<"\nEnter the element for deletion: ";

cin>>element;

cl.delete\_element(element);

cout<<endl;

break;

case 5:cout<<"Enter the elemeent to be searched ";

cin>>ele;

cl.search(ele);

break;

case 6: cl.reverse();

break;

case 7: cl.display\_list();

break;

}

}while(choice!=6);

}

void circular\_llist::add\_end(int value)

{

struct node \*temp;

temp = new(struct node);

temp->info = value;

if (last == NULL)

{

last = temp;

temp->next = last;

}

else

{

temp->next = last->next;

last->next = temp;

last = temp;

}

}

void circular\_llist:: reverse()

{

struct node \*p, \*q, \*r;

p = q = r = last->next;

p = p->next->next;

q = q->next;

r->next = NULL;

q->next = r;

while (p != last)

{

r = q;

q = p;

p = p->next;

q->next = r;

}

last = q;

}

void circular\_llist:: search(int ele)

{

int flag=0;

int count=0;

struct node \*t;

t=last->next;

while(t->next!=last)

{

++count;

if(t->info==ele)

{

flag=1;

cout<<"Element found at position "<<count;

break;

}

t=t->next;

}

if(flag==0)

cout<<"Element not found ";

}

void circular\_llist::add\_begin(int value)

{

if (last == NULL)

{

cout<<"First Create the list."<<endl;

return;

}

struct node \*temp;

temp = new(struct node);

temp->info = value;

temp->next = last->next;

last->next = temp;

}

void circular\_llist::add\_after(int value, int pos)

{

if (last == NULL)

{

cout<<"First Create the list."<<endl;

return;

}

struct node \*temp, \*s;

s = last->next;

for (int i = 0;i < pos-1;i++)

{

s = s->next;

if (s == last->next)

{

cout<<"There are less than ";

cout<<pos<<" in the list"<<endl;

return;

}

}

temp = new(struct node);

temp->next = s->next;

temp->info = value;

s->next = temp;

if (s == last)

{

last=temp;

}

}

void circular\_llist::delete\_element(int value)

{

struct node \*temp, \*s;

s = last->next;

if (last->next == last && last->info == value)

{

temp = last;

last = NULL;

free(temp);

return;

}

if (s->info == value)

{

temp = s;

last->next = s->next;

free(temp);

return;

}

while (s->next != last)

{

if (s->next->info == value)

{

temp = s->next;

s->next = temp->next;

free(temp);

cout<<"Element "<<value;

cout<<" deleted from the list"<<endl;

return;

}

s = s->next;

}

if (s->next->info == value)

{

temp = s->next;

s->next = last->next;

free(temp);

last = s;

return;

}

cout<<"Element "<<value<<" not found in the list"<<endl;

}

void circular\_llist::display\_list()

{

struct node \*s;

if (last == NULL)

{

cout<<"List is empty, nothing to display"<<endl;

return;

}

s = last->next;

cout<<"Circular Link List: "<<endl;

while (s != last)

{

cout<<s->info<<"->";

s = s->next;

}

cout<<s->info<<endl;

}

Program 13: WAP to add two polynomials using linked list representation.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<iomanip>

#include<conio.h>

using namespace std;

struct poly

{

int coeff;

int pow;

poly \*next;

};

class add2poly

{

poly \*poly1, \*poly2, \*poly3;

public:

add2poly(){poly1=poly2=poly3=NULL;}

void addpoly();

void display();

};

void add2poly :: addpoly()

{

int i,p;

poly \*newl=NULL,\*end=NULL;

cout<<"Enter highest power for x\n";

cin>>p;

//Read first poly

cout<<"\nFirst Polynomial\n";

for(i=p;i>=0;i--)

{

newl=new poly;

newl->pow=p;

cout<<"Enter Co-efficient for degree"<<i<<":: ";

cin>>newl->coeff;

newl->next=NULL;

if(poly1==NULL)

poly1=newl;

else

end->next=newl;

end=newl;

}

//Read Second poly

cout<<"\n\nSecond Polynomial\n";

end=NULL;

for(i=p;i>=0;i--)

