



C++语言知识点

C++ 程序框架

```
#include <iostream>
using namespace std;
int main() {

    return 0;

}
```

注释符

单行注释符: //

多行注释符: /* */

注释符注释的内容不会被计算机编译和执行

换行符

1. \n

使用于引号内 " \n "

2. endl

用 << 符号接着写 endl

输出操作符: <<

C++ 提供了一个名为 cout 的对象，可用于在计算机屏幕上显示信息，cout 这个词可以看作是来源于 console output（控制台输出）。

使用方法: cout << "文本";

数据类型

1. 整数 (int)
2. 浮点数 (double、float)
3. 单个字符 (char)

变量的定义

数据类型 变量名;

例: `int a;`
`double s;`

变量的命名规则

- [1] 第一个字符必须是字母或者下划线 (_)。
- [2] 其余部分可以是字母、数字或者下划线 (_)。
- [3] 不能使用关键字 (即 C++ 里面有特殊含义的单词, 例如: `int`、`double`、`float`等)。
- [4] 变量名区分大小写。

变量的赋值

格式：变量=表达式;

例：a=100; //变量a被赋值为100

a=b; //变量a被赋值为b的值

【注意】字符变量需要加单引号，如：

char c='A' ;

变量的交换

把两个变量中的值进行交换，需要再加一个中介变量。

例：把 a 和 b 的值交换，利用变量c

c=a; a=b; b=c;

输入操作符:>>

格式：cin>>变量;

连续输入：cin>>变量1>>变量2>>变量3;

在控制台连续多次输入以空格或者换行表示一次输入结束，不影响变量的存储和输出。

算术运算符

计算机中进行运算的符号与平时数学中使用的符号有所不同

加： +

减： -

乘： *

除： /

C++语言的算术运算是比较严格的，不同于我们平常学习的数学情况。所以只要参与计算（在四则运算的过程）的数值是**整数**，得出来的一定是**整数**（只保留整数部分，小数部分去除），若参与计算（在四则运算的过程）的数值中有**浮点数**，得出来的一定是**浮点数**。例如， $5/2$ 的值是 2，而不是 2.5，而 $5.0/2$ 或 $5/2.0$ 的值是 2.5。

表达式

表达式是由数字、算符、数字分组符号（括号）、自由变量和约束变量等以能求得数值的有意义排列方法所得的组合。

例： $(16+14)*10$

$9 \leq 10$

$a+10*15+b$

如何获得一个三位数个位、十位、百位的数？

取余 “%” 和整除 “/”

个位: $a = n \% 10$

十位: $b = n / 10 \% 10$

百位: $c = n / 100 \% 10$



%

模 (取余)

% (模)：直接取余数。一个数对比它大的数取余等于这个数本身
任何数对1去余都等于0。

C++把基本数据类型归结为以下格式：

- (1)整数 (int)
- (2)浮点数 (double、float)
- (3)单个字符 (char)
- (4)布尔类型 (bool)

用sizeof（数据类型）或sizeof（变量名）可以获得这个数据类型的空间大小。

例如：

```
sizeof(int) ; //获得： 4
char c;
sizeof(c) ; //获得： 1
```

整型数据类型的大小

数据类型	类型标识符	字节数	取值范围
短整型	short	2	$-32768(-2^{15}) \sim 32767(2^{15}-1)$
整型	int	4	$-2147483648(-2^{31}) \sim 2147483647(2^{31}-1)$
超长整型	long long	8	$-9223372036854775808(-2^{63}) \sim 9223372036854775807(2^{63}-1)$

实数数据类型的大小

数据类型	类型标识符	字节数	取值范围
单精度实型	float	4	$-3.4\text{E}-38 \sim 3.4\text{E}+38$
双精度实型	double	8	$-1.7\text{E}+308 \sim 1.7\text{E}+38$
长双精度实型	long double	16	$-3.4\text{E}+4932 \sim 1.1\text{E}+4932$

字符类型和布尔数据类型的大小

数据类型	类型标识符	字节数	取值范围
字符	char	1	-128~127
布尔变量	bool	1	0或1

C++语言中，不同数据类型的运算对象进行混合运算，或者需要将一个表达式的结果转换成期望的类型时，就需要依据数据类型转换规则进行转换。

数据的类型转换分为隐式转换和强制转换

隐式转换

概念：隐式转换是指不需要我们干预，电脑自己对数据类型进行了转换。

转换原则：隐式转换是以数据类型的取值范围来作为转换基础，都是从取值范围小的类型自动向取值范围大的类型的原则进行转换的。

隐式转换发生条件：

- 1.混合运算时产生的数据自动类型转换
- 2.不同数据类型之间的赋值操作

强制类型转换

当自动类型转换不能实现目的时，可以强制地进行数据类型转换

强制类型转换格式：

(数据类型)变量;

(数据类型)(表达式);

例如：(double)a; → 将变量a强制转化为double类型

(int)(x+y); → 将x+y的值强制转换为整型

【注意】进行数据类型转换时，都是为了本次运算而对数据进行临时性的转换，不改边变量定义的数据类型。

```
a = a + 3;  
b = b - 6;  
c = c * 9;  
d = d / 2;  
e = e % 8;
```

复合赋值运算符

+=

a += 3;

-=

b -= 6;

*=

c *= 9;

/=

d /= 2;

%=

e %= 8;

