DATA STRUCTURES AND ALGORITHMS

LAB # 07

Hassaan Akbar Cheema

Regn no. # 174351

Instructor : Sir Abid Rauf

**Lab Tasks**

**Task 1:**

**Stack Operations**

1. void Push(element) – pushes an element on top of stack.
2. element Pop() – removes and displays the element on top of stack.
3. bool isEmpty() – checks if the stack is empty or not.
4. bool isFull() – checks if the stack is full or not.

5.. void Peek() – displays the contents of top element of stack.

**Code:**

**#include<iostream>**

**#define MAX 10**

**using namespace std;**

**int STACK[MAX], TOP = -1;**

**//check it is empty or not**

**int isEmpty(){**

**if (TOP == -1)**

**return 1;**

**else**

**return 0;**

**}**

**//check stack is full or not**

**int isFull(){**

**if (TOP == MAX - 1)**

**return 1;**

**else**

**return 0;**

**}**

**void push(int num){**

**if (isFull()){**

**cout << "STACK is FULL." << endl;**

**return;**

**}**

**++TOP;**

**STACK[TOP] = num;**

**cout << num << " has been inserted." << endl;**

**}**

**void display(){**

**int i;**

**if (isEmpty()){**

**cout << "STACK is EMPTY." << endl;**

**return;**

**}**

**for (i = TOP; i >= 0; i--){**

**cout << STACK[i] << " ";**

**}**

**cout << endl;**

**}**

**//pop - to remove item**

**void pop(){**

**int temp;**

**if (isEmpty()){**

**cout << "STACK is EMPTY." << endl;**

**return;**

**}**

**temp = STACK[TOP];**

**TOP--;**

**cout << temp << " has been deleted." << endl;**

**}**

**void peek(){**

**if (TOP != -1)**

**cout << "element on the top is : " << STACK[TOP] << endl;**

**};**

**int main(){**

**int num;**

**char ch;**

**do{**

**int a;**

**cout << "Choose \n1 to push an element in the stack\n" << "2 to pop an element out of the stack\n" << "3 to display the stack\n" << "4 to peek top element in the stack\n";**

**cout << "Please enter your choice: ";**

**cin >> a;**

**switch (a)**

**{**

**case 1:**

**cout << "Enter an Integer: ";**

**cin >> num;**

**push(num);**

**break;**

**case 2:**

**pop();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**peek();**

**break;**

**default:**

**cout << "An Invalid Choice :(\n";**

**}**

**cout << "Do you want to continue ? : Y for yes N for no : ";**

**cin >> ch;**

**} while (ch == 'Y' || ch == 'y');**

**return 0;**

**}**

**Task 2**

The idea is rather simple: You keep a Stack of braces, and every time you encounter an open brace, you push it into your stack. Every time you encounter a closing brace, you pop the top element from your stack. At the end, you check your stack for being empty. If so, indeed your input string contained balanced braces. Otherwise, it didn't.

**Expected Input**

1. 1 + 2 \* (3 / 4)
2. 1 + 2 \* [3 \* 3 + {4 – 5 (6 (7/8/9) + 10) – 11 + (12\*8)] + 14

3. 1 + 2 \* [3 \* 3 + {4 – 5 (6 (7/8/9) + 10)} – 11 + (12\*8) / {13 +13}] + 14

Your program will determine whether the open brackets (the square brackets, curly braces and the parentheses) are closed in the correct order.

**Expected Output of the above 3 inputs would be**

1. This expression is correct.
2. This expression is NOT correct. Error at character # 10. ‘{‘- not closed.
3. This expression is correct.

Solve the above problem using an **array based stack**.

