Homework#2

CS331 Introduction to C++ & Object-Oriented Programming, Summer 2018

Due: Until Midnight, July 17 (Tuesday)

Note

- This assignment has two parts, Part I and Part II. Part II includes C++ programming practice.
- Make one file (YourLastName_YourFirstName_CS331_HW2.zip) for your submission, and submit it to Blackboard.
- The zip file will have PartI.doc, and PartII_Q1, PartII_Q2 and PartII_Q3 directories for your source code of each question in Part II.
- Each programming question (Part II) directory will include a root directory which contains all the required source code for the assignment including resources (e.g., Visual Studio Project file) but excluding debug directory.

Part I Problem solving (25 points)

1. Consider the following class declaration:

```
class Thing
{
   private:
      int x;
      int y;
      static int z;
   public:
      Thing() {x=y=z;}
      static void putThing(int a) {x=a;}
};
int  Thing::z=0;
```

Assume a program containing the class declaration defines three `Thing` objects with the following statements:

```
Thing one, two, three;
```

(1) How many separate instances of the `x` member exists?

(2) How many separate instances of the `y` member exists?

(3) How many separate instances of the `z` member exists?

(4) What value will be stored in the `x and y` members of each object?

(5) Write a statement that will call the `putThing` member function *before* the `Thing` objects are defined.

2. Describe the difference between making a class a member of another class (object composition) and making a class a friend of another class.

3. The following class declaration has errors. Locate as many as you can and fix them.

```
class Circle
{
   private:
     double diameter;
     int cetnerX;
     int cetnerY;
   public:
     Circle(double d, int x, int y)(diameter=d; centerX=x; centerY=y;}
     //Overloaded = operator
     void Circle=(Circle &right)
        { diameter=right.diamter;
          centerX=right.centerX;
          centerY=right.centerY;}

     //… Other member functions follow ….
};
```

4. Complete the following tables by filling in private, protected, public or inaccessible in the right-hand columns.

(1)

In a private base class, this base class MEMBER access specification…	… becomes this access specification in the derived class.
private	
protected	
public	

(2)

In a protected base class, this base class MEMBER access specification…	… becomes this access specification in the derived class.
private	
protected	
public	

(3)

In a public base class, this base class MEMBER access specification…	… becomes this access specification in the derived class.
private	
protected	
public	

5. Write a function whose prototype is

```
    char lastChar(const char *str)
```

that takes a nonempty C-string as parameter and returns the last character in the string. For example, the call `lastChar(“abc”)` will return the character `c.`

(Hint) Use pointer operators.

Part II. Programming (75 points)

NOTE:

- **Your source codes should be properly indented and documented to have professional appearance. You should also provide proper comments for the program and variables.**
- **Part II evaluation is based on correct implementation and execution.**

1. (15 points) Practice recursion

Ackermann’s function is a recursive mathematical algorithm that can be used to test how well a computer performs recursion. Write a function `A(m, n)` that solves Ackermann’s function. Use the following logic in your function:

If m=0 then return n+1
If n=0 then return A(m-1, 1)
Otherwise, return A(m-1, A(m, n-1))

Design

Function porotype	Description
`long ack(long m, long n)`	A recursive function to compute Ackermann’s function

Test your function in a driver program that displays the following values:

A(0,0) A(0,1) A(1,1) A(1,2) A(1,3) A(2,2) A(3,2)

Test program

```
#include <iostream>
using namespace std;

//Function Prototype
long ack(long m, long n);

int main( )
{
  for (int m = 0; m <=1; m++)
   for (int n = 0; n <=1; n++)
      cout << "A(" << m << "," << n << ") is " << ack(m, n) << endl;

  //On Most computers:
  //The complexity of recursion overflows the stack at this point
  //so this part cannot be executed without generating stack errors.

  for (int row = 1; row <= 3; row ++)
      cout << "A(" << row << "," << 2 << ") is " << ack(row, 2)
           << endl;

  return 0;
}
```

2. (15 points) Practice the `string` Class.

Imagine you are developing a software package that requires users to enter their own passwords. Your software requires that user’s passwords meet the following criteria:

- The password should be at least six characters long.
- The password should contain at least one uppercase and at least one lowercase letter.
- The password should have at least one digit.

Write a program that asks for a password and then verifies that it meets the stated criteria. If it doesn’t, the program should display a message telling the user why.

Design

Function porotypes	Description
`bool isLongEnough(string s)`	A function to check if a string has a minimum length
`bool hasDigit(string s)`	A function to check if a string has at least one digit
`bool hasUpperAndLowerCase(string s)`	A function to check if a string has at least one upper case and at least one lower case letter.

