1.初始

1.1.第一个程序

```
#include<iostream>
using namespace std;
int main
{
   count<< "hello c++" <<end1;
   system("pause");
   return 0;
}
```

1.2.注释

- 单行注释 //
- 多行注释 /* ..... */

1.3变量

语法：数据类型 变量名 =初始值

```
#include<iostream>
using namespace std;
int main ()
{
    int a=10;
    count<<"a="<<a<<endl;
    system("pause");
    return 0;
}
```

1.4常量

C++定义的两种方式：

1：#define 宏常量　#define 常量名 常量值 （通常在文件上方定义，表示一个常量）

2：const修饰的变量　const数据类型 常量名=常量值（通常在变量定义上加关键字const，修饰变量为常量，不可修改）

```
#include<iostream>
using namespace std;
#define day 7
int main()
{
    cout<<"一周总共"<<day<<"天"<<endl;
    system("pause");
    return 0;
}
```

1.5关键字

asm	do	if	return	typedef
auto	double	inline	short	typeid
bool	dynamic_cast	int	signed	typename
break	else	long	sizeof	union
case	enum	mutable	static	unsigned
catch	explicit	namespace	static_cast	using
char	export	new	struct	virtual
class	extern	operator	switch	void
const	false	private	template	volatile
const_cast	float	protected	this	wchar_t
continue	for	public	throw	while
default	friend	register	true	
delete	goto	reinterpret_cast	try	

1.6标识符命名规则

1.标识符不可以是关键字

2.由字母、数字、下划线组成

3.标识符第一个字符只能是字母或下划线

4.区分大小写

2.数据类型

2.1 整形

数据类型	占用空间	取值范围
short(短整型)	2字节	(-2^15 ~ 2^15-1)
int(整型)	4字节	(-2^31 ~ 2^31-1)
long(长整形)	Windows为4字节，Linux为4字节(32位)，8字节(64位)	(-2^31 ~ 2^31-1)
long long(长长整形)	8字节	(-2^63 ~ 2^63-1)

2.2 sizeof关键字

作用：统计数据类型所占内存大小

语法：sizeof(数据类型/变量)

```
cout<<"int类型所占内存空间为："<<sizeof(int)<<endl;
```

2.3实型（浮点型）

数据类型	占用空间	有效数字范围
float	4字节	7位有效数字
double	8字节	15～16位有效数字

```
#include <iostream>
using namespace std;
int main()
{
    float f1=3.14f;//一般会在单精度数字后面加一个f，不然会被系统默认成双精度
    double d1=3.14;
    cout<<"f1="<<f1<<endl;
    cout<<"d1="<<d1<<endl;
    //科学计数法
    float f2 = 3e2; // 3 * 10 ^ 2
    cout << "f2 = " << f2 << endl;

    float f3 = 3e-2;  // 3 * 0.1 ^ 2
    cout << "f3 = " << f3 << endl;
    system("pause");
    return 0;
}
```

2.4字符型

作用：字符型变量用于显示单个字符

语法：char ch ='a';

注意1：在显示字符变量时，用单引号括起来

注意2：单引号内只能由一个字符，不可以是字符串

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

int main() {

    char ch = 'a';
```

```
    cout << ch << endl;
    cout << sizeof(char) << endl;

    //ch = "abcde"; //错误，不可以用双引号
    //ch = 'abcde'; //错误，单引号内只能引用一个字符

    cout << (int)ch << endl;  //查看字符a对应的ASCII码
    ch = 97; //可以直接用ASCII给字符型变量赋值
    cout << ch << endl;
    system("pause");
    return 0;
}
```

2.5转义字符

转义字符	含义	ASCII码值（十进制）
\a	警报	007
\b	退格(BS)，将当前位置移到前一列	008
\f	换页(FF)，将当前位置移到下页开头	012
\n	**换行(LF)，将当前位置移到下一行开头**	**010**
\r	回车(CR)，将当前位置移到本行开头	013
\t	**水平制表(HT) （跳到下一个TAB位置）**	**009**
\v	垂直制表(VT)	011
****	**代表一个反斜线字符""**	**092**
'	代表一个单引号（撇号）字符	039
"	代表一个双引号字符	034
?	代表一个问号	063
\0	数字0	000
\ddd	8进制转义字符，d范围0~7	3位8进制
\xhh	16进制转义字符，h范围0_9，$_a$f，A~F	3位16进制

```
#include <iostream>
using namespace std;
int main() {
    cout << "\\" << endl;   //输出一个字符
    cout << "\tHello" << endl; //输出结果对齐
    /*cout << "aaaa\tHello" << endl;
    cout << "aa\tHello" << endl
    cout << "aa\tHello" << endl*/        //这几个结果输出结尾都是对齐的
    cout << "\n" << endl;   //换行
    system("pause");
    return 0;
}
```

2.6字符串型

c风格字符串

char 变量名[]="zifuchuan"

```
int main()
{
    char str[]="zifuchuan" ;  //要加中括号，等号后面要用双引号
    cout<<str<<endl;
    system("pause");
    return 0;
}
```

c++风格字符串

```
#include <iostream>
using namespace std;
#include <string>  //加入新的头文件
int main()
{
    string str="zifuchuan";
    cout<<str<<endl;
    system("pause");
    return 0;
}
```

2.7 布尔类型bool

作用:代表真和假（true/false）

bool类型占一个字节

```
int main()
{
    bool flag = true;
    cout<<flag<<endl;
    flag=false;
    cout<<flag<<endl;
    cout<<sizeof(bool)<<endl;
    system("pause");
    return 0;
}
```

除了0 其他数字都是1

2.8 数据输入

作用：用于键盘数据获取

关键字 cin>>变量

```
int main(){

    //整型输入
    int a = 0;
```

```
    cout << "请输入整型变量：" << endl;
    cin >> a;
    cout << a << endl;

