National University of Sciences & Technology (NUST)
School of Natural Sciences (SNS)
Department of Mathematics

CS250: Data Structures and Algorithm

Class: BS2k21_Mathematics – Fall 2023

Lab 07:

Implementation of Stacks ADT and its applications

Date: 25th October, 2023

Time: 10:00 am - 01:00 pm

Instructor: Fauzia Ehsan

Submission Deadline: 31st October, 2023

National University of Sciences & Technology (NUST)
School of Natural Sciences (SNS)
Department of Mathematics

Lab 07: Implementation of Stacks in different problems

Introduction
This lab will introduce students with the practical implementation of a stacks ADT and to work on some of its applications

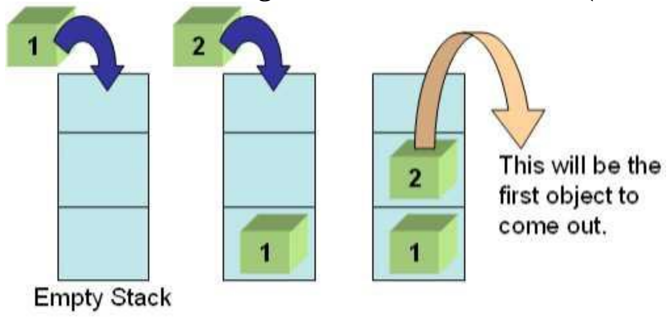
Objectives
Objective of this lab is to enable students to build stack ADT using linked list and arrays, perform the following tasks on it and analyze the performance of each implementation.

Tools/Software Requirement
Python 3/ PyCharm IDE / VS Code

Contents

Stack Operations 3
Lab Tasks 3
Task 1. Expression Parsing 3
Task 02 4
Task 03. Transform the Expression 4
Example 4
Task 04. `Divide by 2 Algorithm` 5
Figure 1. Decimal to Binary Conversion 5
Grade Criteria 6

Description:
A stack meant to mimic the information storage and retrieval in **LIFO (Last In First Out)** order.



Stack Operations

1. **`void Push(element) - pushes an element on the top of stack`**
2. **`element Pop() - removes and display the element on the 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 Clear() - release the memory allocated by stack`**
6. **`void Peek()`** – display the contents of the top element of stack

Lab Tasks

Task 1. Expression Parsing

Write a program that reads in a sequence of characters, and determines whether its parentheses, braces, and curly braces are "balanced."
Hint: for left delimiters, push onto stack; for right delimiters, pop from stack and check whether popped element matches right delimiter.
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 close 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

1. This expression is correct.
2. This expression is NOT correct e.g. error at character # 10. '{'- not closed.
3. This expression is correct.
   Your program should be able to take generic input expression from user. Solve the above problem using an **array based stack**.

Task 02

Alice is rearranging her library. She takes the innermost shelf and reverses the order of books. She breaks the walls of the shelf. In the end, there will be only books and no shelf walls. Print the order of books.

Opening and closing walls of shelves are shown by `'/'` and `'\'` respectively whereas books are represented by lower case alphabets.

Input format: The first line contains string S displaying her library.

Output format: Print only one string displaying Alice's library after rearrangement.

Constraints: $2 \le |s| \le 10^3$

Note:

The first character of the string is `'/'` and the last character of the string is `'\'` indicating outermost walls of the shelf.

Sample Input: `/books/love\i\`

Sample Output: `ilovebooks`

Task 03. Transform the Expression

Transform the algebraic expression with brackets into RPN form (Reverse Polish Notation). Two-argument operators: +, -, *, /, ^ (priority from the lowest to the highest), brackets ( ). Operands: only letters: a,b,...,z.

Input

```
t [the number of expressions <= 100]
expression [length <= 400]
[other expressions]
```

Text grouped in [ ] does not appear in the input file.

Output: The *expressions* in RPN form, one per line.