**CODE:**

**#include"stdafx.h"**

**#include<iostream>**

**#include <string>**

**#define SIZE 50**

**using namespace std;**

**class STACK**

**{**

**private:**

**int element[SIZE];**

**int top;**

**public:**

**STACK() {**

**top = -1;**

**}**

**int isEmpty() {**

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

**return 1;**

**else**

**return 0;**

**}**

**int isFull() {**

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

**return 1;**

**else**

**return 0;**

**}**

**int push(int n) {**

**if (isFull() == 1) {**

**cout << "\nStack is FULL!\n";**

**return 0;**

**}**

**++top;**

**element[top] = n;**

**return n;**

**}**

**int pop() {**

**int temp;**

**if (isEmpty()) {**

**return 0;**

**}**

**temp = element[top];**

**--top;**

**return temp;**

**}**

**void displayItems() {**

**if (isEmpty()) {**

**cout << "\nStack is EMPTY!\n";**

**return;**

**}**

**int i;**

**cout << "STACK is: ";**

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

**cout << element[i] << " ";**

**cout << endl;**

**}**

**void peek() {**

**int p;**

**if (isEmpty()) {**

**cout << "\nStack is EMPTY!\n";**

**return;**

**}**

**else**

**p = element[top];**

**cout << "\nTop element is " << p << endl;**

**}**

**};**

**int main() {**

**int count1 = 0, count2 = 0, count3 = 0, count4 = 0, count5 = 0, count6 = 0, temp1, temp2, temp3;**

**string q;**

**STACK s;**

**do {**

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

**getline(cin, q);**

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

**if (q[i] == '(' || q[i] == '{' || q[i] == '[') {**

**if (q[i] == '(') {**

**temp1 = i + 1;**

**++count1;**

**}**

**if (q[i] == '{') {**

**temp2 = i + 1;**

**++count2;**

**}**

**if (q[i] == '[') {**

**temp3 = i + 1;**

**++count3;**

**}**

**s.push(q[i]);**

**}**

**if (q[i] == ')' || q[i] == '}' || q[i] == ']') {**

**if (q[i] == ')') {**

**++count4;**

**}**

**if (q[i] == '}') {**

**++count5;**

**}**

**if (q[i] == ']') {**

**++count6;**

**}**

**s.pop();**

**}**

**}**

**if (count1 != count4 || count2 != count5 || count3 != count6) {**

**if (count1 != count4) {**

**cout << "\nERROR,Expression is WRONG\n\n";**

**cout << "ERROR is in character # " << temp1 << endl;**

**}**

**else if (count2 != count5) {**

**cout << "\nERROR,Expression is WRONG\n\n";**

**cout << "ERROR is in character # " << temp2 << endl;**

**}**

**else if (count3 != count6) {**

**cout << "\nERROR,Expression is WRONG\n\n";**

**cout << "ERROR is in character # " << temp3 << endl;**

**}**

**}**

**else {**

**cout << "\nExpression is CORRECT\n\n";**

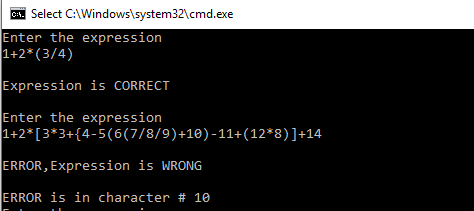
**}**

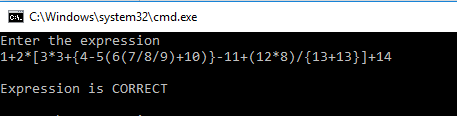
**} while (1);**

**return 0;**

**}**

**OUTPUT:**





**Task 3:**

**Evaluate the correct expression in Task 2 using stack data structure.**

**CODE:**

**#include<iostream>**

**#include<cstring>**

**#include<string.h>**

**#include<ctype.h>**

**#include<stack>**

**#include<math.h>**

**using namespace std;**

**// get weight of operators as per precedence**

**// higher weight given to operators with higher precedence**

**// for non operators, return 0**

**int getWeight(char ch) {**

**switch (ch) {**

**case '[':**

**case ']':**

**case '{':**

**case '}':**

**case '(':**

**case ')': return 4;**

**case '^':return 3;**

**case '/':**

**case '\*': return 2;**

**case '+':**

**case '-': return 1;**

**default: return 0;**

**}**

**}**

**// convert infix expression to postfix using a stack**

**void infix2postfix(char infix[], char postfix[], int size) {**

**stack<char> s;**

**int weight;**

**int i = 