|  |  |  |
| --- | --- | --- |
| **American University of Sharjah**  **School of Engineering**  ­Computer Engineering Department  P. O. Box 26666 Sharjah, UAE |  | **Instructor:** Dr. Ghassan Z. Qadah  **Lab Instructor:** Ms.Praveena Kolli  **Office**: EB2-126  **Phone**: 971-6-515-2352  **e-mail**: pkolli@aus.edu  **Semester**: Spring 2018 |

**CMP305L Data Structures and Algorithms**

**Lab #2 – Stack implementation and applications**

***Note: Exercise 1* and *Exercise* 2 are on dynamic array based Stack ADT**

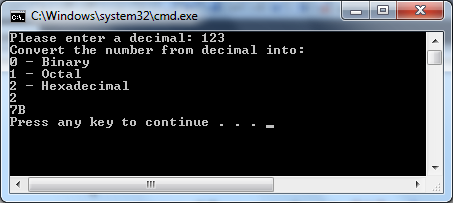
***Exercise 1:***

Add the following functions to the given *dynamic array* *Stack* class (provided in DynamicArrayStack folder):

* + void clear() Removes all items of the stack
  + Overload the assignment operator(=)
* Overload the insertion operator “<<”(to print all items as if they are popped out from the stack. Make sure that the original content of the stack is not disturbed. )

***Exercise 2:***

Write a program that reads in a positive integer and converts into one of the user chosen number system. Give user three options: binary, octal and Hex. You should use stack ADT to store the digits that are converted.



**Note: Exercise 3 and Exercise 4 are on linked structure based Stack ADT**

***Exercise 3:***

Add the following member function to the given *Linked Stack* class (provided in LinkedStack folder):

* bool hasDuplicate(), which returns true if the queue contains a duplicate value, false otherwise.

***Exercise 4:***

One of the *Stack* data structure usages is to evaluate the values of postfix mathematical expressions. Postfix notation is a parenthesis-free way of writing arithmetic expressions, where one places the operator symbol *after* the operator's two operands. For example, the addition of 3 to 2 is written 3 2 +, and the multiplication of the result by 4 is written as 3 2 + 4 \*. Remarkably, *parentheses are never needed*. An example like

((3 + 2) \* 4) / (5 - 1)

is written

3 2 + 4 \* 5 1 - /

and gets manually computed as follows:

3 2 + 4 \* 5 1 - /

=> 5 4 \* 5 1 - /

=> 20 5 1 - /

=> 20 4 /

=> 5

We see that an operator evaluates with the two operands that immediately precede it. This explains why the division operator is written last in the original expression, because the division is performed only after all the other sub expressions are evaluated. To automate the computation of a postfix expression, one may use a stack data structure as follows:

Stack Expression

| |

--- 3 2 + 4 \* 5 1 - /

(empty)

| 3 | 2 + 4 \* 5 1 - /

---

| 2 |

| 3 | + 4 \* 5 1 - /

---

| 5 | 4 \* 5 1 - /

---

| 4 |

| 5 | \* 5 1 - /

---

| 20| 5 1 - /

---

| 5 |

| 20| 1 - /

---

| 1 |

| 5 |

| 20| - /

---

| 4 |

| 20| /

---

| 5 | (finished)

---

Implement an application program that reads a postfix expression from the user and uses a stack to compute its value and returns it to the user. The expression will be assumed to contain integers and the following binary arithmetic operators, namely, + (addition), - (subtraction), \* (multiplication) and / (division).

**Note:** Your program must also handle*integers* that have *mor****e*** than *one digit.*