**Object Oriented Programming**

**Lab report: 13**



|  |  |
| --- | --- |
| Name | Tayyaba |
| Reg no | FA21-BCE-093 |
| Class | BCE- 4 |
| Instructor’s Name | Prof. Sadaf Iqbal |

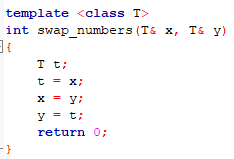
**Lab 13 – *Templates***

**Lab Tasks**

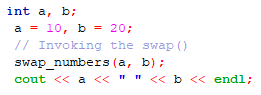
**5.1.** Write a program to swap the contents of two variables. Use template variables as function arguments**.**

**Program:**

**Classes:**



**Main:**



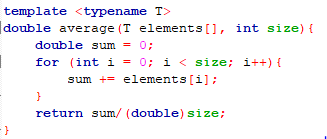
**Output:**



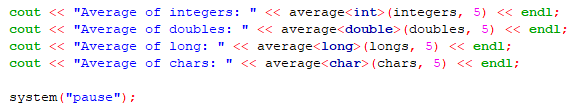
**5.2.** Write a template function that returns the average of all the elements of an array. The arguments to the function should be the array name and the size of the array (type *int*). In main ( ), exercise the function with arrays of type *int*, *long*, *double*, and *char*.

**Program:**

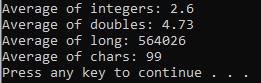
**Classes:**



**Main:**



**Output:**



**Home Task:**

**6.1.** Conceptually, the stack class or abstract data type mimics the information kept in a pile on a desk. Informally, we first consider a material on a desk, where we may keep separate piles for bills that need paying, magazines that we plan to read, and notes we

have taken. These piles of materials have several properties. Each pile contains some papers (information). In addition, for each pile, we can do several tasks:

 Start a new pile,

 Place new information on the top of a pile,

 Take the top item off of the pile,

 Read the item on the top, and

 Determine whether a pile is empty. (There may be nothing at the spot where the pile should be.)

These operations allow us to do all the normal processing of data at a desk. For example, when we receive bills in the mail, we add them to the pile of bills until payday comes. We then take the bills, one at a time, off the top of the pile and pay them until the money runs out.

When discussing these operations it is customary to call the addition of an item to the top of the pile a *Push* operation and the deletion of an item from the top a *Pop* operation.

More formally, a *stack* is defined as a class that can store data and that has the following operations:

 *Make-Stack* Create a new, empty stack object.

 *Empty Empty* returns true or false (1 or 0), depending upon whether the stack contains any items or not.

 *Push Push* adds the specified item to the top of the stack.

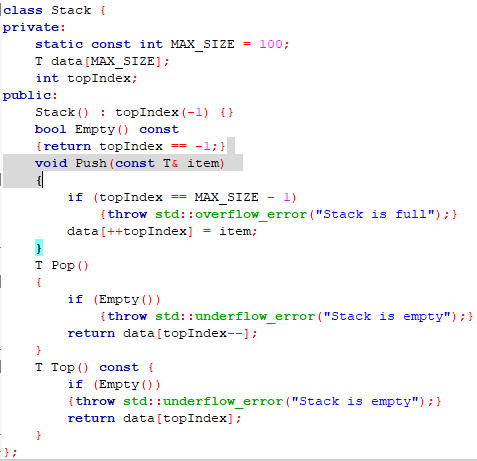
 *Pop* If the stack is not empty, *Pop* removes the top item from the stack, and this item is returned. If the stack is empty, nothing is returned, and an error is reported.

 *Top* If the stack is not empty, the top item is returned, but the contents of the stack are not changed. If the stack is empty, nothing is returned, and an error is reported.

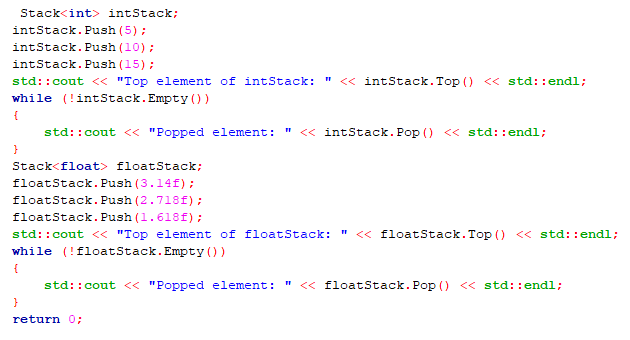
Write a program to create template-based stack. Store *int* and *float* in it.

**Program:**

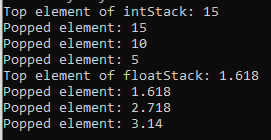
**Classes:**



**Main:**



**Output:**



**Critical analysis:**

In this lab, we understand the concept of templates and how it is efficient for using the same code for different data types. It works at compile time. We know that using a template we can create a single function that can process any type of data i.e. the formal arguments of functions are of generic type. They can accept data of any type such as int, float, long etc.