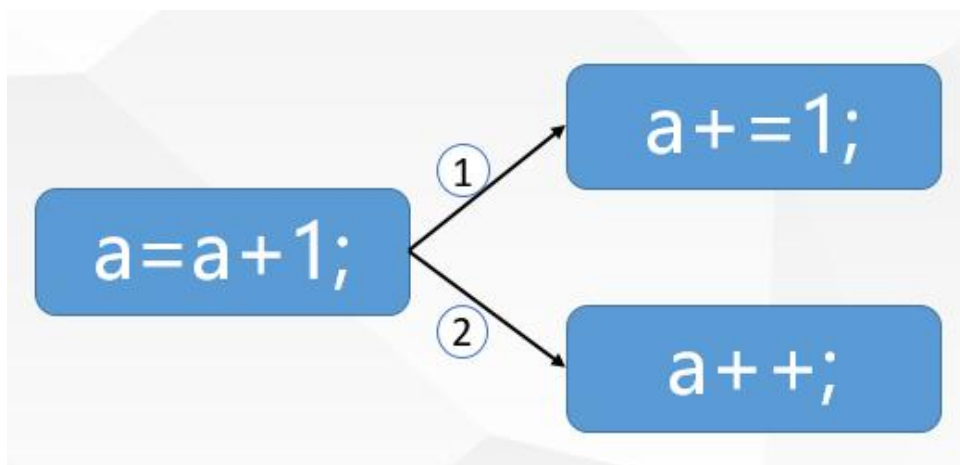
省略右边变量
运算符写到赋值符号的前面

【注意】复合赋值运算符的左边必须是变量。

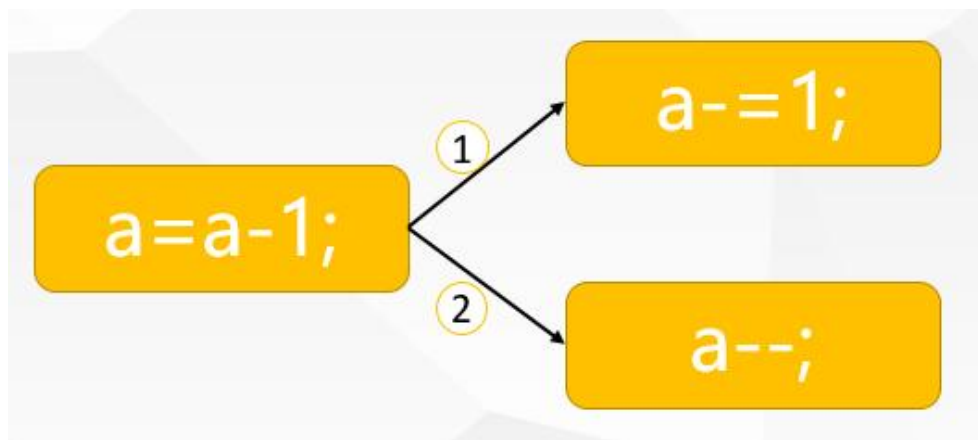
根据运算符的优先级, $+$, $-$, $*$, $/$, $\%$ 运算符优先于赋值运算符, 所以先运行 “=” 右边变量自身进行运算的部分, 再把运算结果赋值给 “=” 左边的变量。

“=” 左边的变量和右边运算的变量是同一个, 相当于相同的那个变量本身进行右边的运算, 再把运算结果赋值回变量自己。

自增自减运算符



`++`符号叫**自增运算符**，运行效果和`a=a+1;`语句一样，都是变量+1，再把运算结果赋值回给变量自身，实现**变量增加1**的结果。



`--`符号叫**自减运算符**，运行效果和`a=a-1;`语句一样，都是变量-1，再把运算结果赋值回给变量自身，实现**变量减去1**的结果。

自增自减符号可放在变量的后面或者前面，结果都能实现变量自增或者自减1，一旦结合其他代码，执行顺序可大不同。

`a++;`

单独一个语句

变量a加1

`++a;`

单独一个语句

变量a加1

`cout<<a++;`

结合其他代码

先使用变量再加1

`cout<<++a;`

结合其他代码

先加1再使用变量

常用数学函数介绍 (添加头文件 `<cmath>`)

1、`floor()`函数

功能：把一个**小数**向下取整(向下取整：比自己小的最大整数)

使用方法：

`floor(需要取整的小数);`

2、`ceil()`函数

功能：把一个**小数**向上取整(向上取整：比自己大的最小整数)

使用方法：

`ceil(需要取整的小数);`

3、`sqrt()`函数

功能：把一个**实数**开平方(开方指求一个数的方根的运算，为乘方的逆运算)，使用方法：

`sqrt(需要开平方的数);`

常用数学函数

绝对值、取整	
int abs(int i)	返回整型参数i的绝对值
long labs(long n)	返回长整型型参数n的绝对值
double fabs(double x)	返回实数型参数x的绝对值
double ceil(double x)	取上整
double floor(double x)	取下整
double round(double x)	四舍五入取整

指数、对数、开方

double log(double x)	返回自然对数$\ln(x)$的值
double log10(double x)	返回常用对数$\lg(x)$的值
double pow10(int p)	返回10的p次幂
double pow (double x,double y)	返回x的y次幂
double sqrt(double x)	返回x的平方根

三角函数 (double x为弧度)

double sin(double x)

返回x的正弦sin(x)的值

double cos(double x)

返回x的余弦cos(x)的值

double tan(double x)

返回x的正切tan(x)的值

double asin(double x)

返回x的反正弦函数的值

double acos(double x)

返回x的反余弦函数的值

double atan(double x)

返回x的反正切函数的值

计算机中的数学函数示例：

```
#include <cmath> //包含数学函数头文件
double d=3.14;
round(d); //对实数四舍五入取整数
floor(d); //对实数向下取整数
ceil(d); //对实数向上取整数
sqrt(d); //对实数开平方（求平方根）
```

左右对齐输出（添加头文件<iomanip>）

1、 设置域宽

功能：结合cout语句控制输出的内容

使用方法：

```
cout<<left<<setw(10)<<"内容"<<endl;
```

scanf 函数

功能：输入语句，作用与cin相同，向计算机输入数据

使用方法：

1、使用前需要加入头文件

#include <stdio.h>

2、使用的时候在括号里面写格式控制符和输入列表
scanf(格式控制符，输入列表);

使用格式:

tips:不写地址符&会报错哦!

`scanf (“格式控制字符串” , 地址列表);`

地址: 地址运算符 “&” + 变量名, 例如: `&a`, `&b`分别表示变量a和变量b的地址。

例如: `int a, b;`

`scanf (“%d%d” , &a, &b);`

相当于

`int a, b;`

`cin>>a>>b;`

注意: 使用scanf语句进行输入数据时, 输入格式要和scanf语句的格式**保持一致**

例如: `int a, b;`

`scanf (“%d, %d” , &a,&b);`

输入格式

`10,20`

格式控制字符串:

指定输出格式, 格式字符串是**以%开头**, 后面跟有各种格式的字符, 用来说明输出数据的类型、形式、小数位等

格式字符	意义
d	以十进制形式表示带符号整数
f	以小数形式表示单精度实数
lf	以小数形式表示双精度实数
c	表示单个字符

取地址符

变量在内存中的地址是由**十六进制**组成的，十六进制包含**0到9和字母A到F（或a~f）**，其中a~f是代表数字10~15。

类似二进制逢二进一，十六进制是逢16进一。

通过**&**可以获取到变量**在内存中的地址**。

例如：

```
int a;  
cout<<&a;
```


printf 函数

功能：输入语句，作用与cout相同，向计算机输出数据

使用方法：

1、使用前需要加入头文件

```
#include <stdio.h>
```

2、使用的时候在括号里面写需要取整的小数

```
printf(格式控制符, 输出列表);
```

例如: `int a=10,b=20;`
`printf ("%d%d\n" ,a,b) ;` 相当于 `int a=10,b=20;`
`cout<<a<<b<<endl;`

`printf ("输出内容\n") ;` 相当于 `cout<<" 输出内容" <<endl;`

注意: 在printf语句里, 换行只能用\n

printf和我们平常使用的cout不太一样,cout只是输出文本和结果, 没有任何格式效果
而printf可以通过不同的输出控制符, 以特定的格式输出结果, 比如**精确小数点后的位数**

1.保留小数

float b = 12.33333 double c = 45.600000 保留三位小数

printf ("%0.3f%0.3lf" ,b,c) ; 输出结果 → 12.33345.600

↑
保留位数,可以省略零

2.输出空格

printf("%0.3l %0.3lf" ,b,c); 输出结果 → 12.333 45.600

↑
空格

printf语句里的格式决定了结果的输出格式

例如: printf("%0.3lf——%0.3lf" ,b,c); 输出结果 → 12.300 ——45.600

tips:格式控制符里写的字符也会正常输出哦

布尔类型 (bool)

