LAB TASK 09 - CLASSES-03

Problem 1:

You are required to design a class that represents a bank account. The only two instance data members for this class are the *accountNumber(long)* and *balance(double)*. Although the balance can be initialized by the user of the class during creation of an instance, the account number cannot be entered by the user because it must be unique.
To prevent duplication of the account number, you need to use a static data member, called *base(int)*, that is initialized to 0 and incremented with the opening of each new account. We add 100000 to this static data member to make it a large number as is customary and store it in variable *accountNumber*. We do not need a static member function because we are not changing the value of the static data member.

Functions:

i) **Account() // Default Constructor**
Initializes balance to zero and update the accountNumber according to given instructions

ii) **Account(double bal) // Parameterized Constructor**
If bal is negative, program must terminate with a message otherwise update balance and accountNumber accordingly

iii) **void checkBalance () const; // Accessor**
Display account details and balance

iv) **void deposit (double amount); // Mutator**
amount > 0.0 to perform this transaction, and display account details afterwards

v) **void withdraw (double amount); // Mutator**
balance >= amount to perfrom this transaction, and display account details afterwards

vi) **~Account()**
Display account is closed with account details and a message that balance is transferred to the authorized person.

Problem 2:

A fraction, also called a rational number, is a ratio of two integers such as 3/4, 1/2, 7/5, and so on. There is no built-in type in C++ that can represent a fraction; we need to create a new type for it with two data members of type *integer*. We call the first the ***numer*** (abbreviation for numerator) and the second the ***denom*** (abbreviation for denominator).
We create three constructors, one destructor, some accessors, and some mutators. We also add a private member function called normalize, which is used to handle class ***invariants***. We add a private member function called gcd that finds the greatest common divisor. This function is

called by the normalize function. Note that we play the roles of both the designer and the user, but in real life these two roles are performed by different people.

Invariants:
The invariants that concern us in the fraction objects are three conditions:

- The numerator and the denominator should not have a common factor. For example 6/9 should be reduced to 2/3.
- The denominator cannot be 0. A fraction such as 2/0 is undefined.
- The sign of the fraction is the product of the sign of the numerator and denominator and should be set as the sign of the numerator.

You will create two private member functions to take care of the invariants. The gcd function finds the greatest common divisor between the numerator and denominator. The normalized function takes care of the three invariants using the gcd function.

Member Functions:

i. **Fraction (int num, int den);**
The parameter constructor gets values for the numerator and denominator, initializes the object, and normalizes the value of the numerator and the denominator according to the conditions defined for invariant in normalize().

ii. **Fraction ();**
The default constructor creates a fraction as 0/1. It does not need validation.

iii. **Fraction (const Fraction& fract);**
The copy constructor creates a new fraction from an existing object. It does not need normalization because the source object is already normalized.

iv. **~Fraction ();**
The destructor simply cleans up a fraction for recycling

v. **int getNumer () const; // Accessors**
The getNumer function is an accessor function returning the numerator of the host object. It needs the const modifier.

vi. **int getDenom () const; // Accessors**
The getDenom function is an accessor function returning the denominator of the host object. It needs the const modifier.

vii. **void print () const; // Accessors**
The print function is an accessor function, it displays the fraction object in the form x/y.

viii. **void setNumer (int num);**

The setNumer is a mutator function that changes the numerator of an existing object. The object needs normalization.

ix. **void setDenom (int den);**
The setDenom is a mutator function that changes the denominator of an existing object. The object needs normalization.

Helping private member functions

i. **void normalize ();**
Normalize function takes care of three fraction invariants.
- Handling a denominator of zero
- Changing the sign of denominator
- Dividing numerator and denominator by gcd

ii. **int gcd (int n, int m);**
The gcd function finds the greatest common divisor between the numerator and the denominator.