《面向对象程序设计A》研究报告

周孙睿、施安然、陈诺、苏亿铭

摘 要 根据《面向对象程序设计A》的课程要求，本小组自定义了String类和Vector类，并分别重新定义了成员函数与运算符：String类中，通过多次重载构造函数，实现了用String对象、C字符数组、n个(默认n=1)C字符及String对象的切片进行初始化，并通过自定义运算符与成员函数，实现了标准库<string>包括empty()，length()，substr()，c\_str()，find()，insert()，replace()，erase()，trim()等函数以及+、=、+=，输入输出流<<、>>，以及逻辑运算符的功能，并支持正负数索引([ ])；vector类中，通过多次重载构造函数，实现了定义空vector对象、指定长度vector对象，并可以用vector对象初始化，通过自定义运算符与成员函数，实现了标准库<vector>包括push\_back ()，empty()，front()，back()，begin()，end()，pop\_back()，erase()，clear()，insert()，find()等函数以及+、=、+=，输入输出流<<、>>，以及逻辑运算符的功能，并支持索引([ ])。Sring类与vector类都使用了申请冗余空间的方式减少内存分配次数，String类将该功能封装至私有成员函数fitmem ()中，便于扩展函数功能。而vector则添加了异常分析，通过自定义异常类，使使用者在不正确使用方法时能够更快排查错误。

关键词 String类；Vector类；运算符重载；异常处理

1 引言

在进行程序设计时，如要存储一组数据，一般会选择使用数组的形式进行存储。然而，数组的长度在声明时就已确定，为了存储不定长度的数据，只能开辟一大片空间以保证空间足够。但这种操作在存储规模较小的数据时，就会造成严重的空间浪费。

为了解决这个问题，在C++程序设计中，一般会使用C++标准模板库（STL）中的字符串类（string）来存储字符串；用向量类（vector）来存储数组，以存储不定长数组/字符串。这种方式不仅能节省空间，同时也能让对数组/字符串的处理更方便、简单。

但是过度依赖STL并不好，因为这样就不能自己为其添加成员函数或重载运算符，影响了程序设计的自由度。因此，本小组将模仿STL中的string类与vector类，编写自己的模板类以存储不定长数组/字符串，同时增加一些STL中没有的成员函数以体现自己编写模板类的优势。

2 字符串类

2.1 成员变量介绍

**2.1.1成员变量介绍**

str用来来存储c风格字符串，指向堆区的空间，最后一位为‘\0’。

**代码展示：**

class String

{

private:

    char \*str = NULL;

};

**2.1.2成员函数介绍**

**1.构造和析构函数**

**（1）有参构造函数**

a.传入字符串。将字符串s传入fitmem成员函数对本串进行初始化。

**代码展示：**

String::String(const char \*s)

{

    fitmem(s);

}

b.传入一个字符c和一个整数n，功能是将n个字符c作为字符串，用fitmem成员函

数进行初始化。

**代码展示：**

String::String(const char c, int n)

{

    int i;

    if (n < 0)

        n = 0;

    char temp[n + 1];

    for (i = 0; i < n; i++)

        temp[i] = c;

    temp[i] = '\0';

    fitmem(temp);

}

**（2）拷贝构造函数**

a.用另一个String类的对象进行初始化。将传入对象储存的字符串传入fitmem成员函

数对本串进行初始化。

**代码展示：**

String::String(const String &Str)

{

    fitmem(Str.str);

}

b. 传入一个String类的对象Str、整数pos和n，功能是用Str对象中str成员指向的字符串第pos位开始的n个字符对本串进行初始化。创建新数组存入要初始化内容，同样用fitmem完成.

**代码展示：**

String::String(const String &Str, int pos, int n)

{

    int m = strlen(Str.str);

    if (m == 0)

    {

        fitmem("");

        return;

    }

    if (pos >= m || pos < -m)

        pos = m - 1;

    else if (pos < 0 && pos > -m)

        pos = m + pos;

    if ((n > 0 && pos + n >= m) || n == 0) //

        n = m - pos;

    else if (n < 0)

        n = 1;

    char temp[n + 1];

    temp[n] = '\0';

    for (int i = 0; i < n; i++)

        temp[i] = Str.str[pos + i];

    fitmem(temp);

}

**（3）析构函数**

释放str成员所指向的堆区空间，并将str置空。

**代码展示：**

String::~String()

{

    if (str != NULL)

    {

        delete[] str;

        str = NULL;

    }

}

**2.运算符重载**

**（1）“[]”重载**

将[]运算符重载为对象的下标索引，本质是对对象的str成员指向的字符数组的索引。在方括号内传入数值index对对象进行下标索引，返回值为str数组下标为index的字符。如果传入的索引值超出了字符数组索引的合法范围，则将传入数值加上数组长度作为索引值或返回数组最后一位。

**代码展示：**

char &String::operator[](int index)

{

    if (index >= 0 && index < length())

        return str[index];

    else if (index < 0 && index > -1 - length())

        return str[length() + index];

    return str[length()];

}

**（2）“=”重载**

功能是用传入的String类的Str对象对本对象进行赋值。具体做法是先比较本对象和传入对象的str成员指向的数组是否相等，若相等则直接返回本对象；若不相等则用fitmem成员函数参照Str对象的str成员对本对象的str成员指向的空间进行大小调整和赋值操作，返回值为本对象。

**代码展示：**

String &String::operator=(const String &Str)

{

    if (&Str == this)

        return \*this;

    fitmem(Str.str);

    return \*this;

}

**（3）“+”重载**

功能是将传入的两个String类的Str1对象和Str2对象各自的str成员数组连接，并返回连接后的新的String对象。具体方法是在栈区生成一个新的String类的对象temp，再在堆区开辟大小为两个传入对象的str成员大小之和的空间，用temp的str成员来接收。将两个传入的str成员分别复制给temp对象的str成员，最后返回temp对象。

**代码展示：**

String operator+(const String &Str1, const String &Str2)

{

    String temp;

    temp.str = new char[strlen(Str1.str) + strlen(Str2.str) + 1];

    strcpy(temp.str, Str1.str);

    strcat(temp.str, Str2.str);

    return temp;

}

**（4）“==”重载**

功能是比较两个传入的String对象储存的c风格字符串是否一致，返回值为bool值。

**代码展示：**

bool operator==(const String &Str1, const String &Str2)