    //浮点型输入
    double d = 0;
    cout << "请输入浮点型变量：" << endl;
    cin >> d;
    cout << d << endl;

    //字符型输入
    char ch = 0;
    cout << "请输入字符型变量：" << endl;
    cin >> ch;
    cout << ch << endl;

    //字符串型输入
    string str;
    cout << "请输入字符串型变量：" << endl;
    cin >> str;
    cout << str << endl;

    //布尔类型输入
    bool flag = true;
    cout << "请输入布尔型变量：" << endl;
    cin >> flag;
    cout << flag << endl;
    system("pause");
    return EXIT_SUCCESS;
}
```

3 运算符

3.1 算术运算符

运算符	术语	示例	结果
+	正号	+3	3
-	负号	-3	-3
+	加	10 + 5	15
-	减	10 - 5	5
*	乘	10 * 5	50
/	除	10 / 5	2
%	取模(取余)	10 % 3	1
++	前置递增	a=2; b=++a;	a=3; b=3;
++	后置递增	a=2; b=a++;	a=3; b=2;
–	前置递减	a=2; b=–a;	a=1; b=1;
–	后置递减	a=2; b=a–;	a=1; b=2;

在除法运算中，除数不能为0。两个小数可以相除。

两个小数不能取模，只有整形变量可以取模运算。

前置递增先对变量进行++，再计算表达式，后置递增相反

3.2 赋值运算符

运算符	术语	示例	结果
=	赋值	a=2; b=3;	a=2; b=3;
+=	加等于	a=0; a+=2;	a=2;
-=	减等于	a=5; a-=3;	a=2;
*=	乘等于	a=2; a*=2;	a=4;
/=	除等于	a=4; a/=2;	a=2;
%=	模等于	a=3; a%2;	a=1;

3.3 比较运算符

运算符	术语	示例	结果
==	相等于	4 == 3	0
!=	不等于	4 != 3	1
<	小于	4 < 3	0
>	大于	4 > 3	1
<=	小于等于	4 <= 3	0
>=	大于等于	4 >= 1	1

C和C++ 语言的比较运算中， “真”用数字“1”来表示， “假”用数字“0”来表示。

3.4 逻辑运算符

运算符	术语	示例	结果
!	非	!a	如果a为假，则!a为真； 如果a为真，则!a为假。
&&	与	a && b	如果a和b都为真，则结果为真，否则为假。
||	或	a || b	如果a和b有一个为真，则结果为真，二者都为假时，结果为假。

4 程序流程结构

C/C++支持最基本的三种程序运行结构：**顺序结构、选择结构、循环结构**

- 顺序结构：程序按顺序执行，不发生跳转
- 选择结构：依据条件是否满足，有选择的执行相应功能
- 循环结构：依据条件是否满足，循环多次执行某段代码

4.1选择结构

4.1.1 if语句

```
int main()
{
    int score ;
    cout<<"input your score:"<<endl;
    cin>>score;
    if score>=600
        cout<<"厉害！"<<endl;
    else
         cout<<"还需要努力"<<endl;
    system("pause");
    return 0;

}
```

注意：if条件表达式后不要加分号,但要有小括号。

嵌套if　(else、else if)

```
int main()
{
    int dps;
    cout<<"请输入你的dps:"<<endl;
    cin>>dps;
    if (dps>50)
    {
        if(dps>60)
            cout<<"大佬！"<<endl;
        if(dps>80)
            cout<<"巨佬!"<<endl;
    }
    else if (dps>30)
        cout<<"副c"<<endl;
    else if (dps>20)
        cout<<"34仔"<<endl;
    else if (dps>10)
        cout<<"拉跨"<<endl;
    else
        cout<<"死混子"<<endl;
}
```

比较数字大小，输出最大数字

```
int main()
{
    int num1,num2,num3,biggest;
    cout<<"请输入体重："<<endl;
    cin>>num1;
    cout<<"请输入体重："<<endl;
    cin>>num2;
    cout<<"请输入体重："<<endl;
    cin>>num3;
```

```
    if(num1<num2&&num2>num3)
        biggest=num2;
    if(num2<num3&&num1<num3)
        biggest=num3;
    if(num1>num2&&num1>num3)
        biggest=num1;
    cout<<"最重的体重为"<<biggest<<endl;
    system("pause");
    return 0;
}
```

4.1.2 三目运算符

作用：实现简单的判断

语法：表达式1?表达式2:表达式3　***有时候需要用小括号括起来***

解释：如果表达式1为真，执行表达式2，并返回表达式2的结果

　　如果表达式1为假，执行表达式3，并返回表达式3的结果

```
int main() {
    int a = 10;
    int b = 20;
    int c = 0;
    c = a > b ? a : b;
    cout << "c = " << c << endl;
    //C++中三目运算符返回的是变量,可以继续赋值
    (a > b ? a : b) = 100;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "c = " << c << endl;
    system("pause");
    return 0;
}
```

4.1.3 switch 语句

作用 执行多条分支语句

语法：

```
switch(表达式)
{
    case 结果1：执行语句;break;
    case 结果2：执行语句;break;
    ...
    default:执行语句;break;
}
```

```
int main() {
    //请给电影评分
    //10 ~ 9    经典
    // 8 ~ 7    非常好
```

```
    // 6 ~ 5   一般
    // 5分以下 烂片
    int score = 0;
    cout << "请给电影打分" << endl;
    cin >> score;

    switch (score)
    {
    case 10:
    case 9:
        cout << "经典" << endl;
        break;
    case 8:
        cout << "非常好" << endl;
        break;
    case 7:
    case 6:
        cout << "一般" << endl;
        break;
    default:
        cout << "烂片" << endl;
        break;
    }
    system("pause");
    return 0;
}
```

注意1：switch语句中表达式类型只能是整型或者字符型

注意2：case里如果没有break，那么程序会一直向下执行

总结：与if语句比，对于多条件判断时，switch的结构清晰，执行效率高，缺点是switch不可以判断区间

4.2循环结构

4.2.1while循环

作用：满足循环条件，执行循环语句

语法： `while(循环条件){ 循环语句 }`

解释：只要循环条件的结果为真，就执行循环语句

