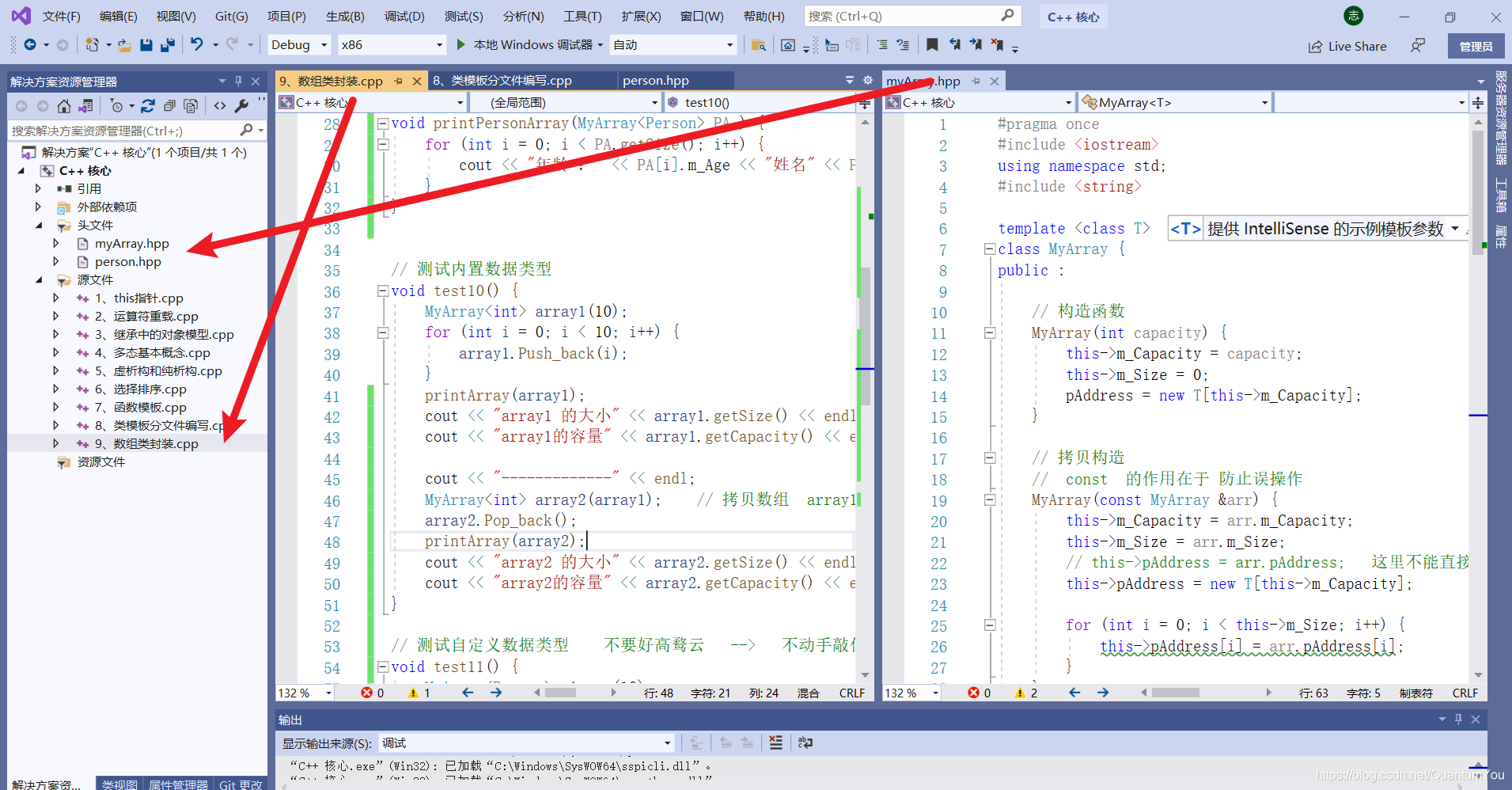
@[toc]

第一次学习要为第二次学习做准备

# 需求分析

* 可以对内置数据类型以及自定义数据类型的数据进行存储
* 将数组中的数据存储到堆区
* 构造函数中可以传入数组的容量
* 提供对应的拷贝构造函数以及operator=防止浅拷贝问题
* 提供尾插法和尾删法对数组中的数据进行增加和删除
* 可以通过下标的方式访问数组中的元素
* 可以获取数组中当前元素个数和数组的容量



# 代码

## myArray.cpp

#pragma once   
#include <iostream>  
using namespace std;  
#include <string>  
  
template <class T>  
class MyArray {  
public :  
  