{

newl=new poly;

newl->pow=p;

cout<<"Enter Co-efficient for degree"<<i<<":: ";

cin>>newl->coeff;

newl->next=NULL;

if(poly2==NULL)

poly2=newl;

else

end->next=newl;

end=newl;

}

//Addition Logic

poly \*p1=poly1,\*p2=poly2;

end=NULL;

while(p1 !=NULL && p2!=NULL)

{

if(p1->pow == p2->pow)

{

newl=new poly;

newl->pow=p--;

newl->coeff=p1->coeff + p2->coeff;

newl->next=NULL;

if(poly3==NULL)

poly3=newl;

else

end->next=newl;

end=newl;

}

p1=p1->next;

p2=p2->next;

}

}

void add2poly :: display()

{

poly \*t=poly3;

cout<<"\n\nAnswer after addition is : ";

while(t!=NULL)

{

cout.setf(ios::showpos);

cout<<t->coeff;

cout.unsetf(ios::showpos);

cout<<"X"<<t->pow;

t=t->next;

}

}

int main()

{

add2poly obj;

obj.addpoly();

obj.display();

return 0;

}

Program 14: Write a menu driven program to implement the following operations in an ordered linked list:

a) Insertion

b) Deletion

c) Merging

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

using namespace std;

class llnode

{

public:

int info;

llnode \*next;

llnode( int a, llnode \*ptr)

{

info=a;

next=ptr;

}

llnode(int a)

{

info=a;

next='\0';

}

llnode()

{

next='\0';

}

};

class llist

{

llnode \*head, \*tail;

public:

llist()

{

head=tail='\0';

}

void insert\_end( int x)

{

llnode \*p;

p= new llnode(x);

if(head=='\0')

{

head=p;

tail=p;

}

else

{

tail->next=p;

tail=p;

}

}

void insert\_ordered( int x)

{

llnode \*p, \*q, \*r;

p= new llnode(x);

q=head;

if(head=='\0')

{

head=p;

tail=p;

}

else if(x<head->info)

{

p->next=head;

head=p;

}

else if(x>tail->info)

{

tail->next=p;

tail=p;

}

else

{

while(q->info<x)

{

r=q;

q=q->next;

}

r->next=p;

p->next=q;

}

}

void delete\_beg()

{

llnode \*p;

p=head;

if(head=='\0')

{

cout<<"\ncannot delete element as list is empty.";

}

else if(head==tail && head!='\0')

{

head='\0';

tail='\0';

}

else

{

head=head->next;

delete p;

}

}

void delete\_end()

{

llnode \*p,\*q,\*r;

p=head;

q=tail;

if(head=='\0')

{

cout<<"\ncannot delete element as list is empty.";

}

else if(head==tail && head!='\0')

{

head='\0';

tail='\0';

}

else

{

while(p!=tail)

{

r=p;

p=p->next;

}

tail=r;

delete q;

}

}

void dbef\_pvalue( int pvalue)

{

llnode \*p,\*q,\*r;

r=head;

if(head=='\0')

{

cout<<"\nNo element in list.";

}

else

{

while(r->info!=pvalue)

{

if(r->info!=pvalue && r->next=='\0')

{

cout<<"\nElement not found!!!";

cout<<"\nExiting";

exit(1);

}

q=r;

r=r->next;

}

if(r==head)

{

cout<<"\nNo element before "<<pvalue;

}

else if( q==head)

{

head=r;

delete q;

}

else

{

p=head;

while(p->next!=q)

{

p=p->next;

}

p->next=r;

delete q;

}

}

}

void daft\_pvalue( int pvalue)

{

llnode \*p,\*q,\*r;

q=head;

if(head=='\0')

{

cout<<"\nNo element in list.";

}

else

{

while(q->info!=pvalue)

{

if(q->info!=pvalue && q->next=='\0')

{

cout<<"\nElement not found!!!";

cout<<"\nExiting";

exit(1);

}

r=q;

q=q->next;

}

if(q==head)

{

if(head==tail)

