Lab – 9 PSEUDOCODE

TASK:

In this lab you are going to write the user-defined types that implement *the Item and Beaded-Bag ADTs.* In other words, you are going to implement two classes in C++.

ADTs

ADTs have “a set of data and a set of operations on that data”.

A. ***Item* ADT**. For data, each *Item* will have a ***name.*** {property} The *Item* ADT will have two operations it can perform on its data:

   1. ***Get The Name***: ***{method}*** This operation will simply return the name of the *Item.*
      a. m_name being a private variable cannot be displayed directly die to encapsulation. Thus this method will return the name using the keyword `*this*` to refer to the current Item that calls the getName method.

   2. ***Is Equal?***: ***{method}*** This operation will simply return whether *this Item* equals another, according to whether the names of the *Item*s match.
      a. Get the names of the items that we are checking using *the `getName()`* method.
      b. Using the equality operator `==` compare the names and then return true if the names are same or false if they are not.

B. ***Beaded-Bag ADT***. will define the data. Like Hermione’s physical bag, the *Beaded-Bag* ADT holds an infinite number of *Item*s.

   I. Completing the *`beadbag.h`* header file:
      a. Initialise all the methods with proper parameters and return types:
      b. e.g. *`void insert (Item  to_insert);`*
      c. All the variables used in the *beadedbag class* are part of the *private member*.

   Having defined the data of the *Beaded-Bag* ADT, we turn our attention to defining its operations. The user of a bag should be able to

1. Insert a new *Item* into the *Beaded Bag*, as long as it is not already in the *Beaded Bag* -- Hermione does *not need* two toothbrushes, after all!
   a. Using the already initialised method *contains()* determine if the item trying to insert is present in the bag or not.
   b. Using a if structure determine if the item needs to be added or not.
   c. return nothing as this will be a void function.

2. Query whether the *Beaded Bag* contains a certain *Item*.
   a. Use the for each method to traverse the vector without worrying about the indexing issues.
   b. Then check if the item's name matches with any of the existing names in the vector. As soon as a match is found return true.
   c. If the loop ends implies there is no match for the new item. Thus returning false.

3. Determine how many *Item*s the *Beaded Bag* actually contains.
   a. Using already existing size method of arrays, return the value.