EEE102 C++ Programming and Software Engineering II

Lab Practice 4

Classes and Objects 2

Notice:

- The aim of this lab is for you to become familiar with the basics of classes and objects.
- Practice with the exercises. These parts are not for submission.

1. Classes and Objects

Exercise 1

Read the following programs, find out the problem of them, correct them and run.

```
//*******************
//*      student.h   *
//*******************
#include <iostream>
#include <string>
using namespace std;
class student
{
public:
   void student(string name,float g=0.0);
   ~student(int i);
   void display();
private:
   string sname;
   float gpa;
};
void student::student(string name,float g)
{
   cout <<"student constructor is running..." <<endl;
   student.sname+=name;
   gpa=g;
}
student::~student(int i)
{
   cout <<"Student No.: "<<i<<endl;
   cout <<"student destructor is running..." <<endl;
}
```

```
void display()
{
   cout <<"Student Name:" <<sname <<endl;
   cout <<"Student GPA:" <<gpa <<endl;
}
```

```
//*******************
//*      main.cpp  *
//*******************
#include <iostream>
#include <student.h>
using namespace std;
void main()
{
   student s1("Venus",3.5);
   student s2("Jon");
   student s3=s2;
   cout<<"Student Name: "<<s1.name <<endl;
   cut <<"Student GPA: "<<s1.gpa <<endl;
   cout<<"Student Name: "<<s2.name <<endl;
   cut <<"Student GPA: "<<s2.gpa <<endl;
   cout<<"Student Name: "<<s3.name <<endl;
   cut <<"Student GPA: "<<s3.gpa <<endl;
}
```

Exercise 2

The class of complex number introduced in lecture 4 is defined below.

```
class complexClass
{
   double x;
   double y;
public:
   complexClass()
   complexClass(double r, double i=0)
   complexClass(complexClass &cNum)
   ~complexClass()

   double real(void) const
   double imag(void) const
   double abs(void)
   double angle(void)

   void display(void);
   void set(int a, int b);
   void assign(complexClass &a);
```

```
    complexClass plus(complexClass a);
    complexClass minus(complexClass a);
};
```

Part1: Complete the definition of the methods; write a testing function to test them;

Part2: Design the methods to provide the "multiple" and "divide" functions to the **`complexClass`**;

2. Composite Class - *Example of a simplest RPG game*

A "container" in an RPG game, is the container to store the items carried by a player. In this example, we only consider about two types of things: "Heal" to increase the HP (health point) of a player, and "MagicWater" to increase the MP (magic point) of the player.

```
//======================
//     container.h
//======================

// The so-called inventory of a player in RPG games
// contains two items, heal and magic water

#ifndef _CONTAINER     // Conditional compilation
#define _CONTAINER

class container        // Inventory
{
protected:
    int numOfHeal;       // number of heal
    int numOfMW;         // number of magic water
public:
    container();         // constuctor
    void set(int heal_n, int mw_n);   // set the items numbers
    int nOfHeal();       // get the number of heal
    int nOfMW();         // get the number of magic water
    void display();      // display the items;
    bool useHeal();      // use heal
    bool useMW();        // use magic water
};
#endif
```

1. Complete the definition of class methods. There function can be easily guessed from the method name and comment description.

2. Add the ***normal constructor*** and ***copy constructor*** to the **`container`** class.

As the character controlled by human or AI, class “player” is defined to illustrate the general properties of a character. The class defined below is just the very fundamental prototype, which will be enriched in later classes.

```
//=======================
//    player.h
//=======================

// The class of players
// including the general properties and methods related to a character

#ifndef _PLAYER
#define _PLAYER
#include <iostream>
#include <string>
#include <time.h>     // use for generating random factor
#include "container.h"
using namespace std;

class player
{
private:
   int HP, HPmax, MP, MPmax, AP, DP, speed, EXP, LV;
   // General properties of all characters
   string name;   // character name
   container bag;  // character's inventory
   bool playerdeath;
public:
   player(int lv_in=1, string name_in="Not Given");
   void isLevelUp();        // level up judgement
   void reFill();     // character's HP and MP resume
   bool death();      // report whether character is dead
   void isDead();     // check whether character is dead
   bool useHeal();    // consume heal
   bool useMW();      // consume magic water
};
#endif
```

1. Try to complete the definition of class methods. There function can be easily guessed from the method name and comment description.

