OOP 错题

```
*ptr = new B(5); delete ptr;
```

假设上述语句中，`new` 申请的内存空间首地址为 `Addr` ，存放 `ptr` 指针变量值的内存空间首地址为 `PAddr` ，则执行 `delete ptr` 语句后，`Addr` 、`PAddr` 指向的内存区域均会被系统收回。(F)

只会回收 PAddr 的空间

用 new 关键字动态申请一个三维数组

`float (* fp)[25][10]; fp = new float[10][25][10];`

注意 `float *fp [25][10]` 定义二维指针数组。

`this` 指针是对象的非静态成员函数的隐含参数，其指向对象自己。(T)

对象的静态成员函数是没有 this 指针的，不能通过对象调用，需要直接通过类调用。

`void f3(int d) const;`

const 成员函数，不能修改类内的成员

```
std::cout  << setprecision(2) << 2.345 << std::endl; // 2.3
std::cout  << fixed << setprecision(2) << 2.345 << std::endl; //2.35
```

- 默认参数必须出现在末尾的若干个参数中。
- Manipulators are objects to be inserted or extracted into/from streams
- It is better to choose **inline function** when the function is not complecated and is to be called frequently.
- `const` data member has to be initialized and can not be modified.
- destructor can not be overloaded

Iterator

迭代器可以分为以下五种：

1. 输入迭代器：只读，只支持自增
2. 输出迭代器：只写，只支持自增
3. 前向迭代器：读写，只支持自增
4. 双向迭代器：读写，可自增可自减
5. 随机访问迭代器：读写

在**随机访问迭代器**中，不仅双向迭代器的所有操作都可以进行，而且还可以进行 `p += i` （`p = p + i`），`p -= i` （`p = p - i`），`p[i]` ，大于小于操作，以及指针相减也具有了意义（两个指针之间的元素数量）。

运算符重载和函数重载

- 运算符重载只能重载为友元函数或者成员函数。
- **()、[]、→、→* ，以及=**等赋值运算符都必须被重载为成员函数。
- 不能重载的运算符
  - 成员相关的 `.` 和 `.*`
  - Resolver
  - 三目运算符
- To make functions overloaded, the parameter list of the functions have to be different from each other(T)

模板

- 源程序调用函数模板，才会生成对应参数类型的模板函数的代码。
- **子类是一般类，父类是模板类，继承时必须在子类里实例化父类的类型参数。**

```
template <typename T>
T func(T x, double y) {
    return x + y;
}

int main() {
    cout << func(2.7, 3) << endl; //5.7
    cout << func(3, 2.7) << endl; // 5
}
```

Constructor

```
class AA {
    public:
        AA() {cout << 1;}
        ~AA() {cout << 2;}
};
class BB : public AA {
    AA aa;
    public:
        BB() {cout << 3; }
        ~BB() {cout << 4;}
};

int main() {
    BB bb;
}
```

上面这段程序输出 113422：需要注意首先构造基类，然后如果派生类的成员函数有基类的话也先构造，最后构造派生类。

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

class A {
    public:
        A() {cout << 1;}
}a;

int main() {
    cout << 2;
    A a;
}
```

上面这段程序的输出是 121. 注意到定义了全局变量 a， 所以在进 main 函数之前，先构造 a。

异常

What is wrong in the following code?

```cpp
v[0] = 2.5;
vector<int> v;
```

The program has a runtime error because there are no elements in the vector.

- 所以编译错误不可以当作异常抛出。

多态

虚函数 是在基类中使用关键字 **virtual** 声明的函数。在派生类中重新定义基类中定义的虚函数时，会告诉编译器不要静态链接到该函数。

我们想要的是在程序中任意点可以根据所调用的对象类型来选择调用的函数，这种操作被称为**动态链接**，或**后期绑定**。

因为静态成员函数不能是虚函数，所以它们不能实现多态。(T)

静态成员函数是没有 this 指针的，而虚函数需要 this 指针找到 vptr，所以静态成员函数不能是虚函数。)

```cpp
#include<iostream>
using namespace std;
class Base{
protected:
    int x;
public:
    Base(int b=0): x(b) { }
    virtual void display() const {cout << x << endl;}
};
```

```
class Derived: public Base{
    int y;
public:
    Derived(int d=0): y(d) { }
    void display() {cout << x << "," << y << endl;}
};
int main()
{
  Base b(1);
  Derived d(2);
  Base *p = &d;
  b.display();
  d.display();
  p->display();
  return 0;
}
```

注意上述代码派生类并没有将基类的虚函数覆盖(没有声明 const)，因此 `p->display` 调用的是基类函数。

```
struct Base { virtual string Speak(int n = 42); };
struct Der : public Base { string Speak(int n = 84); };

string Base::Speak(int n)
{
    stringstream ss;
    ss << "Base " << n;
    return ss.str();
}

string Der::Speak(int n)
{
    stringstream ss;
    ss << "Der " << n;
    return ss.str();
}

int main()
{
    Base b1;
    Der d1;

    Base *pb1 = &b1, *pb2 = &d1;
    Der *pd1 = &d1;
    cout << pb1->Speak() << "\n"    // Base 42
        << pb2->Speak() << "\n"     // Der 42
        << pd1->Speak() << "\n"     // Der 84
        << endl;
}
```

default parameter由object, pointer 或者 reference的静态类型决定.因此

`Base *pb2 = &d1`，调用的是派生类 `Der`， 但是默认参数仍是基类的参数 42。