

Std::cout 是标准的写法

```
int main()
{
    std::cout << "Hello World!\n";
    return 0;
}
```

如果没有用 std::cout, 需要在前面用 using xxxxx std

来防止不同头文件中的函数使用错误

```
using namespace std;
cout << "welcome";
cout << endl;
```

或者在前面用这种方式来声明, 这种方式是最稳定的一种方式, 会保证不会出错。

Cin 是输入 cin>>

```
int main()
{
    using std::cout;
    using std::endl;
    using std::cin;
    cout << "Hello World!\n";
    cout << endl;
    cout<< "nihao";
    return 0;
}
```

换行有两种

1. cout<<endl cout 输出默认 10 进制

2. " \n"

输出具体的数

```
cout << "i have " << book << " book";
```

数学运算的头文件

```
#include <iostream>
#include <cmath>
```

构造函数

Type functionname ()

```
{
}
```

两种方法

1. 在函数构建内部加入 using，只针对这个函数
2. 在函数声明前面加上 using，会让在声明后的所有函数都可以用

```
11 void pri()
12 {
13     using namespace std;
14     cout << "hello";
15 }
```

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 void pri();
5 int main()
6 {
7     //using std::cout;
8     //using std::endl;
9     //using std::cin;
10    pri();
11 }
12 void pri()
13 {
14     cout << "hello";
15 }
16 }
```

Sizeof 的调用要加上#include <climits>头文件

类型转换，低精度向高精度转换不会变化本身的值，而高精度向低精度转换其结果是不确定的

数组的空数组要写' \0'，数组中的内容要写成单引号

计算数组长度的要包含的头文件

用 strlen()

```
#include<cstring>
```

两个 Cin 是用空格作为分割，若你要用回车来区分两个输入就不能用 cin

可以用 cin.get(数组名,数组容量)和 cin.getline(),这两个都是针对数组。Cin 后面可以用 cin.get()来吃掉回车符号，让后再次输入。

```
cout << "你的生日? \n";
int year;
cin>> year;
cin.get();
cout << "你的地址";
char address[80];
cin.get(address, 80);
cout << "date"<<endl;
cout << "地址: "<<address;
```

String 类

String 相比于数组而言更见简便，可以自动处理 string 的大小。

数组直接的数不可以赋值给另一个，而 string 可以。

String 可以直接用 + 来进行相加

```
{
    string da = "sadasf";
    string ba = "dadaf";
    string k = da + ba;
    cout << k;
}
```

删除警告可以用 pragma warning ()

```
1  ~#include <iostream>
2  #include <cmath>
3  #include<climits>
4  #include<cstring>
5  #include<string>
6  #pragma warning(disable:4996)
7  using namespace std;
8  double cov(double x);
9  ~int main()
10 {
11     char charr1[20]="name";
12     char charr2[20]="duttty";
13     string da = "cod";
14     string ba = "duty";
15     string k = da + ba;
16     //strcpy_s(charr1, "dadd");
17     strcat(charr1, charr2);
18     cout << charr1<<endl;
19     cout << charr2;
20 }
21 ~double cov(double x)
22 {
23     double ma;
24     ma = x * 200;
25     cout << "num=" << ma << endl;
26     return ma;
27 }
```

Strcat_s() 添加函数，把 2 加到 1

Strcpy_s() 拷贝函数，把 2 拷贝到 1

```
strcat_s(charr1, charr2);
strcpy_s(charr1, charr2);
```

结构体的构造，结构体可以赋值

```

struct inf
{
    string name;
    float volume;
    double price;
};

```

结构体名字. 项目 其对应的类型便是其项目的类型

```

};
cout << fruit.name<<endl;
cout << fruit.price;

```

结构数组 struct

```

22     inf gifts[30] =
23     {
24         {"call",21,3},
25         {"of",12,3.22}
26     };
27     cout << fruit.name<<endl;
28     cout << fruit.price<<endl;
29     cout << gifts[1].name;

```

共用体 union

共用体是对 int 或 double 或 long，这三个里面的一种进行存储，结构体可以同时存储上面三种。

```

union pail
{
    int year;
    char name[20];
};

```

在调用的时候只能这样，不能像结构体一样直接大括号里面调用。