{

cout<<"\nNo element after "<<pvalue;

}

else if(head->next==tail)

{

p=tail;

tail=head;

delete p;

}

else

{

p=head->next;

r=p->next;

head->next=r;

delete p;

}

}

else if( q==tail)

{

cout<<"\nNo element after "<<pvalue;

}

else

{

p=q->next;

if(p==tail)

{

tail=q;

}

else

{

r=p->next;

q->next=r;

}

delete p;

}

}

}

void dat\_ppos( int ppos)

{

llnode \*p,\*q,\*r;

q=head;

if(ppos<1)

{

cout<<"\nWrong input!!!";

}

else if(ppos==1)

{

if(head=='\0')

{

cout<<"\nCannot delete element as no element in list.";

}

else if( head==tail && head!='\0')

{

head='\0';

tail='\0';

delete q;

}

else

{

head=head->next;

delete q;

}

}

else

{

for(int i=ppos; i>=2; i--)

{

if(q->next=='\0' && i!=2)

{

cout<<"\nWrong position!!";

cout<<"\nExiting";

exit(1);

}

r=q;

q=q->next;

}

if(q==tail)

{

tail=r;

delete q;

}

else

{

p=q->next;

r->next=p;

delete q;

}

}

}

llist merge(llist ob)

{

llist ob2;

llnode \*p, \*q,\*r;

p=head;

q=ob.head;

while(p!='\0' && q!='\0')

{

if(p->info<q->info)

{

ob2.insert\_end(p->info);

p=p->next;

}

else if(p->info==q->info)

{

ob2.insert\_end( p->info );

p=p->next;

q=q->next;

}

else

{

ob2.insert\_end(q->info);

q=q->next;

}

}

if(p!='\0')

{

while(p!='\0')

{

ob2.insert\_end(p->info);

p=p->next;

}

}

else if( q!='\0')

{

while(q!='\0')

{

ob2.insert\_end(q->info);

q=q->next;

}

}

return ob2;

}

void disp()

{

llnode \*p;

p=head;

if(head=='\0')

{

cout<<"\nList is empty.";

}

else

{

cout<<"\nList: ";

while(p!=tail)

{

cout<<p->info<<" ";

p=p->next;

}

cout<<p->info;

}

}

};

int main()

{

int choice, num,num2;

char ch;

llist ob1, ob2, ob3;

do

{

system("cls");

cout<<"\n1. Insert element.";

cout<<"\n2. Delete element.";

cout<<"\n3. Merge two lists.";

cout<<"\n4. Display list.";

cout<<"\n5. Exit.";

cout<<"\n\nEnter your choice: ";

cin>>choice;

system("cls");

switch(choice)

{

case 1: cout<<"\nEnter value for new element: ";

cin>>num;

ob1.insert\_ordered(num);

break;

case 2: cout<<"\n1. Delete element from beginning.";

cout<<"\n2. Delete element from end.";

cout<<"\n3. Delete element before a particaular value.";

cout<<"\n4. Delete element after a particaular value.";

cout<<"\n5. Delete element at a particular index value.";

cout<<"\n\nEnter your choice: ";

cin>>choice;

switch(choice)

{

case 1: ob1.delete\_beg();

break;

case 2: ob1.delete\_end();

break;

case 3: cout<<"\nEneter element before which you want to delete element: ";

cin>>num2;

ob1.dbef\_pvalue(num2);

break;

case 4: cout<<"\nEneter element after which you want to delete element: ";

cin>>num2;

ob1.daft\_pvalue(num2);

break;

case 5: cout<<"\nEnter position at which you want to delete element: ";

cin>>num2;

ob1.dat\_ppos(num2);

break;

default:cout<<"\nWrong input";

break;

}

break;

case 3: cout<<"\nProvide another list(temporary list with insertion at end only):-\n";

cout<<"\nEnter number of elements: ";

cin>>num;

cout<<"\nEnter elements:-\n";

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

{

cin>>num2;

ob2.insert\_end(num2);

