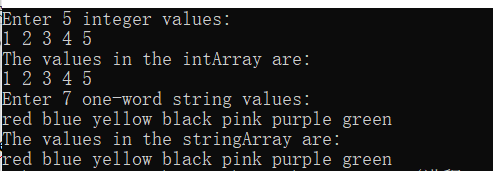
EX1



#include<iostream>

#include<string>

using namespace std;

template <typename elementType,int numberOfElements>

class Array

{

elementType\* target;

public:

Array()

{

//target = new elementType[numberOfElements]{0};

target = new elementType[numberOfElements];

}

elementType& operator[](int index) const

{

return \*(target + index);

}

};

int main()

{

Array<int, 5> myInt;

/\*for (int i = 0; i < 5; i++)

cout << (myInt[i] = i + 1) << " ";\*/

//cout << endl << endl;

cout << "Enter 5 integer values: \n";

for (int i = 0; i < 5; i++)

cin >> myInt[i];

cout << "The values in the intArray are:\n";

for (int i = 0; i < 5; i++)

cout << myInt[i] << " ";

cout << endl;

cout << "Enter 7 one-word string values: \n";

Array<string, 7>myString;

string temp;

for (int i = 0; i < 7; i++)

{

cin >> temp;

myString[i] = temp;

//cout << (myString[i] = temp) << endl;

}

cout << "The values in the stringArray are:\n";

for (int i = 0; i < 7; i++)

{

cout << myString[i] << " ";

}

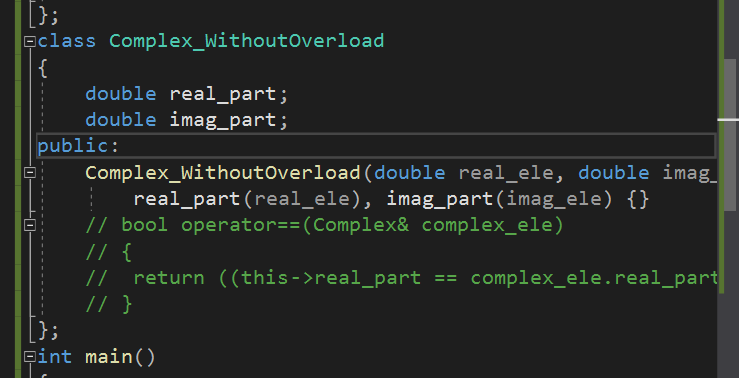
}

问题与思考：

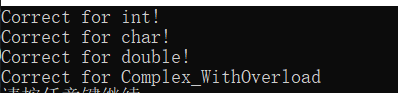
new的时候调用{}初始化调用默认构造，不是很方便，真要初始化还得for循环；

EX2





不重载==的类放到isEqualTo的模板函数里面过不了编译，已经发现比较不了了。



#include<iostream>

using namespace std;

template<typename elementType>

bool isEqualTo(elementType left\_element, elementType right\_element)

{

return (left\_element == right\_element);

}

class Complex\_WithOverload

{

double real\_part;

double imag\_part;

public:

Complex\_WithOverload(double real\_ele, double imag\_ele) :

real\_part(real\_ele), imag\_part(imag\_ele) {}

bool operator==(Complex\_WithOverload& complex\_ele)

{

return ((this->real\_part == complex\_ele.real\_part) && (this->imag\_part == complex\_ele.imag\_part));

}

};

class Complex\_WithoutOverload

{

double real\_part;

double imag\_part;

public:

Complex\_WithoutOverload(double real\_ele, double imag\_ele) :

real\_part(real\_ele), imag\_part(imag\_ele) {}

// bool operator==(Complex& complex\_ele)

// {

// return ((this->real\_part == complex\_ele.real\_part) && (this->imag\_part == complex\_ele.imag\_part));

// }

};

int main()

