## 实验3 Operator Overloading; String and Array Objects

## 学号: 09017423 姓名：杨彬

### 实验目的

In this chapter you will learn:

What operator overloading is and how it can make programs more readable and programming more convenient.

To redefine (overload) operators to work with objects of user‐defined classes.

The differences between overloading unary and binary operators.

To convert objects from one class to another class.

When to, and when not to, overload operators.

To create PhoneNumber, Array, String and Date classes that demonstrate operator overloading.

To use overloaded operators and other member functions of standard library class string. 15

### 实验内容

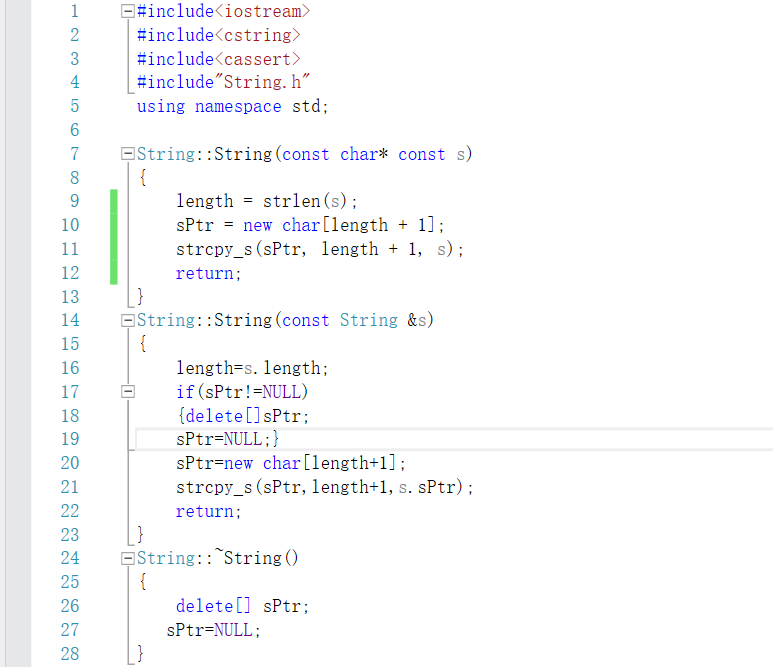
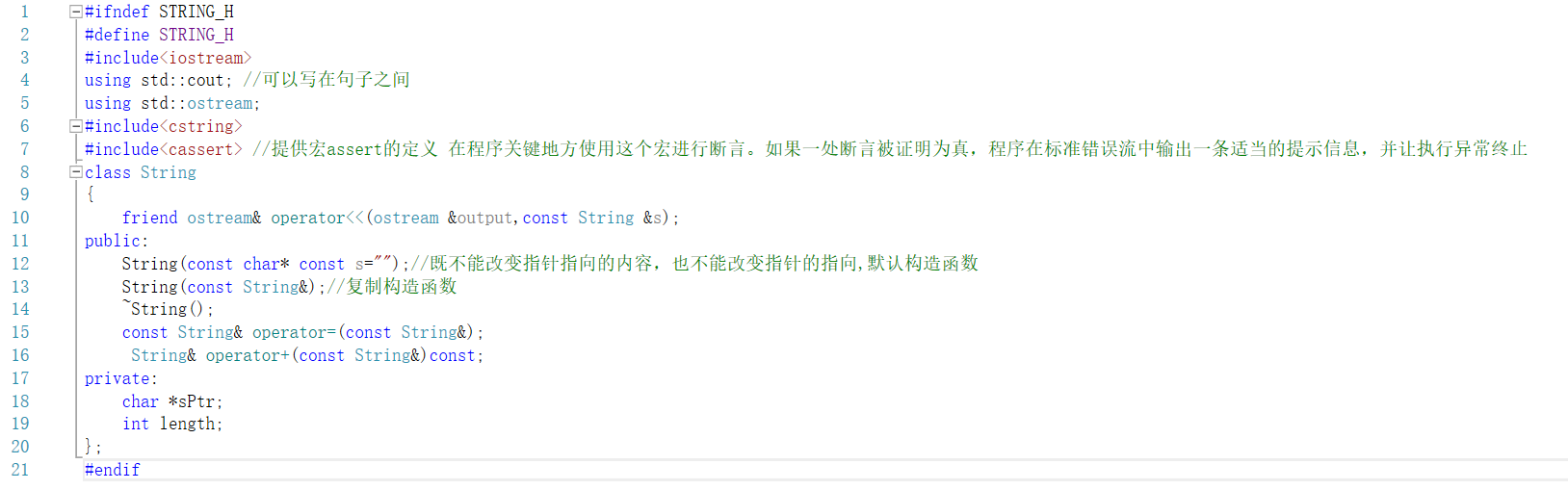
**Exp1:** Description of the Problem