```
int main() {

    int num = 0;
    while (num < 10)
    {
        cout << "num = " << num << endl;
        num++;
    }

    system("pause");

    return 0;
}
```

while循环练习（猜数字）

```
#include<iostream>
using namespace std;
//time系统时间头文件包含
#include<ctime>
int main()
{
    // rand() % 100 + 1 生成0+1到99+1的随机数;
    int num = rand() % 100 + 1;
    int guess;
    //添加随机种子 利用系统时间生成随机数，防止每次每次随机数都一样
    srand((unsigned int)time(NULL));
    cout << "猜一个数字" << endl;

    while (1)
    {
        cin >> guess;
        if (guess > num)
        {
            cout << "大了" << endl;
        }
        else if (guess < num)
        {
            cout << "小了" << endl;
        }
        else
        {
            cout << "答对了" << endl;
            break;
        }
    }
    system("pause");
    return 0;
}
```

4.2.2 do...while 循环语句

作用：满足循环，执行循环

语法：do{ 循环语句 } while(循环条件);

注意：与while的区别在于do...while会先执行一次循环语句，再判断循环条件

练习案例（水仙花数）水仙花数：一个三位数，它的每个位上的数字的三次幂之和等于它本身

例如：1^3+5^3+3^3=153

利用do...while语句。求出3位数中的水仙花数

```
#include<iostream>
using namespace std;
int main()
{
    int num = 100;
    int i, j, k;
```

```
    int h;
    do
    {
        i = num / 100;
        j = (num / 10) % 10;
        k = (num % 100) % 10;
        h = pow(i, 3) + pow(j, 3) + pow(k, 3);
        if (h == num)
        {
            cout << num << endl;
        }
        num++;
    } while (num < 1000);
    system("pause");
    return 0;
}
```

4.2.3 for循环

作用： 满足循环条件，执行循环语句

语法： `for(起始表达式;条件表达式;末尾循环体) { 循环语句; }`

```
int main() {

    for (int i = 0; i < 10; i++)
    {
        cout << i << endl;
    }

    system("pause");

    return 0;
}
```

练习：案例描述：从1开始数到数字100， 输出数字个位含有7，或者数字十位含有7，或者该数字是7的倍数，

```
#include <iostream>
using namespace std;
#include <string>  //加入新的头文件
int main()
{
    int num = 1;
    int i, j, k,h;
    for (num; num < 100; num++)
    {
        i = num % 7;
        j = num / 10;
        k = num % 10;
        if (i == 0 || j == 7 || k == 7)
        {

            cout << num << endl;
```

```
        }
    }
    system("pasue");
    return 0;

}
```

4.2.4 嵌套循环

作用： 在循环体中再嵌套一层循环，解决一些实际问题

```
int main() {

    //外层循环执行1次，内层循环执行1轮
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            cout << "*" << " ";
        }
        cout << endl;
    }

    system("pause");

    return 0;
}
```

案例描述：利用嵌套循环，实现九九乘法表

```
#include <iostream>
using namespace std;
int main() {
    int i;
    int j;
    int k;
    for ( i=1; i <= 9; i++)
    {
        for (j=1; j <= i; j++)
        {
            cout << i<<"*" <<j<<"="<< i * j << "   ";
        }
        cout << endl;
    }
    system("pause");

    return 0;
}
```

4.3跳转语句

4.3.1 break

作用: 用于跳出选择结构或者循环结构

break使用的时机:

- 出现在switch条件语句中，作用是终止case并跳出switch
- 出现在循环语句中，作用是跳出当前的循环语句
- 出现在嵌套循环中，跳出最近的内层循环语句

例子1:

```
int main() {
    //1、在switch 语句中使用break
    cout << "请选择您挑战副本的难度：" << endl;
    cout << "1、普通" << endl;
    cout << "2、中等" << endl;
    cout << "3、困难" << endl;
    int num = 0;
    cin >> num;
    switch (num)
    {
    case 1:
        cout << "您选择的是普通难度" << endl;
        break;
    case 2:
        cout << "您选择的是中等难度" << endl;
        break;
    case 3:
        cout << "您选择的是困难难度" << endl;
        break;
    }
    system("pause");
```

例子2:

```
int main() {
    //2、在循环语句中用break
    for (int i = 0; i < 10; i++)
    {
        if (i == 5)
        {
            break; //跳出循环语句
        }
        cout << i << endl;
    }
    system("pause");
    return 0;
}
```

例子3:

```
int main() {
```

```
    //在嵌套循环语句中使用break，退出内层循环
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            if (j == 5)
            {
                break;
            }
            cout << "*" << " ";
        }
        cout << endl;
    }
    system("pause");
    return 0;
}
```

4.3.2 continue语句

作用：在循环语句中，跳过本次循环中余下尚未执行的语句，继续执行下一次循环

示例：

```
int main() {
    for (int i = 0; i < 100; i++)
    {
        if (i % 2 == 0)
        {
            continue;
        }
        cout << i << endl;
    }

    system("pause");
    return 0;
}
```

注意：continue并没有使整个循环终止，而break会跳出循环

4.3.3 goto语句

作用：可以无条件跳转语句

解释：如果标记的名称存在，执行到goto语句时，会跳转到标记的位置

```
int main()
{
    cout << "1" << endl;
    goto FLAG;
    cout << "2" << endl;
    cout << "3" << endl;
    cout << "4" << endl;
    FLAG:
    cout << "5" << endl;
    system("pause");
    return 0;
}
```

> 注意：在程序中不建议使用goto语句，以免造成程序流程混乱

5 数组

5.1概述

所谓数组，就是一个集合，里面存放了相同类型的数据元素

特点1：数组中的每个数据元素都是相同的数据类型

特点2：数组是由连续的内存位置组成的

5.2.1 一维数组的定义方式

1. `数据类型 数组名[ 数组长度 ];`
2. `数据类型 数组名[ 数组长度 ] = { 值1，值2 ...};`
3. `数据类型 数组名[ ] = { 值1，值2 ...};`

示例：

