3.5模板

模板：抽象的思路*

- 思路：**抽象**
- 各种数据，**抽象**成数据类型。
  - 12,3,……抽象成整型int。
  - 'A','B','C'……抽象成字符型char。
  - 1.0,2.3,17.5,……抽象成浮点型float、double
  - 把一类对象的共性数据抽象成结构体。
  - 把一类对象的共性抽象成类。
- 抽象，再抽象。
  - 抽象可以不断进行，**逐层上升**。
  - 数据类型抽象成模板。
  - 被抽象物是具体的，对应的抽象物可以具体化为不同的被抽象物。

模板

- 模板是一个将**数据类型参数化**的工具，它把“一般性的算法"和其"对数据类型的实现"区分开来。
- 模板分为**函数模板**和**类模**板两种。
- 模板提高了软件的重用性。当函数参数或数据成员可以是多种类型而函数或类所实现的功能又相同时，使用C++模板在很大程度上**简化了**编程。

函数模板

- 函数模板是一种不指定某些参数的数据类型的函数，在函数模板被调用时根据实际参数的类型决定这些函数模板参数的类型。
- 函数重载与函数模板：

```
int my_abs(int val)
{
    returq (val < 0) ? -val : val;
}
float my_abs(float val)
{
    return (val < 0) ? -val : val;
}
//...

template <typename T> // 或 tempLate  <class T>
T my_abs(T val)
{//T为模板参数文（类型参数）
    return (val < 0) ? -val : val;
}
```

- 编译器根据函数实参的数据类型确定模板参数T再自动生成对应的函数，即模板函数。

```cpp
template <typename T>
T my_abs(T val);
int main()
{
    int i = 100;
    long l = -12345L;
    float f = -125.78F;

    cout << my_abs(i) << endl; // T ==> int
    cout << my_abs(l) << endl; // T ==> long
    cout << my_abs(f) << endl; // T ==> float
    return 0;
}
template <typename T> // 或 tempLate  <class T>
T my_abs(T val)
{//T为模板参数文（类型参数）
    return (val < 0) ? -val : val;
}
```

含多个模板参数的函数模板

```cpp
#include <bits/stdc++.h>
using namespace std;

template <typename T1, typename T2>
T1 Max(T1 x, T2 y);

int main()
{
    int i = 100;
    float f = 127.35F;

    cout << Max(i, f) << endl;
    return 0;
}

template <typename T1, typename T2>
T1 Max(T1 x, T2 y)
{
    return (x >= y) ? x : (T1)y;
}
```

类模板（Classtemplate）

- 类模板是一种通用的类：在定义类时不说明某些数据成员、成员函数的形参及返回值类型。
- 类是对象的抽象，类模板是**类的抽象**。
- 类模板被称为带参数（或参数化）的类，也被称为**类工厂**。
- 函数模板只能用于定义非成员函数，它是模板的一个特例。类模板实际上是函数模板的推广，它是一种不确定类的某些数据成员的类型或成员函数的参数及返回值的类型的类。

例子：类模板

```
//内部定义成员函数
// template <typename T>
// class MyTemClass
// {
//     private:
//         T x;
//     public:
//     void setX(T a) {this->x=a;}
//     T getX() {return x;}
// };

template <typename T>
class MyTemClass
{
private:
    T x;

public:
    void setX(T a);
    T getX();
};
//外部定义成员函数
template <typename T1>
void MyTemClass<T1>::setX(T1 x)
{
    this->x = x;
}
template <typename T>
void MyTemClass<T>::getX()
{
    return this->x;
}
```

类模板的实例化

- 类模板的实例化，与函数模板不同，类模板不是通过调用函数时实参的数据类型来确定类型参数具体所代表的类型，而是通过在使用类模板声明对象时所给出的实际数据类型确定类型参数。

实例化语句：

```
MyTemClass<int>intobject;
```

- 编译器首先用int替代模板类定义中的类型参数T,生成一个所有数据类型已确定的类cass；
- 然后再利用这个类创建对象intObject。

含多个参数的类模板

- 多个参数中，可含有已确定类型的参数

```
template <typename T1, int i, typename T2>
class MyClassM{
};
```

- 实例化方法

```
MyClassM<int, 100, float> MyObject;
```

例子：类模板在多文件时的问题

```
#include <bits/stdc++.h>
using namespace std;

// CComplex2.hpp
template <typename T>
class CComplex2
{
private:
    T r, i;

public:
    CComplex2(T r=0, T i=0);
    virtual ~CComplex2();
    CComplex2 operator+(CComplex2 c2);

    friend ostream& operator<<(ostream& os, const CComplex2& c1)
    {
        os << "(" << c1.r << "," << c1.i << ")";
        return os;
    }
};
//CComplex2.cpp
template <typename T>
CComplex2<T>::CComplex2(T r, T i)
{
    this->r = r;
    this->i = i;
}

template <typename T>
CComplex2<T> CComplex2<T>::operator+(CComplex2<T> c2)
{
    CComplex2<T> CTemp;
    CTemp.r = this->r + c2.r;
    CTemp.i = this->i + c2.i;
    return CTemp;
}
template <typename T>
CComplex2<T>::~CComplex2()
{
}

//#include "CComplex2.hpp"
//#include "CComplex2.cpp"
int main()
{
    CComplex2<float> c1(1, 2);
    CComplex2<float> c2(1.1, 2.2);
    CComplex2<float> c3 = c1 + c2;

    cout << c1 << endl;
```

```
52      cout << c2 << endl;
53      cout << c3 << endl;
54      return 0;
55  }
56
```