2020 CUC 春季学期C++整理10.0

一、派生类的构造函数

```cpp
#include<bits/stdc++.h>
using namespace std;
class Bug{
    private:
        int nLegs;
        int nColor;
    public:
        int nType;
        Bug(int legs,int color);
        void PrintBug(){};
};
class FlyBug : public Bug{
    int nWings;
    public:
        FlyBug(int legs,int color,int wings);

};
Bug :: Bug(int legs,int color)
{
    nLegs = legs;
    nColor = color;
}
//错误的构造函数
FlyBug :: FlyBug(int legs,int color,int wings)
{
    nLegs = legs; //无法访问
    nColor = color; //无法访问
    nType = 1; //ok
    nWings = wings;
}
//正确的构造函数
FlyBug :: FlyBug(int legs,int color,int wings) : Bug(legs,color)
{
    nWings = wings;
}
int main()
{
    FlyBug fb(2,3,4);
    fb.PrintBug();
    fb.nType = 1;
    fb.nLegs = 2; //不能访问
    return 0;
 }
```

- 在创建派生类的对象时，**需要调用基类的构造函数**：初始化派生类对象中从基类继承的成员。**在执行一个派生类的构造函数之前，总是先执行基类的构造函数。**

  调用基类构造函数的两种方式

显式方式：在派生类的构造函数中，为基类的构造函数提供参数.
`derived::derived(arg_derived-list):base(arg_base-list)`
隐式方式：在派生类的构造函数中，省略基类构造函数时，派生类的构造函数则**自动调用基类的默认构造函数.**

- 派生类的析构函数被执行时，**执行完派生类的析构函数后，自动调用基类的析构函数。**

```
#include<bits/stdc++.h>
using namespace std;
class Base{
    public:
        int n;
        Base(int i) : n(i){
            cout << "Base" << n << "constructed" << endl;
        }
        ~Base(){
            cout << "Base" << n << "constructed" << endl;
        }
};
class Derived:public Base{
    public:
        Derived(int i):Base (i){
            cout << "Derived constructed" << endl;
        }
        ~Derived(){
            cout << "Derived constructed" << endl;
        }
};
int main()
{
    Derived Obj(3);
    return 0;
 }
//输出结果
Base 3 constructed
Derived constructed
Derived destructed
Base 3 destructed
```

二、封闭类构造函数

```
#include<bits/stdc++.h>
using namespace std;
class Bug{
    private:
        int nLegs;
        int nColors;
    public:
        int nType;
        Bug(int legs,int color);
        void PrintBug(){}
};
class Skill{
    public:
        Skill(int n){}
};
```

```
class FlyBug:public Bug{
    int nWings;
    Skill sk1,sk2;
    public:
    FlyBug(int legs,int color,int wings);
};
FlyBug::FlyBug(int legs,int color,int wings): //使用初始化列表
    Bug(legs,color),sk1(5),sk2(color),nWings(wings){
}
```

在创建派生类的对象时:
1) **先执行基类的构造函数**，用以初始化派生类对象中从基类继承的成员；
2) **再执行成员对象类的构造函数**，用以初始化派生类对象中成员对象。
3) **最后执行派生类自己的构造函数。**

在派生类对象消亡时：
1) 先执行派生类自己的析构函数。
2) 再依次执行各成员对象类的析构函数。
3) 最后执行基类的析构函数。
析构函数的调用顺序与构造函数的调用**顺序相反。**

三、`public`继承的复制兼容规则

```
class base { };
class derived : public base { };
base b;
derived d;
```

1） 派生类的对象可以赋值给基类对象

`b = d;`

2） 派生类对象可以初始化基类引用

`base & br = d;`

3） 派生类对象的地址可以赋值给基类指针

`base * pb = & d;`

如果派生方式是`private`或`protected`，则上述三条不可行。

四、直接基类和间接基类

类A派生类B，类B派生类C，类C派生类D，.....

——类A是类B的**直接基类**、

——类B是类C的**直接基类**，类A是类C的**间接基类**

——类C是类D的**直接基类**，类A、B是类D的**间接基类**

在声明派生类时，只需要列出它的直接基类

——派生类沿着类的层次自动向上继承它的间接基类
——派生类的成员包括

- 派生类自己定义的成员

- 直接基类中的所有成员
- 所有间接基类的全部成员

```
#include <iostream>
using namespace std;
class Base{
    public:
        int n;
        Base(int i) : n(i){
            cout << "Base " << n << " constructed" << endl;
        }
        ~Base(){
            cout << "Base " << n << " destructed" << endl;
        }
};
class Derived:public Base{
    public:
        Derived(int i) : Base(i){
            cout << "Derived constructed" << endl;
        }
        ~Derived(){
            cout << "Derived destructed" << endl;
        }
};
class MoreDerived:public Derived{
    public:
        MoreDerived():Derived(4){
            cout << "More Derived constructed" << endl;
        }
        ~MoreDerived(){
            cout << "More Derived destructed" << endl;
        }
};
int main()
{
    MoreDerived Obj;
    return 0;
}
//输出结果
Base 4 constructed
Derived constructed
More Derived constructed
More Derived destructed
Derived destructed
Base 4 destructed
```