{

    return strcmp(Str1.str, Str2.str) == 0;

}

**（5）“！=”重载**

功能是比较两个传入的String对象储存的c风格字符串是否不一致，返回值为bool值。

**代码展示：**

bool operator!=(const String &Str1, const String &Str2)

{

    return strcmp(Str1.str, Str2.str) != 0;

}

**（6）“>”、“>=”、“<”、“<=”重载**

功能是比较两个传入的两个String对象储存的c风格字符串的大小，使用strcmp函数进行比较。

**代码展示：**

bool operator>(const String &Str1, const String &Str2)

{

    return (strcmp(Str1.str, Str2.str) > 0);

}

bool operator>=(const String &Str1, const String &Str2)

{

    return (strcmp(Str1.str, Str2.str) >= 0);

}

bool operator<(const String &Str1, const String &Str2)

{

    return (strcmp(Str1.str, Str2.str) < 0);

}

bool operator<=(const String &Str1, const String &Str2)

{

    return (strcmp(Str1.str, Str2.str) <= 0);

}

**（7）“<<”重载**

功能是将传入的String对象的str成员传到输出流。

**代码展示：**

ostream &operator<<(ostream &out, const String &Str)

{

    out << Str.str;

    return out;

}

**（8）“>>”重载**

功能是从输入流读取字符存入String类的Str对象的str成员中。具体方法是：在堆区创建一个大小为100的字符数组temp，用in.get()函数读入字符并存入temp中，直到读入空格或换行符停止读入，执行重载之后的str+temp并结束读入。当temp数组存满后，执行重载之后的str+temp，再重新开始读入字符存到temp[0]，重复此过程直到读入空格或换行符。

**代码展示：**

istream &operator>>(istream &in, String &Str)

{

    char tmp;

    int i = 0;

    char temp[100];

    temp[0] = '\0';

    Str = "";

    while (true)

    {

        tmp = in.get();

        if (tmp == '\n' || tmp == ' ')

        {

            Str = Str + temp;

            break;

        }

        temp[i++] = tmp;

        temp[i] = '\0';

        if (i == 99)

        {

            Str = Str + temp;

            i = 0;

            temp[i] = '\0';

        }

    }

    return in;

}

istream &operator>>(istream &in, String &Str)

{

    char tmp;

    int i = 0;

    char temp[100];

    temp[0] = '\0';

    Str = "";

    while (true)

    {

        tmp = in.get();

        if (tmp == '\n' || tmp == ' ')

        {

            Str = Str + temp;

            break;

        }

        temp[i++] = tmp;

        temp[i] = '\0';

        if (i == 99)

        {

            Str = Str + temp;

            i = 0;

            temp[i] = '\0';

        }

    }

    return in;

}

**（9）“+=”运算符重载**

功能是将本对象和传入的String类的Str对象执行重载后的+操作后赋值给本对象并返回自身。

**代码展示：**

String &String::operator+=(const String &Str)

{

    \*this = \*this + Str;

    return \*this;

}

**3.其他成员函数**

**（1）fitmem函数及其重载函数**

接受一个字符串，如果本对象的str成员指向空间大小小于传入字符串大小或大于传入字符串大小的三倍，则释放str原本指向的堆区空间重新在堆区开辟空间，大小为传入字符串大小的两倍，再将传入字符串的内容复制给str成员指向的字符串。本函数为private成员函数。

**代码展示：**

void String::fitmem(const char \*s)

{

    char tmp[strlen(s) + 1];

    strcpy(tmp, s);

    if (\_msize(str) <= strlen(tmp) || \_msize(str) > (strlen(tmp) + 1) \* 3)

    {

        delete[] str;

        str = new char[2 \* (strlen(tmp) + 1)];

    }

    strcpy(str, tmp);

    return;

}

void String::fitmem(char \*s)

{

    if (\_msize(str) <= strlen(s) || \_msize(str) > (strlen(s) + 1) \* 3)

    {

        if (s != str)

        {

            delete[] str;

            str = new char[2 \* (strlen(s) + 1)];

            strcpy(str, s);

        }

        else

        {

            char tmp[strlen(s) + 1];

            strcpy(tmp, s);

            delete[] str;

            str = new char[2 \* (strlen(tmp) + 1)];

            strcpy(str, tmp);

        }

    }

    else

        strcpy(str, s);

    return;

}

**（2）empty函数**

功能是判断对象储存的c字符串是否为空，返回值为bool值。具体方法是用字符串长度是否为0来判断。

**代码展示：**

bool String::empty()

{

    return (strlen((\*this).str) == 0);

}

**（3）length函数、capacity函数、c\_str函数**

返回对象储存字符串的长度、大小；转换为c-字符串。

**代码展示：**

int String::length() const

{

    return strlen(str);

}

int String::capacity() const

{

    return \_msize(str);

}

const char \*String::c\_str()

{

    return str;

}

**(4)substr函数**

功能是取本串位置pos开始的n个字符，构成新对象。具体方法是使用String::String(const String &Str, int pos, int n)构造函数，传入对象自身、子串起始位置和子串字符个数，构造出储存子串的对象并返回。

**代码展示：**

String String::substr(int pos, int n) const

{

    String temp(\*this, pos, n);

    return temp;

}

**(5)insert函数**

功能是将Str所指向的字符串插入本对象指向字符串的pos位置之前。具体方法是在栈区开辟字符数组tmp，tmp最后一位赋值为0。先将本对象的前pos位给temp前pos位赋值，再将Str指向的字符串以及本对象数组的剩余内容先后用strcat函数接到tmp数组尾部，最后用fitmem函数对本对象储存的字符串进行大小调整和赋值。函数返回值为本对象。

**代码展示：**

String &String::insert(int pos, const String &Str)

{

    int len = strlen(str);

    if (pos > len)

        pos = len;

    if (pos < 0)

        pos = 0;

    char tmp[len + strlen(Str.str) + 1];

    strncpy(tmp, str, pos);

    tmp[pos] = '\0';

    strcat(tmp, Str.str);

    strcat(tmp, str + pos);

    fitmem(tmp);

    return \*this;

