模板

函数模板和类模板都可以作为其他类的友元

函数模板

```
template <typename T>
T max(T &a, T &b)
{
    return a > b ? a : b;
}
```

- 关键字`template`声明使用模板
- `<typename T>`类型列表，`T`为模板参数

当编译器遇到调用模板函数的时候，编译器会根据调用函数的参数类型推导出模板的类型，然后使用推导的类型替换模板参数，并生成对应类型的函数定义，这个过程叫**模板实例化**

- 编译器自动推导模板参数，称之为**隐式实例化**
- 手动指定模板参数，称之为**显式实例化**
- 模板参数支持非类型参数，非类型参数可以给默认值，在模板内部当作常量来使用

```
template <typename T, size_t cnt = 100>
size_t foo()
{
    T array[cnt] { 0 };
    return sizeof(array);
}
```

- 模板特化

```
template <>
const char * max<const char *>(const char *s1, const char *s2)
{
    return strcmp(s1, s2) > 0 ? s1 : s2;
}
```

- 只能写在头文件，不能声明和实现分开
- 函数模板只有在调用的时候才会实例化
- C++11标准支持函数模板默认参数

```
template <typename T1, typename T2 = double>
T2 add(T1 num1, T2 num2)
{
    return num1 + num2;
}
```

类模板

```
template <typename T>
class Smart_ptr {
    ...
    T foo();
private:
    T *_ptr;
    ...
};
```

与函数模板类似

- 类模板只能**显式实例化**

```
class Student {
    ...
};

Smart_ptr<Student> sptr;
```

- 类模板的成员函数定义在类外部时，使用以下形式

```
template <typename T>
T Smart_ptr<T>::foo()
{
    ...
}
```

- 只能写在头文件，不能声明和实现分开
- 支持非类型参数，非类型参数可以给默认值，在模板内部当作常量来使用
- 支持函数模板默认参数
- 支持类模板和成员函数模板的特化

拷贝构造与赋值运算

拷贝构造

当定义一个新的对象时调用

```
Testclass t1;
Testclass t2(t1);   // 定义一个新对象
Testcalss t3 = t1;  // 定义一个新对象
```

赋值运算

当一个对象赋值给另一个已经存在的对象的时候调用

```
Testclass t1;
Testclass t2;
t2 = t1;    // 赋值给已经存在的对象
```