Tut02 OOP and String stream

1. (Interaction between classes) Let us define a simple class to represent a point in two-dimensional space. A 2D point consists of the x- and y- coordinates. The specification of the Point class is given below (also in the given **Point.h** header file)

```
class Point
{

private:
     int _x, _y;                  //x- and y- coordinates

public:

     Point();                     //Initialize the point to the origin
     Point( int, int );           //Initialize the point to the given coordinates

     double distanceTo( Point );        //Cartesian distance to another Point object

     string toString();                       //Print out the points in "(x, y)" format
};
```

Complete the implementation in a separate **.cpp** file (e.g. **Point.cpp**). Don't forget to include the header file **<cmath>** if you need any of the predefined math functions.

With the **Point** class above, we can now define various 2D shapes. Let us just pick the simplest shape, the **triangle**. A triangle is defined by three points in a 2D space. Below is a skeleton of a class Triangle. Complete the specification in **Triangle.h**, and provide the coding in Triangle.cpp.

```
class Triangle
{
private:
     //Need 3 points as attributes

public:
     Triangle(   /* fill in parameter here, if any */  );

     double perimeter( /*fill in parameter here, if any */  );
     //Purpose: Calculate the perimeter of the triangle

     double area( /*fill in parameter here, if any */);
     //Purpose: Calculate the area of the triangle

     void toString();
     //Purpose: print out the Triangle with the format "( (x1, y1), (x2, y2), (x3, y3) )"

};
```

Write a test driver to try out the **Point** and **Triangle** classes.

Since there are several source files (classes) involved, a devcpp project file of file type **.dev** is needed in order to link all classes together.

Go through the devcpp help topics on Basic Steps including creating a project, and removing or adding files. Here are the steps needed:

- Click on the first icon to create a new project
- In the new project window, click on empty project
- For the name field enter the name of the project (e.g., testTriangle, or any name that is meaningful) and create the (dev) file in the folder containing all the source files (**`Point.h, Point.cpp, Triangle.h, Triangle.cpp).`** An empty project file (dev file) will be shown in the left window.

- Click on add file icon (the one with a + sign in it) and click on all the above files including the test driver file to add them to the project. All these files will then be shown under the project name.

Now click the compile and run icon, and this will get the project compiled and run if the program is built successfully.

Submit the dev file together with all other source files needed.

==========

Questions 2 and 3 are based on the following scenario: Suppose we are building a simple banking simulation around the **`BankAcct`** class discussed in the lecture. We need to write a program to act as the "**Bank**", which provides the following services:

a. Create new account. A new account will be created, with a unique bank account number.
b. Deposit/Withdraw from an existing account.
c. Print information of all existing accounts.

Assumptions:

- The user will create no more than 5 bank accounts in a single test run.
- The account number will start from 90000001, incremented by 1 for each subsequent new account.

An example test run session is given below (user input in **bold**):

```
Service available (1. New Acct, 2. Deposit, 3. Withdrawal, 4. Print All, 5. exit)
Your choice: 1

Result: [1/5] account created.          //[1/5] means the 1st account out of maximum 5
Account Number: 90000001                //Proceed to print out account information
Balance: $0.00

Service available (1. New Acct, 2. Deposit, 3. Withdrawal, 4. Print All, 5. exit)
Your choice: 1

Result: [2/5] account created.
Account Number: 90000002
Balance: $0.00
```

```
Service available (1. New Acct, 2. Deposit, 3. Withdrawal, 4. Print All, 5. exit)
Your choice: 2
Account Number: 90000002
Amount: 5432.10

Result: Deposit performed
Account Number: 90000002
Balance: $5432.10

Service available (1. New Acct, 2. Deposit, 3. Withdrawal, 4. Print All, 5. exit)
Your choice: 3
Account Number: 90000002
Amount: 32.10

Result: Withdrawal performed
Account Number: 90000002
Balance: $5400.00

Service available (1. New Acct, 2. Deposit, 3. Withdrawal, 4. Print All, 5. exit)
Your choice: 4

Result: 2 existing accounts.
Account Number: 90000001
Balance: $0.00

Account Number: 90000002
Balance: $5400.00

Service available (1. New Acct, 2. Deposit, 3. Withdrawal, 4. Print All, 5. exit)
Your choice: 5
```

You are free to decide what message to print if the operation is unsuccessful, e.g. account number not found, withdrawal amount too large, trying to create more than 5 accounts etc.

