*Experiment No.-1*

**AIM:-**

Write a menu driven program that implements following operations on a linear array:

* Insert a new element at a specified position
* Delete an element either whose value is given or whose position is given
* To find the location of a given element

To display the elements of the linear array

**PROGRAM-**

#include<iostream>

using namespace std;

class array1

{

int size;

int arr[100];

public:

array1()

{

cout<<"enter the size of array\n";

cin>>size;

cout<<"enter the values:\n";

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

{

cin>>arr[i];

}

}

void insert()

{

cout<<"enter the position where you want to insert\n";

int pos,i;

cin>>pos;

for( i=size-1;i>=pos;i--)

{

arr[i+1]=arr[i];

}

cout<<"enter the element you want to insert\n";

int ele;

cin>>ele;

arr[i+1]=ele;

size=size+1;

}

void delete1()

{

int pos1,i;

cout<<"enter the position which you want to delete an element\n";

cin>>pos1;

for(i=pos1;i<size;i++)

{

arr[i]=arr[i+1];

}

size=size-1;

}

void location()

{

int ele1,i;

cout<<"enter the element you want to search\n";

cin>>ele1;

int f=0;

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

{

if(arr[i]==ele1)

{

f=1;

cout<<"Element is found at pos "<<i<<endl;

}

}

if(f==0)

cout<<"Element not found\n";

}

void show()

{

int i;

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

{

cout<<arr[i]<<endl;

}

}

};

int main()

{

array1 obj1;

int ch;

do

{

cout<<"enter 1 to insert a new element at specified position in array\n";

cout<<"enter 2 to delete the given element\n ";

cout<<"enter 3 to search an element in array\n";

cout<<"enter 4 to show array elements\n";

cout<<"enter your choice\n";

cin>>ch;

switch(ch)

{

case 1:

obj1.insert();

break;

case 2:

obj1.delete1();

break;

case 3:

obj1.location();

break;

case 4:

obj1.show();

break;

default:

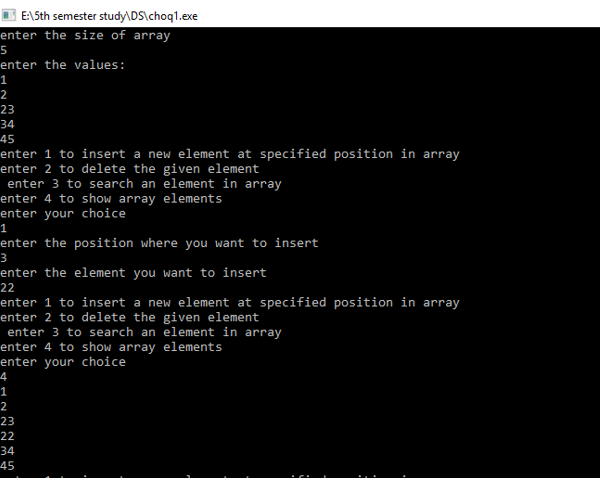
cout<<"you have entered a wrong choice\n";

break;

}

}while(ch>=1 && ch<=4);}

**OUTPUT-**

****

***Experiment No.-2***

**AIM:-** Write a program to accept N numbers from the user and store them in an array. Then, accept another number from the user and search that using Linear Search.

**PROGRAM-**

#include<iostream>

using namespace std;

class array1

{

int size;

int arr[100];

public:

array1()

{

cout<<"enter the size of array\n";

cin>>size;

cout<<"enter the values:\n";

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

{

cin>>arr[i];

}

}

void location()

{

int ele1,i;

cout<<"enter the element you want to search\n";

cin>>ele1;

int f=0;

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

{

if(arr[i]==ele1)

{

f=1;

cout<<"Element is found at pos "<<i<<endl;

}

}

if(f==0)

{

cout<<"element not found\n";

}

}

};

int main()

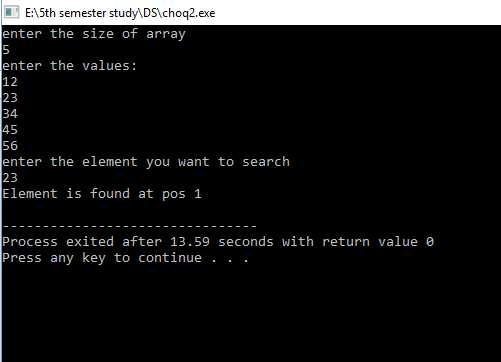
{

array1 obj1;

obj1.location();

}

**OUTPUT-**



***Experiment No.-3***

**AIM:-** Write a program to accept N integers from the user and store them in an array. Sort the array in ascending order using Bubble sort. Then accept another number from the user, search whether that number exists in the array using Binary Search. If it does, display its index and if it doesn’t, **then print that the number is not found in the array.**