 // 构造函数  
 MyArray(int capacity) {  
 this->m\_Capacity = capacity;  
 this->m\_Size = 0;  
 pAddress = new T[this->m\_Capacity];  
 }  
  
 // 拷贝构造  
 // const 的作用在于 防止误操作  
 MyArray(const MyArray &arr) {  
 this->m\_Capacity = arr.m\_Capacity;  
 this->m\_Size = arr.m\_Size;  
 // this->pAddress = arr.pAddress; 这里不能直接这样操作(这是指针，普通则可以) 浅拷贝  
 this->pAddress = new T[this->m\_Capacity];  
  
 for (int i = 0; i < this->m\_Size; i++) {  
 this->pAddress[i] = arr.pAddress[i];  
 }  
 }  
  
  
 // 重载 = 防止浅拷贝的问题  
 MyArray& operator=(const MyArray& myarray) {  
 // 判断原来堆区是否有数据，有则释放数据  
 if (this->pAddress != NULL) {  
 delete[] this->pAddress;  
 this->m\_Size = 0;  
 this->m\_Capacity = 0;  
 }  
  
 this->m\_Capacity = myarray.m\_Capacity;  
 this->m\_Size = myarray.m\_Size;  
 this->pAddress = new T[this->m\_Capacity];  
  
 for (int i = 0; i < this->m\_Size; i++) {  
 this->pAddress[i] = myarray.pAddress[i];  
 }  
 return \*this;  
 }  
  
// 重载 []   
 T& operator[](int index) {  
 return this->pAddress[index];  
}  
  
// 尾插法  
 void Push\_back(const T& val) {  
 if (this->m\_Capacity == this->m\_Size) {  
 return;  
 }  
 this->pAddress[this->m\_Size] = val;  
 this->m\_Size++ ;  
}  
  
// 尾删法  
 void Pop\_back() {  
 if (this->m\_Size == 0) return;  
 this->m\_Size--;  
 }  
  
 int getCapacity() {  
 return this->m\_Capacity;  
 }  
 int getSize() {  
 return this->m\_Size;  
 }  
  
  
 ~MyArray() {  
 if (this->pAddress != NULL) {  
 delete[] this->pAddress;  
 this->pAddress = NULL;  
 this->m\_Capacity = 0;  
 this->m\_Size = 0;  
 }  
 }  
private :  
 T\* pAddress; // 指向一个堆空间 ，存储真正的数据  
 int m\_Capacity ;   
 int m\_Size;  
};

## 数组类封装主代码

#include <iostream>  
using namespace std;  
#include "myArray.hpp"  
#include <string>  
  
  
void printArray(MyArray<int> arr) {  
 for (int i = 0; i < arr.getSize(); i++) {  
 cout << arr[i] << " ";  
 }  
 cout << endl;  
}  
  
  
// 测试自定义数据类型  
class Person {  
public:  
 Person() {}; // 无参数构造  
 Person(string name, int age) {  
 this->m\_Age = age;  
 this->m\_Name = name;  
 };  
public:  
 string m\_Name;  
 int m\_Age;  
};  
// 输出  
void printPersonArray(MyArray<Person> PA ) {  
 for (int i = 0; i < PA.getSize(); i++) {  
 cout << "年龄 ：" << PA[i].m\_Age << "姓名" << PA[i].m\_Name << endl;  
 }  
}  
  
  
// 测试内置数据类型  
void test10() {  
 MyArray<int> array1(10);  
 for (int i = 0; i < 10; i++) {  
 array1.Push\_back(i);  
 }  
 printArray(array1);  
 cout << "array1 的大小" << array1.getSize() << endl;  
 cout << "array1的容量" << array1.getCapacity() << endl;  
   
 cout << "-------------" << endl;  
 MyArray<int> array2(array1); // 拷贝数组 array1   
 array2.Pop\_back();  
 printArray(array2);  
 cout << "array2 的大小" << array2.getSize() << endl;  
 cout << "array2的容量" << array2.getCapacity() << endl;  
}  
  
// 测试自定义数据类型 不要好高鹜云 --> 不动手敲代码  
void test11() {  
 MyArray<Person> pArray(10);  
 Person p1("周杰伦",12);  
 Person p2("张杰", 11);  
 Person p3("周杰", 13);  
 Person p4("钟杰", 21);  
  
 pArray.Push\_back(p1);  
 pArray.Push\_back(p2);  
 pArray.Push\_back(p3);  
 pArray.Push\_back(p4);  
  
 printPersonArray(pArray);  
 cout << "pArray 的大小" << pArray.getSize() << endl;  
 cout << "pArray的容量" << pArray.getCapacity() << endl;  
}  
  
int main() {  
  
 test10();  
 test11();  
  
 system("pause");  
 return 0;  
}

# 结果