```

pail pail1;
pail1.year = 19;
cout << pail1.year << endl;

```

枚举 enum

```
enum spectrum {red,orange,yellow,blue,violet,indigo,ultraviolet};
spectrum band;
int color;
band = blue;
color = band +2;
cout << color<<endl;
```

枚举其中对应的是 int 整形，各类颜色代表 0，1，2，这中整形。

可以定义一个整型在等号左边进行加减,但是不能定义枚举在左边进行加减。

枚举可以进行自己定义里面的数值

```
enum bits {one=1,two=2,four=4,eight=8};
bits myflag;
myflag = bits(four);
```

指针

```
1 int a, b, * p1 = &a, * p2;
2
3 &*p1 的含义:
4 &和*都是自右向左运算符,因此先看p1是指针,*p1即为p1指向的对象,
5 因此*p1等价于a,因此a 的前面加一个&,表示的就是 a 的地址
6 因此: &*p1 ,p1 ,&a 三者等价
7 *&a 的含义: a的地址前加*表示的就是a本身,指针就是用来放地址的,地址前面加*表示的就是这个对象
8 因此: *&a ,a ,*p 三者等价
```

< 和 > 的优先级比 << 要低所以要用括号括起来,来正确运行

```
int x=100;
cout << (x < 1) << endl;
```

Cout.setf(ios_base::boolalpha)可以将 bool 值 0, 1 变成 ture 和 false

```
cout.setf(ios_base::boolalpha);
cout << (x < 1) << endl;
```

For 循环阶乘计算

```
int main()
{
    int i, k;
    long long fac[num], res = 1, f_res;
    fac[0] = 1;
    for (i = 1; i < 10; i++)
    {
        fac[i] = i;
        for (k = 0; k < i; k++)
        {
            res = fac[k] * res;
            f_res=res;
        }
        res = 1;
        cout << i-1<< "!=" << f_res << endl;
    }
}
```

通过 strcmp（名称，比较字符）来进行比较运算，当第一个与第二个相等则停止

```
int main()
{
    char word[5]="date";
    for (char ch='a'; strcmp(word, "mate"); ch++)
    {
        cout << word << endl;
        word[0] = ch;
    }
    cout << "finish!" << endl;
}
```

或者用 string 来进行比较

```
int main()
{
    string word = "date";
    for (char ch = 'a'; word != "mate"; ch++)
    {
        cout << word << endl;
        word[0] = ch;
    }
    cout << "finish!" << endl;
}
```

```

1  #include <iostream>
2  #include <cmath>
3  #include<climits>
4  #include<cstring>
5  #include<string>
6  using namespace std;
7  double cov(double x);
8  const int num = 10;
9  struct inf
10 {
11     string name;
12     float volume;
13     double price;
14 };
15 union pail
16 {
17     int year;
18     char name[20];
19 };
20 int main()
21 {
22     pail pail1;
23     pail1.year = 19;
24     cout << pail1.year << endl;
25     string word = "date";
26     for (char ch = 'a'; word != "mate"; ch++)
27     {
28         cout << word << endl;
29         word[0] = ch;
30     }
31     cout << "finish!" << endl;
32 }
33 double cov(double x)
34 {
35     double ma;
36     ma = x * 200;
37     cout << "num=" << ma << endl;
38     return ma;
39 }

```


类型别名

1. #define 新名字 int/char/double
2. typedef char byte

延时程序

```
#include <ctime>
using namespace std;
double cov(double x);
int main()
{
    cout << "enter your delay time" << endl;
    float secs;
    cin >> secs;
    clock_t delay = secs * CLOCKS_PER_SEC;
    cout << "starting\a\n";
    clock_t start = clock();
    while (clock() - start < delay)
        ;
    cout << "done \a\n";
    return 0;
}
```

For 循环遍历数组

```
int main()
{
    int price[5] = { 1,2,3,4,5 };
    for (int x : price)
    {
        cout << x << endl;
    }
}
```

检测数组内的字符个数