它和int、double、char一样是数据类型，
它的值只有两个：**true**、**false**

true

当**表达式成立**
的时候为真，
使用ture

false

当**表达式不成**
立的时候为假，
使用false

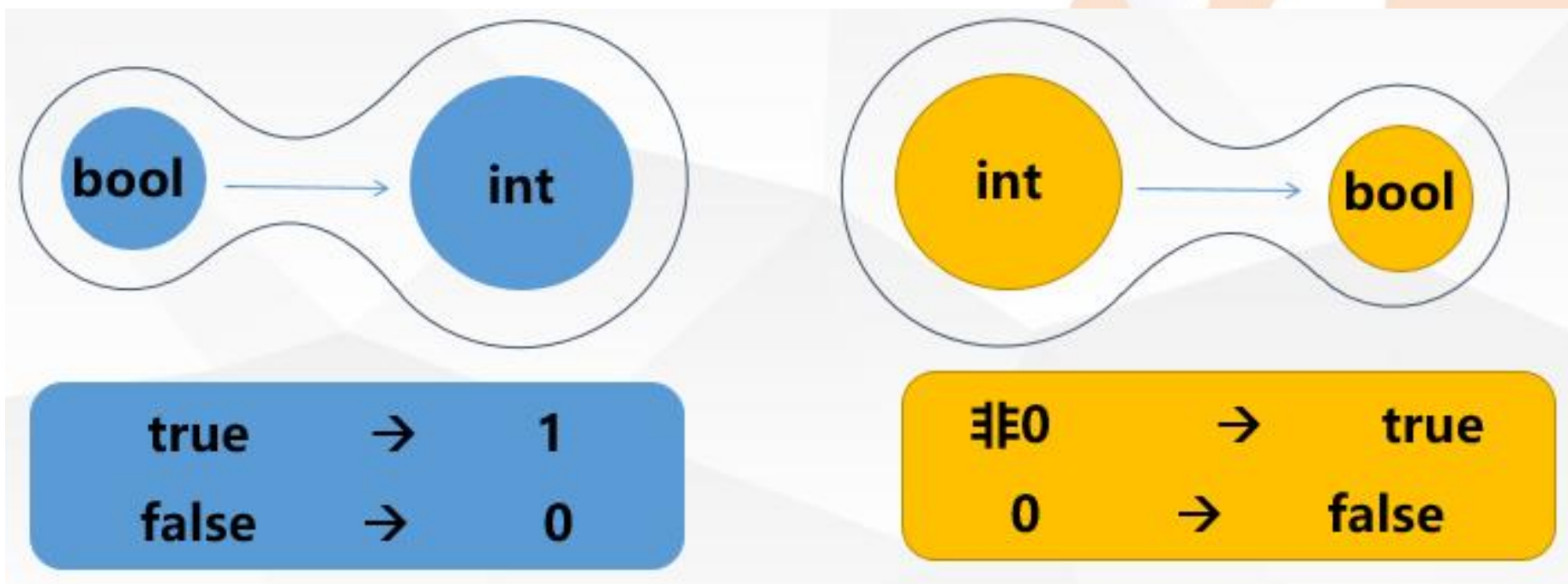
声明方式：bool 变量名;
例如：bool flag;