**PROGRAM-**

#include<iostream>

using namespace std;

class array2

{

int size;

int arr[100];

public:

array2()

{

cout<<"enter the size of an array\n";

cin>>size;

cout<<"enter the array\n";

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

{

cin>>arr[i];

}

}

int temp;

void bubble\_sort()

{

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

{

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

{

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

{

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

}

void binary\_search()

{

int beg=0,end=size-1,mid,value;

cout<<"enter the value u want to search";

cin>>value;

while(beg<end)

{

mid=(beg+end)/2;

if(arr[mid]==value)

{

cout<<"the element "<<arr[mid]<<" is present at index "<<mid+1<<endl;

break;

}

else if(arr[mid]<value)

{

beg=mid+1;

}

else

{

end=mid-1;

}

}

if(beg>end)

{

cout<<"element is not present in array\n";

}

}

void display()

{

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

{

cout<<arr[i]<<endl;

}

}

};

int main()

{

array2 obj;

int ch;

do

{

cout<<"enter 1 for bubble sort\n";

cout<<"enter 2 for binary search\n";

cout<<"enter 3 for displaying the array\n";

cout<<"enter the choice";

cin>>ch;

switch(ch)

{

case 1:

obj.bubble\_sort();

break;

case 2:

obj.binary\_search();

break;

case 3:

obj.display();

break;

default:

cout<<"please enter right choice\n";

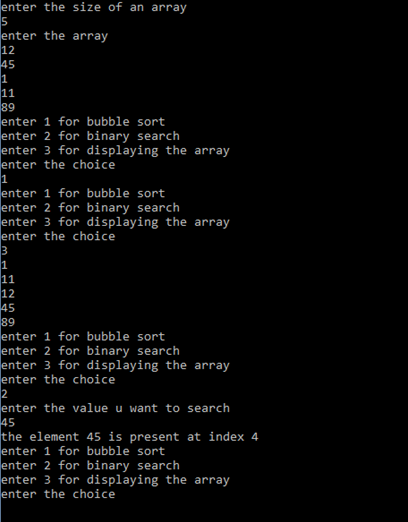
break;

}

}while(ch>=1&&ch<=3);

}

**OUTPUT-**



***Experiment No.-4***

**AIM:-** Write a menu driven program that implements the following operations on a

linked list .

* Insert a new element at the beginning ,end and in-between the given list
* Delete an existing element
* Search an element
* Display all the elements

**PROGRAM-**

#include<iostream>

using namespace std;

class linkedlist

{

int size;

struct node

{

int info;

struct node \*next;

};

struct node \*start=NULL;

struct node \*new\_node;

struct node \*ptr;

public:

void create\_ll()

{

cout<<"enter the size\n";

cin>>size;

cout<<"enter th elements in linked list\n";

int num;

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

{

cin>>num;

new\_node=new node;

new\_node->info= num;

new\_node->next=start;

start=new\_node;

}

}

void insert\_beg()

{

cout<<"enter the element that you want to insert at beg\n";

int ele;

cin>>ele;

new\_node=new node;

new\_node->info=ele;

new\_node->next=start;

start=new\_node;

}

void insert\_end()

{

int ele;

cout<<"enter the element that you want to insert at the end\n";

cin>>ele;

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

new\_node=new node;

new\_node->info=ele;

new\_node->next=ptr->next;

ptr->next=new\_node;

}

void insert\_after()

{

int ele,key;

cout<<"Enter the element after which you want to insert an element\n";

cin>>key;

cout<<"Enter the element that you want to insert in the list\n";

cin>>ele;

ptr=start;

while(ptr!=NULL)

{

if(ptr->info==key)

{

new\_node=new node;

new\_node->info=ele;

new\_node->next=ptr->next;

ptr->next=new\_node;

}

ptr=ptr->next;

}

}

void insert\_before()

{

struct node \*save=NULL;

int key,ele;

cout<<"Enter the element that you want to insert before the given element\n";

cin>>key;

cout<<"enter the element you want to insert\n";

cin>>ele;

ptr=start;

while(ptr!=NULL)

{

if(ptr->info==key)

{

if(save==NULL)

{

insert\_beg();

}

else

{

new\_node=new node;

new\_node->info=ele;

new\_node->next=save->next;

save->next=new\_node;

}

}

save=ptr;

ptr=ptr->next;

}

}

void deletion()

{

int ele;

cout<<"Enter the element that you want to delete\n";

cin>>ele;

struct node \*save=NULL;

ptr=start;

while(ptr!=NULL)

