National University of Computer and Emerging Sciences

Lab Manual 07
Data Structures

Course Instructor	Miss. Arooj Khalil
Lab Instructor	Miss. Saira Arshad Miss Seemab Ayub
Section	BCS-3F2
Semester	Fall 2023

Department of Computer Science
FAST-NU, Lahore, Pakistan

TASK 1

Given an expression containing opening and closing braces, brackets, and parentheses. Implement a function “isBalanced” to check whether the given expression is a balanced expression
or not, using your stack implementation. bool isBalanced(char *&exp,const int &size)

For example:

- {[{}{}]}[()], {{}{}}, and []{}() are balanced expressions
- {()}[) and {(}) are not balanced expressions

In your main function test your function using the given examples.

TASK 2

Using the stack implement a function Infix2Postfix(char expression[]) that
converts the expression given in infix notation into postfix notation. You must use the char based stack for this problem.

Input: expression = "a+b*c".
Output: expression = abc*+
Input: “(AX*(BX*(((CY+AY)+BY)*CX)))”
Output: AXBXCYAY+BY+CX***
Input: “((H*((((A+((B+C)*D))*F)*G)*E))+J)”
Output: HABC+D*+F*G*E**J+

Instructions:

Steps to convert Infix expression to Postfix expression using Stack:

- Scan the infix expression from left to right.
- If the scanned character is an operand, output it.
- Else,
- If the precedence and associativity of the scanned operator are greater than the precedence and associativity of the operator in the stack(or the stack is empty or the stack contains a ‘(‘), then push it.
- ‘^’ operator is right associative and other operators like ‘+’,’-‘,’*’ and ‘/’ are left-associative. Check especially for a condition when both, operator at the top of the stack and the scanned operator are ‘^’. In this condition, the precedence of the scanned operator is higher due to its right associativity. So it will be pushed into the operator stack. In all the other cases when the

top of the operator stack is the same as the scanned operator, then pop the operator from the stack because of left associativity due to which the scanned operator has less precedence.

- Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

If the scanned character is an '(', push it to the stack. If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis. Repeat steps 2-6 until the infix expression is scanned. Print the output Pop and output from the stack until it is not empty.

TASK 3

You are tasked with implementing the functionality of a web browser's "Back" button using a stack data structure. Describe how you would use a stack to keep track of the user's navigation history and allow them to go back to the previously visited web pages.

Instructions:

When you're tasked with implementing the "Back" button in a web browser, you want to create a feature that allows users to go back to the previous web pages they've visited, just like you do in your favorite web browser.
To do this, you can use a special tool called a "stack." In programming, a stack is like a stack of plates, where you can only add or remove plates from the top. In this case, instead of plates, we're going to use the stack to keep track of the web pages the user visits.
Here's how it works:
1.Navigation History:
Every time a user visits a web page, you "push" that page onto the stack. Think of this as adding a new page to the top of the stack. So, if you visit Page A, it goes on the stack. Then, if you visit Page B, it goes on top of Page A, and so on.
2.Going Back:
When the user clicks the "Back" button, you "pop" the top page off the stack. This means you take the top page (the last one they visited) and show it to the user. They've effectively gone back to the previous page.
3.Remembering Forward:
As the user goes back, you don't want to forget where they were. So, you can use another stack to remember the pages they were on before they went back. This is like having a second stack for "forward" navigation.
4.Going Forward:
If the user decides they want to go forward after going back, you "pop" the top page from the forward stack and push it onto the main stack. This brings them back to where they were before going back.
So, in simple terms, you're using a stack to keep track of where the user has been, and you use

the "Back" button to take them to the previous page by popping it from the stack. If they want to go forward, you have another stack to help them do that.
This way, your web browser can keep track of the user's web page history and allow them to navigate back and forth through the pages they've visited. It's like having a stack of web pages, and you can pull pages from the top of the stack when they want to go back in time.