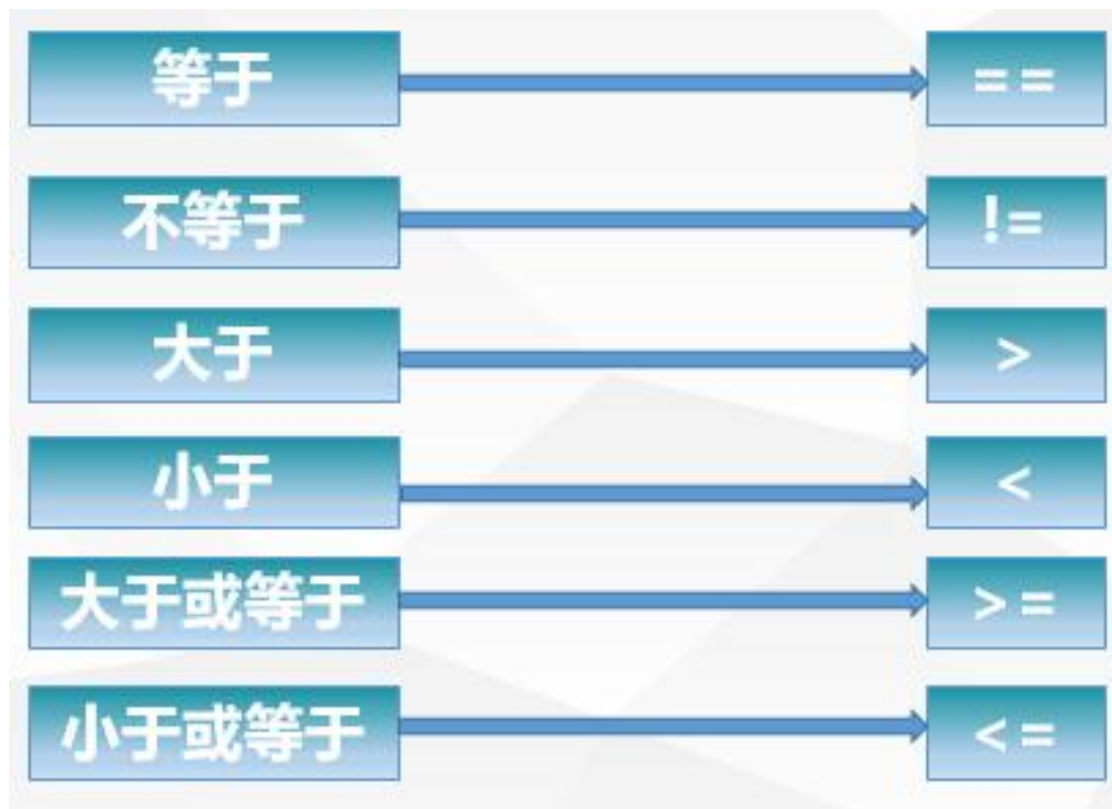
布尔类型的类型转换

在C++语言里是不会输出true和false，如果需要输出布尔类型只会输出1或0。


bool类型和int类型隐式转换：



比较运算符




比较运算符用于比较运算符两边之间的关系，比较运算符和运算符两边构成一个完整的关系表达式，结果返回一个布尔类型的 (**true/false**) 。




判断两边的表达式是否相等，**两边结果都正确返回true，否则返回false。**

&& 与



判断两边是否有正确的结果，**两边至少有一个正确的结果返回true，如两边结果都是错的，返回false。**

|| 或



非是用来否定某个表达式或结果，或者用来反转结果的真假。

! 非

逻辑运算符是用来连接多个(两个以及两个以上)表达式，单个表达式无法使用。

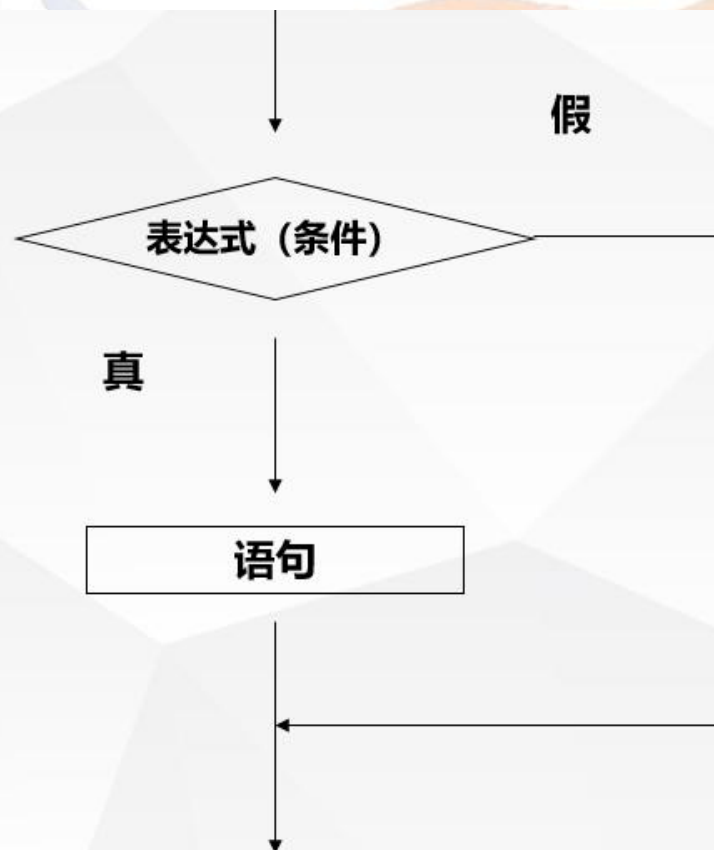
单分支结构

```
if( 条件 ){  
    语句;  
}
```

先判断条件是否成立！再执行接下来的任务！

如果条件达成，结果为true

条件不达成，退出语句



双分支结构

```
if( 条件 ) {  
    语句1;  
}else{  
    语句2;  
}
```

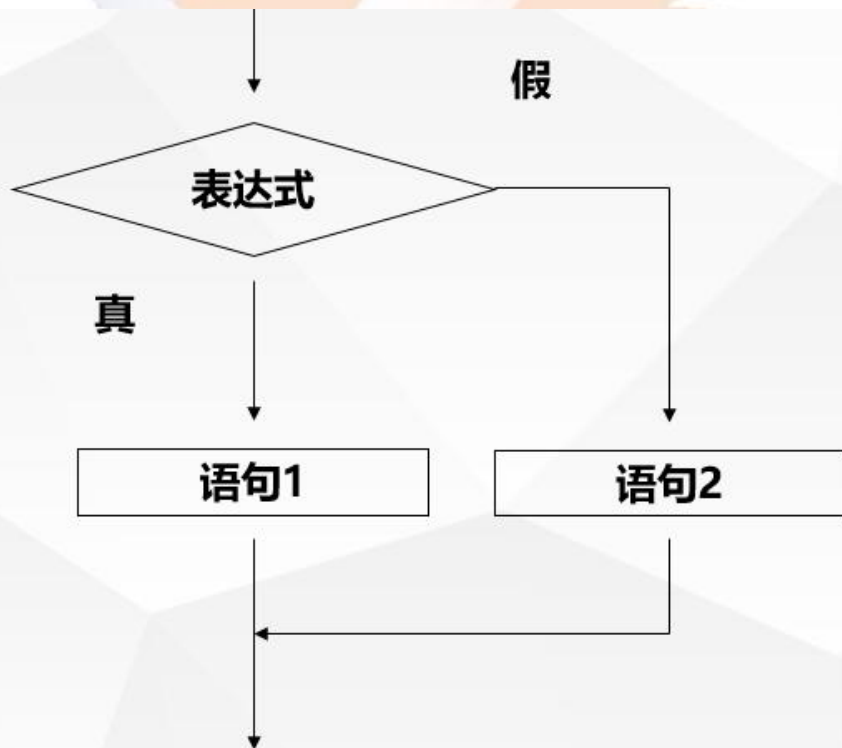
先判断条件是否成立！如果成立执行语句1



如果不成立执行语句2



不管如何总得有一个结果



三目运算 (if-else语句的另外一种形式)

三目运算符：是C++中仅有的需要3个操作数的操作符，格式为：

<表达式1> ? <表达式2> : <表达式3>;

这种形式也称为条件表达式，即条件运算符。

表达式1是一个逻辑值，可以为真或者为假。

如果表达式1为真，那么运算结果就是表达式2；

如果表达式1为假，那么运算结果就是表达式3。

例子：`int res=a>b?a:b; //取两者中的较大值`

嵌套if语句

在if...else...语句里再嵌套的写if或者if...else语句，就叫分支语句的嵌套。

在if里或者else里都能嵌套，可以有多种嵌套方式。根据需要进行程序的编写，但一定要注意逻辑关系的正确处理。

```
if (条件1) {  
    if (条件2) {  
        条件1和条件2都成立时执行的语句;  
    } else {  
        条件1成立但条件2不成立时执行的语句;  
    }  
} else { //条件1不成立  
    if (条件3) {  
        条件1不成立但条件3成立时执行的语句;  
    } else {  
        条件1和条件3都不成立时执行的语句;  
    }  
}
```

if多重选择分支语句

使用if语句时条件要写完整，因为几个if语句是相互独立的，条件不完整可能会有多个输出。

而if多重选择分支语句，条件之间相互作用，判断完一个条件就排除了一种情况，条件可以写得简单些，**只会输出其中一种结果**。如果有多个平级判断条件时候，使用if多重选择分支语句来处理会比较清晰和简单。

```
if ( 判断条件1 ){  
    条件1成立时执行的语句;  
}  
else if ( 判断条件2 ){  
    条件1不成立, 条件2成立时执行的语句;  
}  
else if ( 判断条件3 ){  
    条件1,2不成立, 条件3成立时执行的语句;  
}  
else if ( 判断条件4 ){  
    条件1,2,3不成立, 条件4成立时执行的语句;  
}  
else {  
    以上条件都不成立时执行的语句;  
}
```

switch-case语句



注意：switch语句中的表达式，其取值只能是**整数**、**字符型**、**布尔型**。

break;

作用：退出switch判断语句。

可以放在case语句的最后，代码执行break会退出整个switch语句。

switch语句原理是跳转到符合条件的case位置执行剩下所有的语句（包括其他case里面的）。直到语句的最后或者遇见break为止，因此为了防止把所有的结果都显示，**在每一条语句最后加上break即可。**

default:

作用：默认语句。

所有case语句都不执行的话，就会执行default语句，同时也是为了防止错误的输入导致程序没有结果。

if和switch的区别

- 1、 switch建议判断固定值的时候使用
 - 2、 if建议判断区间或范围的时候使用
- *用switch能做的，用if都能做，反过来则不行。

循环结构的特点:

- 1、有重复的操作
- 2、按照一定的规律重复

for循环结构

```
for(循环变量初始化;循环条件;循环变量改变){  
    循环体;  
}
```

for循环的执行过程

```
#include<iostream>
using namespace std;
int main(){
    for(int ①i=1; i<=②1000; i③++){
        cout<<i<<" ";
    }
    return 0;
}
```

第一次循环

① ② ④ ③

第二次循环

② ④ ③

第三次循环

② ④ ③

第n次循环

② ④ ③

最后执行语句

②

for循环结构

```
for(循环变量初始化;循环条件;循环变量改变){  
    循环体;  
}
```

- 1.最开始执行**循环变量初始化**语句，该语句只在第一次循环的时候执行。
- 2.接着执行**循环条件**语句，**判断**循环是否继续，如果循环条件成立则执行循环体语句，不成立则立即退出循环，每次循环都执行。
- 3.再接着执行**循环体**语句，每次循环都执行。
- 4.再接着执行**循环变量改变**语句，每次循环都执行。
- 5.到第二步执行**循环条件**语句**判断**循环是否继续。

累加累乘

累加	
初始值	公式
0	$s = s + i;$
累乘	
初始值	公式
1	$s = s * i;$



等差数列

等差数列是指从第二项起，每一项与它的前一项的差等于同一个常数的一种数列。这个常数叫做等差数列的公差，公差常用字母 d 表示。

通项公式为： $a_n = a_1 + (n-1) * d$ 。

例如：1, 3, 5, 7, 9, 11..... $2n-1$ 。

首项 $a_1=1$ ，公差 $d=2$ ， $a_n=1+(n-1)*2=2n-1$ 。

前 n 项和公式为： $S_n = a_1 * n + [n * (n-1) * d] / 2$ 或
 $S_n = [(a_1 + a_n) * n] / 2$ 。

水仙花数


水仙花数是三位数，三个数位上数的立方之和等于该数。

例如： $153 = 1 * 1 * 1 + 5 * 5 * 5 + 3 * 3 * 3$ ，所以153是水仙花数

break和continue


当程序运行到**break**语句的时候，程序会**终止**当前整个循环，执行当前循环以外的语句。

```
for(初始条件;判断条件1;循环动作){  
    if(判断条件2){  
        break;  
    }  
    语句1;  
}  
语句2;
```



当程序运行到**continue**语句的时候，程序会跳过本次循环剩余的部分，继续**下一次**循环。

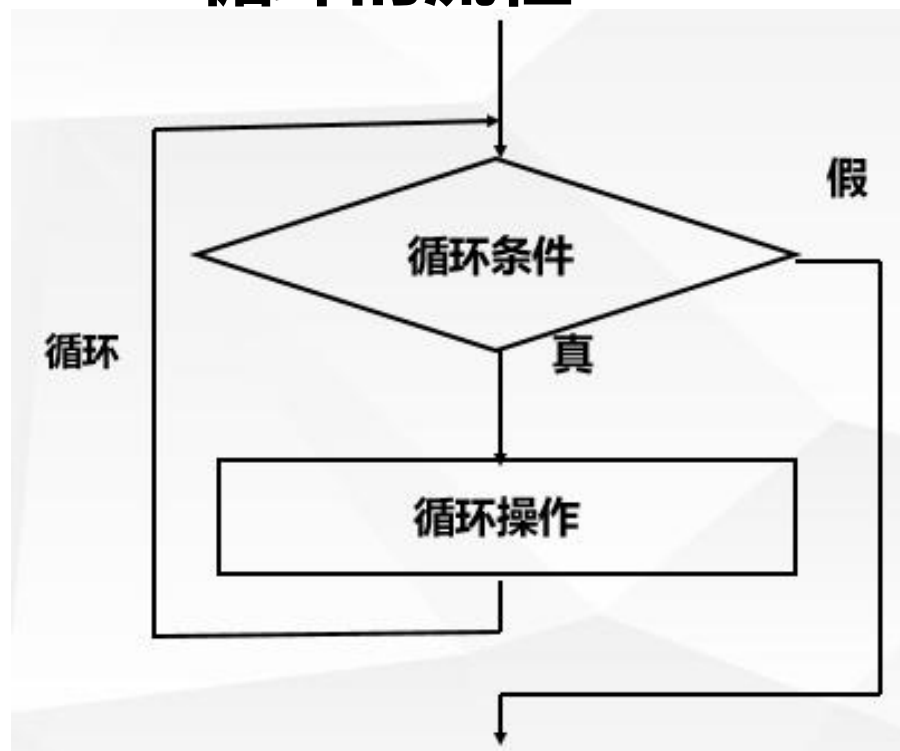
```
for(初始条件;判断条件1;循环动作){  
    if(判断条件2){  
        continue;  
    }  
    语句1;  
}
```



while循环

```
while(循环条件)
{
    循环语句;
}
```

while循环的流程



- 1、 首先判断条件表达式的值，如果它是**真(true)**的，则循环体中的每个语句都被执行;
- 2、 然后，再次测试条件，如果条件表达式仍然为**true**，则循环体中的每个语句将再次被执行;
- 3、 如此循环往复，直到条件表达式被判断为**false**则**终止**。