{

if(ptr->info==ele)

{

save->next=ptr->next;

delete ptr;

break;

}

save=ptr;

ptr=ptr->next;

}

}

void search()

{

int search;

cout<<"enter the element that you want to search\n";

cin>>search;

int c=1,f=0;

ptr=start;

while(ptr!=NULL)

{

c++;

if(ptr->info==search)

{

f=1;

cout<<"found the element at "<<c;

}

}

if(f==0)

cout<<"element does not exist in list\n";

}

void show()

{

ptr=start;

while(ptr!=NULL)

{

cout<<ptr->info<<endl;

ptr=ptr->next;

}

}

};

int main()

{

linkedlist obj1;

int ch;

do

{

cout<<"Enter 1 to create linked list\n";

cout<<"Enter 2 to insert at the beginning of the list\n";

cout<<"Enter 3 to insert at the end of the list\n";

cout<<"Enter 4 to insert after the given element in the lis\n";

cout<<"Enter 5 to insert before the given element in the list\n";

cout<<"Enter 6 to delete the given element in the linked list\n";

cout<<"Enter 7 to search the given element in the linked list\n";

cout<<"Enter 8 to show the linked list\n";

cout<<"Enter your choice !!!\n";

cin>>ch;

switch(ch)

{

case 1:

obj1.create\_ll();

break;

case 2:

obj1.insert\_beg();

break;

case 3:

obj1.insert\_end();

break;

case 4:

obj1.insert\_after();

break;

case 5:

obj1.insert\_before();

break;

case 6:

obj1.deletion();

break;

case 7:

obj1.search();

break;

case 8:

obj1.show();

break;

default:

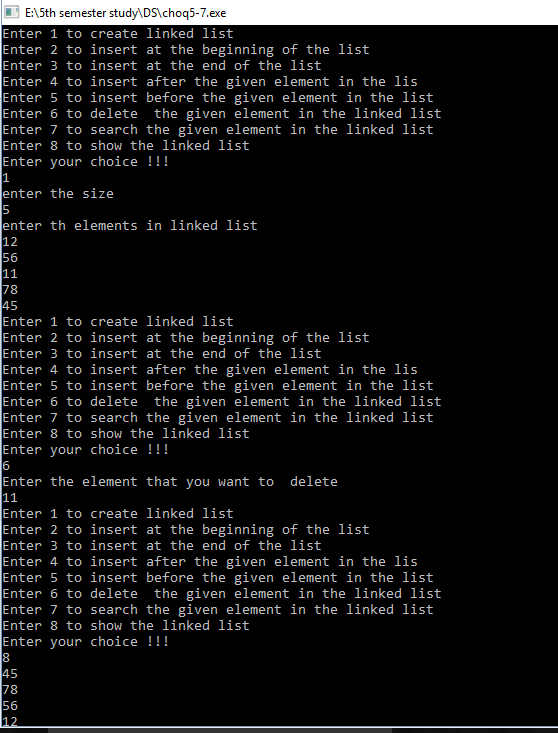
cout<<"you have entered a wrong choice\n";

break;

}

}while(ch>=1 && ch<=8);

}

**OUTPUT- **

***Experiment No.-5***

**AIM:-**Write a menu driven program that implements the following operations on a

Stack(either implement as Linear array or as Linked list) :

* Push
* Pop
* Display Top of the stack

**PROGRAM-**

#include<iostream>

using namespace std;

class stack\_array

{

int stk[200],top;

public:

stack\_array()

{

top=-1;

}

void push(int x);

void pop();

void display();

};

void stack\_array::push(int x)

{

if(top>200)

{

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

return;

}

stk[++top]=x;

cout<<"successfully inserted"<<endl;

}

void stack\_array::pop()

{

if(top<0)

{

cout<<"underflow "<<endl;

return;

}

cout<<"deleted element is:"<<stk[top--]<<endl;

}

void stack\_array::display()

{

if(top<0)

{

cout<<"stack is empty"<<endl;

return;

}

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

cout<<"stack is:"<<stk[i]<<endl;

}

int main()

{

stack\_array sh;

sh.push(1);

sh.push(2);

sh.push(3);

sh.push(4);

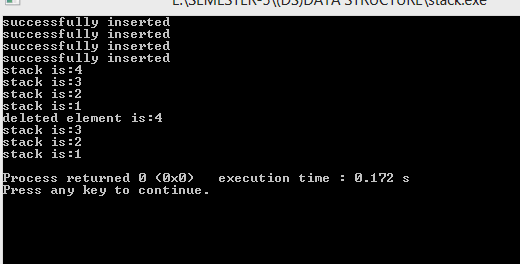
sh.display();

sh.pop();

sh.display();

}

**OUTPUT-**



***Experiment No-6***

**AIM:-** Write a program to demonstrate the use of stack in converting arithmetic expression from infix notation to postfix notation and in evaluating arithmetic postfix expression.

