EEEC10008(515169) S23: Object-Oriented Programming

C++ Overview & Class Introduction

What you will learn from Lab 1

In this laboratory, you will learn the basic concept of C++ and learn how to create objects.

TASK 1-1 NAMESPACES

✔ Please execute the program `lab1-1`. Please try to identify the scope of variable defined in `namespace Complex`.

```
// lab1-1.h
namespace Complex{
    typedef struct{
        double real;
        double image;
    }Cplex;

    const double pi = 3.1416;
    void showComplex(const Cplex &m);
}
```

```
// lab1-1.cpp
#include <iostream>
#include "lab1-1.h"
namespace Complex{
    void showComplex(const Cplex &m)
    {
        std::cout << m.real;
        if (m.image < 0)
           std::cout << m.image << "i" << std::endl;
        else
           std::cout << "+" << m.image << "i" << std::endl;
    }
}
```

```
// lab1-1-main.cpp
#include <iostream>
#include "lab1-1.h"

int main()
{
   Complex::Cplex n;
   n.real = 1 * pi;
   n.image = -0.5;
   Complex::showComplex(n);
   return 0;
}
```

- Please modify the compiler error.
- Hint:
  - Sol1:

```
g++ -c code1.cpp
g++ -c code2.cpp
g++ -o pg.exe code1.o code2.o
```

  - Sol2:

```
g++ -o pg.exe code1.cpp code2.cpp
```

✓ Please modify `lab1-1-main.cpp` as following and execute the program again.

```
// lab1-1-main.cpp
#include <iostream>
#include "lab1-1.h"
using namespace Complex;

int main()
{
   Cplex n;
   n.real = 1 * pi;
   n.image = -0.5;
   showComplex(n);
   return 0;
}
```

- What's the difference between two main files?

TASK 1-2 SCOPE OPERATOR(::)

✓ Please compile and execute the program `lab1-2`.

```
// lab1-2.cpp
#include <iostream>

const int n = 10000;

int main()
{
    int n = 10;
    std::cout << n << " " << ::n << std::endl;
    return 0;
}
```

- Scope operator is usually used to qualify the member defined in specific namespace. For example, scope operator in `std::cout` is used to qualify the member function `cout` under `namespace std`.
- Scope operator can also indicate the member function under global namespace, such as `::n` in this example

TASK 1-3 INLINE FUNCTIONS

✓ The following two examples are used to illustrate the difference between `inline` functions and `define` macro.

```
// lab1-3-1.cpp
#include <iostream>
#define Area(x,y) ((x)*(y))

int main()
{
    double n = Area(3,5.1);
    std::cout << "Area(3,5.1) = " << n << std::endl;
    return 0;
}
```

```
// lab1-3-2.cpp
#include <iostream>
inline int Area(int x,int y) {return x*y;}

int main()
{
    double n = Area(3,5.1);
    std::cout << "Area(3,5.1) = " << n << std::endl;
    return 0;
}
```

- Please observe the difference between `define` and `inline` in `lab1-3-3.cpp`.

```
// lab1-3-3.cpp
#include <iostream>
using namespace std;
#define  D_TRIPLE(n) (n+n+n);
inline int I_TRIPLE(int n) { return n+n+n;}
struct MyClass{
     int m_nValue;
    int GetValue();
};
int MyClass::GetValue(){
    cout<<"hi";
    return m_nValue;
}
int main(){
    MyClass my;
    my.m_nValue=2;
    int r1 = D_TRIPLE(my.GetValue());
    cout<<endl;
    int r2 = I_TRIPLE(my.GetValue());
    cout<< endl;
    cout << r1<<" "<<r2<<endl;
    return 0;
}
```

TASK 1-4 DIRECTIVES

✓ Please compile and execute the `lab1-4` to understand the usage of `ifdef/ifndef` directive.

```
// lab1-4-1.cpp
#include <iostream>
#ifndef PI
#define PI 3.14159
#endif
int main()
{
    std::cout << PI << std::endl;

    return 0;
}
```

```
// lab1-4-2.cpp
#include <iostream>

#define PI 3.1416
#ifndef PI
#define PI 3.14159
#endif

int main()
{
    std::cout << PI << std::endl;
    return 0;
}
```

✓ Please compare the results between two programs and explain the difference.

