1.3：C++变量

1、：C++在创建变量时，必须给变量一个初始值，否则会报错

1.6：标识符和关键字

\* 标识符不能是关键字

\* 标识符只能由字母、数字、下划线组成

\* 第一个字符必须为字母或下划线

\* 标识符中字母区分大小写

> 建议：给标识符命名时，争取做到见名知意的效果，方便自己和他人的阅读

2.3实型（也称浮点型）

1.c++中有效位数包括整数部分

2.float=的数字后最好加个f；

3. //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;

2.4字符型

> 注意1：在显示字符型变量时，用单引号将字符括起来，不要用双引号

> 注意2：单引号内只能有一个字符，不可以是字符串

字符型变量只占一个字节

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

2.5转义字符：

1、换行：\n

2、输出一个\:\\ 注：单数个\会报错，没两个\\输出一个\ \\\\输出两个\\

3、水平制表：\t (8个宽度) 好处：可以整齐地输出

2.6字符串型

1. **\*\*C风格字符串\*\***： `char 变量名[] = "字符串值"`

注意：1.[] 2.用””

2.**\*\*C++风格字符串\*\***：  `string  变量名 = "字符串值"`

注意：1仍要”” 2.前面要包含#include<string>

2.7 bool数据类型

bool数据类型占一个字节大小

bool类型只有两个值：

\* true  --- 真（本质是1）

\* false --- 假（本质是0）

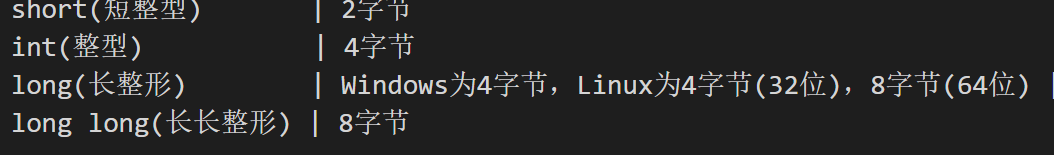
只要是非0的值都代表真

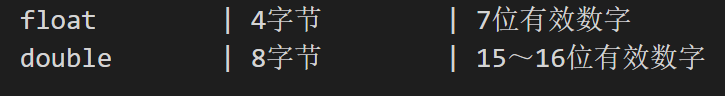
2.8数据的输入

Cin>>

bool的输入不能用false true 要用数字 其中只有0表示假，其他都表示真

2的总结：

各数据类型的大小





字符串28个字节。



3.1算术运算符

1. /:int定义的整数，相除时，若带小数，直接把小数部分抹去。
2. %:10%20=10，q除数比被除数大时，取余得到被除数本身，同时除数也不可以为0.

两个小数不可以做取模运算

前置运算符++a和后置运算符a++区别

//前置递增先对变量进行++，再计算表达式

    int a2 = 10;

    int b2 = ++a2 \* 10;

    cout << b2 << endl; //输出为110

    //后置递增先计算表达式，后对变量进行++

    int a3 = 10;

    int b3 = a3++ \* 10;

    cout << b3 << endl; //输出为100

3.2赋值运算符

a\*=2； 等价于a=a\*2；

其他%/+-同理

3.3 比较运算符

== 等于 ！=不等于 <= >=小大于等于

3.4逻辑运算符

！ 非

&& 与

||或

打印时注意 cout << (a && b) << endl

要加（），使之优先执行

4程序流程结构

4.1 if条件判断

注：if()后面不要加；

1 单行：

If（）

{

}

2.多行

if（）

{

}

else

{

}

注意：（）{}后都没有；

1. 多条件if语句
2. `if(条件1){ 条件1满足执行的语句 }else if(条件2){条件2满足执行的语句}... else{ 都不满足执行的语句}`
3. 嵌套if语句

if(){

if()

{}

Else{

}

}

4.1.2 三目运算符

表达式1 ? 表达式2 ：表达式3

如果表达式1的值为真，执行表达式2，并返回表达式2的结果；

如果表达式1的值为假，执行表达式3，并返回表达式3的结果。

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;

4.1.3 switch语句

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;

}

 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

如果两个条件执行的东西一样，可以这样：

case 1o:

case 9: …;break;

这样9和10执行的东西都一样

4.2循环结构

4.2.1 while循环

While（）

{…

}

注意：在执行循环语句的时候，程序必须提供跳出循环的出口，否则就会出现死循环。

rand()%100 系统随机生成0-99的一个数

但是上面的是伪随机数，每次生成的都一样，要生成真随机数

1. 写上系统时间头文件 #include<ctime>
2. 写随机数种子：srand((unsigned int)time(NULL));

