计算机学院 高级程序语言设计 课程实验报告

实验题目：模板泛型编程，群体类、线性群体类		学号：232300130150
日期：2024.4.25	班级： 23.4	姓名： 王成意

实验目的：
了解并实践模板泛型编程。

实验步骤与内容：
1. 实践第 9 章 PPT，P6-7，例 9-1，函数模板。
代码：

```
#include <iostream>

using namespace std;

template <class T> // 定义函数模板

void outputArray(const T *array, int count)

{

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

        cout << array[i] << " ";

    cout << endl;

}

int main()

{ // 主函数

    const int A_COUNT = 8, B_COUNT = 8, C_COUNT = 20;

    int a[A_COUNT] = {1, 2, 3, 4, 5, 6, 7,

8};                    // 定义 int 数组
```

```
    double b[B_COUNT] = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6,
7.7, 8.8}; // 定义 double 数组
    char c[C_COUNT] = "Welcome to see
you!";                      // 定义 char 数组
    cout << " a array contains:" << endl;
    outputArray(a, A_COUNT); // 调用函数模板,int
    cout << " b array contains:" << endl;
    outputArray(b, B_COUNT); // 调用函数模板，double
    cout << " c array contains:" << endl;
    outputArray(c, C_COUNT); // 调用函数模板，char
    return 0;
}
```

实践结果：

```
  xe' '--interpreter=mi'
  a array contains:
1 2 3 4 5 6 7 8
 b array contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8
 c array contains:
W e l c o m e   t o   s e e   y o u !
PS D:\C++ programs>
```

验证了函数模板的广泛适用性。

2. 实践第 9 章 PPT，P11-13，例 9-2，类模板。

代码：

```
#include <iostream>
#include <cstdlib>
using namespace std;
```

```
struct Student
{
    int id;    // 学号
    float gpa; // 平均分
};
template <class T>
class Store
{ // 类模板：实现对任意类型数据进行存取
private:
    T item;         // item 用于存放任意类型的数据
    bool haveValue; // haveValue 标记 item 是否已被存入内
容
public:
    Store();                  // 缺省形式（无形参）的构造
函数
    T &getElem();             // 提取数据函数
    void putElem(const T &x); // 存入数据函数
};
template <class T> // 默认构造函数的实现
Store<T>::Store() : haveValue(false)
{
}
```

```cpp
template <class T> // 提取数据函数的实现
T &Store<T>::getElem()
{
    // 如试图提取未初始化的数据，则终止程序
    if (!haveValue)
    {
        cout << "No item present!" << endl;
        exit(1); // 使程序完全退出，返回到操作系统。
    }
    return item; // 返回 item 中存放的数据
}
template <class T> // 存入数据函数的实现
void Store<T>::putElem(const T &x)
{
    // 将 haveValue 置为 true，表示 item 中已存入数值
    haveValue = true;
    item = x; // 将 x 值存入 item
}
int main()
{
    Store<int> s1, s2;
    s1.putElem(3);
```

```
    s2.putElem(-7);
    cout << s1.getElem() << "  " << s2.getElem() <<
endl;

    Student g = {1000, 23};
    Store<Student> s3;
    s3.putElem(g);
    cout << "The student id is " << s3.getElem().id <<
endl;

    Store<double> d;
    cout << "Retrieving object D... ";
    cout << d.getElem() << endl;
    // 由于 d 未经初始化,在执行函数 D.getElement()过程中导
致程序终止
    return 0;
}
```

实践结果：

```
3  -7
The student id is 1000
Retrieving object D... No item present!
PS D:\C++ programs>
```