}

ob3=ob1.merge(ob2);

ob3.disp();

break;

case 4: ob1.disp();

break;

case 5: cout<<"\nEXITING...";

exit(100);

default: cout<<"\nWrong input!!!";

break;

}

cout<<"\n\nEnter again(y/n): ";

cin>>ch;

}while(ch=='y');

return 0;

}

Program 15: Write a menu driven program to implement the following operations in a Binary Search Tree:

a) Insertion

b) Deletion by copying

c) Deletion by Merging

d) Search a no. Merging

e) Search a no. in BST

f) Display its preorder, postorder and inorder traversals

g) Display its level-by-level traversals

h) Count the non-leaf nodes and leaf nodes

i) Display height of tree

j) Create a mirror image of tree

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

using namespace std;

#define MAX 20

class bstnode

{

public:

int info;

bstnode \*left, \*right;

bstnode()

{

info=0;

left='\0';

right='\0';

}

bstnode(int x)

{

info=x;

left='\0';

right='\0';

}

};

class stack

{

bstnode \*st[MAX];

int top;

public:

stack()

{

top=-1;

}

int isempty()

{

if(top==-1)

return 1;

else

return 0;

}

void push(bstnode \*x)

{

if(top==(MAX-1))

cout<<"\nStack overflow";

else

{

top=top+1;

st[top]=x;

}

}

bstnode\* ret\_top()

{

return st[top];

}

bstnode\* pop()

{

if(top==-1)

cout<<"\nUnderflow";

else

{

bstnode\* x=st[top];

top=top-1;

return x;

}

}

};

class queue

{

bstnode \*A[MAX];

int front, rear;

public:

queue()

{

front= -1;

rear= -1;

}

int IsEmpty()

{

if((front == -1)&&(rear == -1))

return 1;

else

return 0;

}

int IsFull()

{

if((front == rear+1)||((front == 0)&&(rear == MAX-1)))

return 1;

else

return 0;

}

void Enqueue(bstnode \*x)

{

if(IsFull())

{

cout<<"\nQueue is full.";

return;

}

else

{

if(front == -1)

{

rear++;

front++;

}

else if(rear == MAX-1)

{

rear=0;

}

else

{

rear++;

}

A[rear]=x;

}

}

bstnode\* Dequeue()

{

if(IsEmpty())

{

cout<<"\nQueue is empty.";

}

else

{

bstnode \*x= A[front];

if(front == MAX-1 )

{

front = 0;

}

else if(front == rear)

{

front = rear = -1;

}

else

{

front++;

}

return x;

}

}

};

class bst

{

bstnode \*root;

stack s;

queue q;

public:

bst()

{

root='\0';

}

void insert(int x) //function for inserting values to binary tree.

{

bstnode \*p, \*q, \*prev;

p=new bstnode(x);

q=root;

if(q=='\0')

{

root=p;

}

else

{

while(q!='\0')

{

if(x<q->info)

{

prev=q;

q=q->left;

}

else

{

prev=q;

q=q->right;

}

}

if(x<prev->info)

prev->left=p;

else

prev->right=p;

}

}

bstnode\* ret\_root()

{

return root;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ITERATIVE TRAVERSALS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void pre\_order\_ite(bstnode \*p)

{

bstnode \*q;

if(p!='\0')

{

if(p!='\0')

s.push(p);

while(!s.isempty())

{

q=s.pop();

cout<<q->info<<" ";

if(q->right!='\0')

s.push(q->right);

if(q->left!='\0')

s.push(q->left);

}

}

else

cout<<"\nTree is empty.";

}

void in\_order\_ite( bstnode \*p)

{

if(p!='\0')

{

while(p!='\0')

{

while(p!='\0')

{

if(p->right!='\0')

s.push(p->right);

s.push(p);

p=p->left;

}

p=s.pop();

while( !s.isempty() && p->right=='\0')

{

cout<<p->info<<" ";

p=s.pop();

}

cout<<p->info<<" ";

if(!s.isempty())

p=s.pop();

else

p='\0';

}

}

else

cout<<"\nTree is empty.";