0;**

**int k = 0;**

**char ch;**

**// iterate over the infix expression**

**while (i < size) {**

**ch = infix[i];**

**if (ch == '('|| ch == '{' || ch == '[') {**

**// simply push the opening parenthesis**

**s.push('(');**

**i++;**

**continue;**

**}**

**if (ch == ')'|| ch == '}' || ch == ']') {**

**// if we see a closing parenthesis,**

**// pop of all the elements and append it to**

**// the postfix expression till we encounter**

**// a opening parenthesis**

**while (!s.empty() && s.top() != '(') {**

**postfix[k++] = s.top();**

**s.pop();**

**}**

**// pop off the opening parenthesis also**

**if (!s.empty()) {**

**s.pop();**

**}**

**i++;**

**continue;**

**}**

**weight = getWeight(ch);**

**if (weight == 0) {**

**// we saw an operand**

**// simply append it to postfix expression**

**postfix[k++] = ch;**

**}**

**else {**

**// we saw an operator**

**if (s.empty()) {**

**// simply push the operator onto stack if**

**// stack is empty**

**s.push(ch);**

**}**

**else {**

**// pop of all the operators from the stack and**

**// append it to the postfix expression till we**

**// see an operator with a lower precedence that**

**// the current operator**

**while (!s.empty() && s.top() != '(' &&**

**weight <= getWeight(s.top())) {**

**postfix[k++] = s.top();**

**s.pop();**

**}**

**// push the current operator onto stack**

**s.push(ch);**

**}**

**}**

**i++;**

**}**

**// pop of the remaining operators present in the stack**

**// and append it to postfix expression**

**while (!s.empty()) {**

**postfix[k++] = s.top();**

**s.pop();**

**}**

**postfix[k] = 0; // null terminate the postfix expression**

**}**

**// main**

**int main() {**

**stack<char> s;**

**const int max =100 ;**

**char infix[max];**

**cout << "Enter the Infix expression and add ','";**

**cout<< "after each multidigit and singledigit integer:: "<<endl;**

**cout << "for instance = (12,\*2,)/4, ::" ;**

**cin.getline(infix, max);**

**int size = strlen(infix);**

**char \* postfix;**

**postfix = new char[size];**

**infix2postfix(infix, postfix, size);**

**cout << "\nInfix Expression :: " << infix;**

**cout << "\nPostfix Expression :: " << postfix;**

**cout << endl;**

**for (int i = 0; i < strlen(postfix); i++) // for loop to traverse the string**

**{**

**int a = 0, b = 0;**

**switch (postfix[i]) // switch and case used to check for particular operator**

**{ // or operand in the default case and perform the push and**

**case '+': // pop instructions**

**b = s.top();**

**s.pop();**

**a = s.top();**

**s.pop();**

**s.push(a + b);**

**break;**

**case '-':**

**b = s.top();**

**s.pop();**

**a = s.top();**

**s.pop();**

**s.push(a - b);**

**break;**

**case '\*':**

**b = s.top();**

**s.pop();**

**a = s.top();**

**s.pop();**

**s.push(a\*b);**

**break;**

**case '/':**

**b = s.top();**

**s.pop();**

**a = s.top();**

**s.pop();**

**s.push(a / b);**

**break;**

**case '^':**

**b = s.top();**

**s.pop();**

**a = s.top();**

**s.pop();**

**s.push(pow(a,b));**

**break;**

**default:**

**int p = i + 1, q = 1;// algorithm to check the multi digit numbers**

**int num = 0; // and pushing them in the stack**

**while (postfix[p] != ',') {//',' is used here to check the number of //digits**

**q \*= 10;**

**p++;**

**}**

**p = i;**

**while (postfix[p] != ',') {**

**num += (postfix[p] - '0')\*q;**

**if (q > 1)**

**q = q / 10;**

**p++;**

**}**

**s.push(num);**

**i = p;**

**}**

**}**

**char a = s.top();**

**int c = (int)a;**

**cout << "RESULT OF POSTFIX EVALUATION = " << c << endl;**

**s.pop();**

**return 0;**

**}**

**OUTPUT:**