```
while (ch != '@')
{
    if (isalpha(ch))
        chars++;
    else if (isspace(ch))
        ws++;
    else if (ispunct(ch))
        punct++;
    else if (isdigit(ch))
        dg++;
    else
        others++;
    cin.get(ch);
}
```

? : 运算符 A? B: C

如果 A 条件成立，则 A 的结果是 B

若 A 的条件不成立, A 的结果是 B

Switch 语句

Switch 语句用于多个选择，switch 可以和枚举一起使用。

Switch 只能适用于具体数，不能适用于范围值

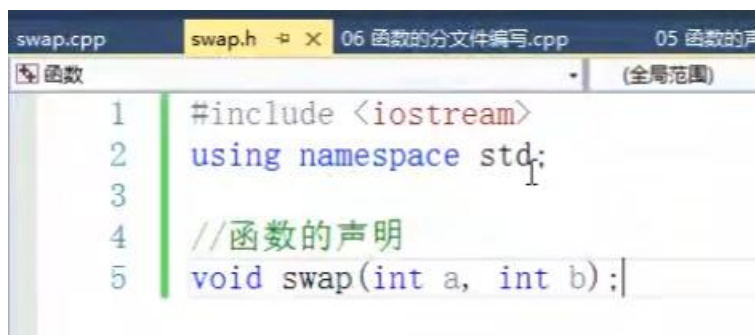
数组的大小 `nums.size();`

h3 6.7 函数的分文件编写

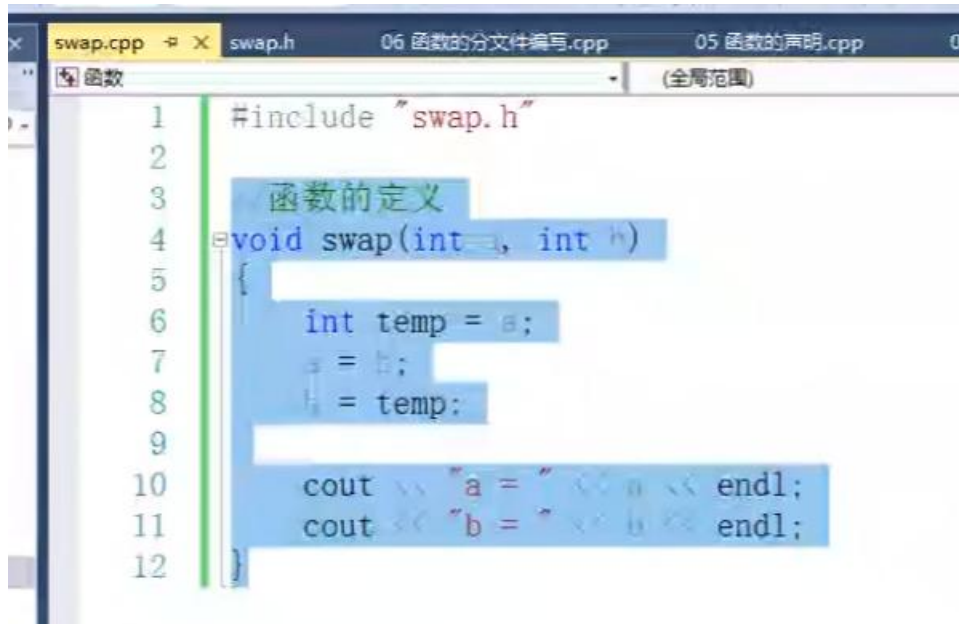
作用：让代码结构更加清晰

函数分文件编写一般有4个步骤

1. 创建后缀名为.h的头文件
2. 创建后缀名为.cpp的源文件
3. 在头文件中写函数的声明
4. 在源文件中写函数的定义



```
swap.cpp  swap.h  06 函数的分文件编写.cpp  05 函数的声明.cpp
函数 (全局范围)
1  #include <iostream>
2  using namespace std;
3
4  //函数的声明
5  void swap(int a, int b);
```



```
swap.cpp  swap.h  06 函数的分文件编写.cpp  05 函数的声明.cpp  04
函数 (全局范围)
1  #include "swap.h"
2
3  函数的定义
4  void swap(int a, int b)
5  {
6      int temp = a;
7      a = b;
8      b = temp;
9
10     cout << "a = " << a << endl;
11     cout << "b = " << b << endl;
12 }
```

