day02

一、Cmake的简单介绍

1.CmakeLists.txt

```
# 清单文件,构建程序的脚本
# 使用cmake的版本
cmake_minimum_required(VERSION 3.17)
# 工程名
project(day02)
# 设置编译软件
# 使用C++14编译代码
set(CMAKE_CXX_STANDARD 14)
# 添加一个可执行程序 程序名叫做main,程序有main.cpp
# 正好main.cpp中有main函数
add_executable(day02 main.cpp)
```

一个程序禁止有两个入口程序

2.数组的遍历

```
#include <iostream>
using namespace std;
// main作为程序的入口
int main() {
    int score[]{10,20,30,40,50,60};
    // 有下标的for循环
    for (int i = 0; i < sizeof(score) / sizeof(int); ++i) {
        cout << i << "=" << score[i] << endl;
    }
    // 无下标的for循环,基于范围的for循环
    /*for (每一个元素:容器){

    }*/
    for (int s : score){
        cout << s << endl;
    }

    return 0;
}
```

3.自动推断类型 auto

- 不能用来定义变量 `auto i;`、只能初始化并赋值 `auto i = 10;`

4.越界检查

- 数组不会有越界的检查机制
- 超过最大下标有时候可以取到值
- 数组的下标取值是地址的偏移
- **要是有检查机制，那么运行效率就会降低**

```
#include <iostream>
using namespace std;
// main作为程序的入口
int main() {
    int score[]{10,20,30};
    cout << "最大值" << score[2] << endl;
    cout << "超过位置" << score[3] << endl;
    return 0;
}
// 运行结果:
最大值30
超过位置0
```

二、字符串

1.字符串表现手法

- C语言中的字符串是实际上是一个数组来存很多数据，末尾有null或者\0来终止一维数组
- C语言的字符串基本操作：
  - 遍历
  - 拷贝 `strcpy()/strscpy()`
  - 拼接字符串 `strcat()`

2.C++字符串

- 定义

```cpp
#include <iostream>
using namespace std;
// 导入string的头文件
#include <string>
// C++ 的字符串
int main() {
    // 定义
    string ss = "abc";
    cout << ss << endl;
    cout << ss[0] << endl;
    return 0;
}
```

- 操作

```cpp
#include <iostream>
using namespace std;
// 导入string的头文件
#include <string>
// C++ 的字符串
int main() {
    // 定义
    string ss = "abc";
    // 取字符
    cout << ss[0] << endl;
    // 拼接字符
    ss = ss + "defg";
    cout << ss << endl;
    // 长度
    cout << ss.length() << "=" << ss.size() << endl;
    // 查找某个字符所在位置
    cout << ss.find('c') << endl;
    // 截取(包含截取值)
    cout << ss.substr(ss.find('c')) << endl;
    return 0;
}
```

三、vector(数组的升级版)

1.Vector是什么

> 很大程度上和数组是一样的，但是数组是定长的，而vector是动态增长
>
> vector在C++STL （标准模板库）中的一个容器，可以看做是对容器的一种扩展。

2.声明并赋值

```cpp
#include <iostream>
// 导入头文件
#include <vector>
using namespace std;
// C++ 的Vector
int main() {

    // 声明
    vector<int> a;
    // 存值
    a.push_back(10);
    a.push_back(20);
    cout << a[1] << endl;
    return 0;
}
```

3.初始化

```cpp
#include <iostream>
// 导入头文件
#include <vector>
using namespace std;
// C++ 的Vector
int main() {
    // 声明并初始化
    vector<int> a{10,20,30,40};
    // 存值
    cout << a[1] << endl;
    return 0;
}
```

4.基本操作

```cpp
#include <iostream>
// 导入头文件
#include <vector>
using namespace std;

// C++ 的Vector
int main() {

    // 声明并初始化
```

```
    vector<int> a{10,20,30,40};
    // 增加
    a.push_back(50);
    a.push_back(60);
    // 删除
//    a.erase();暂时不讲
    // 取值
    cout<<a.at(3)<<endl;
    cout<<a[3]<<endl;
    // 遍历
    for (int d:a){
        cout << d << endl;
    }
    return 0;
}
```