}

**（6）find函数**

查找并返回Str在本串中第一次出现的位置。循环遍历字符串并比较是否有与Str字符串一致的子串，返回值为字串第一次出现的下标。

**代码展示：**

int String::find(const String &Str) const

{

    int i, j, m, n, flag;

    m = strlen(Str.str);

    n = strlen(str);

    if (m > n)

        return -1;

    for (i = 0; i < n - m; i++)

    {

        flag = 1;

        for (j = 0; j < m; j++)

            if (str[i + j] != Str.str[j])

            {

                flag = 0;

                break;

            }

        if (flag == 1)

            return i;

    }

    return -1;

}

**（7）replace函数**

将本对象储存的字符串第pos位起的n个字符替换为Str储存的字符串。具体方法是在堆区创建String类的temp对象并用本对象初始化。去除temp的第pos位起的n个字符，插入Str储存的字符串，最后用fitmem函数修改本对象储存的字符串。

**代码展示：**

String &String::replace(int pos, int n, const String &Str)

{

    String temp = \*this;

    temp = temp.erase(pos, n);

    temp = temp.insert(pos, Str);

    fitmem(temp.str);

    return \*this;

}

**（8）erase函数**

功能是从本串pos位置开始，去除n个字符并返回去除之后的本串。具体方法是将本串第pos位赋值为结束符，再用strcat函数将第pos+n位开始的字符串连接到第pos位之后返回本串。

**代码展示：**

String &String::erase(int pos, int n)

{

    int len = strlen(str);

    if (pos > len)

        pos = len;

    if (n == 0 || pos + n > len)

        n = len - pos;

    if (n < 0)

        return \*this;

    str[pos] = '\0';

    strcat(str, str + pos + n);

    fitmem(str);

    return \*this;

}

**（9）trim函数**

功能是删除字符串前后的空白字符，具体方法是遍历和反向遍历字符串并标记首个和最后一个非空格字符所在位置。如果字符串全部为空格，就用空格字符修改本串。如果有空格字符以外的字符，就在堆区开辟大小为标记区间的temp字符串，并将标记内部的字符串赋值给temp字符串，最后用fitmem函数参照temp字符串修改本串，释放temp空间。函数返回值为本对象。

**代码展示：**

String &String::trim()

{

    int i, j = strlen(str);

    char \*temp = (char \*)"";

    if (j == 0)

        return \*this;

    for (j--; j >= 0 && (str[j] == ' ' || str[j] == '\t'); j--)

        ;

    str[j + 1] = '\0';

    for (i = 0; str[i] == ' ' || str[i] == '\t'; i++)

        ;

    if (i > j)

    {

        fitmem(temp);

    }

    else

    {

        temp = new char[j - i + 2];

        strcpy(temp, str + i);

        fitmem(temp);

        delete[] temp;

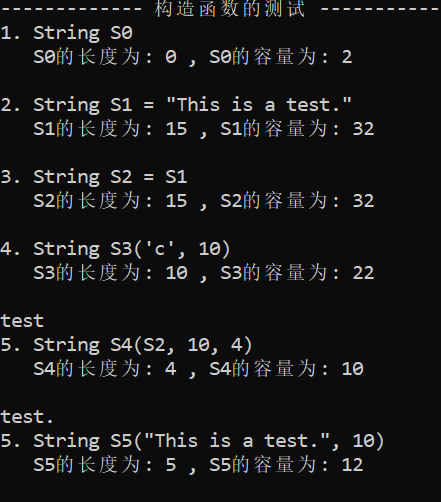
    }

    return \*this;