2. Add a ***copy constructor*** to the `player` class.

3. Use the UML class diagrams to illustrate the members and relationship between these two classes.

3. const

Some objects need to be modifiable and some do not. The programmer may use keyword **`const`** to specify that as object is not modifiable and that any attempt to modify the object should result in a compilation error. The statement

```
const Time noon ( 12, 0, 0 );
```

declares a **`const`** object **`noon`** of class **`Time`** and initializes it to 12:00.

C++ compiler disallows member function calls for **`const`** objects unless the member functions themselves are also declared **`const`**. This is true even for **`get`** member functions that do not modify the objects. A function is specified as **`const`** both in its declaration (Program3.1; lines 19-24) and in its definition (Program3.2; lines 47,53,59 and 65) by inserting the keyword **`const`** after the function's parameter list.

Exercise 3

Read the following programs, run them to learn the usage of const. Answer the question attached at the end.

```
1    // Program 3.1: Time.h
2    // Definition of class Time.
3    // Member functions defined in Time.cpp.
4    #ifndef TIME_H
5    #define TIME_H
6
7    class Time
8    {
9    public:
10      Time( int = 0, int = 0, int = 0 ); // default constructor
11
12      // set functions
13      void setTime( int, int, int ); // set time
14      void setHour( int ); // set hour
15      void setMinute( int ); // set minute
16      void setSecond( int ); // set second
17
18      // get functions (normally declared const)
19      int getHour() const; // return hour
20      int getMinute() const; // return minute
21      int getSecond() const; // return second
22
23      // print functions (normally declared const)
24      void printUniversal() const; // print universal time
25      void printStandard(); // print standard time (should be const)
26
27   private:
   int hour; // 0 - 23 (24-hour clock format)
   int minute; // 0 - 59
   int second; // 0 - 59
}; // end class Time

#endif
```

```
// Program 3.2: Time.cpp
// Member-function definitions for class Time.
#include <iostream>
using std::cout;

#include <iomanip>
using std::setfill;
using std::setw;

#include "Time.h" // include definition of class Time

// constructor function to initialize private data;
// calls member function setTime to set variables;
// default values are 0 (see class definition)
Time::Time( int hour, int minute, int second )
{
   setTime( hour, minute, second );
} // end Time constructor

// set hour, minute and second values
void Time::setTime( int hour, int minute, int second )
{
   setHour( hour );
   setMinute( minute );
   setSecond( second );
} // end function setTime

// set hour value
void Time::setHour( int h )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
} // end function setHour

// set minute value
void Time::setMinute( int m )
{
   minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
} // end function setMinute
```

```

// set second value
void Time::setSecond( int s )
{
   second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
} // end function setSecond

// return hour value
int Time::getHour() const // get functions should be const
{
   return hour;
} // end function getHour

// return minute value
int Time::getMinute() const
{
   return minute;
} // end function getMinute

// return second value
int Time::getSecond() const
{
   return second;
} // end function getSecond

// print Time in universal-time format (HH:MM:SS)
void Time::printUniversal() const
{
   cout << setfill( '0' ) << setw( 2 ) << hour << ":"
      << setw( 2 ) << minute << ":" << setw( 2 ) << second;
} // end function printUniversal

// print Time in standard-time format (HH:MM:SS AM or PM)
void Time::printStandard() // note lack of const declaration
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
      << ":" << setfill( '0' ) << setw( 2 ) << minute
      << ":" << setw( 2 ) << second << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

```
// Program 3.3: testMain.cpp
// Attempting to access a const object with non-const member functions.
#include "Time.h" // include Time class definition

int main()
```

```
{
   Time wakeUp( 6, 45, 0 ); // non-constant object
   const Time noon( 12, 0, 0 ); // constant object

                            // OBJECT     MEMBER FUNCTION
   wakeUp.setHour( 18 );    // non-const  non-const

   noon.setHour( 12 );      // const      non-const

   wakeUp.getHour();        // non-const  const

   noon.getMinute();        // const      const
   noon.printUniversal();   // const      const

   noon.printStandard();    // const      non-const
   return 0;
} // end main
```

Question: can the constructors and destructor be **`const`**?