Question # 01:

A **stack** is a data structure that can be logically thought as linear structure represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called **top** of the stack. Every time an element is added, it goes on the **top** of the stack and the only element that can be removed is the element that is at the top of the stack.

Your task is to define a class **stack** that can be of maximum 6 elements. Initialize the stack with default construct, also provide following operations.

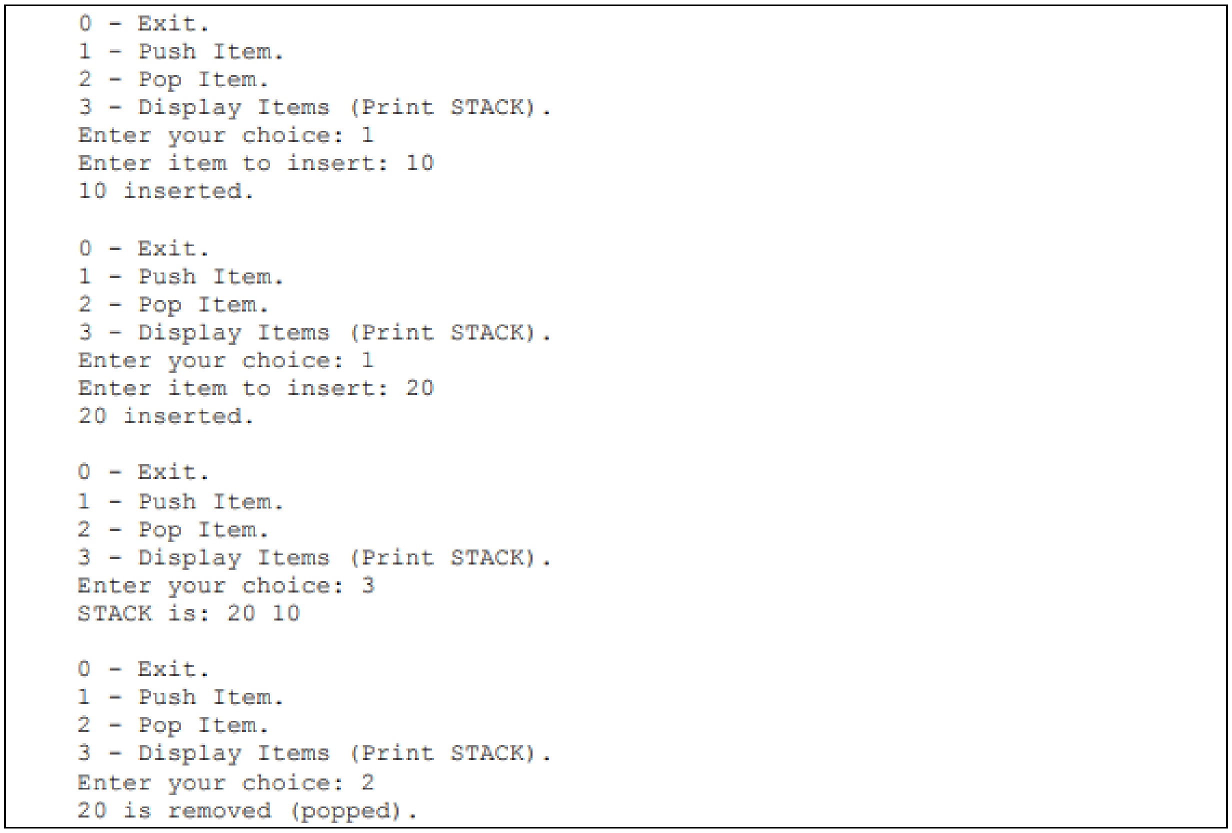
Check_Empty() : used to check if the stack is empty

Check_Full() : used to check if the stack is full.

push(): used to insert new elements into the stack. You cannot perform push operation if the stack is full.

pop(): used to remove an element from the stack. You cannot perform pop operation if the stack is empty.