}

**2.1.3测试**

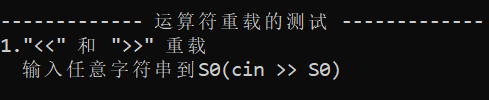
**1.构造函数**

****

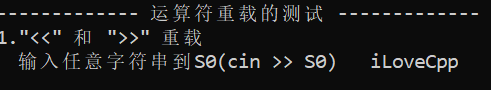
**图2.1.1构造函数的测试**

**2.运算符重载**

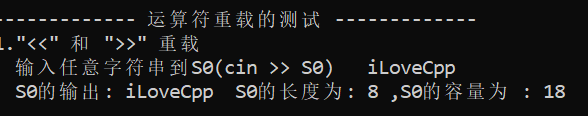
**（1）“<<”和“>>”重载**

****

**图2.1.2输入前**

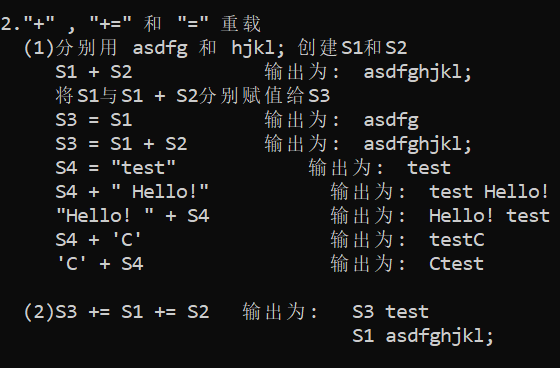
****

**图2.1.3输入中**

****

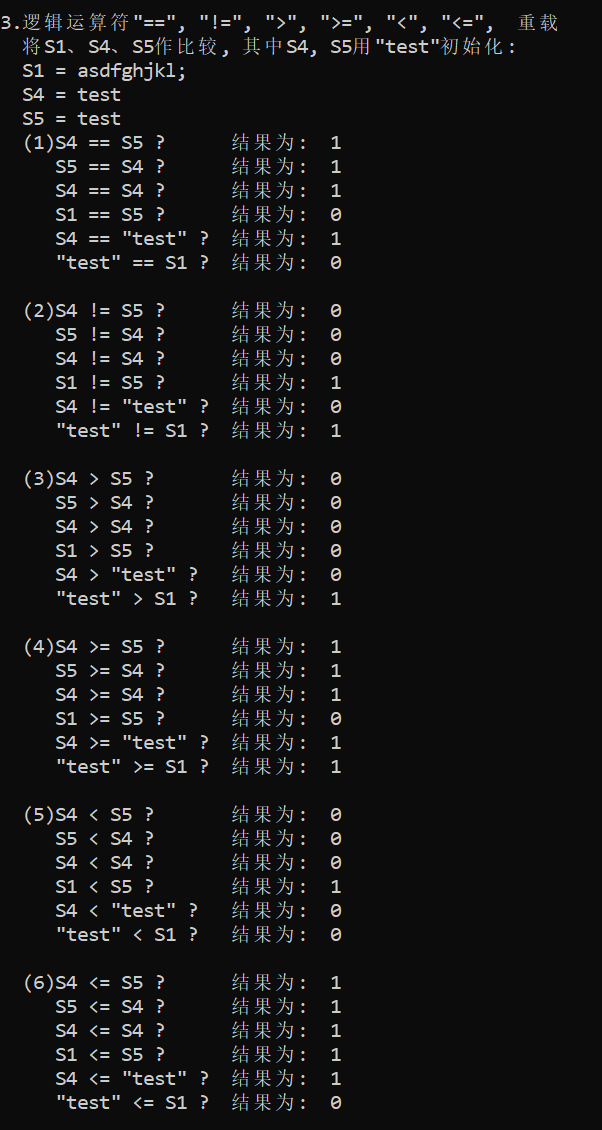
**图2.1.4输入后**

**（2）“+”“+=”“=”重载**

****

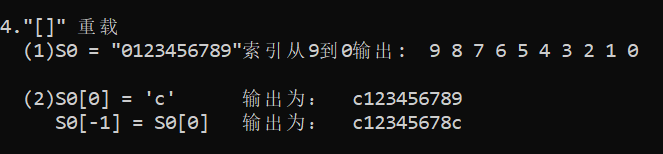
**图2.1.5“+”“+=”“=”测试**

**（3）“==”“！=”“>”“>=”“<”“<=”重载**

****

**图2.1.6“==”“！=”“>”“>=”“<”“<=”测试**

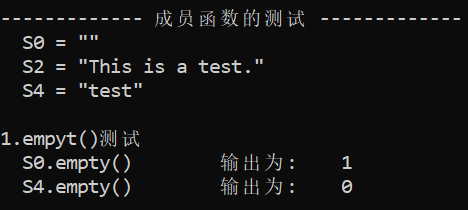
**（4）“[]”重载**

****

**图2.1.7“[]”测试**

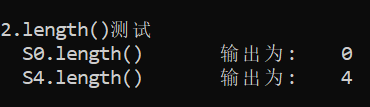
**3.其他成员函数**

**（1）empty()测试**

****

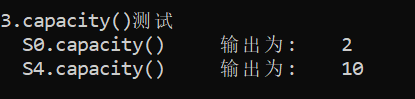
**图2.1.8empty()测试**

**（2）length()测试**

****

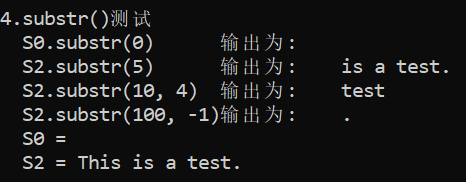
**图2.1.9 length()测试**

**（3）capacity()测试**

****

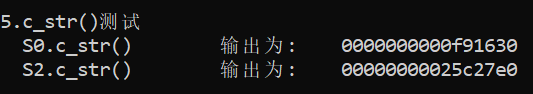
**图2.1.10 capacity()测试**

**（4）substr()测试**

****

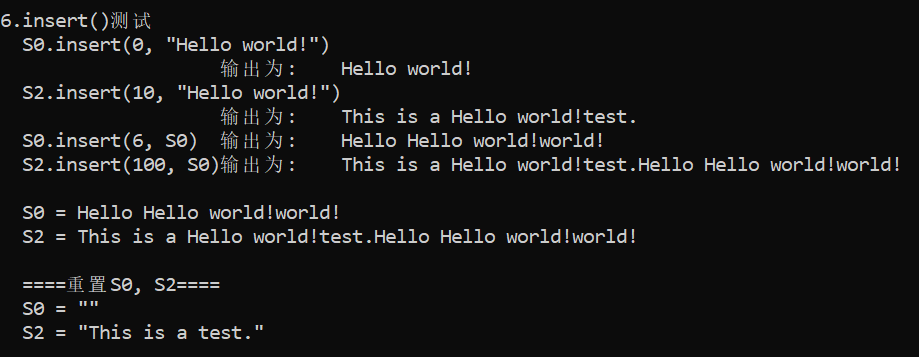
**图2.1.11 substr**(**)测试**

**（5）c\_str()测试**

****

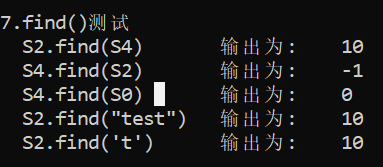
**图2.1.12 c\_str()测试**

**（6）insert()测试**

****

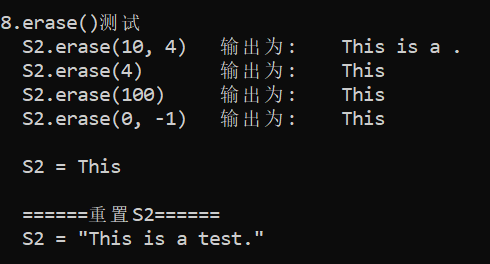
**图2.1.13 insert()测试**

**（7）find()测试**

****

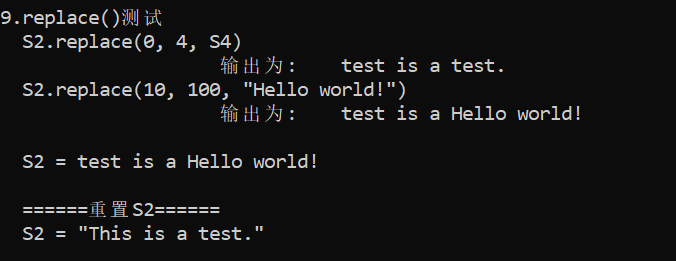
**图2.1.14 find()测试**

**（8）erase()测试**

****

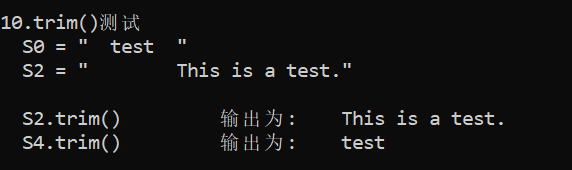
**图2.1.15 erase()测试**

**（9）replace()测试**

****

**图2.1.16 replace()测试**

**（10）trim()测试**

****

**图2.1.17trim()测试**

3 向量类模板

3.1 异常类ERROR类介绍

**3.1.1 成员变量介绍**

**（1）int type**

当type为1时，代表对vector对象的操作出现越界情况

当type为2时，代表vector相加时出现了维度不同无法相加的情况

**代码展示：**

class ERROR

{

private:

    int type; // 1:out of range

};