```
const int * p = &a;
```

常量指针

特点：指针的指向可以修改，但是指针指向的值不可以改

*p = 20; 错误，指针指向的值不可以改

p = &b; 正确，指针指向可以改

```
int * const p = &a;
```

指针常量



特点：指针的指向不可以改，指针指向的值可以改

*p = 20; 正确，指向的值可以改

p = &b; 错误，指针指向不可以改

```
const int * const p3 = &a;
```

//指针的指向 和 指针指向的值 都不可以改

*p3 = 100; // 错误

p3 = &b; // 错误

```
int a[5] = { 1,2,3,4,5 };
```

```
int* p = a;
```

```
cout << "地址为:" << p << endl;
```

```
cout << "内容为:" << *p << endl;
```

```
p++;
```

```
cout << "第二个为: " << *p << endl;
```

```
return 0;
```

```

int main()
{
    int arr[10] = { 4,3,6,9,1,2,10,8,7,5 };
    int len = sizeof(arr)/sizeof(arr[0]);
    bubblesort(arr, len);
    for (int i = 0; i < len - 1; i++)
    {
        cout << arr[i] << endl;
    }
}

void bubblesort(int* arr, int len)
{
    for (int i = 0; i < len - 1; i++)
    {
        for (int j = 0; j < len - i - 1; j++)
        {
            if (arr[j] > arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

```

//2、通过指针指向结构体变量

```
student * p = &s;
```

//3、通过指针访问结构体变量中的数据

```
cout << "姓名: " << p->name << " 年龄: " << p->age << "
```

//将函数中的形参改为指针，可以减少内存空间，而且不会复制新的副本出来

```

void printStudents(const student *s)
{
    //s->age = 150; //加入const之后，一旦有修改的操作就会报错，可以防止我们的误操作
    cout << "姓名: " << s->name << " 年龄: " << s->age << " 得分: " << s->score << endl;
}

```

函数的默认参数

```
21
22 //2、如果函数声明有默认参数，函数实现就不能有默认参数
23 // 声明和实现只能有一个有默认参数
24 int func2(int a = 10, int b = 10);
25
26 int func2(int a = 20, int b = 20)
27 {
28
```

函数的默认值如果有传入则就是传入值，没有值则就是默认值。

函数重载

```
//函数重载的满足条件
//1、同一个作用域下
//2、函数名称相同
//3、函数参数类型不同，或者个数不同，或者顺序不同
void func()
{
    cout << "func 的调用" << endl;
}

void func(int a)
{

```

引用也可以重载

```
1 //函数重载注意事项
2 //1、引用作为重载条件
3
4 void func(int &a)
5 {
6     cout << "func (int &a) 调用 " << endl;
7 }
8
9 void func(const int &a)
10 {
11     cout << "func (const int &a) 调用 " << endl;
12 }
```


内存分区模型

C++程序在执行时，将内存大方向划分为**4个区域**

- 代码区：存放函数体的二进制代码，由操作系统进行管理的
- 全局区：存放全局变量和静态变量以及常量
- 栈区：由编译器自动分配释放，存放函数的参数值，局部变量等
- 堆区：由程序员分配和释放，若程序员不释放，程序结束时由操作系统回收

```
int * func()  
{  
    //在堆区创建整型数据  
    //new返回是 该数据类型的指针  
    int * p = new int(10);  
}
```

```
5         //释放数组 delete 后加 []  
6         delete[] arr;  
7
```

引用的基本使用

****作用：** **给变量起别名

语法： 数据类型 &别名 = 原名

示例：

```
C++  
1  int main() {  
2  
3      int a = 10;  
4      int &b = a;
```

当别名改变时候，其原名的数值也会改变

引用注意事项

- 引用必须初始化
- 引用在初始化后，不可以改变

通过引用参数产生的效果同按地址传递是一样的。引用的语法更清楚简单

封装

封装的意义

封装是C++面向对象三大特性之一

封装的意义：

- 将属性和行为作为一个整体，表现生活中的事物
- 将属性和行为加以权限控制

语法： `class 类名{ 访问权限： 属性 / 行为 };`