**/\*WAP to evaluate a prefix expression\*/**

**#include <iostream>**

**#include <stack>**

**using namespace std;**

**void WriteRPNExpression(char expression[]);**

**void ReverseArray (char arr []);**

**bool isNumber(char &exp);**

**int main()**

**{**

**stack<int> s;**

**char expression [] = "/2\*+435";**

**WriteRPNExpression(expression);//display our rpn expression on the screen**

**cout<<endl;//newline space**

**int n;//a number ot push onto the stack**

**int result;//a result after performing**

**ReverseArray(expression);**

**for(unsigned int i = 0; i < 7; i++)**

**{**

**if(isNumber(expression[i])==true)**

**{**

**char c = expression[i];**

**n = c-'0';//parse the char to an integer**

**s.push(n);**

**}**

**if(expression[i]=='+')**

**{**

**int x = s.top();**

**s.pop();**

**int y = s.top();**

**s.pop();**

**result = x+y;**

**s.push(result);**

**}**

**if(expression[i]=='-')**

**{**

**int x = s.top();**

**s.pop();**

**int y = s.top();**

**s.pop();**

**result = y-x;**

**s.push(result);**

**}**

**if(expression[i]=='\*')**

**{**

**int x = s.top();**

**s.pop();**

**int y = s.top();**

**s.pop();**

**result = x\*y;**

**s.push(result);**

**}**

**if(expression[i]=='/')**

**{**

**int x = s.top();**

**s.pop();**

**int y = s.top();**

**s.pop();**

**result = y/x;**

**s.push(result);**

**}**

**}**

**cout<<"result of expression: "<<s.top();//result should be 17...**

**return 0;**

**}**

**void WriteRPNExpression(char arr [])**

**{**

**for(int i = 0; i < 7; i++)**

**{**

**cout<<arr[i];**

**}**

**}**

**void ReverseArray(char arr [])**

**{**

**int end = 6;**

**int start = 0;**

**char temp;**

**while(start < end)**

**{**

**temp = arr[start];**

**arr[start] = arr[end];**

**arr[end] = temp;**

**start++;**

**end--;**

**}**

**}**

**bool isNumber(char &n)//pass in a char reference**

**{**

**if(!isdigit(n))//check if the char is a number...**

**{**

**return false;**

**}**

**else**

**return true;**

**}**

**/\*WAP to evaluate a prefix expression\*///or**

**#include<iostream>**

**#include<string>**

**#include<cmath>**

**#define max 15**

**using namespace std;**

**template<class T>**

**class Stack**

**{**

**T data[max];**

**int top;**

**public:**

**Stack():top(-1) {}**

**void push(T value)**

**{**

**if(top==max-1)**

**{**

**cout<<"overflow"<<endl;**

**}**

**else**

**data[++top]=value;**

**}**

**T pop()**

**{**

**if(top==-1)**

**{**

**cout<<"underflow"<<endl;**

**}**

**else**

**{**

**return data[top--];**

**}**

**}**

**T peek()**

**{**

**if(top==-1)**

**{**

**cout<<"underflow"<<endl;**

**}**

**else**

**{**

**return data[top];**

**}**

**}**

**void display()**

**{**

**cout<<"------------------XX---------------"<<endl;**

**for(int i=top; i>-1; i--)**

**{**

**cout<<data[i]<<endl;**

**}**

**cout<<"------------------XX---------------"<<endl;**

**}**

**};**

**Stack<char>converter;**

**Stack<int>calculator;**

**// Switch cases for operator**

**int calculate\_result(int x,int y,char symbol)**

**{**

**switch(symbol)**

**{**

**case '+' :**

**return x+y;**

**case '-' :**

**return x-y;**

**case '\*':**

**return x\*y;**

**case '$':**

**return pow(x,y);**

**case '/':**

**return x/y;**

**}**

**return 0;**

**}**

**//evaluation of postfix expression**

**void calculate(string prefix)**

**{**

**int a,b;**

**int result=0;**

**string data,redata;**

**redata.clear();**

**for(int i=prefix.length() -1; i>=0; i--)**

**{**

**if(prefix[i] =='\*' || prefix[i] =='+' || prefix[i] =='-' || prefix[i] =='$'||prefix[i] =='/')**

**{**

**a=calculator.pop();**

**b=calculator.pop();**

**result=calculate\_result(a,b,prefix[i]);**

**calculator.push(result);**

**}**

**else**

**{**

**if (prefix[i]=='\_') {;}**

**else if (prefix[i-1] != '\_')**

**{**

**data+=prefix[i];**

**}**

**else**

**{**

**data+=prefix[i];**

**for(int y=data.length()-1; y>=0; y--)**

**{**

**redata+=data[y];**

**}**

**calculator.push(stoi(redata));**

**//cout<<data<<" changes to "<<redata<<endl;**

**data.clear();**

**redata.clear();**

**}**

**}**

**}**

**cout<<result<<endl;**

**}**

**//precision check**

**int precision\_check(char x)**

**{**

**if(x=='$')**

**{**

**return 3;**

**}**

**else if(x=='\*' || x=='/')**

**{**

**return 2;**

**}**

**else if(x=='+' || x=='-')**

**{**

**return 1;**

**}**

**else**

**{**

**return NULL;**

**}**

**}**

**//infix expression to postfix expression**

**string infix\_to\_Allupostfix(string expression)**

**{**

**string postfix;**

**char y;**

**converter.push('(');**

**for(auto x:expression)**

**{**

**if(x =='(')**

**{**

**converter.push(x);**

**} // if left bracket is encountered**

**else if(x == ')')**

**{**

**while(converter.peek() != '(')**

**{**

**y=converter.pop();**

**postfix+='\_';**

**postfix+=y;**

**}**

**converter.pop();**

**}**

**else if(x =='\*' || x =='+' || x =='-' || x =='$' || x=='/') //if operator is encounter**

**{**

**postfix+='\_';**

**if(converter.peek() =='(' )**

**{**

**converter.push(x);**

**} // if left bracket is at top**

**else if(precision\_check(x)>=precision\_check(converter.peek()))**

**{**

**converter.push(x);**

**} // if operator is at top**

**else**

**{**

**y=converter.pop();**

**postfix+=y;**

**converter.push(x);**

**}**

**}**

**else //if operand or character is encountered**

**{**

**postfix+=x;**

**}**

**}**

**return postfix;**

**}**

**//driver function**

**int main()**

**{**

**string expression;**

**string rev\_expression;**

**string prefixexp;**

**Stack<char>inverse; //for invering sting**

**cout<<"Enter your expression "<<endl;**

**getline(cin,expression);**

**for(auto x:expression)**

**{**

**inverse.push(x);**

**}**

**for(int i=0; i<expression.length(); i++) //for inversing the given expression**

**{**

**if(inverse.peek()==')')**

**{**

**inverse.pop();**

**rev\_expression+='(';**

**}**

**else if(inverse.peek()=='(')**

**{**

**inverse.pop();**

**rev\_expression+=')';**

**}**

**else**

**{**

**rev\_expression+=inverse.pop();**

**}**

**}**

**rev\_expression+=')';**

**cout<<rev\_expression<<endl;**

**string x=infix\_to\_Allupostfix(rev\_expression);**

**cout<<"Before inverse: "<<x<<endl;**

**for(auto i:x) //for inversing the postfix to prefix**

**{**

**inverse.push(i);**

**}**

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

**{**

**prefixexp+=inverse.pop();**

**}**

**cout<<"prefix expression"<<prefixexp<<endl;**

**calculate(prefixexp);**

**return 0;**

**}**