Test program

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

// Prototypes
bool isLongEnough(string s);
bool hasDigit(string s);
bool hasUpperAndLowerCase(string s);

const int LENGTH = 6;  // Minimum length for a safe password

int main()
{
  // Explain program to user and request a password
  cout << "This program checks passwords to see if they are secure.";
  cout << "\nEnter a password to check: ";

  // Read user's input
  string password;
  cin >> password;

  // Check the password
  if (!isLongEnough(password))
  {
      cout << "Password must be at least six characters long.";
      exit(0);
  }
  if (!hasDigit(password))
  {
      cout << "Password must have at least one digit.";
      exit(0);
  }
  if (!hasUpperAndLowerCase(password))
  {
     cout << "Password must have both lower case and upper case letters.";
     exit(0);
  }
  cout << "The password " << password << " is OK.";
  return 0;
}
```

Hints:

- There is a length function in the string class.
- isdigit, isupper, islower functions

3. (30 points) Practice object-oriented programming

Write a program which should

- Create a class `HugeInteger` that uses a 40-element array of digits to store integers as large as 40 digits each.
- For comparing `HugeInteger` objects, provide function isEqualTo., isNotEqualTo, isGreaterTahn, and isLessThan – each of these is a "predicate" function that simply returns `true` if the relationship holds between the two `HugeIntegers` and returns `false` if the relationship does not hold.

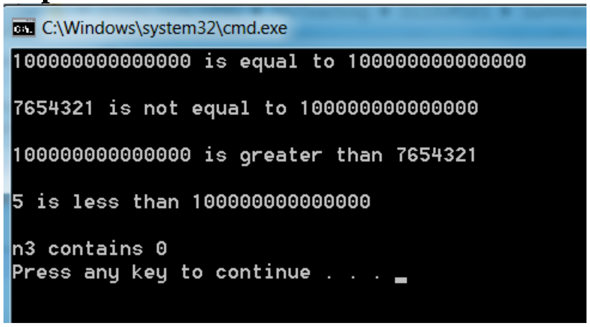
Design:

Class **`HugeInteger`**	
Data Members	Description
`array<short, 40> integer;`	40 element array
Member Functions	Description
`HugeInteger(long = 0);`	default constructor; conversion constructor that converts a long integer into a `HugeInteger` object
`HugeInteger(const std::string&);`	Copy constructor - converts a char string representing a large integer into a `HugeInteger`
`bool isEqualTo(const HugeInteger&) const;`	A function that tests if two `HugeIntegers` are equal
`bool isNotEqualTo(const HugeInteger&) const;`	A function that tests if two `HugeIntegers` are not equal
`bool isGreaterThan(const HugeInteger&) const;`	A function to test if one `HugeInteger` is greater than another
`bool isLessThan(const HugeInteger&) const;`	A function that tests if one `HugeInteger` is less than another
Helper Member Functions	
`string toString() const;`	A function to overload output operator

Given: HugeInteger.h, HugeInteger.cpp and HW2PartIIQ3.cpp.

To do: Complete the isEualTo, isNotEqualTo, isGreatThan, and isLessThan functions.

Expected result:



```
100000000000000 is equal to 100000000000000

7654321 is not equal to 100000000000000

100000000000000 is greater than 7654321

5 is less than 100000000000000

n3 contains 0
Press any key to continue . . .
```

4. (15 points) Practice overloading

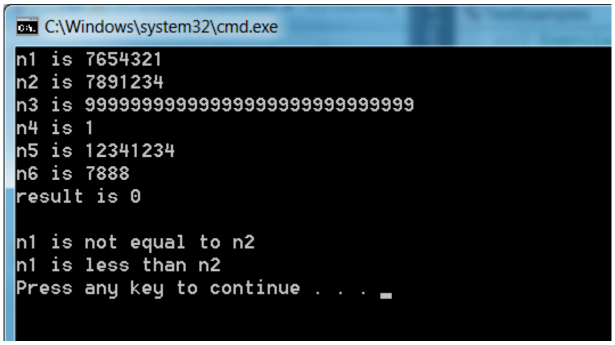
This problem is similar to Part II Q3. Instead of class `HugeInteger,` this problem uses class `HugeInt` to implement operator overloading for the relational operators (>, <) and equality operators (=, !=).

Given: HugeInt.h, HugeInt.cpp, and HW2PartIIQ4.cpp.

To do: Complete HugeInt.h and HugeInt.cpp for the relational operators (>, <) and equality operators (=, !=).

Hint: You may resource most of codes in Q3.

Expected result:



```
n1 is 7654321
n2 is 7891234
n3 is 99999999999999999999999999999
n4 is 1
n5 is 12341234
n6 is 7888
result is 0

n1 is not equal to n2
n1 is less than n2
Press any key to continue . . .
```