```
int main()
{
    //定义方式1
    //数据类型 数组名[元素个数];
    int score[10];
    //利用下标赋值
    score[0] = 100;
    score[1] = 99;
    score[2] = 85;
    //利用下标输出
    cout << score[0] << endl;
    cout << score[1] << endl;
    cout << score[2] << endl;
    //第二种定义方式
    //数据类型 数组名[元素个数] =  {值1，值2 ，值3 ...};
    //如果{}内不足10个数据，剩余数据用0补全
    int score2[10] = { 100, 90,80,70,60,50,40,30,20,10 };
    //逐个输出
    //cout << score2[0] << endl;
    //cout << score2[1] << endl;
    //一个一个输出太麻烦，因此可以利用循环进行输出
    for (int i = 0; i < 10; i++)
    {
```

```
        cout << score2[i] << endl;
    }
    //定义方式3
    //数据类型 数组名[] =  {值1，值2 ，值3 ...};
    int score3[] = { 100,90,80,70,60,50,40,30,20,10 };
    for (int i = 0; i < 10; i++)
    {
        cout << score3[i] << endl;
    }
    system("pause");
    return 0;
}
```

总结1：数组名的命名规范与变量名命名规范一致，不要和变量重名

总结2：数组中下标是从0开始索引

5.2.2 一维数组数组名

一维数组名称的**用途**：

1. 可以统计整个数组在内存中的长度
2. 可以获取数组在内存中的首地址

示例

```
int main() {
    //数组名用途
    //1、可以获取整个数组占用内存空间大小
    int arr[10] = { 1,2,3,4,5,6,7,8,9,10 };
    cout << "整个数组所占内存空间为：  " << sizeof(arr) << endl;
    cout << "每个元素所占内存空间为：  " << sizeof(arr[0]) << endl;
    cout << "数组的元素个数为：  " << sizeof(arr) / sizeof(arr[0]) << endl;
    //2、可以通过数组名获取到数组首地址
    cout << "数组首地址为：  " << (int)arr << endl;
    cout << "数组中第一个元素地址为：  " << (int)&arr[0] << endl;
    cout << "数组中第二个元素地址为：  " << (int)&arr[1] << endl;
    //arr = 100; 错误，数组名是常量，因此不可以赋值
    system("pause");
    return 0;
}
```

> 注意：数组名是常量，不可以赋值

> 总结1：直接打印数组名，可以查看数组所占内存的首地址

> 总结2：对数组名进行sizeof，可以获取整个数组占内存空间的大小

练习1：在一个数组中记录了五只小猪的体重，如：int arr[5] = {300,350,200,400,250};

找出并打印最重的小猪体重。

```
#include <iostream>
using namespace std;
int main()
{
    int arr[5] = { 700,350,200,400,250 };
```

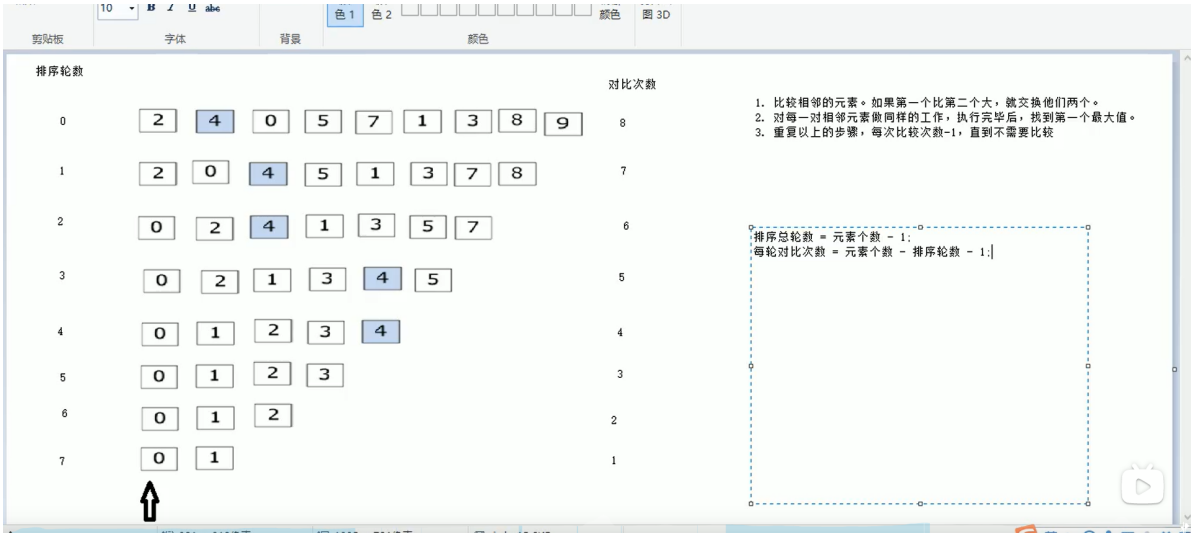
```
    int num = arr[0];
    for (int i = 0; i < 5; i++)
    {
        if (num < arr[i])
        {
            num = arr[i];
        }
    }
    cout << num << endl;
    system("pause");
    return 0;
}
```

练习2：请声明一个5个元素的数组，并且将元素逆置.

(如原数组元素为：1,3,2,5,4;逆置后输出结果为:4,5,2,3,1);