Display() : used to display stack items. Write a program that provides a menu to test the class stack and its functions. Sample output:



```
0 - Exit.
1 - Push Item.
2 - Pop Item.
3 - Display Items (Print STACK).
Enter your choice: 1
Enter item to insert: 10
10 inserted.

0 - Exit.
1 - Push Item.
2 - Pop Item.
3 - Display Items (Print STACK).
Enter your choice: 1
Enter item to insert: 20
20 inserted.

0 - Exit.
1 - Push Item.
2 - Pop Item.
3 - Display Items (Print STACK).
Enter your choice: 3
STACK is: 20 10

0 - Exit.
1 - Push Item.
2 - Pop Item.
3 - Display Items (Print STACK).
Enter your choice: 2
20 is removed (popped).
```

Question # 01:

A **queue** is a data structure that can be logically thought as linear structure represented by a real line of persons or row, a structure where insertion and deletion of items takes place at different end called **rear** and **front** of the queue. The element added first in the queue will be the one to be removed first. Elements are always added to the back or **rear** and removed from the **front**.

Your task is to define a class **queue** that can be of maximum 6 elements. Initialize the queue with default construct, also provide following operations.

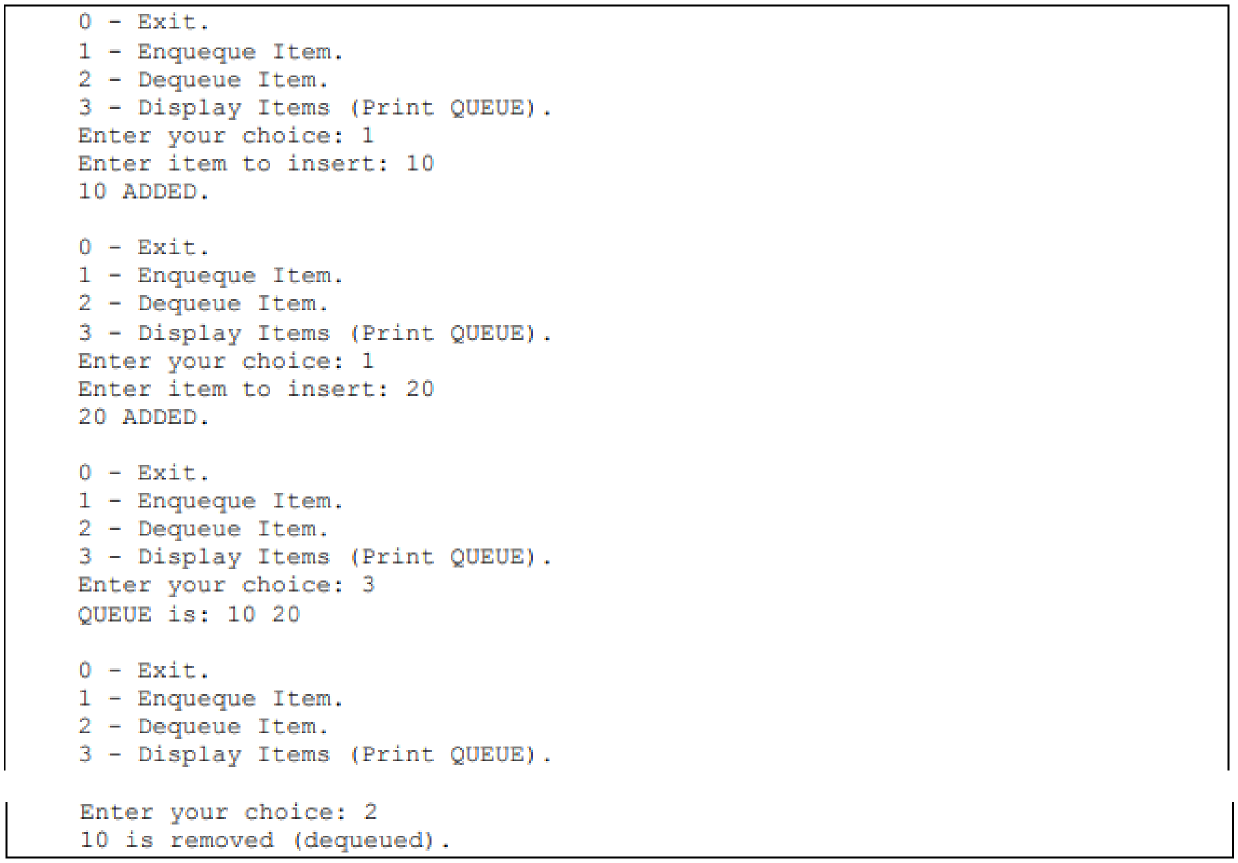
Check_Empty() : used to check if the queue is empty

Check_Full() : used to check if the queue is full.