**3.1.2 成员函数介绍**

**1.构造函数**

**（1）构造函数**

构造时需要参数来确定异常类型

**代码展示：**

    ERROR(int n) { type = n; }

**（2）拷贝构造函数**

将type复制给新对象

**代码展示：**

    ERROR(const ERROR &a) { type = a.type; }

**2.其他成员函数**

**（1）print()**

根据type类型不同，输出具体错误内容

当type未知时输出"Unknown error!"

**代码展示：**

void print()

    {

        if (type == 1)

        {

            cout << "Out of range!";

            return;

        }

        else if (type == 2)

        {

            cout << "Size error! Different size of Vectors can't be added togerther!";

            return;

        }

        cout << "Unknown error!";

        return;

    }

3.2 向量类模板介绍

**3.2.1 成员变量介绍**

运用函数模板，便于存储各种数据类型的数据。data用于存储T类型数据的头指针， size记录向量元素个数，capacity记录向量容量大小。

**代码展示：**

template <typename T>

class Vector

{

private:

    T \*data;      //存储T类型数据的头指针

    int size;     //记录向量元素个数

    int capacity; //记录向量容量大小

}

**3.2.2 成员函数介绍**

**1.构造函数和析构函数**

**（1）无参构造函数**

创建空Vector对象：data指针置NULL，size = 0，capacity = 0

**代码展示：**

template <typename T>

Vector<T>::Vector()

{

    data = NULL;

    size = 0;

    capacity = 0;

}

**（2）有参构造函数**

创建大小为n的Vector对象，size = 0，capacity = 0

**代码展示：**

template <typename T>

Vector<T>::Vector(int n)

{

    data = new T[n]; //开辟大小为n的空间，并返回头指针

    size = 0;

    capacity = n;

}

**（3）拷贝构造函数**

以Vector v为模板创建Vector对象

**代码展示：**

template <typename T>

Vector<T>::Vector(const Vector &v)

{

    int n = v.capacity;

    data = new T[n]; //开辟大小为n的空间，并返回头指针

    for (int i = 0; i < n; i++)

    {

        data[i] = v.data[i];

    }

    size = v.size;

    capacity = v.capacity;

}

**（4）析构函数**

释放data数组的空间，并将data置为NULL

**代码展示：**

template <typename T>

Vector<T>::~Vector()

{

    delete[] data;

    data = NULL;

}

**2.运算符重载**

**（1）“+”重载**

当两个向量维数相同时，将两个向量中对应维数的元素相加，返回相加后的Vector对象，使得函数可以连续调用，否则抛出维数不同异常。

**代码展示：**

template <typename T>

Vector<T> Vector<T>::operator+(const Vector<T> &v) throw(ERROR)

{

    if (size != v.size) //向量维数不同，无法相加

    {

        throw ERROR(2); //抛出异常类对象

    }

    int n = size;

    Vector<T> v2(n);

    for (int i = 0; i < n; i++)

    {

        v2.data[i] = data[i] + v.data[i];

    }

    v2.capacity = capacity;

    v2.capacity = capacity;

    v2.size = size;

    return v2;

}

**（2）“+=”重载**

当两个向量维数相同时，将向量v中的元素加到\*this向量对应维数的元素上，返回值为Vector<T>&，使得函数可以连续调用，否则抛出维数不同异常。

**代码展示：**

template <typename T>

Vector<T> &Vector<T>::operator+=(const Vector<T> &v) throw(ERROR)

{

    if (size != v.size) //向量维数不同，无法相加

    {

        throw ERROR(2); //抛出异常类对象

    }

    else

    {

        int n = size;

        for (int i = 0; i < n; i++)

        {

            data[i] += v.data[i];

        }

    }

    return \*this;

}

**（3）“=”重载**

将向量v中所有成员变量的值赋值给\*this向量。若\*this向量的data指针不为空，则先释放原有内存，然后进行逐一赋值。返回值为Vector<T>&，使得函数可以连续调用。

**代码展示：**

template <typename T>

Vector<T> &Vector<T>::operator=(const Vector<T> &v)

{

    if (\*this == v)

        return \*this; //若自己赋值给自己，则直接返回

    int n = v.size;

    if (data != NULL) //先释放原先的空间

    {

        delete[] data;

        data = NULL;

    }

    data = new T[n];

    for (int i = 0; i < n; i++)

    {

        data[i] = v.data[i];

    }

    capacity = v.capacity;

    size = v.size;

    return \*this;

}

**（4）“<<”重载**

将向量中data数组的元素依次输出，返回值为ostream&，使得函数可以连续调用。

**代码展示：**

template <typename T>

ostream &operator<<(ostream &cout, Vector<T> &v)

{

    int n = v.size;

    for (int i = 0; i < n; i++)

    {

        if (i != n - 1)

        {

            cout << v.data[i] << " ";

        }

        else

        {

            cout << v.data[i];

        }

    }

    return cout;

}

**（5）“>>”重载**

从键盘读入数据，并将数据存储到向量的data数组中，返回值为istream&，使得函数可以连续调用。

**代码展示：**

template <typename T>

istream &operator>>(istream &cin, Vector<T> &v)

{

    T ele;

    while (cin >> ele)

    {

        v.push\_back(ele);

        if (cin.peek() == '\n')

            break;

    }

    return cin;

}

**（6）“==”重载**

将两个向量进行比较，若向量维数相同，且data数组中对应元素相同，则返回true，否则返回false。

**代码展示：**

template <typename T>

bool Vector<T>::operator==(const Vector<T> &v)

{

    if (size != v.size) //向量维数不同，一定不等

    {

        return false;

    }

    else

    {

        int is\_same = 1;

        for (int i = 0; i < size; i++)

        {

            if (data[i] != v.data[i])

