**/\*WAP to evaluate postfix expression using Stack\*/**

**#include <iostream>**

**#include <string.h>**

**using namespace std;**

**// Stack type**

**struct Stack**

**{**

**int top;**

**unsigned capacity;**

**int\* array;**

**};**

**// Stack Operations**

**struct Stack\* createStack( unsigned capacity )**

**{**

**struct Stack\* stack = (struct Stack\*) malloc(sizeof(struct Stack));**

**if (!stack) return NULL;**

**stack->top = -1;**

**stack->capacity = capacity;**

**stack->array = (int\*) malloc(stack->capacity \* sizeof(int));**

**if (!stack->array) return NULL;**

**return stack;**

**}**

**int isEmpty(struct Stack\* stack)**

**{**

**return stack->top == -1 ;**

**}**

**char peek(struct Stack\* stack)**

**{**

**return stack->array[stack->top];**

**}**

**char pop(struct Stack\* stack)**

**{**

**if (!isEmpty(stack))**

**return stack->array[stack->top--] ;**

**return '$';**

**}**

**void push(struct Stack\* stack, char op)**

**{**

**stack->array[++stack->top] = op;**

**}**

**// The main function that returns value of a given postfix expression**

**int evaluatePostfix(char\* exp)**

**{**

**// Create a stack of capacity equal to expression size**

**struct Stack\* stack = createStack(strlen(exp));**

**int i;**

**// See if stack was created successfully**

**if (!stack) return -1;**

**// Scan all characters one by one**

**for (i = 0; exp[i]; ++i)**

**{**

**// If the scanned character is an operand (number here),**

**// push it to the stack.**

**if (isdigit(exp[i]))**

**push(stack, exp[i] - '0');**

**// If the scanned character is an operator, pop two**

**// elements from stack apply the operator**

**else**

**{**

**int val1 = pop(stack);**

**int val2 = pop(stack);**

**switch (exp[i])**

**{**

**case '+':**

**push(stack, val2 + val1);**

**break;**

**case '-':**

**push(stack, val2 - val1);**

**break;**

**case '\*':**

**push(stack, val2 \* val1);**

**break;**

**case '/':**

**push(stack, val2/val1);**

**break;**

**}**

**}**

**}**

**return pop(stack);**

**}**

**// Driver program to test above functions**

**int main()**

**{**

**char exp[] = "231\*+9-";**

**cout<<"postfix evaluation: "<< evaluatePostfix(exp);**

**return 0;**

**}**

**//WAP to evaluate postfix expression using Stack**

**#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 postfix)**

**{**

**int a,b;**

**int result=0;**

**string data;**

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

**{**

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

**{**

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

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

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

**calculator.push(result);**

**}**

**else**

**{**

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

**else if (postfix[i+1] != '\_')**

**{**

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

**}**

**else**

**{**

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

**calculator.push(stof(data));**

**data.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**

**//12+24+45**

**//12\_24\_+\_45\_+**

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

**{**

**string postfix;**

**char y;**

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

**for(auto x:expression)**

**{**

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

**{**

**converter.push(x);**

**}**

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

**{**

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

**{**

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

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

**postfix+=y;**

**}**

**converter.pop();**

**}**

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

**{**

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

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

**{**

**converter.push(x);**

**}**

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

**{**

**converter.push(x);**

**}**

**else**

**{**

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

**postfix+=y;**

**converter.push(x);**

**}**

**}**

**else**

**{**

**postfix+=x;**

**}**

**}**

**//cout<<"test-1:"<<postfix<<endl;**

**calculate(postfix);**

**return postfix;**

**}**

**//driver function**

**int main()**

**{**

**string expression;**

**cout<<"Enter your expression eg:(10+20) :: "<<endl;**

**getline(cin,expression);**

**expression+=')';**

**string x=infix\_to\_postfix(expression);**

**//cout<<x<<endl;**

**}**