//#include <stack>

#include <iostream>

#include <conio.h>

#include <string.h>

#define MAX 20

using namespace std;

char stack[MAX];

int top = -1;

//Push Function

void push(char item){

top++;

stack[top]=item;

}

//Pop Function

char pop(){

char a;

a=stack[top];

top--;

return a;

}

//Function to analyze the precidence of operators

int prcd(char symbol){

switch(symbol) {

case '+':

case '-':

return 2;

case '\*':

case '/':

return 4;

case '(':

case ')':

case '#':

return 1;

}

}

//Function to sort operators from other data

int isoperator(char symbol){

switch(symbol) {

case '+':

case '-':

case '\*':

case '/':

case '(':

case ')':

return 1;

default:

return 0;

}

}

//Function to invert infix to prefix

void convertip(char infix[],char prefix[]){

int i,symbol,j=0;

char test[MAX];

infix=strrev(infix); //Used 'strrev' to reverse string

stack[++top]='#';

for(i=0;i<strlen(infix);i++){

symbol=infix[i];

if(isoperator(symbol)==0){

prefix[j]=symbol;

j++;

}

else{

if(symbol==')'){

push(symbol);

}

else if(symbol=='('){

while(stack[top]!=')'){

prefix[j]=pop();

j++;

}

pop();//pop out (.

}

else{

if(prcd(symbol)>prcd(stack[top])) {

push(symbol);

}

else{

while(prcd(symbol)<=prcd(stack[top])) {

prefix[j]=pop();

j++;

}

push(symbol);

}//end of else.

}//end of else.

}//end of else.

}//end of for.

while(stack[top]!='#'){

prefix[j]=pop();

j++;

}

prefix[j]='\0';//null terminate string.

prefix=strrev(prefix);

}

int main() {

char infix[20],prefix[20];

cout << "Enter the valid infix string: " << endl;

gets(infix);

convertip(infix,prefix);

cout << "The corresponding prefix string is: " << endl;

puts(prefix);

getch();

return 0;

}

…………………………..

#include<iostream>

using namespace std;

template <class T>

class Dynamicstack

{

private:

T \*data;

int top;

int size;

public:

Dynamicstack (int s=10): top (-1), size (s)

{

data = new T [size];

}

~Dynamicstack ()

{

delete []data;

}

void push (T);

T pop();

bool isEmpty();

bool isFull();

T topvalue();

};

template <class T>

void Dynamicstack <T> :: push (T element)

{

if (top==size-1)

{

cerr<<"Stack is full";

}

else

{

top++;

data [top]=element;

}

}

template <class T>

T Dynamicstack <T> :: pop()

{

if (top==-1)

{

cerr<<"List is epmty"<<endl;

}

else

{

T element = data [top];

top--;

return element;

}

}

template <class T>

bool Dynamicstack <T>:: isEmpty ()

{

return top==-1;

}

template <class T>

T Dynamicstack <T>:: topvalue()

{

return data[top];

}

template <class T>

bool Dynamicstack <T>:: isFull()

{

return top ==size -1;

}

……………………….

#include"Dynamicstack.h"

#include<iostream>

using namespace std;

int main()

{

Dynamicstack<int> S1;

int stacksize;

cout<<"Enter size of stack2 "<<endl;

cin>>stacksize;

S1.push(1);

S1.push(2); S1.push(3); S1.push(4); S1.push(5); S1.push(6);

cout<<S1.pop();

cout<<S1.pop();

Dynamicstack<char>S2 (15);

S2.push ('a');

cout<<S2.pop();

return 0;

}

……………………

#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;

/\*

\* Node Declaration

\*/

struct node

{

int info;

struct node \*next;

}\*start;

/\*

\* Class Declaration

\*/

class single\_llist

{

public:

node\* create\_node(int);

void insert\_begin();

void insert\_pos();

void insert\_last();

void delete\_pos();

void sort();

void search();

void update();

void reverse();

void display();

single\_llist()

{

start = NULL;

}

};

/\*

\* Main :contains menu

\*/

main()

{

int choice, nodes, element, position, i;

single\_llist sl;

start = NULL;

while (1)

{

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

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

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

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

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

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

cout<<"4.Sort Link List"<<endl;

cout<<"5.Delete a Particular Node"<<endl;

cout<<"6.Update Node Value"<<endl;

cout<<"7.Search Element"<<endl;

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

cout<<"9.Reverse Linked List "<<endl;

cout<<"10.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<<"Sort Link List: "<<endl;

sl.sort();

cout<<endl;

break;

case 5:

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

sl.delete\_pos();

break;

case 6:

cout<<"Update Node Value:"<<endl;

sl.update();

cout<<endl;

break;

case 7:

cout<<"Search element in Link List: "<<endl;

sl.search();

cout<<endl;

break;

case 8:

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

sl.display();

cout<<endl;

break;

case 9:

cout<<"Reverse elements of Link List"<<endl;

sl.reverse();

cout<<endl;

break;

case 10:

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

exit(1);

break;

default:

cout<<"Wrong choice"<<endl;

}

}

}

/\*

\* Creating Node

\*/

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;

}

}

/\*

\* Inserting element in beginning

\*/

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;

}

/\*

\* Inserting Node at last

\*/

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;

}

/\*

\* Insertion of node at a given position

\*/

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;

}

}

/\*

\* Sorting Link List

\*/

void single\_llist::sort()

{

struct node \*ptr, \*s;

int value;

if (start == NULL)

{

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

return;

}

ptr = start;

while (ptr != NULL)

{

for (s = ptr->next;s !=NULL;s = s->next)

{

if (ptr->info > s->info)

{

value = ptr->info;

ptr->info = s->info;

s->info = value;

}

}

ptr = ptr->next;

}

}

/\*

\* Delete element at a given position

\*/

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;

}

}

/\*

\* Update a given Node

\*/

void single\_llist::update()

{

int value, pos, i;

if (start == NULL)

{

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

return;

}

cout<<"Enter the node postion to be updated: ";

cin>>pos;

cout<<"Enter the new value: ";

cin>>value;

struct node \*s, \*ptr;

s = start;

if (pos == 1)

{

start->info = value;

}

else

{

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

{

if (s == NULL)

{

cout<<"There are less than "<<pos<<" elements";

return;

}

s = s->next;

}

s->info = value;

}

cout<<"Node Updated"<<endl;

}

/\*

\* Searching an element

\*/

void single\_llist::search()

{

int value, pos = 0;

bool flag = false;

if (start == NULL)

{

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

return;

}

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

cin>>value;

struct node \*s;

s = start;

while (s != NULL)

{

pos++;

if (s->info == value)

{

flag = true;

cout<<"Element "<<value<<" is found at position "<<pos<<endl;

}

s = s->next;

}

if (!flag)

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

}

/\*

\* Reverse Link List

\*/

void single\_llist::reverse()

{

struct node \*ptr1, \*ptr2, \*ptr3;

if (start == NULL)

{

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

return;

}

if (start->next == NULL)

{

return;

}

ptr1 = start;

ptr2 = ptr1->next;

ptr3 = ptr2->next;

ptr1->next = NULL;

ptr2->next = ptr1;

while (ptr3 != NULL)

{

ptr1 = ptr2;

ptr2 = ptr3;

ptr3 = ptr3->next;

ptr2->next = ptr1;

}

start = ptr2;

}

/\*

\* Display Elements of a link list

\*/

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;

}

…………………….

#include<iostream>

#include<string>

using namespace std;

template<class T>

class Queue

{

private:

T \*data;

int size;

int front;

int rear;

public:

Queue(int s = 10) :size(s), front(-1), rear(-1)

{

data=new T[size];

}

~Queue()

{

delete[]data;

}

void enqueue(T);

T dequeue();

T topValue();

bool isEmpty();

bool isFull();

};

template<class T>

void Queue<T>::enqueue(T element)

{

if (rear == size - 1)

{

cerr << "Queue is full";

}

else

{

rear++;

data[rear] = element;

if (front == -1)

{

front++;

}

}

}

template<class T>

T Queue<T>::dequeue()

{

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

{

cerr << "Queue is Empty";

}

else

{

T element = data[front];

if (rear == front)

{

rear = -1;

front = -1;

}

else

{

front++;

}

return element;

}

}

template<class T>

T Queue<T>::topValue()

{

return data[front];

}

template<class T>

bool Queue<T>::isEmpty()

{

return front == -1;

}

template<class T>

bool Queue<T>::isFull()

{

return rear == size - 1;

}

………………..

#include<iostream>

#include<string>

#include<process.h>

#include"q.h"

using namespace std;

int main()

{

int size;

cout << "Enter the size of the Queue. ";

cin >> size;

Queue<string> Q1;

// Queue<string> Q1(size);

Q1.enqueue("Data");

Q1.dequeue();

cout << Q1.dequeue();

//Queue<string> Queue;

system("pause");

}