Static

this Pointer:

```
cout << c1.real(this /*隐式传递*/,...); // this is c1's pointer
```

```
class Account{
    static double m_rate; // 与Class类所创建的对象脱离，不属于任何对象
    static void set_rate(const double &x) {m_rate = x;} // 没有隐式this指针，故无法处理non_stati
}

double Account::m_rate = 8.0; // 静态数据要在类外初始化

int main()
{
    // Static 成员函数的调用（两种）
    Account::set_rate(5.0); // 通过类名来调用
    Account a;
    a.set_rate(2.0); // 通过实例来调用（但是编译器此时不会将a的地址this传入函数）
}
```

单例模式 Singleton

```
class A {
public:
    static A& getInstance(); // 唯一接口
    setup() { pass }
private:
    // 构造函数为private,外界不可实例化
    A();
    A(const A& rhs);
    //Static A a; // 类内唯一实例化一个对象
};
// 在静态函数中实例化唯一对象，防止浪费内存空间
A& A::getInstance()
{
    static A a;
    return a;
}
int main()
{
    // 外界只能通过A::getInstance()函数来得到唯一的一个A的实例
    A::getInstance().setup();
}
```

cout

```
class _IO_ostream_withassign : public ostream {
    pass
};
extern _IO_ostream_withassign cout;

class ostream : virtual public ios{
public:
    pass // << 符号的重载
};
```

template 类模板 & 函数模板

```
template<typename T>
class MyClass {
public:
    T func(T t){this->t = t;}
private:
    T t;
};

template<class T>
inline
const T& min(const T& a, const T& b)
{
    return a < b ? a : b; // < 符号在 T 内被重载
}

int main()
{
    MyClass<double> m; // 实例化
}
```

namespace 命名空间

```
namespace std {
    ...
}
using namespace std; // 全开（不推荐）
using std::cout; // 指定某条开
int main()
{
    int a,b;
    std::cin >> a >> b;
    std::cout << "hello world" << std::endl; // 提倡
```

```
    cout << "hello world" << std::endl;
}
```