TASK1-5 STRUCT

✓ Please try to compile and execute the program `lab1-5-1` and `lab1-5-2`, and observe the results.

```
//lab1-5-1.cpp
#include<iostream>
#include<string>

using namespace std;

typedef struct person_t{
    string name;
    unsigned age;
}person;

void printInfo(person p){
    cout<<"Name: "<<p.name<<" | Age: "<<p.age<<endl;
}

int main()
{
    person p = {"Janet", 20};
    printInfo(p);
    return 0;
}
```

```
//lab1-5-2.cpp
#include<iostream>
#include<string>

using namespace std;

typedef struct person_t{
    string name;
    unsigned age;
    void printInfo(){
        cout<<"Name: "<<name<<" | Age: "<<age<<endl;
    }
}person;

int main()
{
    person p = {"Janet", 20};
    p.printInfo();
    return 0;
}
```

➢ In C++, we can declare a function in a `struct`.

TASK1-6 CLASS: CONSTRUCTOR

✓ Please try to compile and execute the program `lab1-6-1`, and observe the results.

```
// lab1-6-1.cpp
#include <iostream>

class Point2D
{
private:
    int x;
    int y;
    double value;

public:
    void assignPoint2D(int n1, int n2, double v);
    void displayPoint2D();
};

void Point2D::assignPoint2D(int n1, int n2, double v)
{
    x = n1;
    y = n2;
    value = v;
}

void Point2D::displayPoint2D()
{
    std::cout << "(" << x << "," << y << ") = ";
    std::cout << value << std::endl;
}

int main()
{
    Point2D ptArray[10];
    for (int i=0;i<10;i++)
       ptArray[i].displayPoint2D();
    return 0;
}
```

✓ We add ***constructor*** to class Point2D and make program `lab1-6-2` work as expect.

```
// lab1-6-2.cpp
…
class Point2D
{
private:
    int x;
    int y;
    double value;
```

```
public:
    Point2D();          // default constructor
    void assignPoint2D(int n1, int n2, double v);
    void displayPoint2D();
};

Point2D::Point2D()
{
    x = 0;
    y = 0;
    value = 0.0;
}
…
```

- You can also use a parenthesized expression list to build your default constructor. In the above example, you can replace the declaration and definition of default constructor as `Point2D():x(0), y(0), value(0.0) {}`.

✓ Please modify your class `Point2D` to make the `lab1-6-3` work.

```
// lab1-6-3.cpp
…
int main()
{
    Point2D pt1;
    Point2D pt2(1,2);
    Point2D pt3(3,2,1.9);

    pt1.displayPoint2D();
    pt2.displayPoint2D();
    pt3.displayPoint2D();
    pt3.assignPoint2D(2,1,0.0);
    pt3.displayPoint2D();

    return 0;
}
```

- Both `Point2D pt3(3,2,1.9)` and `pt3.assignPoint2D(2,1,0.0)` can assign the value to pt3's private member. Can you explain their difference?

TASK1-7 CLASS: DESTRUCTOR

✓ In class `Point2D`, we do not specific the destructor `~Point2D()` since the compiler will generate one. However, if you use new or delete memory in the object, you need constructor to allocate memory and destructor to release it.

✓ Please modify the class `Point2D` as `PointND`, which is used to record the N-dimensional

coordinate using an integer array.

```
// lab1-7.cpp
#include <iostream>
#include <assert.h>

const int num = 10;

class PointND
{
private:
    int *coord;
    double value;

public:
    PointND();
    ~PointND();
    void assignValue(double v);
    void assignCoord(int *vec, int len);
    void displayPointND();
};

PointND::PointND()
{
    value = 0.0;
    coord = new int [num];
    for (int i=0;i<num;i++) coord[i] = 0;
}

PointND::~PointND()
{
    delete []coord;
}

void PointND::assignValue(double v)
{
    value = v;
}

void PointND::assignCoord(int *vec, int len)
{
    assert(len <= num);          // make sure len <= num
    for (int i=0;i<len;i++)
        coord[i] = vec[i];
}

void PointND::displayPointND()
{
    std::cout << "(";
    for (int i=0;i<num;i++)
    {
        std::cout << coord[i];
```

```
        if (i!=num-1)
           std::cout << ", ";
    }
    std::cout << ") = " << value << std::endl;
}

int main()
{
    PointND pt1;
    pt1.displayPointND();

    PointND pt2;
    pt2.assignValue(1.0);
    pt2.displayPointND();

    int *vec = new int [num];
    for (int i=0;i<num;i++) vec[i] = i;

    PointND pt3;
    pt3.assignValue(4.3);
    pt3.assignCoord(vec,num);
    pt3.displayPointND();

    delete []vec;
    return 0;
}
```