```
#include <iostream>
using namespace std;
int main()
{
    int arr[5] = {1,2,3,4,5};
    int temp[5];
    int start = 0;
    int end = sizeof(arr) / sizeof(arr[0])-1;
    for (int i = 0; i < 5; i++)
    {
        temp[start] = arr[end];
        start++;
        end--;
    }
    for (int j = 0; j < 5; j++)
    {
        cout << temp[j] << endl;
    }
    system("pause");
    return 0;
}
```

练习：冒泡排序




```
#include <iostream>
using namespace std;
int main()
{
    int arr[10] = {10,9,8,7,6,5,4,3,2,1};
    int temp[1];
    for (int i = 0; i < 10-1; i++)
    {
        for (int j=0;j<10-i-1;j++)
        {
            if (arr[j] > arr[j+1])
            {
                temp[0] = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp[0];
            }
        }
    }
    for (int k = 0; k < 10; k++)
    {
        cout << arr[k] << endl;
    }
    system("pause");
    return 0;
}
```

5.3 二维数组

5.3.1 二维数组的定义方式

1. `数据类型 数组名[ 行数 ][ 列数 ];`
2. `数据类型 数组名[ 行数 ][ 列数 ] = { {数据1，数据2 } ，{数据3，数据4 } };`
3. `数据类型 数组名[ 行数 ][ 列数 ] = { 数据1，数据2，数据3，数据4};`
4. `数据类型 数组名[ ][ 列数 ] = { 数据1，数据2，数据3，数据4};`

建议：以上4种定义方式，利用第二种更加直观，提高代码的可读性

```
int main() {
    //方式1
```

```
    //数组类型 数组名 [行数][列数]
    int arr[2][3];
    arr[0][0] = 1;
    arr[0][1] = 2;
    arr[0][2] = 3;
    arr[1][0] = 4;
    arr[1][1] = 5;
    arr[1][2] = 6;
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
    //方式2
    //数据类型 数组名[行数][列数] = { {数据1，数据2 } ，{数据3，数据4 } };
    int arr2[2][3] =
    {
        {1,2,3},
        {4,5,6}
    };
    //方式3
    //数据类型 数组名[行数][列数] = { 数据1，数据2 ,数据3，数据4  };
    int arr3[2][3] = { 1,2,3,4,5,6 };
    //方式4
    //数据类型 数组名[][列数] = { 数据1，数据2 ,数据3，数据4  };
    int arr4[][3] = { 1,2,3,4,5,6 };
    system("pause");
    return 0;
}
```

5.3.2 二维数组数组名

- 查看二维数组所占内存空间
- 获取二维数组首地址

```
int main() {
    //二维数组数组名
    int arr[2][3] =
    {
        {1,2,3},
        {4,5,6}
    };
    cout << "二维数组大小：  " << sizeof(arr) << endl;
    cout << "二维数组一行大小：  " << sizeof(arr[0]) << endl;
    cout << "二维数组元素大小：  " << sizeof(arr[0][0]) << endl;
    cout << "二维数组行数：  " << sizeof(arr) / sizeof(arr[0]) << endl;
    cout << "二维数组列数：  " << sizeof(arr[0]) / sizeof(arr[0][0]) << endl;
    //地址
    cout << "二维数组首地址：" << arr << endl;
    cout << "二维数组第一行地址：" << arr[0] << endl;
    cout << "二维数组第二行地址：" << arr[1] << endl;
    cout << "二维数组第一个元素地址：" << &arr[0][0] << endl;
```

```
    cout << "二维数组第二个元素地址：" << &arr[0][1] << endl;
    system("pause");
    return 0;
}
```

总结1：二维数组名就是这个数组的首地址

总结2：对二维数组名进行sizeof时，可以获取整个二维数组占用的内存空间大小

5.3.3 二维数组应用

考试成绩统计：

案例描述：有三名同学（张三，李四，王五），在一次考试中的成绩分别如下表，**请分别输出三名同学的总成绩**

	语文	数学	英语
张三	100	100	100
李四	90	50	100
王五	60	70	80

```
#include <iostream>
using namespace std;
#include<string>;
int main()
{
    string name[3] = { "刻晴","甘雨","神里菱华" };
    int score[3][3] =
    {
        {100,100,100},
        {90,87,100},
        {88,88,99}
    };
    for (int j = 0; j < 3; j++)
    {
        int sum = 0;
        for (int k = 0; k < 3; k++)
        {
            sum += score[j][k];
        }
        cout << name[j] << "的总分为:" << sum << endl;
    }
    system("pause");
    return 0;
}
```

6 函数

6.1概述

作用：将一段经常使用的代码封装起来，减少重复代码

一个较大的程序，一般分为若干个程序块，每个模块实现特定的功能。

6.2函数的定义

函数的定义一般主要有5个步骤：

1、返回值类型

2、函数名

3、参数表列

4、函数体语句

5、return 表达式

语法

```
返回值类型 函数名 （参数列表）
{
        函数体语句
        return表达式
}
```

- 返回值类型 ： 一个函数可以返回一个值。在函数定义中
- 函数名：给函数起个名称
- 参数列表：使用该函数时，传入的数据
- 函数体语句：花括号内的代码，函数内需要执行的语句
- return表达式： 和返回值类型挂钩，函数执行完后，返回相应的数据

示例：定义一个加法函数，实现两个数相加

```
//函数定义
int add(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

6.3 函数的调用

功能：使用定义好的函数

语法：`函数名（参数）`

```
//函数定义
int add(int num1, int num2) //定义中的num1,num2称为形式参数，简称形参
{
    int sum = num1 + num2;
    return sum;
}

int main() {

    int a = 10;
    int b = 10;
    //调用add函数
    int sum = add(a, b);//调用时的a，b称为实际参数，简称实参
    cout << "sum = " << sum << endl;
```

```
    a = 100;
    b = 100;

    sum = add(a, b);
    cout << "sum = " << sum << endl;

    system("pause");

    return 0;
}
```