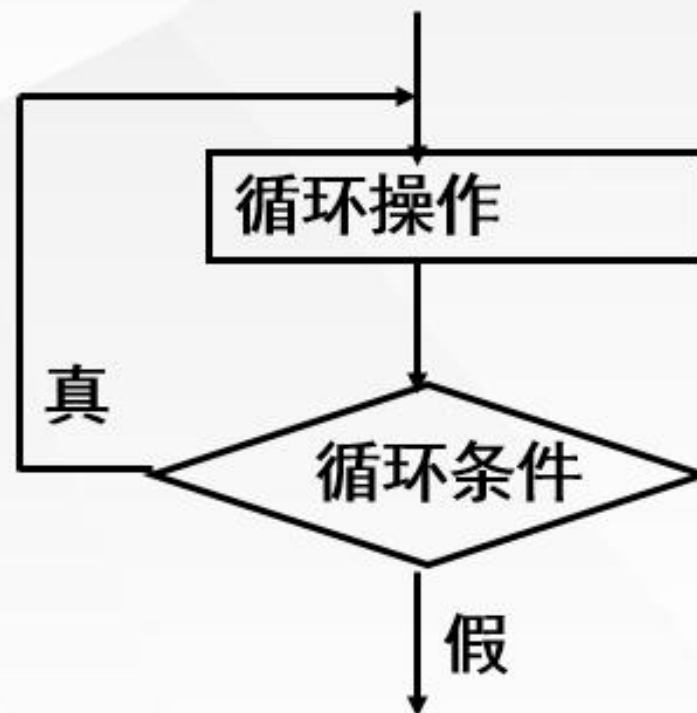
do-while循环

```
do {
```

循环操作

```
}while ( 循环条件 );
```

特点：先执行，再判断



while和for的区别

1、 不同场合使用不同的循环

for循环适用于循环的**开始和结束已知**，循环次数确定的场合

while循环适用于变化过程不连续且**循环次数不确定**的场合

2、 循环变量的使用

循环变量是否需要在循环外部使用。如需使用，则使用while循环

或者把循环变量定义到for循环外部，再使用for循环

数组的概念

数组就是一组类型相同的变量，往往用来表示同一批对象的统一属性。

数组可以用来存储一个**固定大小的,相同类型元素**的数列。

数组可以是一维的，也可以是二维的，甚至是多维的。

一维数组的定义

定义一维数组的一般格式：

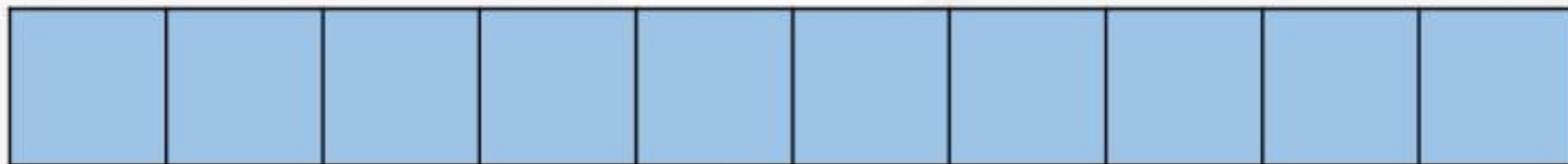
数据类型 数组名[元素个数];

例如：int a[10];

数组下标

数组定义后，会自动产生一排**有序**的号码来标记数组的存储空间，这些序号称为**下标**。借助下标我们可以找到数组的任一存储空间以及任意一个数组元素（也就是存储空间里的**值**）。

a数组的元素存放空间（房间）



a数组的下标（房间号）

0 1 2 3 4 5 6 7 8 9

需要注意的是：数组的下标一般从**0**开始，因此如果你定义了一个含有n个元素的数组，下标只能从**0**引用到**n-1**。

例如：

```
int a[10]; //这个数组的下标为0-9。
```

数组名加上数组的下标可以让计算机在内存中准确地存放数据和读取数据，这个特性通常叫做**随机存取**。

例如：

```
int i=1, j=2, a[10];  
a[8]=2;  
a[i+j]=4;
```

一维数组初始化

数据类型 数组名[数组长度]={值1,值2,值3,值4,.....};

在初始化的过程中，当初始化列表中的值少于数组元素个数时，编译器会把剩余元素初始化为0。

```
int a[5]={1,2,3,4,5}; //将数组中的元素分别初始化为1,2,3,4,5
```

```
int a[10]={0}; //将数组中的所有元素初始化为0
```

```
int a[10]={1}; //将数组中的第1个元素，也就是下标为0 的元素初始化为1，其他元素初始化为0
```

```
int a[10]={1,2}; //将数组中的第1个元素初始化为1，第2个元素初始化为2，其他元素初始化为0
```

```
int a[]={1,2,3,4,5}; //编译器自动匹配数组的长度为5
```


单独给某个数组元素读入数据或者赋值。

```
cin >> a[7];
```

```
a[6] = a[3] + 2;
```

```
a[5] = 100;
```

```
a[15] = a[20];
```

```
s = 7;  
a[2] = s + 9;
```

```
m = 4;  
a[m] = 30;
```

单独输出元素

```
cout << a[7];
```

```
m = 5;  
cout << a[m];
```

数组整体输入

数组名代表的并不是一个变量，而是一批变量，因而不能直接读入整个数组，而是要逐个读入数组元素，通常用循环结构来完成这一功能。从键盘读入n个数组元素的值最常用的方法：

```
for(int i=0;i<n;i++)  
    cin>>a[i];
```

数组整体输出

和数组元素的输入相同，数组元素的输出通常也用循环结构来实现的。

输出n个数组元素：

```
for(int i=0;i<n;i++)  
    cout<<a[i]<<' ';
```

数组越界

使用数组时，根据C++语言规定需要注意：

- (1) 数组元素的下标值为0和正整数。
- (2) 在定义元素个数的下标范围内使用。

C++语言中，数组越界访问时不会报任何的错误，但是，程序进行超出数组边界进行读/写，造成了内存数据混乱。所以为了防止数组越界，在编写完程序后，认真阅读程序是否有按照设计的来。

二维数组的定义

二维数组的定义和一维数组是类似的，不同之处只是多了一个下标：

二维数组定义的一般格式:

数据类型 数组名[行数][列数] ;

例如： `int a[4][10];`

a数组类似一个有4行、10列的表格，表格中有40（行数*列数）个存储空间，可存储40个数据，分析二维数组都尽量用表格来类比。

[illegible]

例如：int a[3][5];

表示a是二维数组（相当于一个3*5的表格），共有 $3*5=15$ 个元素，它们是：

a[0][0] a[0][1] a[0][2] a[0][3] a[0][4]

a[1][0] a[1][1] a[1][2] a[1][3] a[1][4]

a[2][0] a[2][1] a[2][2] a[2][3] a[2][4]

因此可以看成是一个**矩阵**（表格），a[2][3]即表示第3行第4列的元素。

二维数组的初始化

二维数组的初始化分为两种，一种是**顺序初始化**，一种是**按行初始化**。

```
int a[3][2]={4,2,5,6,7,8}; //顺序初始化
```

```
int b[3][2]={{4,2},{5,6},{7,8}}; //按行初始化,推荐使用
```

所谓按顺序初始化就是**先从左向右再由上而下**地初始化，即第一行所有元素都初始化好以后再对第二行初始化。而按行初始化则是用一对大括号来表示每一行，跳过前一行没有初始化的元素，在行内从左向右地进行初始化。**对于没有初始化的元素，则都是一个不确定的值。**

二维数组的元素存取

二维数组的元素存取与一维数组元素存取类似，区别在于二维数组元素的存取必须给出两个下标。

存取的格式为：

数组名[行下标][列下标];

简单记为：**先行后列。**

【说明】显然，每个下标表达式取值不应超出下标所指定的范围，否则会导致致命的**越界错误**。

二维数组的整体输入方式

输入二维数组有两种方式： (1) 按行输入 (2) 按列输入

按行输入

同一行中，行号一样，列号从0开始逐渐递增，
读下一行时，列号又从0开始逐渐递增。

```
int a[5][5];  
for(int i=0;i<5;i++){//i表示行号  
    for(int j=0;j<5;j++){//j表示列号  
        cin>>a[i][j];  
    }  
}
```

按列输入

同一列中，列号一样，行号从0开始逐渐递增，
读下一列时，行号又从0开始逐渐递增。

```
int a[5][5];  
for(int i=0;i<5;i++){//i表示列号  
    for(int j=0;j<5;j++){//j表示行号  
        cin>>a[j][i];  
    }  
}
```