}

void post\_order\_ite(bstnode \*p)

{

if(p!='\0')

{

do

{

while(p!='\0')

{

if(p->right!='\0')

s.push(p->right);

s.push(p);

p=p->left;

}

p=s.pop();

if( s.ret\_top()==p->right && p->right!='\0')

{

s.pop();

s.push(p);

p=p->right;

}

else

{

cout<<p->info<<" ";

p='\0';

}

}while(!s.isempty());

}

else

cout<<"\nTree is empty.";

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RECURSIVE TRAVERSALS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void pre\_order\_rec(bstnode \*p)

{

if(p!='\0')

{

cout<<p->info<<" ";

pre\_order\_rec(p->left);

pre\_order\_rec(p->right);

}

else

cout<<"\nTree is empty.";

}

void in\_order\_rec(bstnode \*p)

{

if(p!='\0')

{

pre\_order\_rec(p->left);

cout<<p->info<<" ";

pre\_order\_rec(p->right);

}

else

cout<<"\nTree is empty.";

}

void post\_order\_rec(bstnode \*p)

{

if(p!='\0')

{

pre\_order\_rec(p->left);

pre\_order\_rec(p->right);

cout<<p->info<<" ";

}

else

cout<<"\nTree is empty.";

}

void bfs(bstnode \*p) //Breadth first search traversal.

{

if(p!='\0')

{

cout<<p->info<<" ";

do

{

if(p->left)

q.Enqueue(p->left);

if(p->right)

q.Enqueue(p->right);

p=q.Dequeue();

cout<<p->info<<" ";

}while(!q.IsEmpty());

}

else

cout<<"\nTree is empty.";

}

int height(bstnode \*p)

{

if(p=='\0')

return 0;

else

{

int lht= height(p->left);

int rht= height(p->right);

if(lht>rht)

return lht+1;

else

return rht+1;

}

}

void mirror( bstnode \*p)

{

bstnode \*temp;

if(p!='\0')

{

mirror(p->left);

mirror(p->right);

temp=p->left;

p->left=p->right;

p->right=temp;

}

}

int getLeafCount(bstnode \*p)

{

if( p=='\0')

return 0;

if(p->left == '\0' && p->right=='\0')

return 1;

else

return getLeafCount(p->left) + getLeafCount(p->right);

}

int totalNode(bstnode \*p)

{

if(p=='\0')

return 0;

else

return 1+totalNode(p->left)+totalNode(p->right);

}

int search(bstnode \*p, int x)

{

if(p!='\0')

{

if(p->info==x)

return 1;

else if( p->info<x)

search(p->right,x);

else

search(p->left,x);

}

else

return 0;

}

bstnode\* FindMin(bstnode \*node)

{

if(node==NULL)

{

/\* There is no element in the tree \*/

return NULL;

}

if(node->left) /\* Go to the left sub tree to find the min element \*/

return FindMin(node->left);

else

return node;

}

bstnode \* DeletebyCopying(bstnode \*node, int info)

{

bstnode \*temp;

if(node=='\0')

{

cout<<"\nElement Not Found";

}

else if(info < node->info)

{

node->left = DeletebyCopying(node->left, info);

}

else if(info > node->info)

{

node->right = DeletebyCopying(node->right, info);

}

else

{

if(node->right && node->left) //Two child

{

temp = FindMin(node->right);

node -> info = temp->info; //Copy minimum value of right sub-tree onto node to be deleted.

node -> right = DeletebyCopying(node->right,temp->info); //Delete node copied onto node to be deleted.

}

else // Only child or One child.