            {

                is\_same = 0;

                break;

            }

        }

        if (is\_same == 1)

        {

            return true;

        }

        else

            return false;

    }

}

**（7）“!=”重载**

将两个向量进行比较，若向量维数相同，且data数组中对应元素相同，则返回false，否则返回true。

**代码展示：**

template <typename T>

bool Vector<T>::operator!=(const Vector<T> &v)

{

    if (\*this == v)

    {

        return false;

    }

    else

        return true;

}

**（8）“[]”重载**

用“[i]”可以访问向量data中第i号元素的值，返回值为T&，使得可以更改对应元素的值。若i越界，则抛出越界异常。

**代码展示：**

template <typename T>

T &Vector<T>::operator[](int i) throw(ERROR)

{

    if (i > (size - 1) || i < 0)

    {

        throw ERROR(1); //抛出越界异常

    }

    return data[i];

}

**3.其他成员函数**

**(1)push\_back(T )**

功能：用于在Vector对象的末尾增加一个指定类型的值

实现：如果空间足够，直接增加；如果空间不够，则new一片大小足够的新空间，并将原来的数据复制到新空间，接着delete旧空间，最后再增加新数据。

**代码展示：**

template <typename T>

void Vector<T>::push\_back(T ele)

{

    if (size < capacity)

    {

        data[size] = ele;

        size++;

    }

    else //如果空间不够，则new一片大小足够的新空间，并将原来的数据复制到新空间，接着delete旧空间

    {

        int n = size + 1;

        Vector<T> v(\*this);

        if (data != NULL)

        {

            delete[] data;

            data = NULL;

        }

        data = new T[n];

        for (int i = 0; i < size; i++)

        {

            data[i] = v.data[i];

        }

        capacity = n;

        data[size] = ele;

        size++;

    }

}

**(2) getsize()**

功能：返回Vector的长度

**代码展示：**

template <typename T>

int Vector<T>::getcapacity()

{

    return capacity;

}

**(3) getcapacity()**

功能：返回Vector实际占用的空间长度

**代码展示：**

template <typename T>

int Vector<T>::getcapacity()

{

    return capacity;

}

**(4) empty()**

功能：判断Vector是否为空

实现：如果size为0则返回true，否则返回false

**代码展示：**

template <typename T>

bool Vector<T>::empty()

{

    if (size)

        return false;

    return true;

}

**(5) front()**

功能：返回Vector首元素的引用

实现：如果Vector为空，新建一个ERROR类报错；否则返回data[0]

**代码展示：**

template <typename T>

T Vector<T>::front() throw(ERROR)

{

    if (this->size == 0)

        throw ERROR(1);

    else

        return this->data[0];

}

**(6)back()**

功能：返回Vector尾元素的引用

实现：如果Vector为空，新建一个ERROR类报错；否则返回data[size - 1]

**代码展示：**

template <typename T>

T Vector<T>::back() throw(ERROR)

{

    if (this->size == 0)

        throw ERROR(1);

    else

        return this->data[this->size - 1];

}

**(7)begin()**

功能：返回Vector头指针

实现：如果Vector为空，新建一个ERROR类报错；否则返回data

**代码展示：**

template <typename T>

T \*Vector<T>::begin() throw(ERROR)

{

    if (this->size == 0)

        throw ERROR(1);

    else

        return this->data;

}

**(8)end()**

功能：返回Vector尾指针

实现：如果Vector为空，新建一个ERROR类报错；否则返回data + size

**代码展示：**

template <typename T>

T \*Vector<T>::end() throw(ERROR)

{

    if (this->size == 0)

        throw ERROR(1);

    else

        return this->data + this->size;

}

**(9)pop\_back()**

功能：将尾元素删除

实现：如果Vector为空，新建一个ERROR类报错；否则将size – 1

**代码展示：**

template <typename T>

void Vector<T>::pop\_back() throw(ERROR)

{

    if (this->size == 0)

        throw ERROR(1);

    size--;

}

**(10) erase(T\* p)**

功能：将p指向的元素移除出Vector

实现：先判断p指向的元素是否在Vector中，不在则新建ERROR类报错；否则将地址为p + 1 至 data + size – 1 的元素向前移一位，并将size-1

**代码展示：**

template <typename T>

void Vector<T>::erase(T \*p) throw(ERROR)

{

    if (p >= this->end() || p < this->begin())

        throw ERROR(1);

    size--;

    for (T \*i = p; i < this->end(); i++)

        \*i = \*(i + 1);

    return;

}

**(11) clear()**

功能：清空Vector

实现：将size置为0

**代码展示：**

void Vector<T>::clear()

{

    size = 0;

    return;

}

**(12) insert(T\* pos, T ele)**

功能：在地址pos处插入元素ele

实现：先判断pos指向的地址是否在Vector中，不是则新建ERROR类报错；是则先调用push\_back()函数使Vector的容量足以存储size+1个元素，再将地址为pos至data+size-1的元素向后移动一位，最后在pos处写入ele

**代码展示：**

template <typename T>

void Vector<T>::insert(T \*pos, T ele) throw(ERROR)

{

    if (pos > this->end() || pos < this->begin())

        throw ERROR(1);

    this->push\_back(ele);

    for (T \*i = this->end() - 1; i > pos; i--)

    {

        \*i = \*(i - 1);

    }

    \*pos = ele;

    return;

}

**(13) find(T\* p, T ele)**

功能：从p处向后查询元素ele，直到data+size-1。若找到则返回该元素第一次出现的地址，若未找到则返回尾地址。

思路：先判断p指向的地址是否在Vector中，不是则新建ERROR类报错；否则通过while循环从p处逐个向后查询，找到则直接返回地址，若未找到则返回尾地址。

**代码展示：**

template <typename T>

T \*Vector<T>::find(T \*p, T ele) throw(ERROR)

{

    if (p >= this->end() || p < this->begin())

        throw ERROR(1);

    while (p != this->end())

    {

        if (\*p == ele)

            break;

        p++;

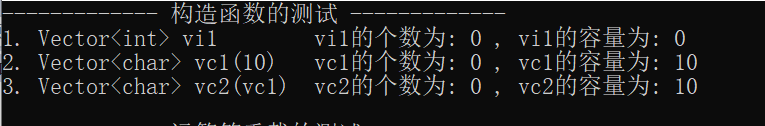
    }

    return p;

