2017 Computer Programming Note

Harry Chang
NTUEE

This is a lecture note of **Computer Programming** held by professor W.J. Liao, NTUEE. This note is mainly about **object oriented** computer programming in `C++`.
Date: From Nov. 14, 2017 to.

Contents

1 Concepts of Classes

1.1 Basic Structure

```
class myclass{
public:
    // member functions
    myclass(...); // constructor
    void function1(...);
    int function2(...);
private:
    // data member
    int variable1;
    double variable2;
};

myclass::myclass(...){...}
void myclass::function1(...){...}
int myclass::function2(...){...}
```

While defining functions **outside** the scope of `class`, must add `myclass::` before function name.

1.2 Preprocess Wrapper

```
#ifndef MYCLASS_H
#define MYCLASS_H
// myclass code
#endif
```

We can put the definition of `myclass` in `myclass.h`, and then define every function in `myclass.cpp`. Last, we can include `"myclass.h"` to use this class in other `.cpp` files. Because **linker** will link all those files together.

1.3 Constructor and Destructor

Constructor: Initialize object.

```
myclass(...);  // constructor
~myclass(...); // destructor
```

Type 1 `myclass()`

```
myclass jizz;
```

Type 2 `myclass(int,int,int)`

```
myclass jizz(a,b,c);
```

Constructor overloading is allowed. We can set several constructors at the same time.
However, destructor overloading is **not** allowed!
`static` won't be destructed until the program ends.
For `auto` variables, last declared variable destroys first.

2 A Deeper Look in Class

2.1 Constant Object

In `myclass`:

```
1 void function1() const; // constant member function
2 const int variable; // constant data member
```

Outside `myclass`:

```
1 myclass var1; // non-constant object
2 const myclass var2; // constant object
```

When a member function doesn't have to change values in data member, we **must** declare these functions `const` because if we accidentally put a `const` object into a `non-const` member function, it would result in a `CE`.

Constant data member **must** be initialized using `member initializers`:

```
1 class myclass{
2 public:
3     myclass(int _x, int _y, int _id)
4     :x(_x), y(_y), id(_id){ // member initializers
5         // empty
6     }
7 private:
8     int x, y;
9     const int id; // we don't want to change this
10 };
```

Both `non-const` and `const` data members can be initialized using `member initializers`.

`Default copy constructor` copies each corresponding data member to initialize the new object:

```
1 myclass(const myclass &);
```

2.2 Friend

```
1 class my_class{
2     friend void my_friend(my_class &,int,int);
3 public:
4     my_class():x(0),y(0){}
5     void function(...);
6 private:
7     int x,y;
8 };
9
10 void my_friend(my_class &c,int _x,int _y){
11     c.x=_x; c.y=_y;
12 }
```

`Friend` functions aren't member functions, they have **granted the right to access** private data members.
Advantage: **Enhance performance**.

2.3 Static Class Members

```
class my_class{
public:
    my_class():x(0){count++;}
    void function1(...);
    static void set_count(int c){count=c;}// static member function
    static int get_count(){return count;} // static member function
private:
    int x;
    static int count; // static data
};
```

There are two methods to access (in `main()`):
Method 1

```
myclass a;
myclass b;
myclass::get_count();
myclass::set_count(5);
myclass::get_count();
```

2.4 `this` Pointer

```
my_class &set_x(int _x){x=_x; return *this;}
my_class &set_y(int _y){y=_y; return *this;}
```

Use:

```
my_class a;
cin >> x >> y;
a.set_x(x).set_y(y); // using this pointer
```

可以串接!

3 Operator Overloading

3.1 Introduction

`a=a+b` v.s. `a=a.add(b)`
Operators must be overloaded for that class.
Exceptions (We can use them in our class without overloading.):

- Assignment operator: `=`
- Address operator: `&`
- Comma operator: `,`

Overloading `?:` operator is **prohibited**!
While overloading an operator with two different classes, we must overload it **globally**.

E.g.

```
class vec{ // vector class (not std::vector)
public:
    vec(){
        x=0;y=0;
    }
    vec(int _x,int _y){
        x=_x;y=_y;
    }
    void print(){
        printf("(%d,%d)\n",x,y);
    }
private:
    int x,y;
};

int main(){
    vec v1,v2(1,2);
    v1.print(); v2.print();
    return 0;
}
```