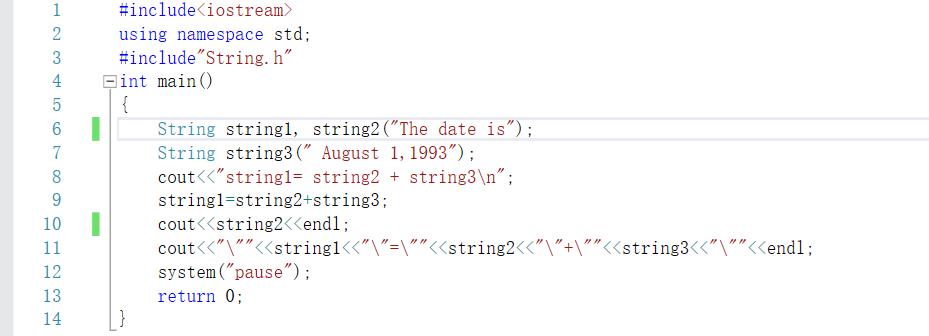
String concatenation requires two operands—the two strings that are to be concatenated. In the text, we showed how to implement an overloaded concatenation operator that concatenates the second String object to the right of the first String object, thus modifying the first String object. In some applications, it is desirable to produce a concatenated String object without modifying the String arguments. Implement operator+ to allow operations

**Exp2:** Description of the Problem

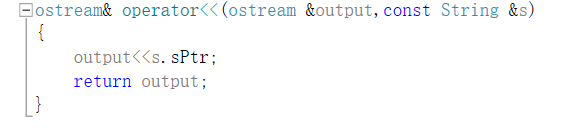
A machine with 32‐bit integers can represent integers in the range of approximately –2 billion to +2 billion. This fixed‐size restriction is rarely troublesome, but there are applications in which we would like to be able to use a much wider range of integers. This is what C++ was built to do, namely, create powerful new data types. Consider class HugeInt of Figs. 11.8–11.10. Study the class carefully, then overload the relational and equality operators.[Note: We do not show an assignment operator or copy constructor for class HugeInt, because the assignment operator and copy constructor provided by the compiler are capable of copying the entire array data member properly.]

