Q1 : Implement two stacks in an array.

#include<iostream>

using namespace std;

class stack

{

int \*arr;

int size;

int top1, top2;

public:

stack(int n)

{

size=n;

arr=new int[n];

top1=-1;

top2=size;

}

void push1(int x)

{

if(top1<top2-1)

{

top1++;

arr[top1]=x;

}

else

cout<<"Stack Overflow";

}

void push2(int x)

{

if(top1<top2-1)

{

top2--;

arr[top2]=x;

}

else

cout<<"stack overflow";

}

int pop1()

{

if(top1>=0)

{

int x=arr[top1];

top1--;

return x;

}

else

cout<<"Stack Underflow";

}

int pop2()

{

if(top2<size)

{

int x=arr[top2];

top2++;

return x;

}

else

cout<<"stack underflow";

}

};

int main()

{

stack s(5);

s.push1(2);

s.push1(4);

s.push1(6);

s.push2(8);

s.push2(10);

cout<<"Popped element from stack 1 is "<<s.pop1();

cout<<endl;

cout<<"Popped element from stack 2 is "<<s.pop2();

return 0;



Q2) Implement Queue using two stacks.

#include<iostream>

#include<stack>

using namespace std;

class queue

{

stack <int> s1,s2;

public:

void enqueue(int x)

{

while(!s1.empty())

{

s2.push(s1.top());

s1.pop();

}

s1.push(x);

while(!s2.empty())

{

s1.push(s2.top());

s2.pop();

}

}

int dequeue()

{

if(s1.empty())

{

cout<<"the queue is empty";

return -1;

}

int x=s1.top();

s1.pop();

return x;

}

};

int main()

{

queue q;

q.enqueue(1);

q.enqueue(2);

int k=q.dequeue();

if(k!=-1)

cout<<"Popped element is "<<k<<endl;

k=q.dequeue();

if(k!=-1)

cout<<"Value of popped second is "<<k<<endl;

k=q.dequeue();

return 0;

}



Q3) Implement sorted linked list.

#include<iostream>

using namespace std;

class node

{

public:

int data;

node\*next;

node(int d)

{

data=d;

next=NULL;

}

};

class lnklst

{

public:

node\*head;

lnklst()

{

head=NULL;

}

node\*search(int d)

{

node\*ptr=head;

while(ptr!=NULL)

{

if(ptr->data==d)

return ptr;

else if(ptr->data>d)

break;

else ptr=ptr->next;

}

cout<<"data not available"<<endl;

return NULL;

}

void insert(int d)

{

node\*n=new node(d);

if(head==NULL)

{

head=n;

head->next=NULL;

return;

}

node\*ptr=head;

node\*prev=NULL;

while(ptr!=NULL&&ptr->data<d)

{

prev=ptr;

ptr=ptr->next;

}

if(prev==NULL)

{

n->next=head;

head=n;

}

else

{

n->next=prev->next;

prev->next=n;

}

}

int del(int d)

{

int item;

if(head==NULL)

{

cout<<"empty"<<endl;

return -1;

}

node\*prev=NULL;

node\*ptr=head;

while(ptr!=NULL)

{

if(ptr->data==d)

{

if(prev==NULL)

{

item=head->data;

head=head->next;

delete ptr; //CHECK

return item;

}

else

{

item=ptr->data;

prev->next=ptr->next;

delete ptr;

return item;

}

}

if(ptr->data>d)

{

cout<<"item unavailable"<<endl;

return -1;

}

prev=ptr;

ptr=ptr->next;

}

}

void printll()

{

if(head==NULL)

return;

node\*ptr=head;

while(ptr!=NULL)

{

cout<<ptr->data<<" - >";

ptr=ptr->next;

}

}

};

int main()

{

lnklst l;

l.insert(1);

l.insert(3);

l.insert(2);

l.insert(0);

l.insert(5);

l.printll();

cout<<endl;

l.del(1);

l.del(5);

l.del(2);

l.printll();

return 0;

}



Q5) Implement quicksort using Doubly Linked List.

#include<iostream>

#include<stack>

using namespace std;

class Node

{

public:

int data;

Node \* next;

Node \* prev;

Node(int data)

{

this -> data = data;

this -> next = this -> prev = this;

}

};

Node\* head = NULL;

Node\* takeinput()

{

cout<<"enter the elements to be sorted"<<endl;

int data;

cin >> data;

while(data != -1)

{

Node \*newNode = new Node(data);

if(head == NULL)

{

Node \* newNode = new Node(data);

head = newNode;

}

else

{

Node \* newNode = new Node(data);

Node \* tail = head -> prev;

tail -> next = newNode;

newNode -> next = head;

newNode -> prev = tail;

head -> prev = newNode;

}

cin >> data;

}

return head;

}

Node \* swapTwoNodes(Node \* start, Node\* temp1, Node\* temp2)

{

if(temp1 == temp2)

return head;

if(temp1 -> next == temp2)

{

Node \* prev = temp1 -> prev;

