Zewail University of Science and Technology

CIE 205	**Data Structures & Algorithms Lab**	**Fall 2018**
	LAB #1	
	Revision: Pointers, Arrays, Functions and	
	Classes	

Exercises

Exercise 1:

Write a function that takes 2 sorted <u>arrays of integers</u> A and B then allocates an <u>array of integers</u> A_minus_B that contains elements found in A and not found in B.
It's required the following:

      a. Decide the inputs and output of the function and the passing type of each of them then write function prototype
      b. Implement the function
      c. Write a main function that reads from the use A and B arrays then call the function and print <u>outside the function</u> A_minus_B returned as an output from the function call.
      d. Make the appropriate de-allocations of any dynamically allocated arrays

Exercise 2: BAG/Shop

1- Create a class that presents **bag** objects in a shop. For each bag you store:
      a. **Size** (<u>float</u>)
      b. **Slots (number of slots in the bag)**
      c. **Price**

2- Member Functions of class **Bag**:
      a. <u>**Constructor**</u>: takes the following 2 parameters with these default values:
            i. size: **15.6**
            ii. number of Slots = **5**
      The constructor should initialize price, quantity, total profit with **0** value.
      [Note]: You can skip here the validation of const. data members.

      b. <u>**Setters for all members**</u>: If the passed parameter is valid, the data member should be changed and the function should return **true**. Otherwise, the member variable **won't be affected** and the function should return **false**.)

      c. <u>**Getters for all members**</u>

      d. <u>**Compare:**</u> this function should compare the current bag with another bag passed to the function. It returns **1** if the current bag is *bigger* than the passed bag, and returns **-1** if the passed bag is bigger than the current one and **0** if they are equal. If the sizes are equal, the bag with more number of slots is considered bigger.

      e. <u>**PrintInfo**</u>: Print all the details of the bag in the following format.

      f. <u>**ReadInfo**</u>: Reads the information of the bag from the user

3- Create a class **Shop** to represent bags shop with members:

a. **BagList** (Array of 50 pointers to **Bag**)
b. **Quantity** (Number of bags <u>available</u> in the shop).
c. **Total profit** (<u>Total</u> profit for the <u>sold</u> bags)

d. **Constructor**: to initialize Shop members.

e. **IsAvailabe**: This function takes bag size and no. of slots returns the index of the first bag with the passed parameters. Otherwise, it returns -1.

f. **AddBag**: Add a new bag to the BagList if there is a vacant place. Then reads the data of the new bag from the user.

g. **GetBag**: takes and index and returns a pointer to the bag stored at this index if any. Otherwise, returns NULL. If index is out of range, function returns NULL too.

h. **Sell:** this function takes bag size and no. of slots and searches for the first bag with the passed parameters. If found, it removes (sells) the bag from the list and adds its price to the total profit.
[Note]: Call **IsAvailable** function to check the availability of the required bag.

i. **PrintAllBags**: Prints information of all bags in the shop

4- Write a program (**main function**) that:
   a. Create a Shop object **S**
   b. Add 3 bags to the shop with arbitrary values for their members.
   c. Print the info of all bags.
   d. Declare Bag **Pointer Bp**.
   e. <u>Dynamically</u> allocate a bag using the following constructor parameters:
      (Size = **15.6**, Number of slots = **7**) and <u>make **Bp** point to it</u>.
   f. Set the price of **Bp** to **130**
   g. Call GetBag and point to the returned Bag by pointer **ptr**.
   h. Compare the object pointed to by **Bp** with the object pointed to by **ptr** and print the comparison result"
   i. Sell the 2nd bag in the shop.
   j. Print the info of all bags.

Exercise 3: Faculty

The problem keeps track of the graduate and undergraduate students of a faculty. The system consists of 4 classes: **Student** class which is the base class of **Graduate** and **Undergraduate** classes. It also includes class **Faculty** that contains a list of students.

1- Create class **Student** that contains:
   - ❑ student name and id data members
   - ❑ **a non-default constructor** that initializes its data members and validates them
   - ❑ **getters** for data members
   - ❑ a member function **PrintInfo()** that prints the data members of the student

2- Create class **Graduate** student that contains:
   - ❑ grad_year data member which represents the graduation year
   - ❑ **a non-default constructor** that initializes its data members and validates them
   - ❑ a member function **PrintInfo()** that prints the data members of the graduate student

3- Create class **Undergraduate** student that contains:
   - ❑ current_year that represents the faculty year the student currently in
     (**Assume:** the number of years of this faculty is 4 years)
   - ❑ **a non-default constructor** that initializes its data members and validates them
   - ❑ **getters** for data members
   - ❑ a member function **bool Pass()** that increments the current_year of the student and returns true if he passed his 4th year and graduated, otherwise returns false.
   - ❑ a member function **PrintInfo()** that prints the data members of the undergraduate student

4- Create another class, **Faculty**, which contains:
   - ❑ an array of 200 **Student pointers**
   - ❑ **a default constructor** that makes any needed initializations
   - ❑ a member function **AddStudent (Student * pS)** that adds a student to the list
   - ❑ a member function **DropStudent (int index)** that takes an array index and <u>drops</u> the student (that pointed to by the pointer of this index) from the array by:
     - o making its pointer points to the last array element and making the pointer of the last element points to NULL then decrementing the elements count of the array.
   - ❑ a member function **PassAll()** that:
     - o calls function **Undergraduate::Pass()** for all <u>undergraduate</u> students in the list
     - o if the **Undergraduate::Pass()** of a student returns true (he finished the 4 years of the faculty), the **PassAll()** function should:
       - ▪ <u>drops</u> this student from the list using function **DropStudent**
       - ▪ creates a new **<u>Graduate</u>** student with the same information of the just-graduated student (***Assume:*** the current year is 2017)
       - ▪ adds this graduate student to the list using **AddStudent** function
   - ❑ a member function **PrintInfo()** that prints this information of the faculty:
     - o the number of its <u>undergraduate students</u>
     - o the number of its <u>graduate students</u>
     - o the basic information of its <u>graduate</u> students.