4.2.2 do{}while(); 循环语句

和while的区别，dowhile会先执行一次

**4.2.3 for循环语句**

**\*\*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; }`

注：for可以拆分，

例：

for（int a=0;a<10;a++）{…}

等价于

int a=0;

for(;;)

{

if(a>=10)

{break;}

a++;

}

注意：for循环中的表达式，要用分号进行分隔

> 总结：while , do...while, for都是开发中常用的循环语句，for循环结构比较清晰，比较常用

Continue: 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

Goto语句：

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

例：

    cout << "1" << endl;

    goto FLAG;

    cout << "2" << endl;

    cout << "3" << endl;

    cout << "4" << endl;

    FLAG:

    cout << "5" << endl;

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

**\*\*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

* + 1. 一维数组

1. 一维数组定义的三种方式：
2. 1. ` 数据类型  数组名[ 数组长度 ]; `
3. 2. `数据类型  数组名[ 数组长度 ] = { 值1，值2 ...};`
4. 3. `数据类型  数组名[ ] = { 值1，值2 ...};`

注：用2.时，按顺序赋值，没赋值的数默认为0；

数组定义时一定要有确定的长度

5.2.2

一维数组名的用途：

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

eg1:sizeof(数组名)

eg2:直接打印数组名 打印出来的是16进制的地址， 前面加int可以把16进制强行转为十进制。

&arr[3]可以知道第四个元素的地址。

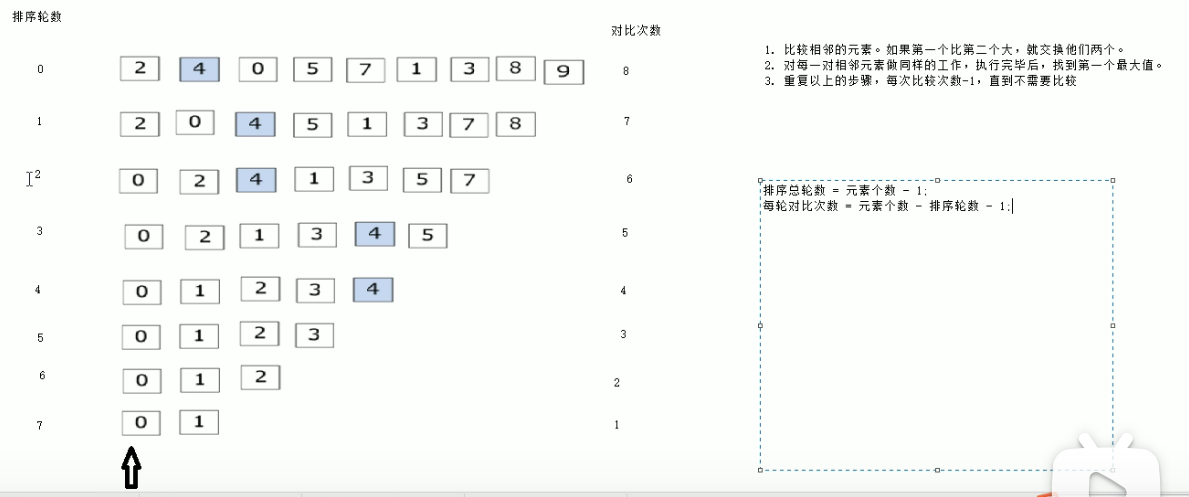
：数组名是常量，不可以赋值

5.2.3 冒泡排序

1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。

2. 对每一对相邻元素做同样的工作，执行完毕后，找到第一个最大值。

3. 重复以上的步骤，每次比较次数-1，直到不需要比较



排序总轮数=元素个数-1，

每轮对比次数=元素个数-排序轮数-1

int a[10] = { 3,6,24,5,2,8,0,65,5,8 };

int t = sizeof(a) / sizeof(a[0]) - 1;

for (int i = 0; i < t-1;i++) //排序总轮数=元素个数-1

{

for (int j = 0; j <= t - i - 1; j++) //每轮对比次数=元素个数-排序轮数-1

{

if (a[j] > a[j + 1])

{

int b = a[j];

a[j] = a[j + 1];

a[j + 1] = b;

}

}

}

for (int i = 0; i < 10; i++)

{

cout << a[i] << endl;