Node \* next = temp2 -> next;

prev -> next = temp2;

temp2 -> prev = prev;

temp2 -> next = temp1;

temp1 -> prev = temp2;

temp1 -> next = next;

next -> prev = temp1;

if (start == temp1)

return temp2;

}

if(temp2 -> next == temp1)

{

Node \* prev = temp2 -> prev;

Node \* next = temp1 -> next;

prev -> next = temp1;

temp1 -> prev = prev;

temp1 -> next = temp2;

temp2 -> prev = temp1;

temp1 -> next = next;

next -> prev = temp2;

if (start == temp2)

return temp1;

}

Node \* prev1 = temp1 -> prev;

Node \* prev2 = temp2 -> prev;

Node \* next1 = temp1 -> next;

Node \* next2 = temp2 -> next;

prev1 -> next = temp2;

temp2 -> prev = prev1;

temp2 -> next = next1;

next1 -> prev = temp2;

temp1 -> next = next2;

prev2 -> next = temp1;

temp1 -> prev = prev2;

next2 -> prev = temp1;

if (start == temp2)

{

return temp1;

}

else if(start == temp1)

{

return temp2;

}

return start;

}

void swap\_data(Node\* temp1, Node\* temp2)

{

int temp = temp1 -> data;

temp1 -> data = temp2 -> data;

temp2 -> data = temp;

}

Node\* quick(Node\* head, Node\* beg, Node\* end)

{

Node\* left = beg;

Node\* right = end;

Node\* loc = left;

while(1)

{

while(loc -> data <= right -> data && loc != right)

right = right -> prev;

if(loc == right)

break;

if(loc -> data > right -> data)

{

swap\_data(loc, right);

loc = right;

}

while(loc -> data >= left -> data && loc != left)

left = left -> next;

if(loc == left)

break;

if(loc -> data < left -> data)

{

swap\_data(loc, left);

loc = left;

}

}

return loc;

}

void quickSort(Node\* head)

{

if(head == NULL || head -> next == head)

return;

stack<Node\*> upper;

stack<Node\*> lower;

lower.push(head);

upper.push(head -> prev);

while(!upper.empty() && !lower.empty())

{

Node\* beg = lower.top();

Node\* end = upper.top();

lower.pop();

upper.pop();

Node\* loc = quick(head, beg, end);

if(beg != loc)

{

lower.push(beg);

upper.push(loc -> prev);

}

if(end != loc && loc -> next != end)

{

lower.push(loc -> next);

upper.push(end);

}

}

}

void printL(Node\* head)

{

if(head == NULL)

return;

Node\* ptr = head;

cout<< ptr -> data << " ";

ptr = ptr -> next;

while(ptr != head)

{

cout << ptr -> data << " ";

ptr = ptr -> next;

}

}

int main()

{

Node \* head = takeinput();

quickSort(head);

printL(head);

return 0;

}



Q6) Evaluate infix arithmetic expression.

#include<iostream>

#include<stack>

using namespace std;

int exp\_eval(string);

int operation(int,int,char);

bool isoperator(char);

bool predence(char,char);

int main()

{

string exp;

int ans;

cout<<"Enter Infix Arithmetic Expression: ";

cin>>exp;

ans= exp\_eval(exp);

cout<<"The value of the Infix Arithmetic Expression is: "<<ans<<endl;

return 0;

}

int exp\_eval(string exp)

{

stack<int> opr; //stack1

stack<char> op; //stack2

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

{

if(isdigit(exp[i]))

{

int num=0;

while(isdigit(exp[i]))

{

num\*=10;

num+=exp[i]-'0';

i++;

}

opr.push(num);

i--;

}

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

op.push(exp[i]);

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

{

while(op.top()!='(')

{

char c=op.top();

op.pop();

int a=opr.top();

opr.pop();

int b= opr.top();

opr.pop();

int result= operation(a,b,c);

opr.push(result);

}

op.pop();

}

else if(isoperator(exp[i]))

{

while(!op.empty() && predence(op.top(),exp[i]))

{

char c=op.top();

op.pop();

int a=opr.top();

opr.pop();

int b= opr.top();

opr.pop();

int result= operation(a,b,c);

opr.push(result);

}

op.push(exp[i]);

}

}

while(!op.empty())

{

char c=op.top();

op.pop();

int a=opr.top();

opr.pop();

int b= opr.top();

opr.pop();

int result= operation(a,b,c);

opr.push(result);

}

return opr.top();

}

int operation(int a, int b, char c)

{

if(c=='+')

return b+a;

else if(c=='-')

return b-a;

else if(c=='\*')

return b\*a;

else if(c=='/')

return b/a;

else if(c=='^')

{

int value=1;

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

value\*=b;

return value;

}

return 0;

}

bool isoperator(char c)

{

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

return 1;

else return 0;

}

bool predence(char a, char b)

{

if( a=='^')

return 1;

else if(a=='/'||a=='\*'&& b!='^')

return 1;

else if(a=='+'||a=='-'&& b=='+'||b=='-')

return 1;

else

return 0;

}