，验证了类模板的广泛适用性。

3. 实践第 9 章 PPT，P18-22 例 9-3，Array.h 直接访问数组类声明；

代码：

```
#ifndef ARRAY_H
#define ARRAY_H
#include <cassert>
template <class T> // 数组类模板定义
class Array
{
private:
    T *list;  // 用于存放动态分配的数组内存首地址
    int size; // 数组大小（元素个数）
public:
    Array(int sz = 50);                          // 构造函
数
    Array(const Array<T> &a);                    // 拷贝构
造函数
    ~Array();                                    // 析构函
数
    Array<T> &operator=(const Array<T> &rhs); // 重载
"="
    T &operator[](int i);                        // 重载
"[]"
    const T &operator[](int i) const;
    operator T *(); // 重载到 T*类型的转换
```

```
    operator const T *() const;
    int getSize() const; // 取数组的大小
    void resize(int sz); // 修改数组的大小
};
template <class T>
Array<T>::Array(int sz)
{                       // 构造函数
    assert(sz >= 0);    // sz 为数组大小（元素个数），应当非负
    size = sz;          // 将元素个数赋值给变量 size
    list = new T[size]; // 动态分配 size 个 T 类型的元素空间
}
template <class T>
Array<T>::~Array()
{ // 析构函数
    delete[] list;
}
template <class T>
Array<T>::Array(const Array<T> &a)
{                   // 拷贝构造函数
```

```
    size = a.size; // 从对象 x 取得数组大小，并赋值给当前对象的成员
    // 为对象申请内存并进行出错检查
    list = new T[size];             // 动态分配 n 个 T 类型的元素空间
    for (int i = 0; i < size; i++) // 从对象 X 复制数组元素到本对象
        list[i] = a.list[i];
}
// 重载"="运算符，将对象 rhs 赋值给本对象。实现对象之间的整体赋值
template <class T>
Array<T> &Array<T>::operator=(const Array<T> &rhs)
{
    if (&rhs != this)
    {
        // 如果本对象中数组大小与 rhs 不同，则删除数组原有内存，然后重新分配
        if (size != rhs.size)
        {
            delete[] list;      // 删除数组原有内存
            size = rhs.size;    // 设置本对象的数组大小
```

```
            list = new T[size]; // 重新分配 n 个元素的内存
        }
        // 从对象 x 复制数组元素到本对象
        for (int i = 0; i < size; i++)
            list[i] = rhs.list[i];
    }
    return *this; // 返回当前对象的引用
}
// 重载下标运算符，实现与普通数组一样通过下标访问元素，并且具有越界检查功能
template <class T>
T &Array<T>::operator[](int n)
{
    assert(n >= 0 && n < size); // 检查下标是否越界
    return list[n];             // 返回下标为 n 的数组元素
}
template <class T>
const T &Array<T>::operator[](int n) const
{
    assert(n >= 0 && n < size); // 检查下标是否越界
    return list[n];             // 返回下标为 n 的数组元素
```

```
}
// 重载指针转换运算符，将 Array 类的对象转换为 T 类型的指针
template <class T>
Array<T>::operator T *()
{                  // 转换为 T 类型的指针 T*的类型转换函数
    return list; // 返回当前对象中私有数组的首地址
}
template <class T>
Array<T>::operator const T *() const
{
    return list; // 返回当前对象中私有数组的首地址
}
// 取当前数组的大小
template <class T>
int Array<T>::getSize() const
{
    return size;
}
// 将数组大小修改为 sz
template <class T>
void Array<T>::resize(int sz)
{
```

```
    assert(sz >= 0); // 检查 sz 是否非负
    if (sz == size)  // 如果指定的大小与原有大小一样，什么也不做
        return;
    T *newList = new T[sz];          // 申请新的数组内存
    int n = (sz < size) ? sz : size; // 将 sz 与 size 中较小的一个赋值给 n
    // 将原有数组中前 n 个元素复制到新数组中
    for (int i = 0; i < n; i++)
        newList[i] = list[i];
    delete[] list;  // 删除原数组
    list = newList; // 使 list 指向新数组
    size = sz;      // 更新 size
}
#endif // ARRAY_H
```

