**C++ 语 言 程 序 设 计**

实

验

报

告

实 验 三

姓名： 刘健恒

学号： SZ170320207

班级： 自动化2班

**一 实验项目**

* 1. 基于模板，编写vector类，实现动态的建立，插入，删除等
  2. 调用不同的构造函数建立不同的vector
  3. 测试vector的插入删除等操作
  4. 附上vector类的代码，测试代码，以及实验结果。

**二 实验原理**

1. 说明（1）为什么要重载赋值运算符 = ；（2）重载时需要注意哪些问题？（3）为什么要有返回值？（4）如果出现类似于执行a=a的情况，代码应如何处理保证操作正确。
2. 标准库中=只可对单个数值进行赋值，若不进行重载，则赋值为地址，实际左值的改变会影响右值，因此需要重载创建新的地址给左值。
3. 重载时应对左值进行判断是否存在足够空间，若无则需要新建空间存放
4. 若出现a=b+c的情况，需要通过返回值对数值进行暂存以保证运行。
5. 由于对=进行了函数重载，使用=时会存在空间变换，因此需确保数组空间存在，需要新建空间保证存取数值安全。
6. 说明重载[]的两个函数的异同，以及应用场景？

目标功能相同，使用const修饰函数使得即使对常成员也可进行读取及引用，而无const修饰的函数则无法对常成员进行上述操作，但对非常成员可以。

由此可以确保希望被保护的数据不被意外修改。

1. 针对自己的代码，考虑运行效率问题，以及可以如何提升效率？

大概可以减少判断的使用，减少空间的增减处理，减少循环的重复使用。初始化时直接给数据赋值，非在初始化函数中赋值。少使用i++而使用++i。

#ifndef VECTOR\_SS\_H

#define VECTOR\_SS\_H

#include<iostream>

using namespace std;

template<class T>

class vector\_ss

{

private:

int v\_size;//记录当前数组大小

T\* buf; //存放数组空间首地址

public:

void push\_back(T t)

{

if((sizeof(buf)<v\_size+1)||(sizeof(buf)==0))

{

T\* temp = buf;

buf = new T[v\_size+1];

for (int i=0;i<v\_size;i++)

buf[i] = temp[i];

delete []temp;

}

buf[v\_size] = t;

v\_size++;

};//在数组末尾插入元素

T pop\_back()

{

buf[v\_size-1] = 0;

v\_size--;

return buf[v\_size];

};//删除并返回末尾元素

T front()

{

return buf[0];

};//返回数组首元素

void clear()

{

delete []buf;

buf = new T[0];

v\_size = 0;

};//清空数组

bool empty()

{

if (v\_size == 0)

return 1;

else

return 0;

};//返回数组是否为空

void insert(int pos,T data)

{

v\_size++;

if((sizeof(buf)<v\_size)||(sizeof(buf)==0))

{

T\* temp = buf;

buf = new T[v\_size];

for (int i=0;i<v\_size;i++)

buf[i] = temp[i];

delete []temp;

}

for (int i = v\_size-1;i>=pos-1;i--)

{

buf[i] = buf[i-1];

}

buf[pos-1] = data;

};//在数组的第pos个位置插入data

void erase(int pos)

{

v\_size--;

for (int i = pos-1;i<v\_size;i++)

{

buf[i] = buf[i+1];

}

};//删除pos位置的数据

void erase(int begin, int end)

{

v\_size -= end - begin + 1;

for (int i = begin -1;i<v\_size;i++)

{

buf[i] = buf[i+end-begin+1];

}

};//删除从begin位置到end位置的数据

const int& size() const {return v\_size;};//返回数组的大小

const vector\_ss& operator=(const vector\_ss & v)

{

v\_size = v.size();

if((sizeof(buf)<v\_size)||(sizeof(buf)==0))

{

T\* temp = buf;

buf = new T[v\_size];

for (int i=0;i<v\_size;i++)

buf[i] = temp[i];

delete []temp;

}

for (int i=0;i<v\_size;i++)

{

buf[i] = v[i];

}

};//重载幅值函数

const T& operator()(int i) const {return buf[i];};//重载()

T& operator[](int n){return buf[n];};//重载[]

const T& operator[](int n) const {return buf[n];}; //重载[]

void display()

{

for(int i=0;i<v\_size;i++)

cout<<buf[i]<<" ";

cout<<endl;

};

public:

vector\_ss()

{

buf = new T[0];

v\_size = 0;

};//构造函数

vector\_ss(int n)

{

buf = new T[n];

v\_size = n;

};//构造函数，产生一个大小为n的数组

vector\_ss(int n, T t)

{

buf = new T[n];

v\_size = n;

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

buf[i] = t;

};//构造函数，产生一个大小为n的数组，且每个元素均为t

vector\_ss(const vector\_ss &v)

{

v\_size = v.size();

buf = new T[v\_size];

for(int i=0;i<v\_size;i++)

{

buf[i] = v[i];v\_size = v.size();

}

};//拷贝构造函数

vector\_ss(const vector\_ss &v, int start, int end)

{

v\_size = end + 1 - start;

buf = new T[v\_size];

for(int i=0;i<v\_size;i++)

buf[i] = v[start-1+i];

};

//拷贝构造函数，数组元素为v的start位置到end位置的元素

~vector\_ss()

{

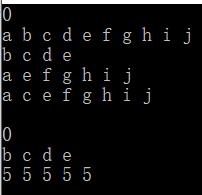
delete []buf;

};//析构函数

};

#endif // VECTOR\_SS\_H

实验结果：



**三 实验总结与建议**

（总结实验实施过程，说明实验过程中遇到的问题与解决方案；提出实验环节的建议）

本次实验学习使用了动态数组，一开始打算一次性创建足够数组便能轻松完成，然而可能存在越界问题，由此还是重新编写，添加数组大小判别，数组的创建与删除等等，最终完善程序。

其中也学习了常函数，常重载函数等等受益匪浅。