总结：函数定义里小括号内称为形参，函数调用时传入的参数称为实参

6.4 值传递

- 所谓值传递，就是函数调用时实参将数值传入给形参
- 值传递时，如果形参发生，并不会影响实参

示例

```
void swap(int num1, int num2)
{
    cout << "交换前: " << endl;
    cout << "num1 = " << num1 << endl;
    cout << "num2 = " << num2 << endl;

    int temp = num1;
    num1 = num2;
    num2 = temp;

    cout << "交换后: " << endl;
    cout << "num1 = " << num1 << endl;
    cout << "num2 = " << num2 << endl;

    //return ; 当函数声明时候，不需要返回值，可以不写return
}

int main() {

    int a = 10;
    int b = 20;

    swap(a, b);

    cout << "mian中的 a = " << a << endl;
    cout << "mian中的 b = " << b << endl;

    system("pause");

    return 0;
}
```

总结： 值传递时，形参是修饰不了实参的

6.5 函数的常见样式

常见的函数样式有4种

1. 无参无返
2. 有参无返
3. 无参有返
4. 有参有返

示例

```
//函数常见样式
//1、 无参无返
void test01()
{
    //void a = 10; //无类型不可以创建变量,原因无法分配内存
    cout << "this is test01" << endl;
    //test01(); 函数调用
}

//2、 有参无返
void test02(int a)
{
    cout << "this is test02" << endl;
    cout << "a = " << a << endl;
}

//3、无参有返
int test03()
{
    cout << "this is test03 " << endl;
    return 10;
}

//4、有参有返
int test04(int a, int b)
{
    cout << "this is test04 " << endl;
    int sum = a + b;
    return sum;
}
```

6.6函数的声明

作用： 告诉编译器函数名称及如何调用函数。函数的实际主体可以单独定义。

- 函数的**声明可以多次**，但是函数的**定义只能有一次**（如果函数定义放在了主函数的后面，那必须在主函数前，进行函数声明，如果函数定义在主函数之前，那么函数声明就可有可无了）

示例

```
//声明可以多次，定义只能一次
//声明
int max(int a, int b);
int max(int a, int b);
//定义
```

```
int max(int a, int b)
{
    return a > b ? a : b;
}
int main()
{
    int a = 100;
    int b = 200;
    cout << max(a, b) << endl;
    system("pause");
    return 0;
}
```

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

示例

```
//swap.h文件
#include<iostream>
using namespace std;

//实现两个数字交换的函数声明
void swap(int a, int b);
```

```
//swap.cpp文件
#include "swap.h"

void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
}
```

```
//main函数文件
#include "swap.h"
int main()
{
    int a = 100;
    int b = 200;
    swap(a, b);
    system("pause");
    return 0;
}
```

7 指针

7.1 指针的基本概念

指针的作用： 可以通过指针间接访问内存

- 内存编号是从0开始记录的，一般用十六进制数字表示
- 可以利用指针变量保存地址

7.2 指针变量的定义和使用

指针变量定义语法： `数据类型 * 变量名;`

```
int main() {
    //1、指针的定义
    int a = 10; //定义整型变量a

    //指针定义语法：  数据类型 * 变量名 ;
    int * p;

    //指针变量赋值
    p = &a; //指针指向变量a的地址
    //int *p = &a;
    cout << &a << endl; //打印数据a的地址 0053FA88
    cout << p << endl;  //打印指针变量     0053FA88

    //2、指针的使用
    //通过*操作指针变量指向的内存
    cout << "*p = " << *p << endl;   10

    system("pause");

    return 0;
}
```

指针变量和普通变量的区别

- 普通变量存放的是数据,指针变量存放的是地址
- 指针变量可以通过" * "操作符，操作指针变量指向的内存空间，这个过程称为解引用

> 总结1： 我们可以通过 & 符号 获取变量的地址

> 总结2：利用指针可以记录地址

> 总结3：对指针变量解引用，可以操作指针指向的内存

7.3指针变量的定义和使用

提问：指针也是种数据类型，那么这种数据类型占用多少内存空间？

```
int main() {

    int a = 10;

    int * p;
    p = &a; //指针指向数据a的地址

    cout << *p << endl; //* 解引用
    cout << sizeof(p) << endl;
    cout << sizeof(char *) << endl;
    cout << sizeof(float *) << endl;
    cout << sizeof(double *) << endl;

    system("pause");

    return 0;
}
```

总结：所有指针类型在32位操作系统下是4个字节,64位操作系统占8个字节

7.4 空指针和野指针

空指针：指针变量指向内存中编号为0的空间

用途：初始化指针变量

注意：空指针指向的内存是不可以访问的

示例1：空指针

```
int main() {

    //指针变量p指向内存地址编号为0的空间
    int * p = NULL;

    //访问空指针报错
    //内存编号0 ~255为系统占用内存，不允许用户访问
    cout << *p << endl;

    system("pause");

    return 0;
}
```

野指针：指针变量指向非法的内存空间

示例2：野指针

```
int main() {

    //指针变量p指向内存地址编号为0x1100的空间
    int * p = (int *)0x1100;

    //访问野指针报错
    cout << *p << endl;

    system("pause");

    return 0;
}
```

总结：空指针和野指针都不是我们申请的空间，因此不要访问。

7.5 const修饰指针

const修饰指针有三种情况

1. const修饰指针 — 常量指针
2. const修饰常量 — 指针常量
3. const即修饰指针，又修饰常量

示例：

```
int main() {

    int a = 10;
    int b = 10;

    //const修饰的是指针，指针指向可以改，指针指向的值不可以更改
    const int * p1 = &a;
    p1 = &b; //正确
    //*p1 = 100;  报错


    //const修饰的是常量，指针指向不可以改，指针指向的值可以更改
    int * const p2 = &a;
    //p2 = &b; //错误
    *p2 = 100; //正确

    //const既修饰指针又修饰常量
    const int * const p3 = &a;
    //p3 = &b; //错误
    //*p3 = 100; //错误

    system("pause");

    return 0;
}
```