字符的输入和输出

1、使用cin和cout进行字符输入输出的一般格式：

`cin >> 变量名;`

`cout << 变量名;`

2、使用scanf()和printf()进行字符输入输出的一般格式：

`scanf("%c" , 字符变量);`

`printf("%c" , 字符变量);`

3、使用getchar()和putchar()进行字符输入输出的一般格式：

`字符变量=getchar();`

`putchar(字符变量);`

总结:

(1) cin在读入字符时, **不会读入空格或者换行**, 会将空格或者换行作为字符间的分隔符。

(2) getchar()和scanf()在读入字符时, 会依次读取输入的**任意字符**, 所有输入都是合法的字符输入, 包括**空格**、**回车**等空白符。

字符的赋值

使用格式:

char 变量名 = '变量值';

例如: char c = 'c';

注意: 字符的值一定是用单引号";

高四位 低四位		ASCII 码控制字符												ASCII 码打印字符													
		0000						0001						0010		0011		01/0		0101		0110		0111			
		0						1						2		3		4		5		6		7			
		十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符
0000	0	0		^@	NUL	\0	空字符	16	►	^P	DLE		数据链路转义	32		48	0	64	@	80	P	96	`	112	p		
0001	1	1	☺	^A	SOH		标题开始	17	◄	^Q	DC1		设备控制1	33	!	49	1	65	A	81	Q	97	a	113	q		
0010	2	2	☹	^B	STX		正文开始	18	↕	^R	DC2		设备控制2	34	"	50	2	66	B	82	R	98	b	114	r		
0011	3	3	♥	^C	ETX		正文结束	19	!!	^S	DC3		设备控制3	35	#	51	3	67	C	83	S	99	c	115	s		
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4		设备控制4	36	\$	52	4	68	D	84	T	100	d	116	t		
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK		否定应答	37	%	53	5	69	E	85	U	101	e	117	u		
0110	6	6	♠	^F	ACK		肯定应答	22	—	^V	SYN		同步空闲	38	&	54	6	70	F	86	V	102	f	118	v		
0111	7	7	●	^G	BEL	\a	响铃	23	↑↓	^W	ETB		传输块结束	39		55	7	71	G	87	W	103	g	119	w		
1000	8	8	▣	^H	BS	\b	退格	24	↑	^X	CAN		取消	40	(56	8	72	H	88	X	104	h	120	x		
1001	9	9	○	^I	HT	\t	横向指标	25	↓	^Y	EM		介质结束	41)	57	9	73	I	89	Y	105	i	121	y		
1010	A	10	▣	^J	LF	\n	换行	26	→	^Z	SUB		替代	42	*	58	:	74	J	90	Z	106	j	122	z		
1011	B	11	♂	^K	VT	\v	纵向制表	27	←	^[BSC	\e	溢出	43	+	59	;	75	K	91	[107	k	123	{		
1100	C	12	♀	^L	FF	\f	换页	28	L	^\	FS		文件分隔符	44	,	60	<	76	L	92	\	108	l	124			
1101	D	13	♪	^M	CR	\r	回车	29	↔	^]	GS		组分分隔符	45	-	61	=	77	M	93]	109	m	125	}		
1110	E	14	🎵	^N	SOH		移出	30	▲	^^	RS		记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~		
1111	F	15	☼	^O	SI		移入	31	▼	^-	US		单元分隔符	47	/	63	?	79	O	95	_	111	o	127	△	^Backspace 代码:DEL	

注：表中的ASCII字符可以用“Alt + 小键盘上的数字键”方法输入

ASCII码

信息在计算机上是用二进制表示的，这种表示法让人理解就很困难。因此计算机上都配有输入和输出设备，这些设备的主要目的就是，以一种人类可阅读的形式将信息在这些设备上显示出来供人阅读理解。为保证人类和设备，设备和计算机之间能进行正确的信息交换，人们编制的统一的信息交换代码，这就是ASCII码表，它的全称是“美国信息交换标准代码”。

第一部分是：ASCII非打印控制字符

ASCII表上的数字0–31分配给了控制字符，用于控制像打印机等一些外围设备。例如，12代表换页/新页功能。此命令指示打印机跳到下一页的开头。

第二部分是：ASCII打印字符

数字 32–126 分配给了能在键盘上找到的字符，当您查看或打印文档时就会出现，数字127代表 DELETE 命令，共128个。

C语言字符数组的定义

字符数组是指元素为**字符**的数组，字符数组中的每个元素存放一个字符。字符数组具有一般数组的共同属性，其定义类似，所不同的是数据类型是字符型（char）。

char 数组名[个数];

例如：**char str[10];**

字符数组的初始化

字符数组可以存放一个或多个字符，也可以存放字符串，两者的区别是字符串有一个结束符（**'\0'**）。

初始化字符数组有两种形式：

(1) 用字符初始化 (2) 用字符串初始化

用字符初始化

```
char chr[10]={'H','E','L','L','O'}; //从数组首元素开始赋值，剩余的元素默认为空字符
```

```
char chr[10]; //各个元素逐一初始化  
chr[0]='H', chr[1]='E', chr[2]='L', chr[3]='L', chr[4]='O';
```

用字符串初始化

用字符串初始化数组有两种形式：

一种是用字符初始化的方式并加一个结束符（**'\0'**），另外一种形式是**直接用双引号**赋值。

```
char chr[10]={'H','E','L','L','O', '\0'};
```

```
char chr[10]=" HELLO" ;
```

```
char chr[10]=" HELLO" ;
```

直接用双引号初始化一个一维数组，这种方式在字符数组中自动添加一个结束符（**'\0'**），所以**数组的容量**至少要比实际存储字符串中的字符数**多1**。

数组中字符串末尾的**'\0'**是字符串结束符，它会告知程序当处理字符串时，如果**遇到字符串结束符就表示处理结束**，不需要再输出了。

罗列数组初始化的方式

```
char a[5] = { 'a' , 'b' , 'c' , 'd' , 'e' };
```

//各元素分别赋初始值

```
char b1[5] = { 'a' };
```

//仅第一个元素设为a, 其余为空

```
char b2[5] = { 'a' , 'b' , 'c' };
```

//b2[3]、b2[4]自动赋值为空字符

```
char c[] = { "abcdefg" };
```

//数组长度自动设为8

```
char d[5] = { 'a' , 'b' , 'c' , 'd' , 'e' , 'f' };
```

//错误, 初始化数据过多

字符数组的输入和输出

字符数组的输入和输出可以按照整型数组元素来处理，但字符数组有自己的特性，每个字符数组都有一个特殊的结束符（**'\0'**），所以也可以无须像整型数组那样通过循环输入和输出字符数组。

1、cin和cout

cin>>字符数组名称;

cout<<字符数组名称;

例如：

```
char a[10];  
cin>>a;  
cout<<a;
```

2、scanf()和printf()

scanf("%s" ,字符数组名称);

printf("%s" ,字符数组名称);