{

temp = node;

if(node->left == NULL)

node = node->right;

else if(node->right == NULL)

node = node->left;

delete (temp);

}

}

return node;

}

};

int main()

{

bst ob;

char ch;

int num, num2;

int leaf, nleaf;

do

{

system("cls");

cout<<"\n\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*\*";

cout<<"\n1. Insert element.";

cout<<"\n2. Pre-order traversal.";

cout<<"\n3. In-order traversal.";

cout<<"\n4. Post-order traversal.";

cout<<"\n5. Deletion by copying.";

cout<<"\n6. Deletion by merging.";

cout<<"\n7. Height of tree.";

cout<<"\n8. Mirror image of tree.";

cout<<"\n9. Count leaf and non-leaf nodes.";

cout<<"\n10. Search any value in tree.";

cout<<"\n11. Exit.";

cout<<"\nEnter your choice: ";

cin>>num;

switch(num)

{

case 1: system("cls");

do{

cout<<"\nEnter value: ";

cin>>num2;

ob.insert(num2);

cout<<"\nEnter more values(y/n): ";

cin>>ch;

}while(ch=='y');

break;

case 2: system("cls");

cout<<"\n1. Using iterative.";

cout<<"\n2. Using recursive.";

cout<<"\nEnter your choice: ";

cin>>num;

if(num==1)

ob.pre\_order\_ite( ob.ret\_root());

else

ob.pre\_order\_ite( ob.ret\_root());

break;

case 3: system("cls");

cout<<"\n1. Using iterative.";

cout<<"\n2. Using recursive.";

cout<<"\nEnter your choice: ";

cin>>num;

if(num==1)

ob.in\_order\_ite( ob.ret\_root());

else

ob.in\_order\_ite( ob.ret\_root());

break;

case 4: system("cls");

cout<<"\n1. Using iterative.";

cout<<"\n2. Using recursive.";

cout<<"\nEnter your choice: ";

cin>>num;

if(num==1)

ob.post\_order\_ite( ob.ret\_root());

else

ob.post\_order\_ite( ob.ret\_root());

break;

case 5: cout<<"\nValue to be deleted from tree: ";

cin>>num2;

ob.DeletebyCopying( ob.ret\_root(), num2);

break;

case 6:

break;

case 7: num2=ob.height( ob.ret\_root());

cout<<"\nHeight of tree: "<<num2;

break;

case 8: ob.mirror( ob.ret\_root());

break;

case 9: leaf=ob.getLeafCount( ob.ret\_root());

nleaf=ob.totalNode( ob.ret\_root()) -leaf;

if(leaf==1 && nleaf==0)

{

cout<<"\nOnly root node present.";

}

else

{

cout<<"\nNumber of leaf nodes: "<<leaf;

cout<<"\nNumber of non-leaf node: "<<nleaf;

}

break;

case 10:cout<<"\nEnter value for searching: ";

cin>>num2;

if( ob.search( ob.ret\_root(), num2) )

cout<<"\nValue found in tree.";

else

cout<<"\nValue not found.";

break;

case 11: cout<<"\nExiting...";

exit(100);

break;

default: cout<<"\nWrong input!!!";

break;

}

cout<<"\n\nReturn to menu(y/n): ";

cin>>ch;

}while(ch=='y');

return 0;

}

Program 16: Write a menu driven program to implement the following sorting and searching algorithms:

a) Bubble Sort

b) Insertion

c) Selection

d) Merge Sort

e) Quick Sort

f) Linear Search

g) Binary Search

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

using namespace std;

int partition(int A[], int low, int high);

void inp\_arr(int A[],int size)

{

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

{

cin>>A[i];

}

}

void display(int A[], int size)

{

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

{

cout<<A[i]<<" ";

}

}

void bubble\_sort( int A[], int size)

{

for(int i=0; i<size-1; i++)

{

for(int j=0; j<size-i-1; j++)

{

if( A[j]>A[j+1])

swap(A[j],A[j+1]);

}

}

}

void insertion\_sort( int A[], int size)

{

int temp,j;

for(int i=1; i<size; i++)

{

temp=A[i];

for( j=i-1; temp<A[j] && j>=0; j--)

{

A[j+1]=A[j];

}

A[j+1]=temp;

}

}

void selection\_sort( int A[], int size)

{

for(int i=0; i<size-1; i++)

{

for(int j=i+1; j<size; j++)

{

if( A[i]>A[j])

swap(A[i],A[j]);

}

}

}

void merge\_sort( int A[], int size)