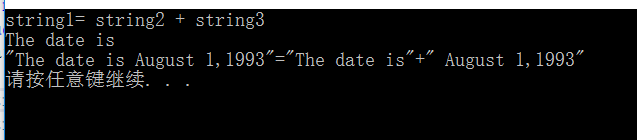
### 实验代码及结果：

**Exp1:** 

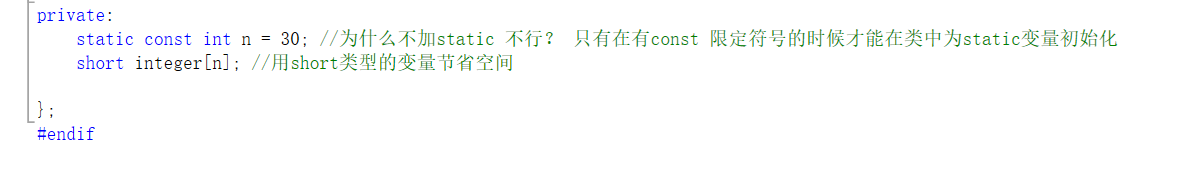


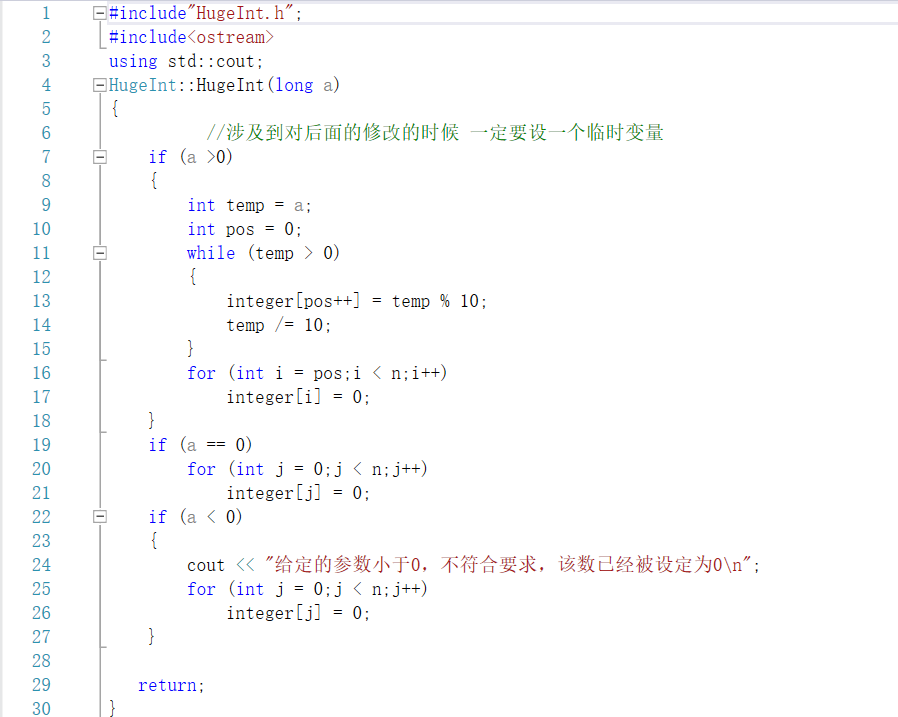


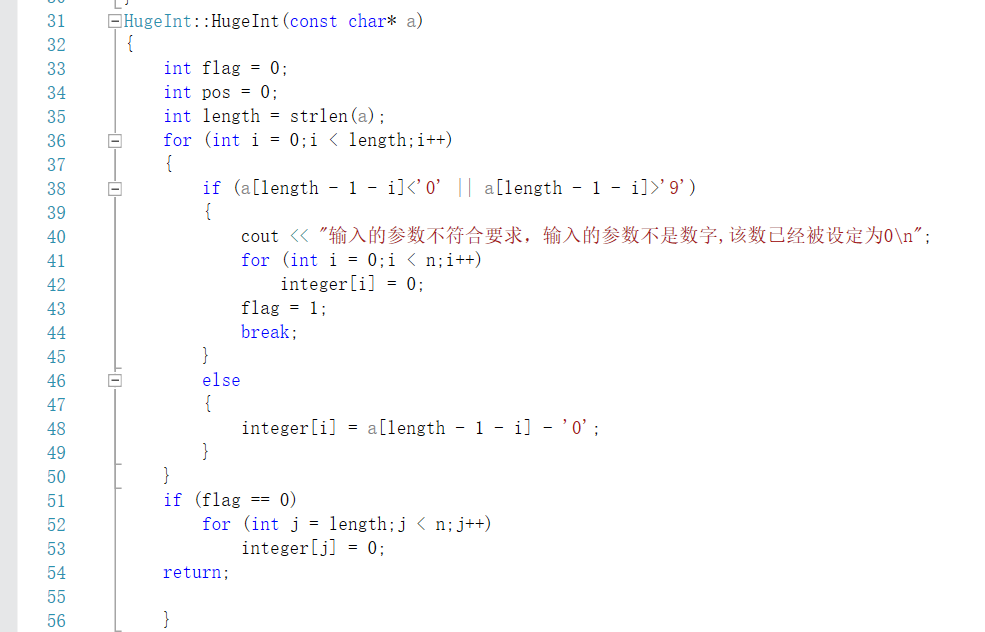
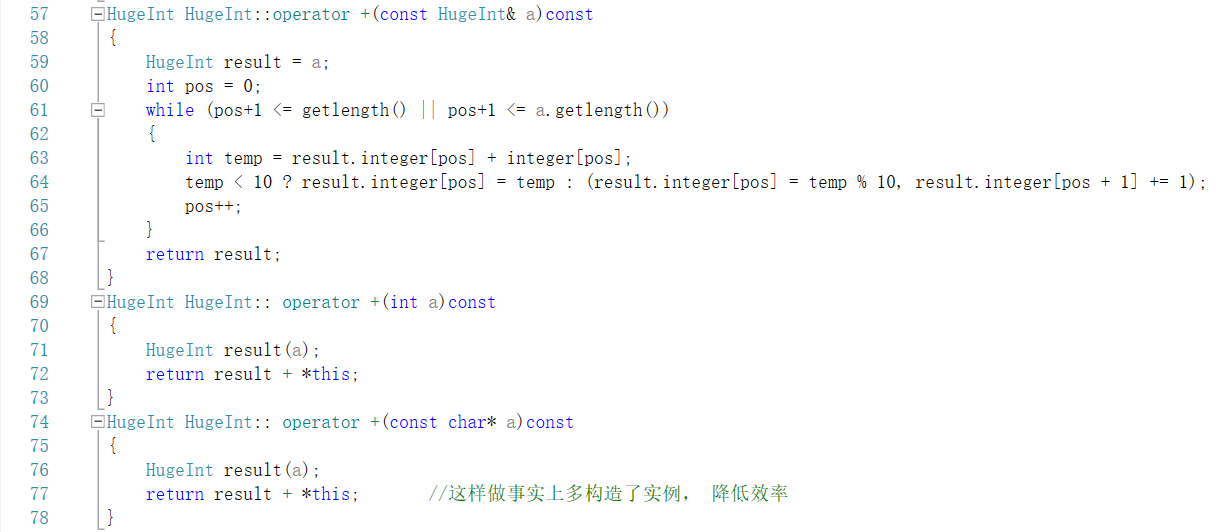