}

**3.2.3 测试**

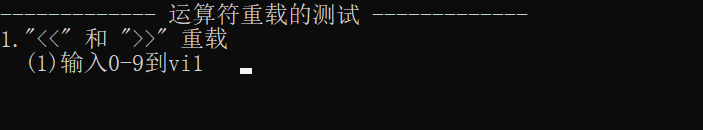
**1.构造函数**

****

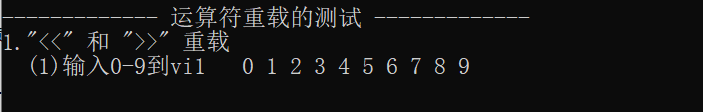
**图3.2.1 构造函数的测试**

**2.运算符重载**

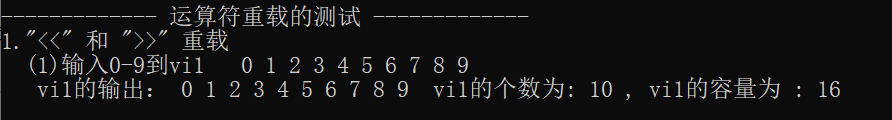
**（1）“<<”和“>>”重载**

****

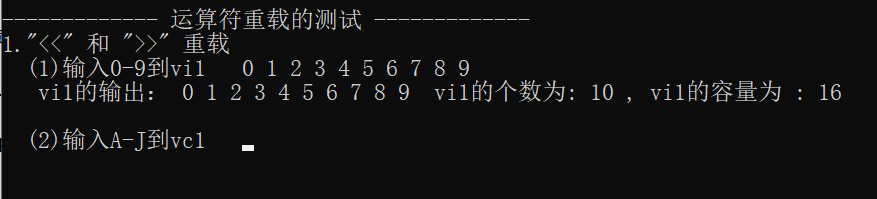
**图3.2.2 vi1输入前**

****

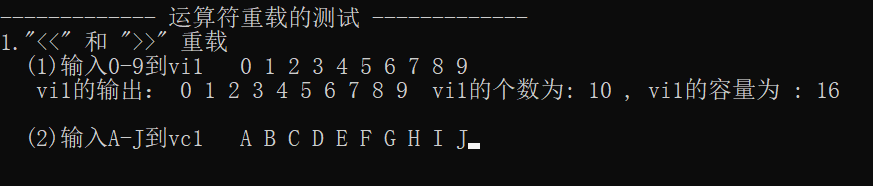
**图3.2.3 vi1输入中**

****

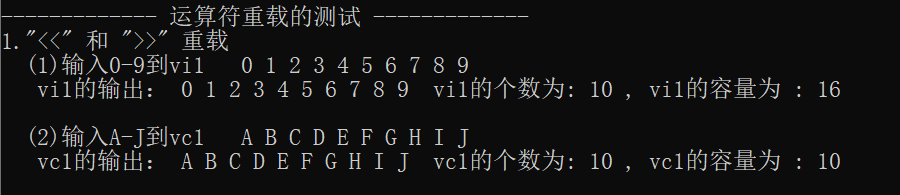
**图3.2.4 vi1输入后**

****

**图3.2.5 vc1输入前**

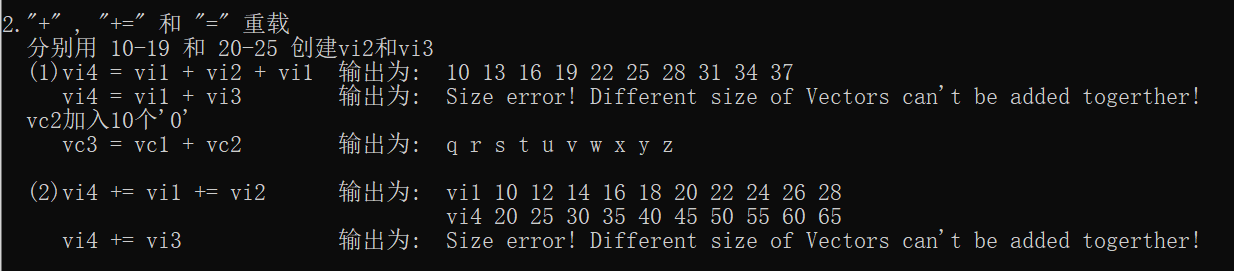
****

**图3.2.6 vc1输入中**

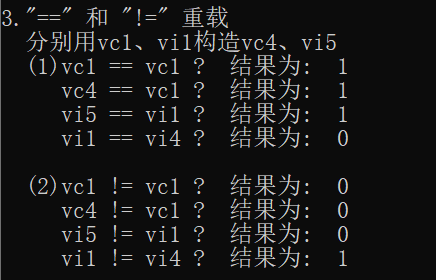
****

**图3.2.7 vc1输入后**

**（2）“+”和“+=”和“=”重载**

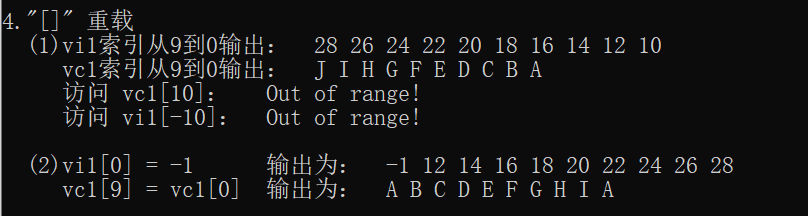
**图3.2.8 “+”和”+=”和”=”测试**

**（3）“==”和“!=”重载**

****

**图3.2.9 “==”和“！=”测试**

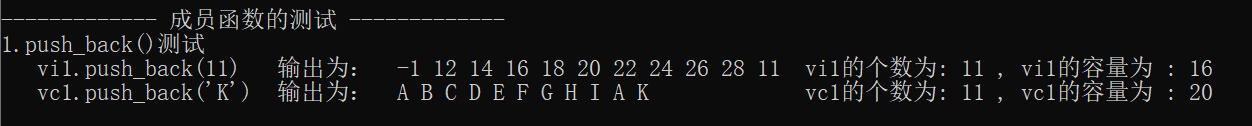
**（4）“[]”重载**

****

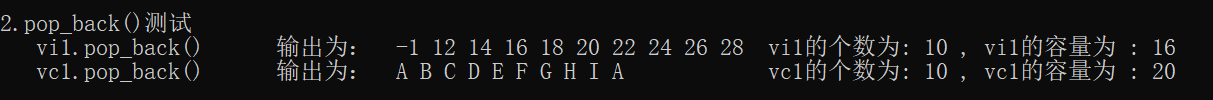
**图3.2.10 “[]”测试**

**3.其他成员函数**

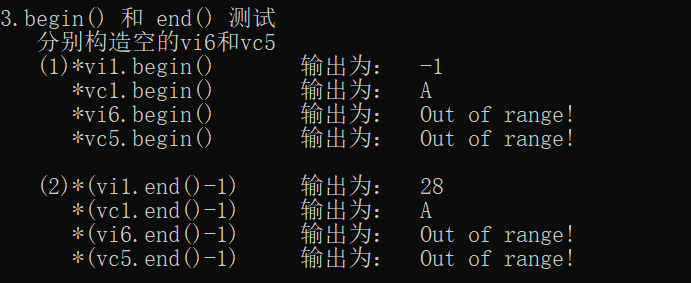
**（1）push\_back()测试**

**图3.2.11 push\_back()测试**

**（2）pop\_back()测试**

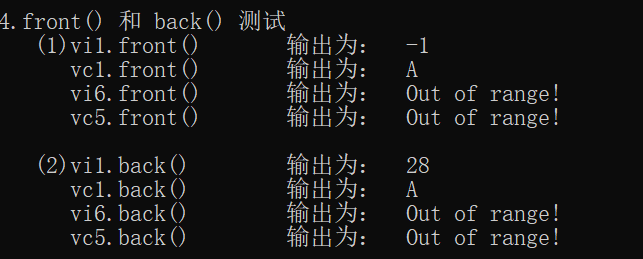