{

if (!isEqualTo(int(5), int(7)) && isEqualTo(int(5), int(5)))

cout << "Correct for int!\n";

if (!isEqualTo(char('a'), char('b')) && isEqualTo(char('a'), char('a')))

cout << "Correct for char!\n";

if (!isEqualTo(double(1.2), double(2.2)) && isEqualTo(double(2.2), double(2.2)))

cout << "Correct for double!\n";

if (!isEqualTo(Complex\_WithOverload(4, 5), Complex\_WithOverload(3, 7)) && isEqualTo(Complex\_WithOverload(1, 1), Complex\_WithOverload(1, 1)))

cout << "Correct for Complex\_WithOverload\n";

system("pause");

//过不了编译

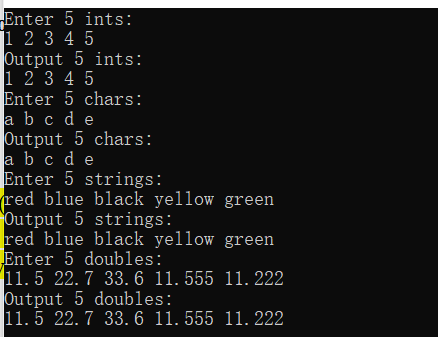
//isEqualTo(Complex\_WithoutOverload(3, 7), Complex\_WithoutOverload(5, 4));

}

问题与思考：

函数模板与类模板得进行区分，用两个template。

EX3



#include<iostream>

#include<string>

using namespace std;

template<typename elementType>

class Vector

{

elementType\* iterator;

int Vector\_size;

template<typename elementType2>

friend ostream& operator<<(ostream& cout, const Vector<elementType2>& target);

template<typename elementType3>

friend istream& operator>>(istream& cin, const Vector<elementType3>& target);

public:

Vector()

{

iterator = nullptr;

Vector\_size = 0;

}

Vector(unsigned int size)

{

if (!size)

{

Vector();

return;

}

iterator = new elementType[size];

Vector\_size = size;

}

//int size()

//{

// return Vector\_size;

//}

//elementType& operator[](int i)

//{

// return \*(iterator + i);

//}

};

template<typename elementType>

ostream& operator<<(ostream& cout, const Vector<elementType>& target)

{

for (int i = 0; i < target.Vector\_size; i++)

cout << \*(target.iterator + i) << " ";

return cout;

}

template<typename elementType>

istream& operator>>(istream& cin, const Vector<elementType>& target)

{

for (int i = 0; i < target.Vector\_size; i++)

cin >> (\*(target.iterator + i));

return cin;

}

int main()

{

Vector<int>Coli(5);

printf("Enter 5 ints:\n");

cin >> Coli;

printf("Output 5 ints:\n");

cout << Coli;

cout << endl;

Vector<char>Colc(5);

printf("Enter 5 chars:\n");

cin >> Colc;

printf("Output 5 chars:\n");

cout << Colc;

cout << endl;

Vector<string>Cols(5);

printf("Enter 5 strings:\n");

cin >> Cols;

printf("Output 5 strings:\n");

cout << Cols;

cout << endl;

Vector<double>Cold(5);

printf("Enter 5 doubles:\n");

cin >> Cold;

printf("Output 5 doubles:\n");

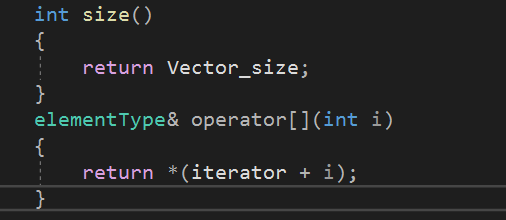
cout << Cold;

cout << endl;

}

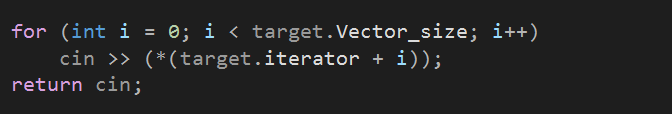
问题与思考：

这里因为要求重载流插入>>和流提取<<运算符，所以直接操作整个对象，不然我会这么写：



在外就通过for循环对小于size范围内进行任意读写操作。

char\*类型当其中一种类型发现出现了问题，因为我没有为char\*赋予指向的内存。



（外部）函数模板得和类的区分开，不然编译是不给过的（不可能使得创建一个特定类型类而将外部函数的参数类型确定下来，模板会显得无意义）