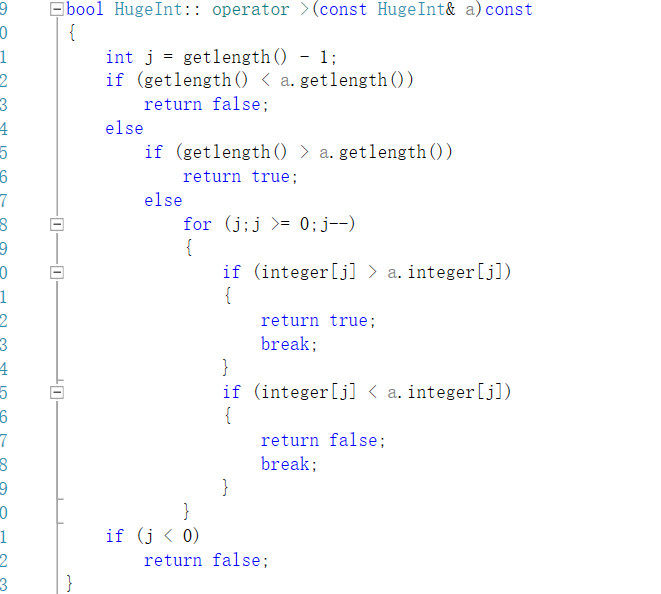


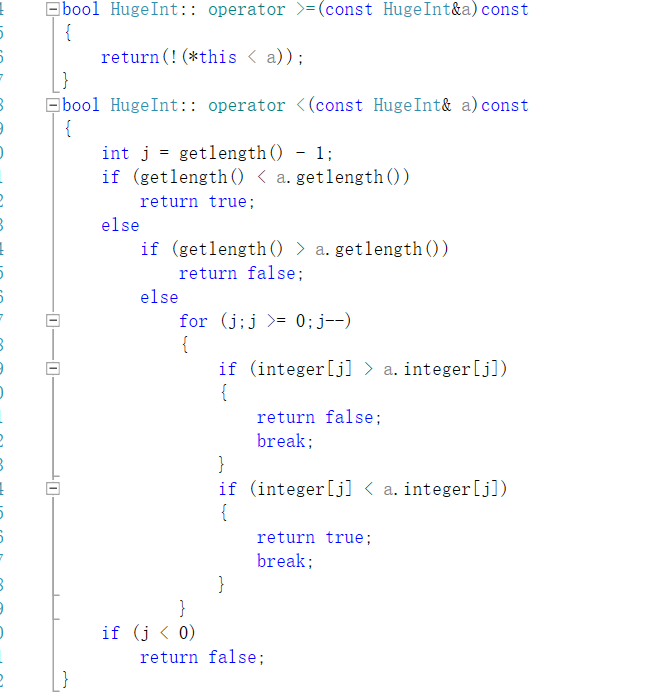
**Exp2:** 

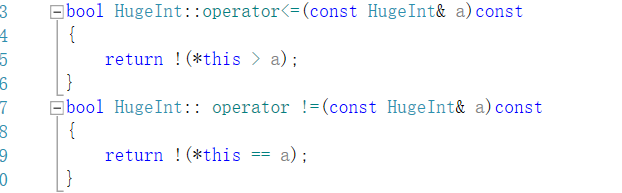


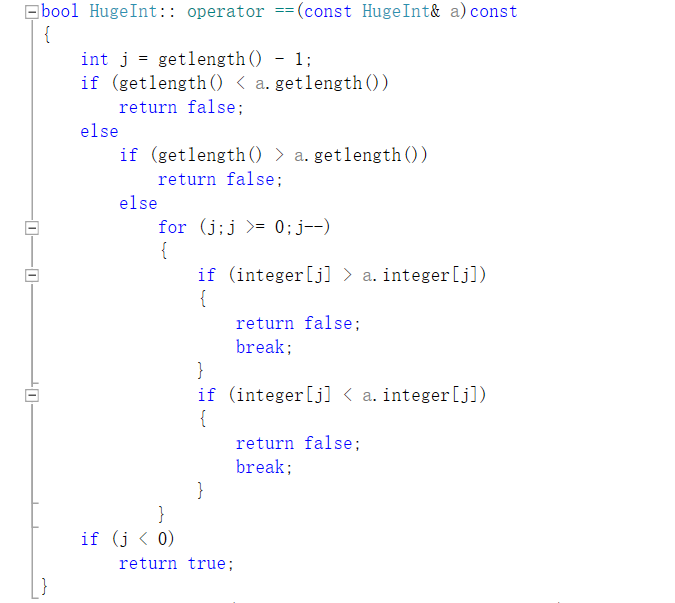


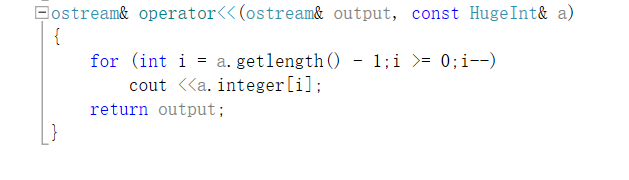
 

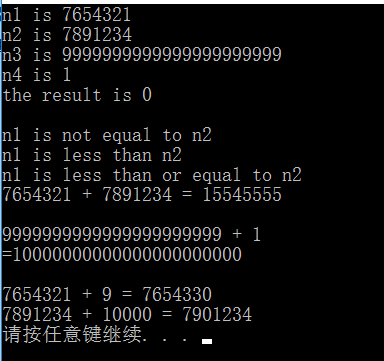




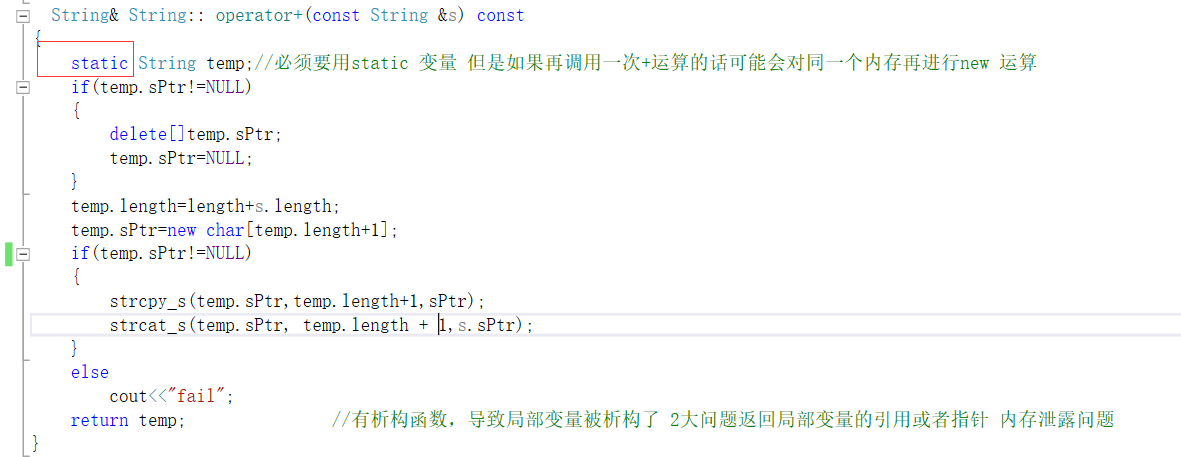


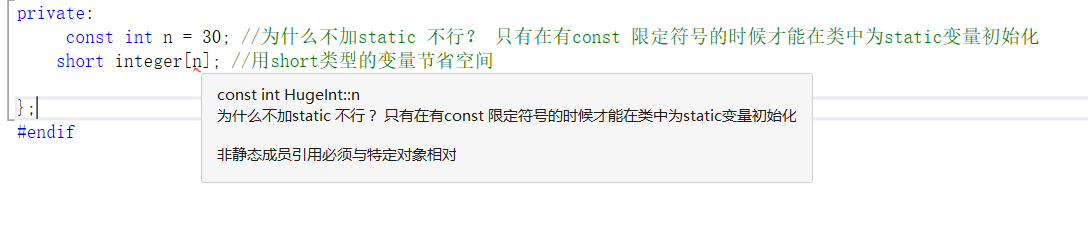


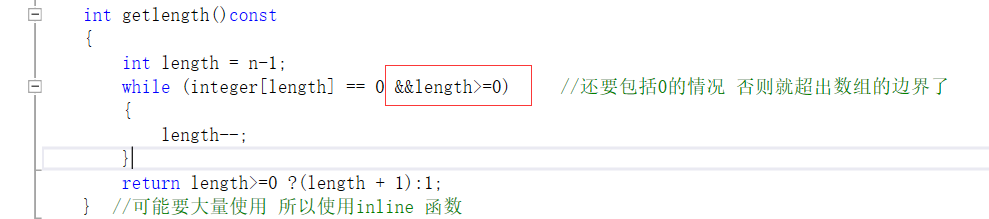


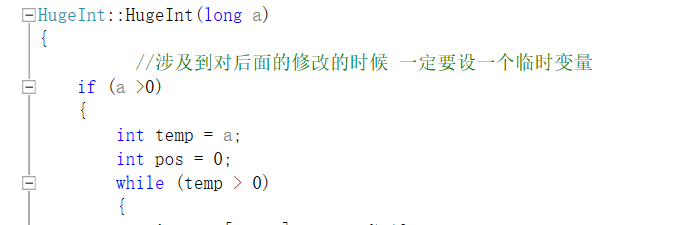