}

5.2.4 二维数字定义方式

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

可以省略行数，但一定不可以省略列数

5.2.5 二维数组名

\* 查看二维数组所占内存空间

\* 获取二维数组首地址

二维数组的大小/第一行的大小=行数

第一行大小/单个元素的大小=列数

cout<<arr[1]<<endl;可以知道第二行的地址

具体元素的地址需要加& 例： cout<<&arr[0][1]<<endl;

**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：\*\***

```C++

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 ////注：参数列表要写参数类型，如int char

\* 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

```C++

//函数定义

int add(int num1, int num2)

{

    int sum = num1 + num2;

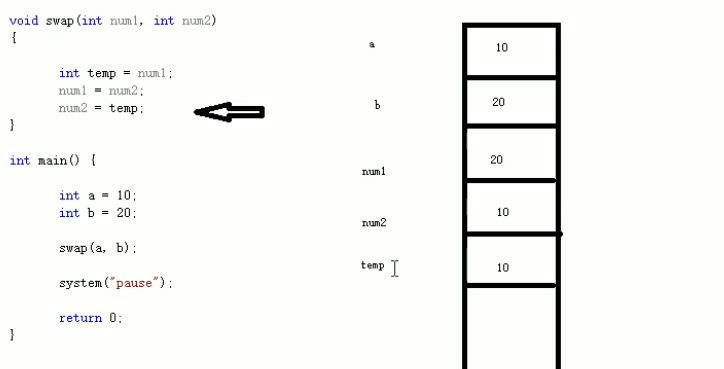
    return sum;

}

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==



用了函数过后对主函数的a b的值没影响。

6.5函数的声明

函数的**\*\*声明可以多次\*\***，但是函数的**\*\*定义只能有一次\*\***

函数写在主函数前面的时候可以不用声明，但是写在主函数后面的时候，一定要声明。

例：

int max(int a,int b);

6.6函数的分文件编写

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

解决方案资源管理器左键头文件和源文件可以分别添加.cpp和.h的头文件和源文件。

对3：要写函数的基本框架

#include<iostream>

Using namespace std;

对4.cpp源文件：要有#include”max.h” //max.h是头文件的文件名 //双引号代表这是自定义的头文件 //这样编译器就知道头文件和源文件是有关联的

最后，在要用的文件中只要开头有#include”max.h” 就能用了

7指针

7.1: **指针的作用：\*\*** 可以通过指针间接访问内存

指针定义语法： 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

    cout << &a << endl; //打印数据a的地址

    cout << p << endl;  //打印指针变量p

    //2、指针的使用

    //通过\*操作指针变量指向的内存

    cout << "\*p = " << \*p << endl;

    system("pause");

    return 0;

}

使用指针的解引用可以改变变量的值。

int a=10；

int\* p=&a;

\*p=50;

感觉int\* p中\*的作用是告诉编译器p是指针变量；而\*p=50；中的\*是解引用，两个\*本质应该不同

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指针所占内存空间

所有指针类型在32位操作系统下是4个字节

64位8个字节

7.3&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

int main() {

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

    cout << \*p << endl; //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

    system("pause");

    return 0;

}

**野指针\*\***：指针变量指向非法的内存空间

int main() {

    //指针变量p指向内存地址编号为0x1100的空间

    int \* p = (int \*)0x1100;//0x1100是一个随机乱写的野指针

    //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：谁在前谁不就不能改

1. const在前   --- 常量指针：指针变量，即可以改为另一个地址，但是原地址中的数不能改

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

int a=10;

int b=20;

const int \* p1 = &a;

可以：p1=&b;

不可以：\*p1=100;

2. const在后   --- 指针常量：指针变量为定值，即指针变量指向的地址为定值不能改，但是这个地址中的值可以改

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 a=10;

int b=20;

int \* const p2 = &a;

可以：

\*p2=b；

不可以：

p2=&b；

1. const即修饰指针，又修饰常量

const int \* const p3 = &a;

7.6指针和数组

P指向数组首地址时，利用指针p++，就可以访问第二个数组元素，可借此来遍历数组

7.7函数和指针

利用指针作函数参数，可以修改实参的值

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

**指针、数组、函数**

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结构体的定义和使用

定义：的语法：struct 结构体名 { 结构体成员列表 }；

使用：创建结构题的语法：

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；

> 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] = {  {} , {} , ... {} }`

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

(\*指针变量名).成员名

指针变量名->成员名

传入指针的函数中时两者是等价的

结构体变量.成员名

在主函数中用时，三者等价

**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

例子：teacher.stduent.name就是老师结构体中学生结构体中学生的姓名

**### 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

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的使用场景

**作用：**用const来防止误操作