A skeleton file **`BankingSimulation.cpp`** is given. You can try to code Q2 and Q3 described below.

2. (Simple Class) Let us first flesh out the **`BankAcct`** class:

```
class BankAcct
{

private:
   int _acctNum;
   double _balance;

public:
   BankAcct( int aNum) { /*Refer to lecture note */ }
   BankAcct( int aNum, double bal) { /*Refer to lecture note */ }

   int withdraw( double amount ) {    /*Refer to lecture note */ }

   void deposit( double amount ) {  /*Part (a)*/ }
   int getAccountNumber( ) { /*Part (b)*/ }

   void toString() { /*Part (c)*/ }
};
```

3. (Main Program) With the **`BankAcct`** class in (2), we can now write a main program to provide the services. As the main program serves to **drive the execution** by creating objects and calling appropriate method, we commonly refer to the main program as the **driver code**.

Suggestion:
Use an array of pointers to point to the **`BankAcct`** object(s). Since the **`BankAcct`** objects are created dynamically, you need to use ***object pointers*** (to point to the objects).

Code fragment needed:

```
BankAcct *  baList[5];      //An array of 5 BankAcct pointers

//Initialize them to NULL as good practice
for (int i = 0; i < 5; i++)
     baList[i] = NULL;

//To create an account
baList[0] = new BankAcct( ... );        //use the appropriate constructor,
                                        // index 0 is just an example

//To use a method
baList[0] -> print();
```

Let us modularize the program to make it more manageable:

a. Write a function

```
int findBankAcct( BankAcct* baPtrArr[], int size, int targetAcctNum );
```

Search through the bank accounts in baPtrArr to locate the **index** of bank account object with the target account number. The baPtrArr contains size number of valid bank account object pointers.
Return the index if found, -1 if not found.

b. Write a function

```
void printAllBankAcct( BankAcct* baPtrArr[], int size );
```

Print out the bank account information for each bank account objects in the baPtrArr array of size number of elements.

c. Write a main function to:

i. Print available services. Read user input.
ii. Perform the appropriate service from the user input.

The main function will make use of the functions / methods specified in Q2 and Q3.

4. After dinner at a Chinese restaurant, Batman and Robin had a fortune cookie each. Both cookies contained the same fortune written on a single line on a strip of paper. After reading their fortunes, they both tore the strip of paper into 2 pieces and left promptly. You manage to find two pieces of the torn papers, one containing the left portion of the sentence, the other the right portion. Unfortunately, they may not from the same piece of paper!

From these two pieces, can you work out the content of the original fortune? If it cannot be determined, then print "***The future remains a mystery***".
Hint: A few possible cases:

Original Fortune: You will never never say die.

Case 1: 1st piece: You will never
2nd piece: never say die.
Output: The fortune "You will never say die."

Case 2: 1st piece: You
2nd piece: die.
Output: The fortune remains a mystery.

Case 3: 1st piece: You will never never
2nd piece: never say die.
Output: The fortune: You will never never say die.

Notice that at times the fortune predicted it could be shorter than the actual one.

Sample Input:
er will coat every surface of the batmobile with silly putty.
On 6 September 2011, The Joker will coat eve

Output:
On 6 September 2011, The Joker will coat every surface of the batmobile with silly putty.

Write a test driver to show the working of this function called Fortune. The solution file Fortune.cpp is stored in the sub folder Q4 of folder solution. It is simpler if the test data are embedded within the driver. If the test data are stored in separate file, then you will have to write a testharness describing the use of these data. In this case, the sub folders input and output are needed as their use in a lab exercise.