技巧：看const右侧紧跟着的是指针还是常量，是指针就是常量指针，是常量就是指针常量

7.6 指针和数组

作用：利用指针访问数组中元素

示例：

```
int main() {

    int arr[] = { 1,2,3,4,5,6,7,8,9,10 };

    int * p = arr;  //指向数组的指针

    cout << "第一个元素：  " << arr[0] << endl;
    cout << "指针访问第一个元素：  " << *p << endl;

    for (int i = 0; i < 10; i++)
    {
        //利用指针遍历数组
        cout << *p << endl;
        p++;                        //指针往后偏移四个字节
    }

    system("pause");

    return 0;
}
```

7.7 指针和函数

作用：利用指针作函数的参数，可以修改实参的值

示例：

```
//值传递
void swap1(int a ,int b)
{
    int temp = a;
    a = b;
    b = temp;
}
//地址传递
void swap2(int * p1, int *p2)
{
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

int main() {

    int a = 10;
    int b = 20;
    swap1(a, b); // 值传递不会改变实参

    swap2(&a, &b); //地址传递会改变实参
```

```
    cout << "a = " << a << endl;

    cout << "b = " << b << endl;

    system("pause");

    return 0;
}
```

总结：如果不想修改实参，就用值传递，如果想修改实参，就用地址传递

7.8 指针，数组，函数

案例描述：封装一个函数，利用冒泡排序，实现对整型数组的升序排序

例如数组：int arr[10] = { 4,3,6,9,1,2,10,8,7,5 };

示例：

```
//冒泡排序函数
void bubbleSort(int * arr, int len)  //int * arr 也可以写为int arr[]
{
    for (int i = 0; i < len - 1; i++)
    {
        for (int j = 0; j < len - 1 - i; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

//打印数组函数
void printArray(int arr[], int len)
{
    for (int i = 0; i < len; i++)
    {
        cout << arr[i] << endl;
    }
}

int main() {

    int arr[10] = { 4,3,6,9,1,2,10,8,7,5 };
    int len = sizeof(arr) / sizeof(int);

    bubbleSort(arr, len);

    printArray(arr, len);

    system("pause");
```

```
    return 0;
}
```

总结：当数组名传入到函数作为参数时，被退化为指向首元素的指针

8 结构体

8.1结构体的基本概念

结构体属于用户自定义的数据类型，允许用户存储不同的数据类型

8.2结构体的使用

通过结构体创建变量的方式有三种：

- struct 结构体名 变量名
- struct 结构体名 变量名 = { 成员1值， 成员2值...}
- 定义结构体时顺便创建变量

示例：

```
//结构体定义
struct student
{
    //成员列表
    string name;  //姓名
    int age;      //年龄
    int score;    //分数
}stu3; //结构体变量创建方式3


int main() {

    //结构体变量创建方式1
    struct student stu1; //struct 关键字可以省略

    stu1.name = "张三";
    stu1.age = 18;
    stu1.score = 100;

    cout << "姓名：" << stu1.name << " 年龄：" << stu1.age  << " 分数：" <<
stu1.score << endl;

    //结构体变量创建方式2
    struct student stu2 = { "李四",19,60 };

    cout << "姓名：" << stu2.name << " 年龄：" << stu2.age  << " 分数：" <<
stu2.score << endl;


    stu3.name = "王五";
    stu3.age = 18;
    stu3.score = 80;
```

```
    cout << "姓名：" << stu3.name << " 年龄：" << stu3.age   << " 分数：" <<
stu3.score << endl;

    system("pause");

    return 0;
}
```

总结1：定义结构体时的关键字是struct，不可省略

总结2：创建结构体变量时，关键字struct可以省略

总结3：结构体变量利用操作符 “.” 访问成员

8.3结构体数组

作用：将自定义的结构体放入到数组中方便维护

语法：`struct 结构体名 数组名[元素个数] = { {} , {} , ... {} }`

```
#include<iostream>
using namespace std;
#include<string>
struct meinv                    //定义结构体
{
    string name;
    int age;
    string cup;
};

int main()
{
    struct meinv arry[3] =                  //定义结构体数组
    {
        {"刻晴",24,"A"},
        {"神里凌华",23,"A"},
        {"甘雨",3000,"B"}
    };
    arry[1].name = { "xxxx" };              //如何修改结构体数组元素
    arry[1].age = { 25 };
    for (int i = 0; i < 3; i++)             //如何遍历结构体数组
    {
        cout << "姓名:" << arry[i].name << "年龄:" << arry[i].age << "CUP:" <<
arry[i].cup << endl;
    }
    system("pause");
    return 0;
}
```

8.4结构体指针

作用：通过指针访问结构体中的成员

- 利用操作符 `->`可以通过结构体指针访问结构体属性

```
#include<iostream>
```

```
using namespace std;
#include<string>
struct meinv                //定义结构体
{
    string name;
    int age;
    string cup;
};

int main()
{
    struct meinv arry = { "神里凌华",23,"A" };
    struct meinv * p = &arry;
    p->name= "xxxx";

    cout << "姓名:" << p->name << "年龄:" << p->age << "CUP:" << p->cup << endl;

    system("pause");
    return 0;
}
```

8.5结构体--嵌套结构体

作用： 结构体中的成员可以是另一个结构体

```
#include<iostream>
using namespace std;
#include<string>
//学生结构体定义
struct student
{
    //成员列表
    string name;  //姓名
    int age;      //年龄
    int score;    //分数
};

//教师结构体定义
struct teacher
{
    //成员列表
    int id; //职工编号
    string name;  //教师姓名
    int age;   //教师年龄
    struct student stu; //子结构体 学生
};


int main() {

    struct teacher t1;
    t1.id = 10000;
    t1.name = "老王";
    t1.age = 40;
```

```
    t1.stu.name = "张三";
    t1.stu.age = 18;
    t1.stu.score = 100;

    cout << "教师 职工编号： " << t1.id << " 姓名： " << t1.name << " 年龄： " <<
t1.age << endl;

    cout << "辅导学员 姓名： " << t1.stu.name << " 年龄：" << t1.stu.age << " 考试分
数： " << t1.stu.score << endl;

    system("pause");

    return 0;
}
```