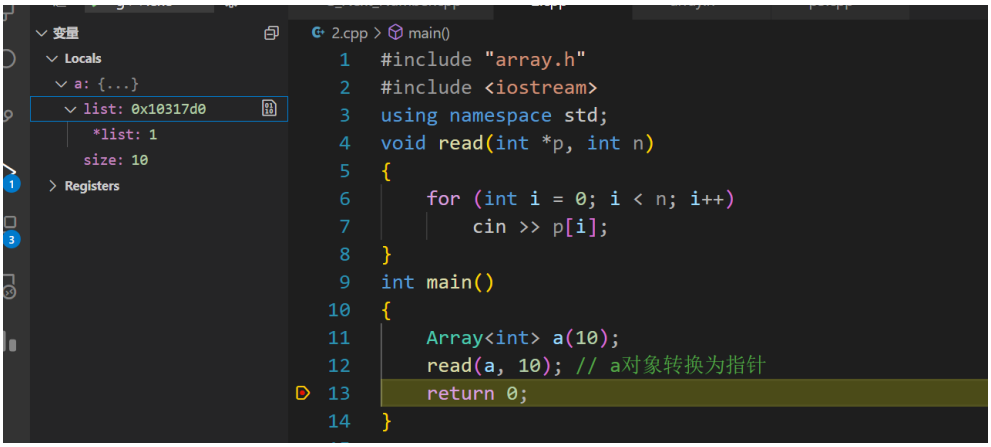
问：为何很多成员函数有两种声明（const）？

```
T &operator[](int i);                    //重载"[]"
const T &operator[](int i) const;
operator T *(); //重载到T*类型的转换
operator const T *() const;
```

答：

当我们外部对象定义为 const 对象的时候，编译器则会自动使用 const 形式的成员函数，而且 const 成员函数的返回值必须为 const，const 成员函数虽然能让 const 对象和非 const 外部对象都访问，但是他的返回值为 const，不是我们期望的，因为我们的非 const 对象我们是期望它能修改的其中元素的，而返回了 const 类型，又无法修改元素。也就是说只有两种声明才能满足要求。

实践 P25 页 PPT 右栏的例子。



说明运行成功。

实践第 9 章 PPT，P27，例 9-4，利用上面的动态数组类求质数。
代码（main）：

```
#include <iostream>

#include <iomanip>

#include "array.h"

using namespace std;

int main()

{

    Array<int> a(10); // 用来存放质数的数组，初始状态有

10 个元素。

    int n, count = 0;

    cout << "Enter a value >= 2 as upper limit for

prime numbers: ";

    cin >> n;

    for (int i = 2; i <= n; i++)

    {
```

```
        bool isPrime = true;
        for (int j = 0; j < count; j++) // 检查 i 是否能
被比它小的质数整除
            if (i % a[j] == 0)
            { // 若 i 被 a[j]整除，说明 i 不是质数 20
                isPrime = false;
                break;
            }
        if (isPrime)
        {
            if (count == a.getSize())
                a.resize(count * 2);
            a[count++] = i; // 第 1 次将 2 放入 a 数组
        }
    }
    for (int i = 0; i < count; i++)
        cout << setw(8) << a[i];
    cout << endl;
    return 0;
}
```

运行结果：

```
Enter a value >= 2 as upper limit for prime numbers: 20
       2       3       5       7      11      13      17      19
```

，说明运行成功。

5. 编写程序提示用户输人一个班级中的学生人数 n 再依次提示用户输入 n 个人在课程 A 中的考试成绩，然后计算出平均成绩，显示出来。请使用教材第 9 章中的数组类模板 Array 定义浮点型数组储存考试成绩。（习题 9.1）

代码（main）：

```
#include"array.h"

#include<cstdio>

#include<iostream>

using namespace std;

int n;

int main()

{

    printf("input n:");

    scanf("%d", &n);

    Array<double> a(n);

    printf("input grades(n in total):");

    for (int i = 0; i < a.getSize();++i)

    {

        double x;

        scanf("%lf", &x);

        a[i] = x;

    }
```

```
    double tot = 0;
    for (int i = 0; i < a.getSize();++i)
    {
        tot += a[i];
    }
    tot /= a.getSize();
    printf("average grade: %f\n", tot);
    return 0;
}
```

结果：

```
f' '--dbgExe=D:\mingw64\bin\gdb.exe' '--interp
input n:5
input grades(n in total):55 99 77 98 40
average grade: 73.800000
```

结论分析与体会：

本实验使我进一步了解并实践了模板泛型编程。