Example

Input:	Output:
3	abc*+
(a+(b*c))	ab+zx+*
((a+b)*(z+x))	at+bac++cd+^*
((a+t)*((b+(a+c))^(c+d)))	

Task 04. `Divide by 2 Algorithm`

Write a program that reads in a positive integer and prints the **binary representation** of that integer using stacks. Hint: divide the integer by 2

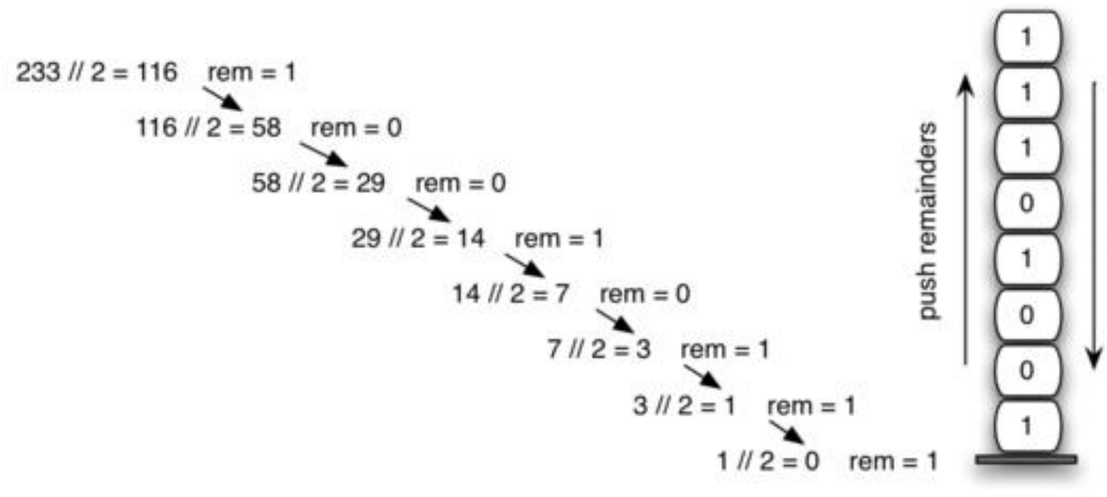


Figure 1. Decimal to Binary Conversion

Description:

Integer values are common data items. They are used in computer programs and computation all the time. We learn about them in math class and of course represent them using the decimal number system, or base 10. The decimal number 233_{10} and its corresponding binary equivalent 11101001_2 are interpreted respectively as

$2\times10^2+3\times10^1+3\times10^0$

and

$1\times2^7+1\times2^6+1\times2^5+0\times2^4+1\times2^3+0\times2^2+0\times2^1+1\times2^0$

But how can we easily convert integer values into binary numbers? The answer is an algorithm called **"Divide by 2" that uses a stack to keep track of the digits for the binary result.**

The **Divide by 2 algorithm** assumes that we start with an integer greater than 0. A simple iteration then continually divides the decimal number by 2 and keeps track of the remainder. The first division by 2 gives information as to whether the value is even or odd. An even value will have a remainder of 0. It will have the digit 0 in the ones place. An odd value will have a remainder of 1 and will have the digit 1 in the ones place. We think about building our binary number as a sequence of digits; the first remainder we compute will actually be the last digit in the sequence. As shown in Figure 1, we again see the reversal property that signals that a stack is likely to be the appropriate data structure for solving the problem.

Grade Criteria

This lab is graded. Min marks: 0. Max marks: 30

Tasks Shown during Lab (At least 2 Tasks must be shown to LE during lab working) 10 marks	Modern Tool Usage 5 marks	Lab Ethics 5 Marks	Lab Report and Tasks Final Submission 5 marks	Lab Viva/Quiz 5 Marks
Total Marks: 30	Obtained Marks: _____			

Happy Coding!

Deliverables

Comment your program heavily. Intelligent comments and a clean, readable formatting of your code account for 20% of your grade.

You should submit your codes and report as a compressed zip file. It should contain all files used in the exercises for this lab.

The submitted file should be named cs250_firstname_lastname_lab07.zip