**PROGRAM-**

1. Infix to postfix:

#include<iostream>

#include<stack>

#include<string>

#include<algorithm>

using namespace std;

class infixToPostfix

{

stack<char> S;

public:

string conversion(string expr);

int precedence(char op1, char op2);

bool IsOperator(char C);

bool IsOperand(char C);

int IsRightAssociative(char op)

{

if(op == '^') return true;

return false;

}

int OperatorWeight(char op)

{

int weight=0;

switch(op)

{

case '^':

weight = 3;

break;

case '\*':

case '/':

weight = 2;

break;

case '+':

case '-':

weight = 1;

break;

default:

weight=0;

break;

}

return weight;

}

};

string infixToPostfix :: conversion(string expr)

{

string postfix = " ";

for(int i = 0;i< expr.length();i++)

{

if(expr[i] == ' ' || expr[i] == ',') continue;

else if(IsOperator(expr[i]))

{

while((!S.empty()) && (S.top() != '(') && (precedence(S.top(),expr[i])))

{

postfix+= S.top();

S.pop();

}

S.push(expr[i]);

}

else if(IsOperand(expr[i]))

postfix +=expr[i];

else if (expr[i] == '(')

S.push(expr[i]);

else if(expr[i] == ')')

{

while(!S.empty() && S.top() != '(') {

postfix += S.top();

S.pop();

}

S.pop();

}

}

while(!S.empty())

{

postfix += S.top();

S.pop();

}

return postfix;

}//func conversion ends

bool infixToPostfix :: IsOperand(char C)

{

if(C >= '0' && C <= '9') return true;

if(C >= 'a' && C <= 'z') return true;

if(C >= 'A' && C <= 'Z') return true;

else

return false;

}//func isoperand ends

bool infixToPostfix :: IsOperator(char C)

{

if(C == '+' || C == '-' || C == '\*' || C == '/' || C== '^')

return true;

else

return false;

}

int infixToPostfix :: precedence(char op1, char op2)

{

int op1Weight = OperatorWeight(op1);

int op2Weight = OperatorWeight(op2);

if(op1Weight == op2Weight)

{

if(IsRightAssociative(op1)) return false;

else return true;

}

return op1Weight > op2Weight ? true: false;

}

int main()

{

infixToPostfix i;

string expr;

cout<<"Enter Infix Expression \n";

getline(cin,expr);

string postfix = i.conversion(expr);

cout<<"Output = "<<postfix<<"\n";

}

1. Evaluation postfix expression:

#include<iostream>

#include<cctype>

#include<stack>

using namespace std;

class postfixEvaluation

{

public:

int eval(int op1, int op2, char operate);

int evalPostfix(char postfix[], int size);

};

int postfixEvaluation:: eval(int op1, int op2, char operate)

{

switch (operate)

{

case '\*': return op2 \* op1;

case '/': return op2 / op1;

case '+': return op2 + op1;

case '-': return op2 - op1;

default : return 0;

}

}

int postfixEvaluation:: evalPostfix(char postfix[], int size)

{

stack<int> s;

int i = 0;

char ch;

int val;

while (i < size)

{

ch = postfix[i];

if (isdigit(ch))

s.push(ch-'0');

else

{

int op1 = s.top();

s.pop();

int op2 = s.top();

s.pop();

val = eval(op1, op2, ch);

s.push(val);

}

i++;

}

return val;

}

// main

int main()

{

postfixEvaluation p;

char postfix[] = {'9','8','6','+','\*','2','/'};

int size = sizeof(postfix);

cout<<"postfix expression is:"<<postfix;

int val = p.evalPostfix(postfix, size);

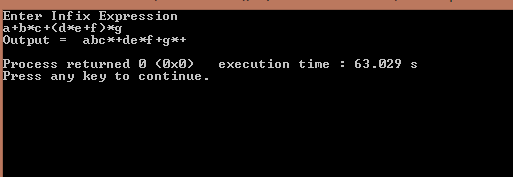
cout<<"\nExpression evaluates to "<<val;

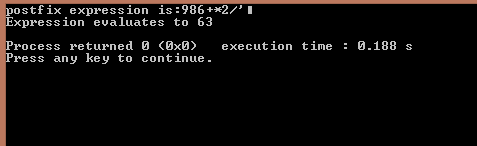
cout<<endl;

return 0;

}

**OUTPUT-**





***Experiment No-7***

**AIM:-** Menu driven Program to demonstrate the implementation of various operations on a Circular queue (using a linear array or a linked list).