### 遇到的问题及解决过程

**Exp1:**这个实验我遇到了一个很大的问题：在调试的时候程序总是提示说无法申请内存。后来经过设置断点的方法，我发现问题在于在写这个函数的时候 一开始没有写static关键词 导致类被返回的时候就被析构了，那么自然就导致我在C=A+B这个表达式中作为参数的类的length变量的大小就随机了，导致length过大无法申请内存，当加上static的时候，这个变量就变成了静态变量，返回的时候就不会被析构了，这样就解决了问题。

**Exp2:** 这个实验由于与实验1 的类似，本次实验没有什么大问题。1但是在private的变量的时候 即 如果不加static 那么编译器会报错，这是因为只有静态变量才能在没有任何一个实例被创建的情况下存在。

2其次：在写代码的时候一定要考虑特殊情况，特别涉及到数组的边界，0，空指针 等等问题的时候 比如在下面这个函数 我因为没有加上判断条件导致错误

3 在写

的时候 我因为没用重新声明临时变量temp 导致后面的判断全部出错。重新定义临时变量就解决了问题。

### 实验体会

**Exp1:** 1从这个实验的错误 我知道了 永远不要返回一个临时变量的指针或者引用，就想在这个实验中我遇到的错误一样。这样会导致返回的指针变成了野指针造成不可预料的后果，其次在函数中还有一个问题需要注意的就是内存泄露问题，在函数中如果声明指针并且为其分配空间，那么在函数返回前应该用delete语句销毁空间否则会造成内存泄露。2在类的运算符重载，或者类的函数中。如果临时类的实例，这个实例涉及到new 运算或者是 指针，那么如果不在临时变量声明的语句前面加上static关键词让它成为一个静态变量，即使是返回这个auto变量的值而不是引用，依然是十分危险的，会造成严重的后果就像我在这个实验中所犯的错误。

2函数中的临时变量如果是类的话，那么返回值之后，函数将会自动调用其析构函数将其销毁，同时如果返回值，也会调用类的构造函数来返回值，参数传递同理。

**Exp2:** 1 通过这个实验，我尝试着应用了inline 函数，即getlength函数，对于这种代码比较短，而且使用频繁的函数可以使用lnline函数比如将set get 等直接与底层数据接触的函数写成inline的形式。

2 在这个实验中，我第一次尝试了对数据的检查工作，采用条件语句来对数据进行检查。这对类的可靠性和实用性 相当重要，同样的我也知道了，设计一个合理的类还应该考虑各种特殊的极端情况，这对类的设计者来说是一个很大的挑战。

### 