函数模板和类模板

- 函数模板
- 类模板

9.1.1 函数模板

- 函数模板可以用来创建一个通用功能的函数，以支持多种不同形参，进一步简化重载函数的函数体设计。
- 定义方法：

  template <模板参数表>

  函数定义
- 模板参数表的内容
  - 类型参数：class（或typename） 标识符

例：求绝对值函数的模板

```
template<typename T>
T abs(T x) {
  return x < 0? -x : x;
}
int main() {
  int n = -5;
  double d = -5.5;
  cout << abs(n) << endl;
  cout << abs(d) << endl;
  return 0;
}
```

9.1.2 类模板

- 类模板的作用

  使用类模板使用户可以为类声明一种模式，使得类中的某些数据成员、某些成员函数的参数、某些成员函数的返回值，能取任意类型。

类模板的声明

- 类模板：

```
template <模板参数表>
class 类名
{类成员声明}
```

- 如果需要在类模板以外定义其成员函数，则要采用以下的形式：

```
template <模板参数表>
类型名 类名<模板参数标识符列表>::函数名（参数表）
```

例9-2 类模板示例

```
template <class T>
class Store {//类模板：对任意类型数据进行存取
private:
  T item;   // item用于存放任意类型的数据
  bool haveValue;  // 标记item是否已被存入内容
public:
  Store();              // 无参构造函数
  T &getElem();   //提取数据函数
  void putElem(const T &x);  //存入数据函数
};
```

```
template <class T>          //默认构造函数的实现
Store<T>::Store(){ haveValue=false; }

template <class T>      //提取数据函数的实现
T &Store<T>::getElem() {
   if (!haveValue) {
         cout << "No item present!" << endl;
         exit(1);          //使程序完全退出，返回到操作系统。
   }
   return item;        // 返回item中存放的数据
}

template <class T>          //存入数据函数的实现
void Store<T>::putElem(const T &x) {
   haveValue = true;// 置为true，表示item中已存入数值
   item = x;                        // 将x值存入item
}
```

例9-2 (续)

```
struct Student {
  int id;                 //学号
  float gpa;              //平均分
};
int main() {
   Store<int> s1;
   s1.putElem(3);
   cout << s1.getElem() << endl;

   Student g = { 1000, 23 };
   Store<Student> s3;
   s3.putElem(g);
   cout << s3.getElem().id << endl;
}
```