EXERCISE 1-1

✓ Write a `class` **Double** with following variables and functions:

Private Variables:

num(double)

Public Functions:

Constructor:

Double(double): set num to the corresponding values

Destructor:

~Double()

void showResult(): print the result of private functions

Private Functions:

double Round(): return the rounding value of num

double Ceil(): return the ceil value of num

double Floor(): return the floor value of num

✓ The main structure of the program becomes,

```
// Double.h
#ifndef DOUBLE_H
#define DOUBLE_H
/* Write class declaration for Double including constructor and
destructor*/

#endif
```

```
// Double.cpp
#include <iostream>
using namespace std;

#include "Double.h"

// Member-function definitions for class Double.
```

```
// ex1-1.cpp
```

```
#include <iostream>
...
#include "Double.h"
int main()
{
    //Declare a variable(Double).
    ...

    //Use showResult() to print the arithmetic results of the numbers.
    ...

    return 0;
}
```

✓ Execution Result:

```
$ ./ex1-1
Please enter the number: 3.1514 ↵
the beginning of the function(showResult)
Round(3.1514) = 3
Ceil(3.1514) = 4
Floor(3.1514) = 3
the end of the function(showResult)
```

EXERCISE 1-2

✓ Write a `class` **Array** with following variables and functions:

Private Variables:

num_arr(int*)

length(int)

Public Functions:

Constructor:

Array(int*, int): set private variables to the corresponding values

Destructor:

~Array()

void showArray(): print all of the numbers in num_arr.

void add_num(): ask user to enter a number n and add number n at the end of num_arr

int count_n(): ask user to enter a number n and return the number of n in num_arr

✓ The main structure of the program becomes,

```
// Array.h
#ifndef ARRAY_H
#define ARRAY_H
/* Write class declaration for Array including constructor and
destructor*/

#endif
```

```
// Array.cpp
#include <iostream>
using namespace std;

#include " Array.h"

// Member-function definitions for class Array.
```

```
// ex1-2.cpp
#include <iostream>
...
#include " Array.h"
int main()
{
   //Declare a variable(Array).
   ...

   //Execute the function
   ...

   return 0;
}
```

✓ Execution Result:

```
$ ./ex1-2
please enter the initial length: 4↵
please enter 4 integer numbers: 3 5 5 7↵
```

```
The number in num_arr is 3 5 5 7          //begin function(showArray)
please enter a counting number: 7 ↵         //begin function(count_n)
The counting number appears 1 times in the num_arr
please enter a new number: 10 ↵             //begin function(add_num)
The number in num_arr is 3 5 5 7 10         //begin function(showArray)
please enter a counting number: 5 ↵         //begin function(count_n)
The counting number appears 2 times in the num_arr
please enter a new number: 5 ↵              //begin function(add_num)
The number in num_arr is 3 5 5 7 10 5       //begin function(showArray)
please enter a counting number: 5 ↵         //begin function(count_n)
The counting number appears 3 times in the num_arr
```

EXERCISE 1-3

✓ Please rewrite the `class` **Array** from Exercise 1-2 to a `namespace`.

```
// Array.h
namespace Array{
    typedef struct{
        int* num_arr;
        int length;
    }Arr;

    void initialize(Arr &A, int* arr, int len);
    void showArray(Arr &A);
    void add_num(Arr &A);
    int count_n(Arr &A);
}
```

```
// ex1-3.cpp
#include <iostream>
...
int main()
{
    //Declare a variable(Array).
    ...

    //Execute function
```

```
    ...

    return 0;
}
```

✓ Execution Result:

```
$ ./ex1-3
please enter the initial length: 4↵
please enter 4 integer numbers: 3 5 5 7↵
The number in num_arr is 3 5 5 7          //begin function(showArray)
please enter a counting number: 7↵         //begin function(count_n)
The counting number appears 1 times in the num_arr
please enter a new number: 10↵              //begin function(add_num)
The number in num_arr is 3 5 5 7 10         //begin function(showArray)
please enter a counting number: 5↵          //begin function(count_n)
The counting number appears 2 times in the num_arr
please enter a new number: 5↵               //begin function(add_num)
The number in num_arr is 3 5 5 7 10 5       //begin function(showArray)
please enter a counting number: 5↵          //begin function(count_n)
The counting number appears 3 times in the num_arr
```