**PROGRAM-**

#include<iostream>

using namespace std;

class queue

{

int rear;

int front;

int data;

int size;

int \*a;

public:

queue()

{

cout<<"enter size"<<endl;

cin>>size;

a=new int[size];

rear=-1;

front=-1;

}

int empty()

{

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

{

return 1;

}

else

return 0;

}

int full()

{

if(rear==(size-1) && front==0)

return 1;

else

return 0;

}

void enque(int data)

{

if(full()||rear==front-1)

{

cout<<"overflow"<<endl;

}

else if(empty())

{

front++;

rear++;

a[rear]=data;

}

else if(rear==size-1)

{

rear=0;

a[rear]=data;

}

else

{

rear=rear+1;

a[rear]=data;

}

}

void dequeue()

{

int t;

if(empty())

{

cout<<"underflow"<<endl;

}

else if(front==rear)

{

front=-1;

rear=-1;

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

}

else if(front==size-1 && rear<front)

{

front=0;

}

else

{

t=a[front];

front=front+1;

}

}

void print()

{

if(empty())

{

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

}

else if(rear<front)

{

rear=0;

for(int i=rear;i<=front;i++)

{

cout<<a[i]<<" ";

}

cout<<endl;

}

else

{

for(int i=front;i<=rear;i++)

{

cout<<a[i]<<" ";

}

cout<<endl;

}

}

};

int main()

{

queue obj;

int x;

int m;

int n;

while(1)

{

cout<<"1. To enter"<<endl<<"2. To delete"<<endl<<"3. To print"<<endl;

cin>>x;

switch(x){

case(1):

cout<<"enter data"<<endl;

cin>>m;

obj.enque(m);

break;

case(2):

obj.dequeue();

break;

case(3):

obj.print();

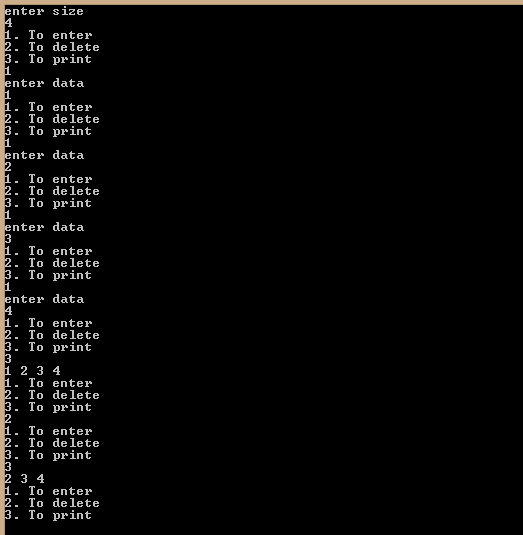
break;

}

}

}

**OUTPUT-**

****

***Experiment No-8***

**AIM:-** **Write a program to accept N numbers in an array, and then sort the array using Insertion Sort. Then accept a number from the user and insert it in the array according to the sequential order.**

**PROGRAM-**

#include<iostream>

using namespace std;

class insertion\_sort

{

int n,i,j,k,key;

int a[20];

public:

void getdata()

{

cout<<"Enter number of elements:"<<endl;

cin>>n;

cout<<"Enter elements:"<<endl;

for(i=0;i<n;i++)

{

cin>>a[i];

}

}

void sorting()

{

for(k=0;k<n-1;k++)

{

for(j=1;j<n;j++)

{

key=a[j];

for(i=j-1;i>=0;i--)

{

if(key<a[i])

a[i+1]=a[i];

else

break;

}

a[i+1]=key;

}

}

}

void display()

{

cout<<"elements are:"<<endl;

for(i=0;i<n;i++)

cout<<a[i]<<endl;

}

};

int main()

{

insertion\_sort is;

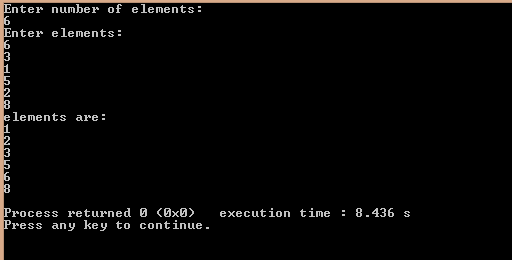
is.getdata();

is.sorting();

is.display();