**图3.2.12 pop\_back()测试**

**（3）begin()和end()测试**

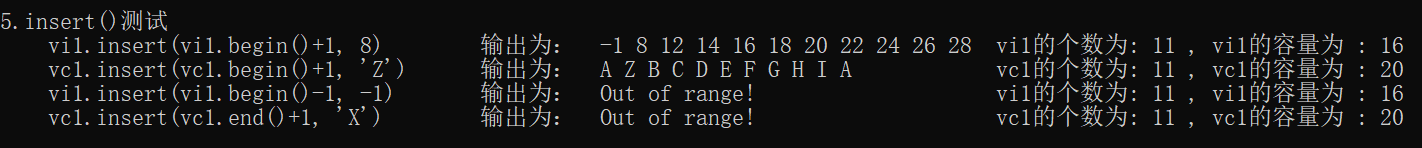
****

**图3.2.13 begin()和end()测试**

**（4）front()和back()测试**

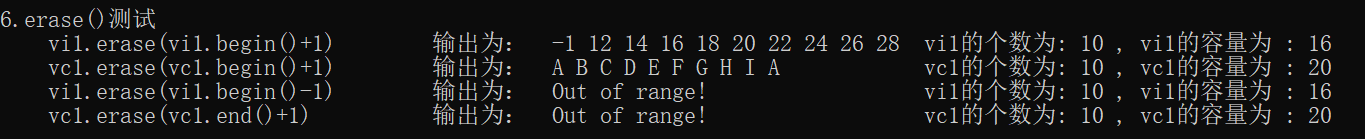
****

**图3.2.14 front()和back()测试**

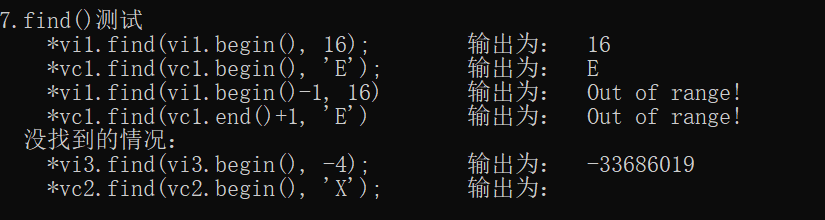
**（5）insert()测试**

**图3.2.15 insert()测试**

**（6）erase()测试**

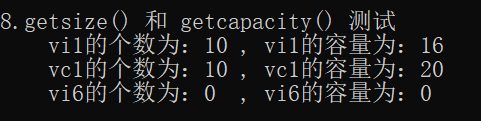
**图3.2.16 erase()测试**

**（7）find()测试**

****

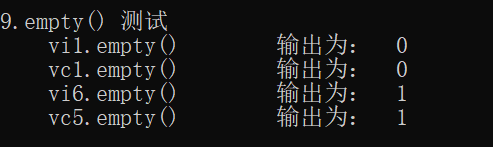
**图3.2.17 find()测试**

**（8）getsize()和getcapacity()测试**

****

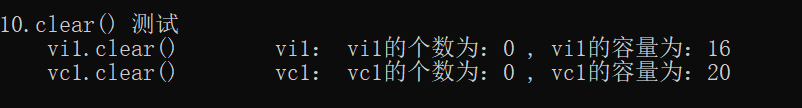
**图3.2.18 getsize()和getcapacity()测试**

**（9）empty()测试**

****

**图3.2.19 empty()测试**

**（10）clear()测试**

****

**图3.2.20 clear()测试**

4 小组分工

组长：周孙睿：Vector类模板构造和析构、运算符重载的编写，以及Vector类的测试

组员：施安然：Vector类模板其他成员函数，以及其异常类的编写

陈诺：部分成员函数的编写，对内存调用方法进一步优化，以及String类的测试

苏亿铭：String类构造、析构函数以及部分成员函数的编写

5 结语

在本次实验中，我们自己动手编写了string类和向量类模板，实现了它们大部分的成员函数(包括运算符重载和其他成员函数)，使我们对于他们的底层原理有了进一步的了解。此外，我们还编写了他们各自的异常类，用于处理他们各自成员函数中不同的异常信息，同时也便于今后的维护。本实验将我们之前所学的知识点：类的构造和析构、运算符重载、异常处理等全部有机结合了起来，使我们在进行项目编写的同时，又复习巩固了一遍，加深我们对于他们的理解。