- (1) 输入时字符数组名称前不用加取地址符 (&) ,
- (2) 输入字符常量后, 系统后自动在后面加'\0'标志,
- (3) 当输入多个字符数组时, 以空格隔开。

例如:

```
char a[10],b[10];  
scanf("%s %s",a,b);  
printf("%s %s",a,b);
```

3、gets()和puts()

gets(字符数组名称);

puts(字符数组名称);

(1) gets()只能输入一个字符数组，可以读入空格

(2) puts()会输出一个字符数组和一个换行符

例如：

```
char a[10];
```

```
gets(a); // 替换为cin.getline(a,10);
```

```
puts(a);
```


cin、scanf()、gets()的区别

(1) cin和scanf()以**空格**或者**换行**区分输入的字符数组，当输入的字符数组带有**空格**时，则无法使用cin、scanf()成功输入。

(2) gets()以**换行**区分输入的字符数组，使用**gets()**可以输入带有空格的字符数组。

cout、printf()、puts()的区别

puts()在输出字符数组时，会自动**添加换行**。

C语言字符数组常用函数（需加入头文件：`#include <cstring>`）

函数语法

**strcmp(字符数组a,
字符数组b);**

举例说明

字符数组比较函数，比较规则：

(1) 从第一个字符开始比较每个位置上字符的ASCII码值，直到遇到字符不相等或者其中一个字符串的结束符'\0'，比较结束，返回值。

(2) 返回值具体情况如下：

若**a > b**(不相等的字符的ASCII码值)，**返回1**。

若**a < b**(不相等的字符的ASCII码值)，**返回-1**。

若**a == b**(不相等的字符的ASCII码值)，**返回0**。

`char a[10] = "abc" , b[10] = "cde" ;`

`cout << strcmp(a,b);`结果为-1

字符数组常用函数（需加入头文件：#include<cstring>）

函数语法	举例说明
strncmp(字符数组a, 字符数组b, 长度n);	比较前n个字符，strncmp()函数比较规则如同strcmp()函数。 char a[10]= "abc" ,b[10]= "cde" ; cout<<strncmp(a,b,2);结果为-1
strlen(字符数组a);	获取字符数组a的长度（不包括'\0'）。 char a[10]= "abc" ; cout<<strlen(a); 结果为3

字符数组常用函数（需加入头文件：#include<cstring>）

函数语法

举例说明

**strcpy(字符数组1,
字符数组2);**

将字符数组2复制到字符数组1，返回字符数组1的值。

```
char a[10]= "abc" ,b[10]= "cde" ;  
strcpy(a,b);  
cout<<a;结果为cde
```

**strncpy(字符数组1,
字符数组2,长度n);**

将字符串2的前n个字符复制到字符数组1，返回字符数组1的值。

```
char a[10]= "abc" ,b[10]= "cde" ;  
strcpy(a,b,2);  
cout<<a;结果为cd
```

字符数组常用函数（需加入头文件：`#include <cstring>`）

函数语法

举例说明

strcat(字符数组1,字符数组2);

将字符数组2**连接到字符数组1的结尾**，返回字符数组1的值。（确保字符数组1指向的**存储空间足以存储连接后的字符串**）

```
char a[10]= "abc" ,b[10]= "cde" ;  
strcat(a,b);  
cout<<a;结果为abccde
```

strncat(字符数组1,字符数组2,长度n);

将字符数组2的**前n个字符连接到字符数组1的结尾**，返回字符数组1的值。（确保字符数组1指向的存储空间足以存储连接后的字符串）

C++ string类

在C++中，std::string是一个类，它表示可变长度的字符序列，支持许多便捷的操作如连接、子串等。

string类型变量的赋值

函数语法	举例说明
<code>string s(字符串);</code>	定义并初始化，例如： <code>string s("abc 123");</code>
<code>string s(个数, 字符);</code>	定义并初始化为若干相同字母，例如： <code>string s(10, '*');</code>
<code>string s=字符串;</code>	定义并初始化赋值，例如： <code>string s= "abc 123" ;</code>

string类型变量的连接操作列表

函数语法	举例说明
s=字符或字符串;	<code>string s1, s2, s3;</code> <code>s1= 'x' ;</code> <code>s2= "abc" ;</code>
字符串变量或常量+ 字符串变量或常量	运算符 “+” 连接两个字符串, <code>s3=s1+s2;</code>
s+=字符/字符串	运算符 “+=” 是自身加赋值运算符, 例如: <code>string s= "Test:" ;</code> <code>char i= 'a' ;</code> <code>s+=i;</code>

string类型的常用函数与运算

函数语法	举例说明
size()	求字符串长度 ，等同于length()函数，例如： s="12 34"; cout << s.size (); 结果是:5
s[下标i]	取字符串的某个字符 ，等同于at(下标i)函数，例如： s="abcd"; cout<<s[0]<<s.at (2); 结果是:ac

string类型的常用函数与运算

函数语法	举例说明
<code>getline (cin ,s);</code>	读入一整行(直到换行)，包括读入空格
<code>substr(开 始位置i,子 串长度len);</code>	取字符串的子串。当i+len超过原字符串长度时，只取字符串范围内的。提醒:i要在字符串长度内。例如: <code>s="abcdef";</code> <code>cout << s.substr (3,2)<<s.substr (3,20);</code> 结果是:dedef

string类型的常用函数与运算

函数语法	举例说明
insert(插入位置i, 插入字符串s);	在字符串的第i个位置插入s，例如： <code>s="abcdef";</code> <code>s.insert(2,"+");</code> <code>cout <<s;</code> 结果是:ab+cdef
erase(开始位置i, 删除长度len);	删除字符串的第i个位置后的len个字符，例如： <code>s="abcdef";</code> <code>s.erase (2,3);</code> <code>cout <<s;</code> 结果是:abf

string类型的常用函数与运算

函数语法	举例说明
replace(开始位置i,长度len,要换上的字符串ss);	用字符串ss替换字符串中i开始长度是len的一段, 例如: <pre>s="abcdef"; s.replace (2,1,"123") ; cout <<s;结果是:ab123def</pre>
find(子串subs)	查找子串subs第1次出现的位置, 没有找到返回string::npos, 例如: <pre>s="abcdef"; int i=s.find("cd"); cout << i;结果是:2</pre>

string类型变量的转换总结

知识点	举例说明
char型可以直接当整数使用	值是其ASCII码，例如： 'A'-'0' -5 结果是:65-48-5=12 '7'-'0' 结果是:字符'7'转为数字7
整数型转 char型要强制类型	例如: char(5+'0') 结果是:字符'5'
任意类型间转换可使用字符串流	例如: stringstream tempIO << "123456"; int x; tempIO >> x;

string类型变量的转换总结

知识点	举例说明
string类可直接按字典序比较大小	可直接使用关系运算符: >, >=, <, <=, ==, != 例如:"ABC" > "BCD"结果为假
string类型可转换到C风格的字符数组	使用函数c_str(), 例如: string s="filename"; printf("%s",s.c_str());
C风格的字符数组可直接赋给string类型	例如: char cs[]="filename"; string s=cs;