5- Write the **main program** to test your classes. You first need to
   - ❑ create <u>one</u> object of class **Graduate** student
   - ❑ create <u>two</u> objects of class **Undergraduate** student with current_year: 3 and 4
   - ❑ creates a **Faculty** object and adds the 3 created students to it using **AddStudent**
   - ❑ calls **PrintInfo()** function of the faculty object
   - ❑ repeats the following 2 steps <u>3 times:</u>
     - o calls **PassAll()** function of the faculty object
     - o calls **PrintInfo()** function of the faculty object

Exercise 4: Social Media

1. Create an abstract class ***Profile:***
   a. Data members:
      i.   Name: **char***
      ii.   N_Followers**: int**
      iii.   N_Following**: int**
      iv.   Followers: **Profile* []**
      v.   Following: **Profile* []**
   b. Member functions:
      i.   **Non-default constructor:** sets the name of the profile, and initializes the **Followers** and **Following** fields to 0.
      ii.   **Setters and Getters.**
      iii.   **IsFollow:** takes as a parameter another **Profile** and checks whether the current **Profile** follows the passed profile.
      iv.   **IsFollowed:** takes as a parameter another **Profile** and checks whether the current **Profile** is followed by the passed profile.
      v.   **Follow:** takes as a parameter another **Profile**. If the current profile doesn't follow the passed profile:
         -   The current profile should add the passed profile to its list of followers.
         -   The current profile should increment its followers.
         -   The passed profile should add the current profile to its list of following.
         -   The passed profile should increment its following.
         Hint 1: This function should be pure virtual function.
         Hint 2: It's better to define 2 functions: **AddToFollowers** and **AddToFolowing** that take a **Profile** and add it in the appropriate array and increment its counter. Then, function **Follow** should use these two functions inside it. Should these 2 functions be non-virtual, virtual or pure virtual?
      vi.   **Unfollow***: takes as a parameter another **Profile**. If the current profile already follows the passed profile, it should do the reverse operation of **Follow**.
         Hint: you can define 2 more functions for removing from the arrays and updating the counters too.
      vii.   **PrintInfo:** Prints all information of the profile.

2. Create class **Twitter Profile** that is derived from class **Profile**:
   a. Data members:
      i.   Tweets : **int**
      ii.   isProtected**: bool**
   b. Member functions**:**
      i.   **Constructor**: An initial Twitter profile should be **unprotected** and have **zero** tweets.
      ii.   **SetProtected**: sets the profile to be protected or unprotected.
      iii. **Setters and Getters**.
      iv.   **Follow**: Override **Follow** function in the base class to support the new feature that protected profiles should be notified first. It should print a message to the user that "You have a new follow request from *profile X, Do you accept it*?"
         -   If they **accept** the request, the profile should be added.
         -   If they **reject** the request, the profile shouldn't be added and nothing changes.
         Note: Only Twitter profiles can follow Twitter profiles. An error message should be printed otherwise.
      v.   **PrintInfo**: Override **printInfo** function in the base class to print the new information.

3. Create class **LinkedIn Profile** that is derived from class **Profile:**
   a. Extra data members:
      i.   University: **char***
      ii.   Work**: char***

      b. Member functions:
-      **i.**   **Constructor with default parameters:** sets the name of the university and the company of the account holder
-      **ii.**   **Setters and Getters.**
-      **iii.**   **Follow**: Override **Follow** function in the base class to support the new feature that LinkedIn profiles can follow each other **ONLY** if they were in the same university or if they work at the same company or both.
Note: Only LinkedIn profiles can follow LinkedIn profiles. An error message should be printed otherwise.
-      **iv.**   **PrintInfo**: Override **printInfo** function in the base class to print the new information.

4. Write a main function that does the following:
     a. Declare six pointers to **Profile** and dynamically allocate them as follows:
-      **3 LinkedIn** profiles with fields:
  - o "**Ahmed**" studied in **Cairo University** and works in **IBM.**
  - o **"Mai"** is currently studying at **Cairo University**.
  - o **"Omar"** studied in **Alexandria University** and works in **Microsoft**.
-      **3 Twitter** profiles with profile names "**Adam**", "**Kareem**", and "**Dina**".
   b. **Ahmed** follows **Mai** and **Omar**.
   c. Print the information of **Ahmed**, **Mai** and **Omar**.
   d. **Omar** follows **Mai**.
   e. Print the information of **Ahmed**, **Mai** and **Omar**.
   f. **Kareem** sets his profile protected.
   g. **Kareem** follows **Adam** and **Dina.**
   h. **Adam** follows back **Kareem.**
   i. Print the information of **Adam, Kareem** and **Dina.**
   j. **Kareem** unprotects his profile.
   k. **Dina** follows back **Kareem**.
   l. Print the information of **Adam, Kareem** and **Dina.**
   m. **Adam** unfollows **Kareem**.
   n. Check whether **Adam** follows each of **Dina** and **Kareem.**
   o. **Adam** follows **Omar**
   p. Print the information of all profiles.