}

**OUTPUT-**

****

***Experiment No-9***

**AIM:-** **Write a program to accept N numbers in an array, and then sort the array using Quick Sort.**

**PROGRAM-**

#include<iostream>

using namespace std;

int a[30];

int n, lb, loc, ub, left, right, temp, temp1;

void quicksort(int[10],int,int);

int pivot(int[],int,int);

int main()

{

cout<<"Enter size of array"<<endl;

cin>>n;

cout<<"Enter Array Elements "<<endl;

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

{

cin>>a[i];

}

quicksort(a,0,n-1);

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

{

cout<<" "<<a[i];

}

}

void quicksort(int a[], int lb, int ub)

{

int p;

if(lb<ub)

{

p=pivot(a,lb,ub);

quicksort(a,lb,p-1);

quicksort(a,p+1,ub);

}

}

int pivot( int a[],int lb,int ub )

{

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

{

cout<<" "<<a[i];

}

cout<<endl;

int left =lb;

int right = ub;

int loc =lb;

while((a[loc]<=a[right]) && (loc!=right))

{

right=right-1;

}

if(loc==right)

{

return loc;

}

temp=a[loc];

a[loc]=a[right];

a[right]=temp;

loc=right;

while((a[left]<=a[loc]) && (loc!=left))

{

left=left+1;

}

if(loc==left)

{

return loc;

}

temp1=a[loc];

a[loc]=a[left];

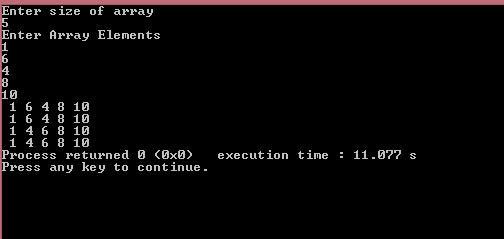
a[left]=temp1;

loc=left;

quicksort(a,0,n-1);

}

**OUTPUT-**

****

***Experiment No-10***

**AIM:-** Write a program to accept N numbers from the user in one array and M numbers in another array. Then, sort the arrays using Selection Sort and then merge these two arrays using Merge Sort.

**PROGRAM-**

#include <iostream>

using namespace std;

class sort {

public :

int \*arr;

int size;

void input() {

cin>>size;

arr = new int [size];

cout<<"Enter the elements of the array : ";

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

cin>>arr[i];

}

}

void selectionsort() {

int pivot;

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

pivot = i;

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

if(arr[j] < arr[pivot]) {

pivot = j;

}

}

if(pivot != i) {

int temp = arr[i];

arr[i] = arr[pivot];

arr[pivot] = temp;

}

}

}

void merge(int l, int m, int r, int \*aux) {

int i = l;

int j = m + 1;

int k;

for(k = l; k <= r; k++) {

if(i > m)

aux[k] = arr[j++];

else if (j > r)

aux[k] = arr[i++];

else if(arr[i] > arr[j])

aux[k] = arr[j++];

else

aux[k] = arr[i++];

}

for(k = l; k <= r; k++) {

arr[k] = aux[k];

}

}

void msort(int l, int r, int \*aux) {

if(l >= r) {

return;

}

int m = (l + r) / 2;

msort(l, m, aux);

msort(m + 1, r, aux);

merge(l, m, r, aux);

}

void mergesort(sort s1, sort s2) {

int i, j, k = 0;

size = s1.size + s2.size;

arr = new int [size];

for(i = 0; i < s1.size; i++) {

arr[k++] = s1.arr[i];

}

for(j = 0; j < s2.size; j++) {

arr[k++] = s2.arr[j];

}

static int aux[1000];

msort(0, size-1, aux);

}

void print() {

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

cout<<arr[i]<<" ";

}

cout<<endl;

}

};

int main() {

sort s1, s2, s3;

cout<<"Enter the size of 1st array : ";

s1.input();

cout<<"Enter the size of 2nd array : ";

s2.input();

s1.selectionsort();

s2.selectionsort();

s3.mergesort(s1, s2);

cout<<"1st array after selection sort : ";

s1.print();

cout<<"2nd array after selection sort : ";

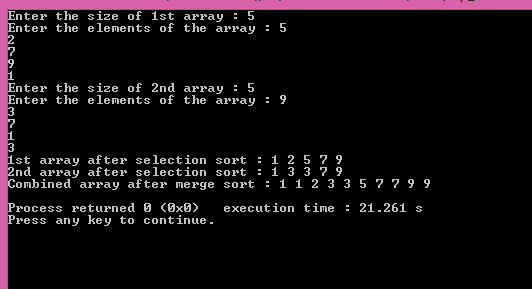
s2.print();

cout<<"Combined array after merge sort : ";

s3.print();

