中山大学移动信息工程学院本科生实验报告

（2015-2016 学年春季学期）

课程名称：**Data structures and algorithms**　　任课教师：**张子臻、黄淦**

年级	15	专业（方向）	软件工程(移动信息工程)
学号	15352408	姓名	张镓伟
电话	13531810182	Email	709075442@qq.com
开始日期	2016.5.23	完成日期	2016.5.23

1. 实验题目

1000:

编写一个模板类 Stack，用于实现栈的相关功能，模板参数有两个，一个是类型 typename，另一个是 int 型的 capacity，指明这个栈的容量。

2. 实验目的

A. 初步接触泛型编程。

B. 学习编写模板类，模板函数。

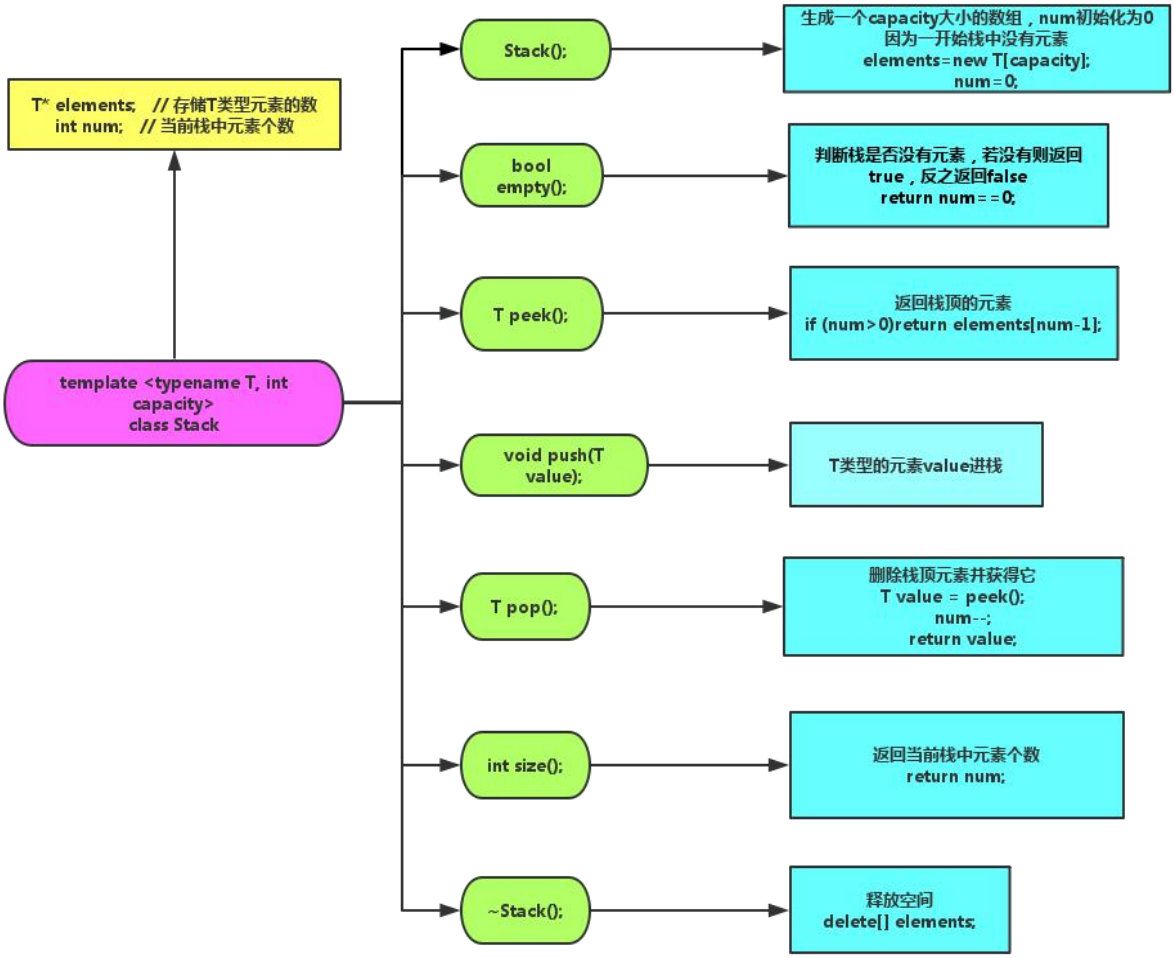
3. 程序设计

1000:

模板类的定义：

```
template<typename T, int capacity>//capacity用来指明栈的容量大小
class Stack{
    public:
        Stack(); // Constructs an empty stack.
        bool empty();// Returns true if the stack is empty.
        T peek(); // Returns the element at the top of the stack without removing it from the stack.
        void push(T value); // Stores an element into the top of the stack.
        T pop();// Removes the element at the top of the stack and returns it.
        int size(); // Returns the number of elements in the stack.
        ~Stack();
    private:
        T* elements; // Points to an array that stores elements in the stack.
        int num;  // The number of the elements in the stack.
};
```

