DHA Suffa University
Department of Computer Science
CS 2001L – Data Structures and Algorithms Lab
Spring 2021

Lab 07 – Stack

Objective:

To learn about

- Stack Operations
- Expression conversion

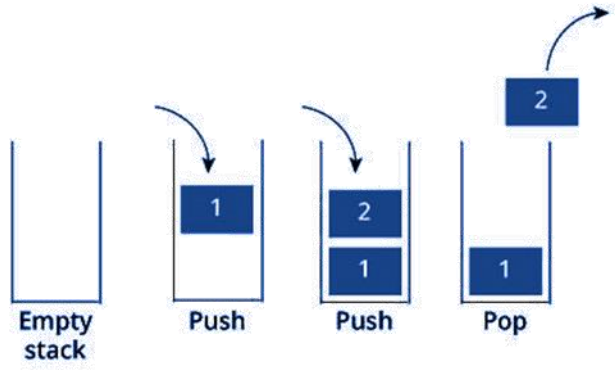
Stack

A stack is a list of elements in which an element may be inserted or deleted only at one end, called the top of the stack. This means, in particular, that elements are removed from a stack in the reverse order of that in which they were inserted into the stack.

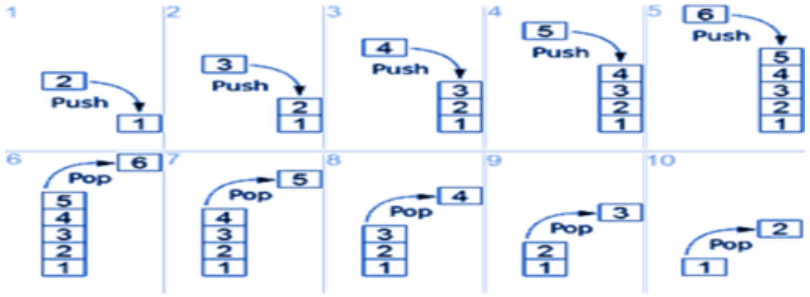
Special terminology is used for two basic operations associated with stacks:

- "Push" is the term used to insert an element into a stack.
- "Pop" is the term used to delete an element from a stack.

The operation of adding (Pushing) an item onto a stack and the operation of removing (Popping) an item from a stack may be implemented, respectively, by the following procedures, called PUSH and POP.



In executing the procedure PUSH, one must first test whether there is room in the stack for the new item; if not, then we have the condition known as overflow. Analogously, in executing the procedure POP, one must first test whether there is an element in the stack to be deleted; if not, then we have the condition known as underflow.



Following are the basic operations

Push(x): Insert element x at the top of a stack

```
push (int stack[ ] , int x) {
   isFull ( )
     print "Stack is full. Overflow condition!"
   else
      top = top +1
      stack[ top ] = x
end push
```

Pop( ): Removes an element from the top of a stack

```
pop (int stack[ ] )
   if isEmpty ( )
     print "Stack is empty. Underflow condition! "
   else
     top = top - 1
end pop
```

Top ( ) or Peek ( ): Access the top element of a stack

```
topElement( )
      return stack[ top ]
```

isEmpty ( ): Check whether a stack is empty

```
isEmpty ( )
      if ( top == -1 )
        return true
        else
        return false
```

isFull ( ): Check whether a stack is full

```
isFull ( )
      if ( top == Size-1 )
        return true
      else
        return false
```

Stack Operations using Array

A stack can be implemented using array as follows:

Before the implementation of actual operations, first follow the steps mentioned below 0 to create an empty stack.

Step 1: Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 2: Declare all the functions used in stack implementation.

Step 3: Create a one dimensional array with fixed size (int stack[SIZE])

Step 4: Define an integer variable 'top' and initialize with '-1'. (int top = -1)

Step 5: In main method display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

Stack using Linked List

The major problem with the stack implemented using array is, it works only for fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using linked list data structure. The stack implemented using linked list can work for unlimited number of values.

In linked list implementation of a stack, every new element is inserted as 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its next node in the list. The next field of the first element must be always NULL.

Operations

To implement stack using linked list, we need to set the following things before implementing actual operations.

Step 1: Include all the header files which are used in the program. And declare all the user defined functions.

Step 2: Define a 'Node' class with two members, data and next.

Step 3: Define a Node pointer 'top' in LinkedList class and set it to NULL.

Step 4: Define following methods for insertion and deletion in stack, with consideration of the Last in First out (LIFO) manner.

- insertFirst method: this method will add/push element on top of the stack.
- deleteFirst method: this method will remove and return the top of the stack.

Step 5: Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.

Expression Conversion

Infix expression: The expression of the form a op b. When an operator is in-between every pair of operands.

Postfix expression: The expression of the form a b op. When an operator is followed for every pair of operands.

Infix to Postfix conversion:

The compiler scans the expression either from left to right or from right to left.

Consider the below expression: a + b * c + d

The compiler first scans the expression to evaluate the expression b * c, then again scan the expression to add a to it. The result is then added to d after another scan.

The repeated scanning makes it very in-efficient. It is better to convert the expression to postfix(or prefix) form before evaluation.

The corresponding expression in postfix form is: abc*+d+. The postfix expressions can be evaluated easily using a stack.

Algorithm:

1. Read the element from the input.

2. If it is an operand, print it.

3. If it is an opening parenthesis, push it on stack.

4. If it is an operator then

   i. If the stack is empty, push the operator on stack.
   ii. If the top is opening parenthesis, then push the operator on stack.
   iii. If it has higher precedence than the top of the stack, then push it on stack.
   iv. If it has lower precedence than the top of the stack, pop and print the top. Then, test the operator against the new top of the stack.
   v. If it has equal precedence to the top of the stack, use associativity rule.

      a. If associativity is L to R, then pop and print the top of the stack and then push the incoming operator.
      b. If associativity is R to L, then push the incoming operator.

5. If closing parenthesis, pop operators from stack and output them until opening parenthesis is encountered. Pop and discard the opening parenthesis.

6. If there are more elements in input, go to step 1.

7. If there are no more elements in input, pop the remaining elements from stack.

LAB ASSIGNMENT

1. Implement stack using Arrays. Make proper functions for each operation.

2. Implement stack using Linked list. Make proper functions for each operation.

3. Create a program for conversion of infix expressions to postfix expressions using Stacks.

SUBMISSION GUIDELINES

- Take a screenshot of each task (code and its output), labeled properly.
- Place all the screenshots and the code files (properly labeled) in a single folder labeled with Roll No and Lab No. e.g. **'cs191xxx_Lab01'**.
- Submit the folder at LMS
- **-100%** policies for plagiarism.