}

**OUTPUT-**

****

***Experiment No-11***

**AIM:-** Write a menu driven program that implements the following operations on a

Binary search tree :

* Insert a new element
* Delete an existing element
* Traversing the tree
* Pre-order Traversal
* In-order Traversal
* Post-order Traversal

**PROGRAM-**

#include<iostream>

using namespace std;

struct node

{

public:

int data;

node\* left;

node\* right;

int value;

};

class binary\_search\_tree

{

public:

node\* insertion(node \*root,int key)

{

if(root==NULL)

{

node \*nn=new node;

nn->data=key;

nn->left=NULL;

nn->right=NULL;

root=nn;

}

else if(key<root->data)

root->left=insertion(root->left,key);

else if(key>root->data)

root->right=insertion(root->right,key);

return root;

}

void print\_pre\_order(node \*t)

{

if(!t)

return;

else

{

cout<<t->data<<endl;

print\_pre\_order(t->left);

print\_pre\_order(t->right);

}

}

void print\_in\_order(node \*t)

{

if(!t)

return;

else

{

print\_in\_order(t->left);

cout<<t->data<<endl;

print\_in\_order(t->right);

}

}

void print\_post\_order(node \*t)

{

if(!t)

return;

else

{

print\_post\_order(t->left);

print\_post\_order(t->right);

cout<<t->data<<endl;

}

}

node\* deletenode(node \*root,int value)

{

if(root==NULL)

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

else if(root->data>value)

root->left=deletenode(root->left,value);

else if(root->data<value)

root->right=deletenode(root->right,value);

else

{

if(root->left==NULL)

{

node\* temp=root->right;

delete root;

return temp;

}

if(root->right==NULL)

{

node\* temp=root->left;

delete root;

return temp;

}

}

return root;

}

};

int main()

{

binary\_search\_tree bst;

node \*root=NULL;

int n,i;

cout<<"Enter number of elements in array:"<<endl;

cin>>n;

int a[n];

cout<<"Enter element you want in ll:"<<endl;

for(i=0;i<n;i++)

{

int key;

cin>>key;

root=bst.insertion(root,key);

}

cout<<"output according to IN order is:"<<endl;

bst.print\_in\_order(root);

cout<<"output according to POST order is:"<<endl;

bst.print\_post\_order(root);

cout<<"output according to PRE is:"<<endl;

bst.print\_pre\_order(root);

cout<<"Enter node you want to delete:"<<endl;

cin>>n;

node\* y=bst.deletenode(root,n);

cout<<"IN order output after deleting "<<endl;

bst.print\_in\_order(y);

}

**OUTPUT:**

****

***Experiment No.-12***

**AIM- Sort the list of integers using heap tree (Heap sort)**

**PROGRAM-**

#include<iostream>

#include<algorithm>

#include<stack>

using namespace std;

class heaps

{

int \*arr;

int n,left,right,largest;

public:

heaps()

{

n=0,left=0,right=0,largest=0;

cout<<"Enter the no of elements\n";

cin>>n;

arr=new int[n];

cout<<"Enter the elements\n";

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

{

cin>>arr[i];

}

}

void max\_heapify(int arr[],int k)

{

left=2\*k;

right=2\*k +1;

if(left<=n&&arr[left]>arr[k])

{

largest=left;

}

else

largest=k;

if(right<=n&&arr[right]>arr[largest])

{

largest=right;

}

if(largest!=k)

{

swap(arr[largest],arr[k]);

max\_heapify(arr,largest);

}

}

void create\_heap()

{

for(int i=n/2; i>=1; i--)

{

max\_heapify(arr,i);

}

}

void display()

{

cout<<"The heap is:\n";

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

cout<<arr[i]<<"\t";

}

void insert\_ele()

{

int key=0,i=0;

cout<<"Enter the element to be inserted\n";

cin>>key;

n=n+1;

arr[n]=key;

i=n;

while(i>1&&arr[i]>arr[i/2])// greater than 1 cz if ele to be inserted is greater than root, then after it comes to root position, then also comparison will continue, so to avoid extra comparison, take condition >1 and not >=1

{

swap(arr[i],arr[i/2]);

i/=2;

}

}

void heap\_sort()

{

stack<int>s;

while(n>0)

{

// cout<<arr[1]<<" ";

s.push(arr[1]);

arr[1]=arr[n];

n=n-1;

if(n==0)

break;

max\_heapify(arr,1);

}

cout<<"The sorted output is:\n";

while(!s.empty())

{

cout<<s.top()<<" ";

s.pop();

}

}

};

int main()

{

heaps h;

