Neh Samir Joshi

19BEC1112

07/01/2022

CSE2003

Data Structures and Algorithms

[LAB]

LAB – 1

Stacks

Aim: To learn about stacks, its basic operations and its implementation in C/C++.

Software Required: Code editor (e.g. VS Code, Dev C++), GCC/G++ compiler

Task 1: To create a stack using C++ array and perform push() and pop() operations:

Code:

```
#include <iostream>
using namespace std;
#define stack_size 5

int stack[stack_size], top = -1;

bool isFull(){
    return top==stack_size-1;
}
bool isEmpty(){
    return top==-1;
}
void push(int value){
    if (isFull()){
        cout << "Stack overflow!" << endl;
        return;
    }
```

```
    else {
    top++;
    stack[top] = value;
    }
}
void pop(){
    if (isEmpty()){
        cout << "Stack is already empty!" << endl;
        return;
    }
    else {
        stack[top] == 0;
        top--;
    }
}

void view(){
    for (int i=0; i<=top; i++){
        cout << "Element " << i << ": " << stack[i] << endl;
    }
}

int main(){
    cout << "Pushing 2,3,4,5,6 to stack (19BEC1112)" << endl;
    push(2);
    push(3);
    push(4);
    push(5);
    push(6);
    view();
    cout << "Popping one element from stack" << endl;
    pop();
    view();
    push(7);
    return 0;
}
```

Output:

```
PS C:\Users\OS\Desktop\6th Semester\CSE2003 Data Structures and Algorithms\Code\LAB1> cd "c:\Users\OS\Desktop\6th Semester\CSE2003 Da
ta Structures and Algorithms\Code\LAB1\" ; if ($?) { g++ stack.cpp -o stack } ; if ($?) { .\stack }
Pushing 2,3,4,5,6 to stack (19BEC1112)
Element 0: 2
Element 1: 3
Element 2: 4
Element 3: 5
Element 4: 6
Popping one element from stack
Element 0: 2
Element 1: 3
Element 2: 4
Element 3: 5
PS C:\Users\OS\Desktop\6th Semester\CSE2003 Data Structures and Algorithms\Code\LAB1>
```

Task 2: To convert an infix expression to a postfix expression using stack

Algorithm:

1. Iterate through the given string
2. If character is an operand, add it to the result string.
3. If the character is an open bracket, i.e. '(', push it to the operator stack.
4. If the character is a close bracket, i.e. ')', add all the operators of the stack to the result string till the stack becomes empty
5. Else, if the character is one of the arithmetic operators, add the top operator of the stack to the result string if its precedence is greater than that of the arithmetic operator, and repeat this till the stack is empty. After that, push the arithmetic operator to the stack.
6. Return the result string

Code:

```
#include <iostream>
using namespace std;
#define stack_size 100
//Code to create the stack
// ------------
char stack[stack_size];
int top = -1;

bool isFull(){
    return top==stack_size-1;
```

```
}
bool isEmpty(){
    return top==-1;
}
void push(char value){
    if (isFull()){
        cout << "Stack overflow!" << endl;
        return;
    }
    else {
    top++;
    stack[top] = value;
    }
}
void pop(){
    if (isEmpty()){
        cout << "Stack is already empty!" << endl;
        return;
    }
    else {
        stack[top] == '\0';
        top--;
    }
}
// ------
int calcPredidence(char c){ //Calculate precedence of each operator according to
BODMAS
    if (c=='^'){
        return 3;
    }
    else if(c=='*' || c=='/'){
        return 2;
    }
    else if (c=='+' || c=='-'){
        return 1;
    }
    else return -1;
}

string convert(string s){
    string result;
    for (int i=0; i<s.length(); i++){
        if (s[i] >= 'a' && s[i] <= 'z'){
            result += s[i];
        }
```

```
        else if (s[i]=='('){
            push(s[i]);
        }
        else if (s[i]==')'){
            while (!isEmpty() && stack[top]!='('){
                result += stack[top];
                pop();
            }
            if (!isEmpty()){
                pop();
            }
        }
        else {
            while (!isEmpty() &&
calcPredidence(stack[top])>calcPredidence(s[i])){
                result += stack[top];
                pop();
            }
            push(s[i]);
        }
    }
    while (!isEmpty()){
        result += stack[top];
        pop();
    }
    return result;
}

int main(){

    string s = "(a+b)*c";
    cout << "Conversion of: " << s << " to postfix is: " << convert(s) << endl;
    return 0;
}
```

Output:

```
PS C:\Users\OS\Desktop\6th Semester\CSE2003 Data Structures and Algorithms\Code\LAB1> cd
ta Structures and Algorithms\Code\LAB1\" ; if ($?) { g++ infix_postfix.cpp -o infix_postf
Conversion of: (a+b)*c to postfix is: ab+c*
PS C:\Users\OS\Desktop\6th Semester\CSE2003 Data Structures and Algorithms\Code\LAB1>
```

Task 3: Given a postfix expression, evaluate it and return the answer using stacks in C/C++.

Algorithm:

1. Iterate through the postfix string
2. Create a result variable of type double
3. If the character is an operand, push it to the stack
4. Else, if it is an operator, retrieve two of the stack operands, and depending on the operator, perform the operation on the operands and store it in the result variable.
5. Push the result to stack
6. Do this for all items in the string (continue the loop)
7. Return the result variable

Code:

```
#include <iostream>
using namespace std;
#define stack_size 100
//Code to create the stack
// ------------
double stack[stack_size];
int top = -1;

bool isFull()
{
    return top == stack_size - 1;
}
bool isEmpty()
{
    return top == -1;
}
void push(double value)
{
    if (isFull())
    {
        cout << "Stack overflow!" << endl;
        return;
    }
    else
```

```
    {
        top++;
        stack[top] = value;
    }
}
void pop()
{
    if (isEmpty())
    {
        cout << "Stack is already empty!" << endl;
        return;
    }
    else
    {
        stack[top] = 0;
        top--;
    }
}
// ------
double evaluate(string s)
{
    double result = 0;

    for (int i = 0; i < s.length(); i++)
    {
        if (s[i] >= '0' && s[i] <= '9')
        {
            push(s[i] - '0');
        }
        else
        {
            int op2 = stack[top];
            pop();
            int op1 = stack[top];
            pop();
            switch (s[i])
            {
            case '+':
                result = op1 + op2;
                break;
            case '-':
                result = op1 - op2;
                break;
            case '*':
                result = op1 * op2;
```

```
                break;
            case '/':
                result = op1 / op2;
                break;
            }
            push(result);
        }
    }
    return result;
}

int main()
{

    string s = "42/5+";
    cout << "Evaluation of postfix: " << s << " = " << evaluate(s) << endl;
    return '\0';
}
```

Output:

```
PS C:\Users\OS\Desktop\6th Semester\CSE2003 Data Structures and Algorithms\Code\LAB1> cd "c:\
ta Structures and Algorithms\Code\LAB1\" ; if ($?) { g++ evaluate_postfix.cpp -o evaluate_post
Evaluation of postfix: 42/5+ = 7
PS C:\Users\OS\Desktop\6th Semester\CSE2003 Data Structures and Algorithms\Code\LAB1>
```

Conclusion

Thus we have learnt how to implement stack and perform basic operations on it using C++. Hence the experiment is complete.

Neh Samir Joshi

19BEC1112