Enqueue(): Adds an element to the back of the queue if the queue is not full otherwise it will print "OverFlow"

Dequeue(): Removes the element from the front of the queue if the queue is not empty otherwise it will print "UnderFlow".

Display() : Display the elements of queue. Write a program that provides a menu to test the class queue and its functions.



```
0 - Exit.
1 - Enqueque Item.
2 - Dequeue Item.
3 - Display Items (Print QUEUE).
Enter your choice: 1
Enter item to insert: 10
10 ADDED.

0 - Exit.
1 - Enqueque Item.
2 - Dequeue Item.
3 - Display Items (Print QUEUE).
Enter your choice: 1
Enter item to insert: 20
20 ADDED.

0 - Exit.
1 - Enqueque Item.
2 - Dequeue Item.
3 - Display Items (Print QUEUE).
Enter your choice: 3
QUEUE is: 10 20

0 - Exit.
1 - Enqueque Item.
2 - Dequeue Item.
3 - Display Items (Print QUEUE).

Enter your choice: 2
10 is removed (dequeued).
```

Question # 01

A supermarket chain has asked you to develop an automatic checkout system. All products are identifiable by means of a barcode and the product name. Groceries are either sold in packages or by weight. Packed goods have fixed prices. The price of groceries sold by weight is calculated by multiplying the weight by the current price per kilo.

Develop the classes needed to represent the products first and organize them hierarchically. The Product class, which contains generic information on all products (barcode, name, etc.), can be used as a base class.

The Product class contains two data members for storing barcodes and the product name. Define a constructor with parameters for both data members. Add default values for the parameters to provide a default constructor for the class. In addition to the access methods setCode() and getCode(), also define the methods scanner() and printer(). For test purposes, these methods will simply output product data on screen or read the data of a product from the keyboard.

The next step involves developing special cases of the Product class. Define two classes derived from Product, PrepackedFood and Fresh- Food. In addition to the product data, the PrepackedFood class should contain the unit price and the FreshFood class should contain a weight and a price per kilo as data members. In both classes define a constructor with parameters providing default-values for all data members. Use both the base and member initializer. Define the access methods needed for the new data members.Also redefine the methods scanner() and printer() to take the new data members into consideration.

Test the various classes in a main function that creates three objects each of the types Product, PrepackedFood and FreshFood. One object of each type is fully initialized in the object definition. Use the default constructor to create the other object and test the scanner() method. For the third object, test the get and set methods. Display the products on screen for all objects.

Question # 01:

A complex number is a number in the form *a* + *bi*, where *a* and *b* are real numbers and *i* is √ -1. The numbers *a* and *b* are known as the real part and imaginary part of the complex number, respectively. You can perform addition, subtraction, multiplication, and division for complex numbers using the following formulas:

$$a + bi + c + di = (a + c) + (b + d)i$$

$$a + bi - (c + di) = (a - c) + (b - d)i$$

$$(a + bi)*(c + di) = (ac - bd) + (bc + ad)i$$

$$(a + bi)/(c + di) = (ac + bd)/(c^2 + d^2) + (bc - ad)i/(c^2 + d^2)$$

Design a class named Complex for representing complex numbers and the function *add(), subtract(), multiply()* , *divide()* for performing complex-number operations, and the *toString()* function for returning a string representation for a complex number. The *toString()* function returns *(a + bi)* as a string. If *b* is *0* , it simply returns *a* .

Provide three constructors *Complex(a, b)* , *Complex(a)* , and *Complex()* . *Complex()* creates a *Complex* object for number *0* and *Complex(a)* creates a *Complex* object with *0* for *b* .

Overload the operators +, -, *, /, [ ], << and >> for input and output complex numbers..

Overload the operators +, -, *, / as non-member functions, Overload [ ] so that [0] returns a and [1] returns b.it should also throw a run time exception when subscript is out of range i.e [2].

Question # 01:

Create a class RationalNumber (fractions) with the following capabilities:

a) Create a constructor that prevents a 0 denominator in a fraction, reduces or simplifiesfractions that are not in reduced form and avoids negative denominators. Use appropriate exception handling.
b) Overload the addition, subtraction, multiplication and division operators for this class.
c) Overload the relational and equality operators for this class.