{

}

void quick\_sort( int A[], int low, int high)

{

int pi;

if( low<high)

{

pi=partition(A,low,high);

quick\_sort(A,low,pi-1);

quick\_sort(A,pi+1,high);

}

}

int partition(int A[], int low, int high)

{

int pivot=A[high];

int i=low-1, j;

for( j=low; j<high; j++)

{

if(A[j]<=pivot)

{

i++;

swap(A[i],A[j]);

}

}

swap(A[i+1],A[high]);

return i+1;

}

int lsearch\_ite(int A[], int num, int size)

{

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

{

if(A[i]==num)

return i+1;

}

return 0;

}

int lsearch\_rec(int A[],int num,int index, int size)

{

if(index<size)

{

if(A[index]==num)

return index+1;

else

lsearch\_rec(A,num,index+1,size);

}

else

return 0;

}

int bsearch\_ite(int A[], int num, int size)

{

int beg=0, end=size;

int mid;

while(beg<end)

{

mid=(beg+end)/2;

if(A[mid]==num)

return mid+1;

else if(A[mid]>num)

end=mid-1;

else

beg=mid+1;

}

return 0;

}

int bsearch\_rec(int A[], int num, int beg, int end)

{

int mid=(beg+end)/2;

if(beg>end)

return 0;

else if(A[mid]==num)

return mid+1;

else if(A[mid]>num)

return bsearch\_rec(A,num,beg,mid-1);

else

return bsearch\_rec(A,num,mid+1, end);

}

int main()

{

int num,num2, size, found;

char ch;

cout<<"\nEnter total number of values: ";

cin>>size;

int A[size], B[size];

cout<<"\nEnter elements:-\n";

inp\_arr(A,size);

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

{

B[i]=A[i];

}

do{

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

{

A[i]=B[i];

}

system("cls");

display(A,size);

cout<<"\n\*\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout<<"\n1. Bubble sort.";

cout<<"\n2. Insertion sort.";

cout<<"\n3. Selection sort.";

cout<<"\n4. Merge sort.";

cout<<"\n5. Quick sort.";

cout<<"\n6. Linear serach.";

cout<<"\n7. Binary search.";

cout<<"\n8. Exit.";

cout<<"\n\nEnter your choice: ";

cin>>num;

switch(num)

{

case 1: bubble\_sort(A,size);

display(A,size);

break;

case 2: insertion\_sort(A,size);

display(A,size);

break;

case 3: selection\_sort(A,size);

display(A,size);

break;

case 4: merge\_sort(A,size);

display(A,size);

break;

case 5: quick\_sort(A,0,size-1);

display(A,size);

break;

case 6: system("cls");

cout<<"\nEnter number to be searched: ";

cin>>num2;

cout<<"\n1. Search using iteration.";

cout<<"\n2. Search using recursion.";

cout<<"\nEnter your choice: ";

cin>>num;

if(num==1)

{

if(found=lsearch\_ite(A,num2,size))

cout<<"\nNumber found at "<<found<<" position.";

else

cout<<"\nNot found.";

}

else

{

if(found=lsearch\_rec(A,num2,0,size))

cout<<"\nNumber found at "<<found<<" position.";

else

cout<<"\nNot found.";

}

break;

case 7: system("cls");

cout<<"\nEnter number to be searched: ";

cin>>num2;

cout<<"\n1. Search using iteration.";

cout<<"\n2. Search using recursion.";

cout<<"\nEnter your choice: ";

cin>>num;

if(num==1)

{

if(found=bsearch\_ite(A,num2,size))

cout<<"\nNumber found at "<<found<<" position.";

else

cout<<"\nNot found.";

}

else

{

if(found=bsearch\_rec(A,num2,0,size-1))

cout<<"\nNumber found at "<<found<<" position.";

else

cout<<"\nNot found.";

}

break;

case 8: cout<<"EXITING...";

exit(100);

break;

default: cout<<"\nWrong input!!!";

break;

}

cout<<"\nShow Menu again(y/n): ";

cin>>ch;

}while(ch=='y');

return 0;

}