总结：在结构体中可以定义另一个结构体作为成员，用来解决实际问题

8.6 结构体做函数参数

作用：将结构体作为参数向函数中传递

传递方式有两种：

- 值传递
- 地址传递

```
#include<iostream>
using namespace std;
#include<string>
struct character
{
    string name;
    int age;
    string cup;
};
void printCharacter1(struct character c1)
{
    c1.cup = "C";
    cout << "角色名字:" << c1.name << endl << "角色年龄:" << c1.age << endl << "角色
cup:" << c1.cup << endl;
}
void printCharacter2(struct character *c1)
{
    c1->cup = "D";
    cout << "角色名字:" << c1->name << endl << "角色年龄:" << c1->age << endl << "角
色cup:" << c1->cup << endl;
}
int main() {

    struct character c1 = { "刻晴",24,"B" };
    printCharacter1(c1);
    //cout << "角色名字:" << c1.name <<endl<< "角色年龄:" << c1.age <<endl<< "角色
cup:" << c1.cup << endl;
    printCharacter2(&c1);
    cout << "主函数中的角色名字:" << c1.name << endl << "主函数中的角色年龄:" << c1.age
<< endl << "主函数中的角色cup:" << c1.cup << endl;
```

```
    system("pause");

    return 0;
}
```

总结：如果不想修改主函数中的数据，用值传递，反之用地址传递

8.7结构体中const使用场景

作用：用const来防止误操作

```
#include<iostream>
using namespace std;
#include<string>
struct character
{
    string name;
    int age;
    string cup;
};
void printCharacter( const character *c1)   //地址传递比值传递生内存
{
    //c1->cup = "D";       //const 将传递的值变为只能读取，不可写入的值    ，改变传入值就会报错
    cout << "角色名字:" << c1->name << endl << "角色年龄:" << c1->age << endl << "角色cup:" << c1->cup << endl;
}
int main()
{
    struct character c1 = { "刻晴",24,"B" };
    printCharacter(&c1);
    system("pause");
    return 0;
}
```

8.8.1 结构体练习（案例1）

案例描述：

学校正在做毕设项目，每名老师带领5个学生，总共有3名老师，需求如下

设计学生和老师的结构体，其中在老师的结构体中，有老师姓名和一个存放5名学生的数组作为成员

学生的成员有姓名、考试分数，创建数组存放3名老师，通过函数给每个老师及所带的学生赋值

最终打印出老师数据以及老师所带的学生数据。

```
#include<iostream>
using namespace std;
#include<string>
#include<ctime>
struct student
{
    string name;
    int score;
};
struct teacher
```

```
{
    string name;
    struct student s_array[5];
};
void allocate(teacher t_array[], int len)
{
    for (int i = 0; i < len; i++)
    {
        string t_name = "教师";
        string s_name = "学生";
        string nameseed = "ABCDE";
        t_array[i].name = t_name + nameseed[i];
        for (int j = 0; j < 5; j++)
        {
            t_array[i].s_array[j].name = s_name + nameseed[j];
            t_array[i].s_array[j].score = rand() % 61 + 40;      //40~100
        }
    }
}
void printinfo(teacher t_array[],int len)
{
    for (int i = 0; i < len; i++)
    {
        cout << "教师的姓名:" << t_array[i].name << endl;
        for (int j = 0; j < 5; j++)
        {
            cout << "\t学生的姓名:" << t_array[i].s_array[j].name << "学生的分数:"
<< t_array[i].s_array[j].score << endl;
        }
    }
}
int main()
{
    srand((unsigned int)time(NULL));    //随机数种子 头文件 #include <ctime>
    teacher t_array[3];
    int len = sizeof(t_array) / sizeof(t_array[0]);
    allocate(t_array,len);
    printinfo(t_array, len);
    system("pause");
    return 0;
}
/*要取得 [a,b) 的随机整数，使用 (rand() % (b-a))+ a;

要取得 [a,b] 的随机整数，使用 (rand() % (b-a+1))+ a;

要取得 (a,b] 的随机整数，使用 (rand() % (b-a))+ a + 1;

通用公式: a + rand() % n；其中的 a 是起始值，n 是整数的范围。

要取得 a 到 b 之间的随机整数，另一种表示：a + (int)b * rand() / (RAND_MAX + 1)。

要取得 0~1 之间的浮点数，可以使用 rand() / double(RAND_MAX)。*/
```

8.8.2 结构体案例练习2

案列描述设计一个英雄的结构体，包括成员姓名，年龄;创建结构体数组，数组中存放5名英雄。

通过冒泡排序的算法，将数组中的英雄按照年龄进行升序排序，最终打印排序后的结果。

```
#include<iostream>
using namespace std;
#include<string>
struct hero
{
    string name;
    int age;
};
void bubblesort(hero array[],int len)
{
    for (int i = 0; i < len - 1; i++)
    {
        for (int j = 0; j < len - i - 1; j++)
        {
            if (array[j].age < array[j + 1].age)
            {
                hero temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}
void printhero(hero array[], int len)
{
    for (int i = 0; i < 5; i++)
    {
        cout << "姓名:" << array[i].name << "\t年龄:" << array[i].age << endl;
    }
}
int main()
{
    struct hero array[5] =
    {
        {"刻晴",24},
        {"神里凌华",23},
        {"可莉",10},
        {"七七",300},
        {"甘雨",260}
    };
    int len = sizeof(array) / sizeof(array[0]);
    bubblesort(array,len);
    printhero(array, len);
    system("pause");
    return 0;
}
```