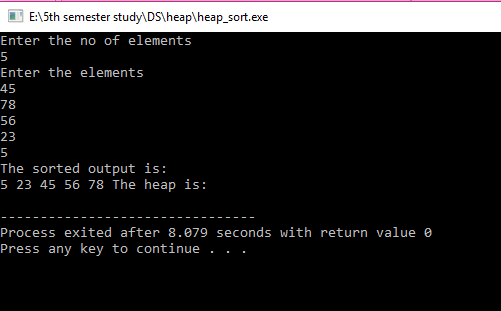
h.create\_heap();

h.heap\_sort();

h.display();

}

**OUTPUT:-**

****

***Experiment No.-12***

**AIM-**  **Program including all Operations on Graph and illustrate the traversals using DFS and BFS**

**PROGRAM-**

**DFS:**

#include <iostream>

#include <stack>

#include <algorithm>

using namespace std;

class dfs{

int \*A;

int arr[10][10];

int v;

public:

dfs()

{

int i,j;

cout<<"enter no. of vertices: ";

cin>>v;

A=new int[v];

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

{

A[i]=0;

}

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

{

for(j=1;j<=v;j++)

{

arr[i][j]=0;

}

}

}

void create(int ed)

{

int x,y;

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

{

cout<<"directed : ";

cin>>x>>y;

arr[x][y]=1;

}

}

void logic()

{

stack<int>s;

int source;

cout<<"enter source: ";

cin>>source;

A[source]=1;

s.push(source);

int res;

while(!s.empty())

{

res=s.top();

cout<<res<<" ";

s.pop();

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

{

if(arr[res][i]==1)

{

if(A[i]!=1)

{

A[i]=1;

s.push(i);

break;//to exit the loop;

}

}

}

}

}

}

int main()

{

dfs d;

int ed;

cout<<"enter edges: ";

cin>>ed;

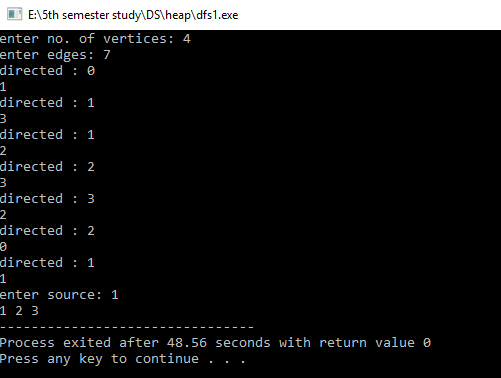
d.create(ed);

d.logic();

return 0;

}

**OUTPUT:-**

****

**BFS:-**

#include <iostream>

#include <queue>

#include <stack>

#include <algorithm>

using namespace std;

class bfs{

int v;

int \*A;

int arr[10][10];

int x,y;

public:

bfs()

{

cout<<"enter total vertices: ";

cin>>v;

int i,j;

A=new int[v];

//int \*arr=(int \*)malloc(v\*v\*sizeof(int));

for(i=0;i<v;i++)

{

A[i]=0;

}

for(i=0;i<v;i++)

{

for(j=0;j<v;j++)

{

arr[i][j]=0;//adjancey matrix;

}

}

}

void create(int ed)

{

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

{

cout<<"enter directed: ";

cin>>x>>y;

arr[x][y]=1;

}

}

void logic()

{

int source;

cout<<"enter source: ";

cin>>source;

queue<int>q;

A[source]=1;

q.push(source);

//cout<<source<<" ";

while(!q.empty())

{

int res=q.front();

cout<<res<<" ";

q.pop();

for(int i=0;i<v;i++)//to check the neighbors;

{

if(arr[res][i]==1)

{

if(A[i]!=1)

{

A[i]=1;

q.push(i);

}

}

}

}

}

};

int main()

{

bfs b;

int edge;

cout<<"enter no. of edges: ";

cin>>edge;

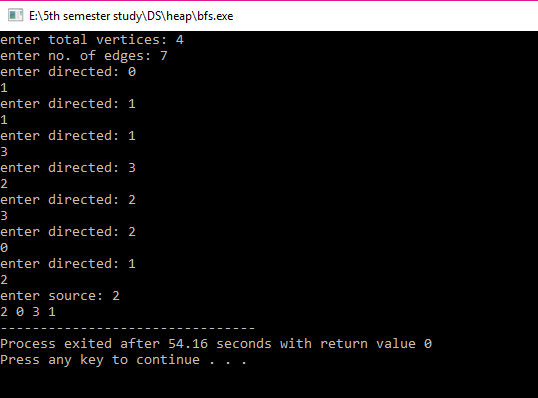
b.create(edge);

b.logic();

return 0;

}

**OUTPUT:-**

****