5.细节部分

如果超过下标，中括号的方式一般不报错，.at()这种方式会检测下标越界

6.二维Vector

```
#include <iostream>
// 导入头文件
#include <vector>
using namespace std;
// C++ 的Vector
int main() {
    vector<vector<int>> vi{{10,20},{30,40}};
    for (int i = 0; i < vi.size(); ++i) {
        for (int j = 0; j < vi[0].size(); ++j) {
            cout << vi[i][j] << "\t";
        }
        cout << endl;
    }
    return 0;
}
```

四、函数

1.函数的定义

```
返回值类型 函数名(参数1,参数2){
    return 返回值;
}
```

```cpp
#include <iostream>
using namespace std;
// main作为程序的入口
void sayHi();
int main() {
    sayHi();
    return 0;
}
void sayHi(){
    std::cout << "Hello, World!" << std::endl;
    return;
}
```

2.函数的四种表现形式

- 无返回值，无参数

```cpp
void sayHi(){
    cout << "hello" << endl;
}
```

- 有返回值，无参数

```cpp
int sayHi(){
    cout << "hello" << endl;
    return 1;
}
```

- 无返回值，有参数

```cpp
void sayHi(int a){
    cout << "hello" << a << endl;
}
```

- 有返回值，有参数

```cpp
int sayHi(int a){
    cout << "hello" << a << endl;
    return 1;
}
```

3.函数的原型

所谓的函数原型，指的就是在编译之前先把函数的返回值、函数的名称、函数的参数、实现注册和登记一下，仅仅是函数的声明

意义：防止源码泄露

4.函数头文件和源文件的实现

- 头文件放声明的代码、或者是简单的变量定义
- 导入自己写的头文件时，应该用 “” 导入
- 头文件不能重复定义
- 头文件里的三句话，是为防止重复调用

5.函数重载

- 指的是函数参数的 **个数或者类型** 不同，但是 **函数名称** 和 **返回值类型相同**

```
#include <iostream>
using namespace std;
// main作为程序的入口
int add(int a,int b){
    return a+b;
}
int add(double a,int b){
    return a+b;
}
int add(int a,int b,int c){
    return a+b+c;
}
int main() {
    cout << add(10,5) << endl;
    cout << add(10,5,5) << endl;
    return 0;
}
```

6.函数传参

- 值传递

```
#include <iostream>
using namespace std;
// main作为程序的入口

void changScore(int s) {
    s = 100;
}
int main() {
    int score = 50;
    cout << score << endl;
    changScore(score);
    // 相当于
    /*
        s = score;
        s = 100;
    */
    cout << score << endl;
    return 0;
}
```

```
// 运行结果:
50
50
```

- 地址传递

```
#include <iostream>
using namespace std;
// main作为程序的入口

void changScore(int* score){
    *score = 100;
}
int main() {
    int score = 50;
    cout << score << endl;
    changScore(&score);
    cout << score << endl;
    return 0;
}
// 运行结果:
50
100
```

如果要在一个函数内部遍历一个数组，那么就要在外部传入这个数值的长度

7.函数的引用

引用就是原数据的一个别名，不会有新的空间开辟，和原数据说的是同一个位置

1.简单定义方法

```
int a = 10;
int & b = a;
```

2.引用的函数传递

```
void changeScore(int & s){
    s = 100;
}

void main(){
    int score = 10;
    changeScore(score);
    cout << "分数:" << score << endl;
}
// 运行结果:
分数:100
```

8.内联函数

如果函数只调用一次，那么就可以用 `inline` 关键字对这个函数进行定义，这样add就不会进行开辟空间

```
inline int add(int a,int b){
    return a + b;
}
int main(){
    int a = 10;
    int b = 20;
    cout << add(a,b) << endl;
}
```

补

```
// 代码块，当程序运行结束的时候，就回收数据
{

}
```

静态局部变量只会初始化一次