总流程：



本题完整可测试代码详见附件 1000.cpp

4. 程序运行与测试

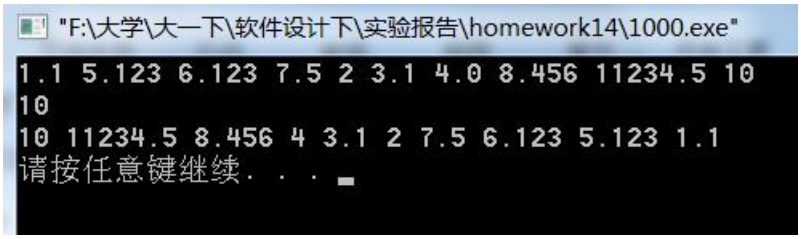
1000:

测试一：

```
int main(){
    Stack<int,15> intStack;
    for (int i = 0; i < 15; i++) intStack.push(i);
    cout<<intStack.size()<<endl;
    while (!intStack.empty()) {
        cout << intStack.peek() << " ";
        intStack.pop();
    }
    cout << endl;
}
```

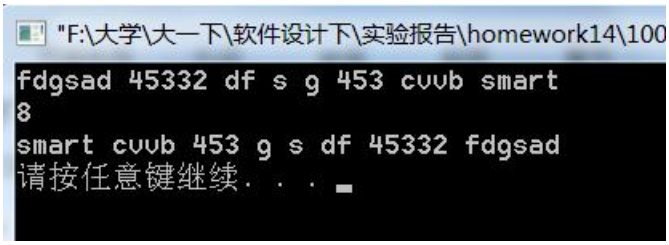
测试二：

```
int main(){
    Stack<double,15> intStack;
    double x;
    for (int i = 0; i < 10; i++) {
        cin>>x;
        intStack.push(x);
    }
    cout<<intStack.size()<<endl;
    while (!intStack.empty()) {
        cout << intStack.peek() << " ";
        intStack.pop();
    }
    cout << endl;
}
```



测试三：

```
int main(){
    Stack<string,8> intStack;
    string x;
    for (int i = 0; i < 8; i++) {
        cin>>x;
        intStack.push(x);
    }
    cout<<intStack.size()<<endl;
    while (!intStack.empty()) {
        cout << intStack.peek() << " ";
        intStack.pop();
    }
    cout << endl;
}
```



5. 实验总结与心得

1. 所谓泛型编程就是以独立于任何特定类型的方式编写代码。使用泛型程序时，我们需要提供具体程序实例所操作的类型或值。简单来说，我们希望写出的代码可以支持多种数据类型，而由于算法相同只是数据类型不同，为了避免复制粘贴大量重复的代码，我们可以使用泛型编程。**在C++中，模板是泛型编程的基础。模板是创建类或函数的蓝图或公式**。

2.函数模板：

A.模板定义**以关键字 template 开始，后接模板形参表**，模板形参表是用尖括号括住的**一个或多个模板形参**的列表，形参之间以逗号分隔。**模板形参表不能为空**。

B.模板形参表很像函数形参表，函数形参表定义了特定类型的局部变量但并不初始化那些变量，在运行时再提供实参来初始化形参。

C.模板形参可以是表示类型的类型形参，也可以是表示常量表达式的非类型形参。非类型形参跟在类型说明符之后声明,类型形参跟在关键字 class 或 typename 之后定义，例如，class T 是名为 T 的类型形参，class 和 typename 没有区别。习惯上 类型是类的时候用 class，非类的时候用 typename。

D.使用函数模板时，编译器会推断哪个（或哪些）模板实参绑定到模板形参。一旦编译器确定了实际的模板实参，就称它实例化了函数模板的一个实例。实质上，编译器将确定用什么类型代替每个类型形参，以及用什么值代替每个非类型形参。推导出实际模板实参后，编译器使用实参代替相应的模板形参产生编译该版本的函数。**编译器承担了为我们使用的每种类型而编写函数的单调工作**。

3.类模板:

类模板也是模板，因此必须以关键字 template 开头，后接模板形参表。

A.除了模板形参表外，类模板的定义看起来与任意其他类相似。类模板可以定义数据成员、函数成员和类型成员，也可以使用访问标号控制对成员的访问，还可以定义构造函数和析构函数等等。在类和类成员的定义中，可以使用模板形参作为类型或值的占位符，在使用类时再提供那些类型或值。

B.与调用函数模板形成对比，使用类模板时，**必须为模板形参显式指定实参。编译器使用实参来实例化这个类的特定类型版本**。

4.模板形参与函数形参一样，我们为模板形参选择的名字没有任何意义。可以给模板形参赋予的唯一含义是区别形参是类型形参还是非类型形参。如果是类型形参，我们就知道该形参表示未知类型，如果是非类型形参，我们就知道它是一个未知值。

5.模板形参的名字可以在**声明为模板形参之后直到模板声明或定义的末尾处使用**。模板形参遵循常规名字屏蔽规则,与**全局作用域中声明的对象、函数或类型同名的模板形参会屏蔽全局名字。用作模板形参的名字不能在模板内部重用(不能再次作为类型来使用)**。

6.模板在为我们提供了便利的同时我们也要注意其缺点，程序员在使用模板类或者函数时应保证该模板对你所用到的类型都支持。比如我们将交换函数 swap 模板化,其参数为 int 、long long、double 等基本类型时是可行的，当如果参数是两个数组就会出错，因为数组并不支持直接整体赋值。

附录、提交文件清单

实验报告一份：实验报告.pdf
代码一份:1000.cpp