C基础

一、语法组成

1、简单程序

```c
#include<stdio.h>
int main(){
  printf("hello C\n");
  return 0;
}
```

2、编译和链接

C 语言是编译型语言，源代码需要编译成为机器码才能被计算机执行。

C 语言编译的步骤：

- **预处理**：首先程序会被交给预处理器。预处理器执行以#开头的指令。预处理器可以给程序添加内容，也可以修改程序。
- **编译**：经过预处理修改过的程序可以直接进入编译器。编译器可以把程序翻译成为机器指令（目标代码）。
- **链接**：在最后一个步骤中，链接器把由编译器产生的目标代码和所需要的其他附加代码整合在一起，这样就最终产生了完全可以执行的程序。附加代码是程序中需要用到的库函数。

3、程序基本语句和结构

C 语言程序的基本形式：

```
指令
主函数{
  语句
}
```

指令：在编译 C 语言之前，通常是由预处理器对程序中的指令进行预处理，C 语言通过指令来添加修改程序。如：#include 指令就是引入程序需要的功能库。所有的指令都是以#开头的。这个字符是将指令与其他代码区别。指令默认独占一行，末尾没有结束符号。

主函数：函数是特定的功能模块。是用来构建程序的构建块。事实上，C 语言就是函数的集合。函数可以由程序员自己编写，C 语言也会提供一部分已经实现的**库函数**可以供调用。一个程序中可以有多个函数，**但是只能有一个主函数--main()**，而且 main 函数是必须有的，函数的调用，语句的执行主要是 main 函数来操作的。main 函数非常特殊，**在执行程序时系统会自动调用 main 函数**。函数前面的数据类型标识符是标识函数的返回值类型 int main()就表示 main 函数的返回值类型是整数，main 函数的返回值类型也只能是整数。

语句：语句是程序运行时执行命令。

4、注释

单行注释// 多行注释/* */

```
//C 语言注释
/*
C 语言
注释
*/
```

5、变量和赋值

变量是数据的存储单元，声明变量后，程序会在内存中开辟一段空间用来存储数据。C 语言是强类型，也就是 C 语言中的变量是需要声明数据类型的，所以声明变量的语句通常是：数据类型 变量名。

```
int a;
float b;
```

定义变量的行为被称为声明。变量可以单独声明，相同类型的变量也可以共同声明

```
int a,b;
```

声明变量的目的还是存储值。变量通过赋值获得值。

```
int a;
float b;
a=1;
b=0.1;
```

也可以将变量的声明和赋值放到一起就是变量的初始化。

```
int a=1;
float b=0.1;
```

6、常量和标识符

程序中还包含常量，常量是通过预处理指令被定义的。被称为宏定义

```
#define INT 100
```

在编写程序时，需要对变量、函数、宏或其他实体命名时，这些命名被称为标识符，需要遵循一定的命名规范。C 语言中标识符由字母、数字、下划线组成。但必须由字母或下划线开头，C 语言标识符区分大小写。C 语言还包含有关键字不能被标识符使用。

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

1999年12月16日，ISO推出了C99标准，该标准新增了5个C语言关键字：

inline	restrict	_Bool	_Complex	_Imaginary

2011年12月8日，ISO发布C语言的新标准C11，该标准新增了7个C语言关键字：

_Alignas	_Alignof	_Atomic	_Static_assert	_Noreturn	_Thread_local	_Generic

---

二、输入输出

1、printf 函数

printf()是格式化输出函数，被设计用来显示格式化字符串的内容。调用 printf 函数必须提供格式化字符串。

```
printf(格式串,表达式 1,表达式 2);
```

printf 显示的值可以是常量、变量、或者更加复杂的表达式

格式串中要包含普通字符和转换说明（占位符）。

占位符：

- `%a`：十六进制浮点数，字母输出为小写。
- `%A`：十六进制浮点数，字母输出为大写。
- `%c`：**字符。**
- `%d`：**十进制整数。**
- `%e`：使用科学计数法的浮点数，指数部分的 `e` 为小写。
- `%E`：使用科学计数法的浮点数，指数部分的 `E` 为大写。
- `%i`：整数，基本等同于 `%d`。
- `%f`：**小数（包含 `float` 类型和 `double` 类型）。**
- `%g`：6 个有效数字的浮点数。整数部分一旦超过 6 位，就会自动转为科学计数法，指数部分的 `e` 为小写。
- `%G`：等同于 `%g`，唯一的区别是指数部分的 `E` 为大写。
- `%hd`：十进制 short int 类型。
- `%ho`：八进制 short int 类型。
- `%hx`：十六进制 short int 类型。
- `%hu`：unsigned short int 类型。
- `%ld`：十进制 long int 类型。
- `%lo`：八进制 long int 类型。
- `%lx`：十六进制 long int 类型。

- `%lu`：unsigned long int 类型。
- `%lld`：十进制 long long int 类型。
- `%llo`：八进制 long long int 类型。
- `%llx`：十六进制 long long int 类型。
- `%llu`：unsigned long long int 类型。
- `%Le`：科学计数法表示的 long double 类型浮点数。
- `%Lf`：long double 类型浮点数。
- `%n`：已输出的字符串数量。该占位符本身不输出，只将值存储在指定变量之中。
- `%o`：八进制整数。
- `%p`：**指针。**
- `%s`：**字符串。**
- `%u`：无符号整数（unsigned int）。
- `%x`：十六进制整数。
- `%zd`：`size_t` 类型。
- `%%`：输出一个百分号。

转义序列：

转义字符	代表含义	输出字符的结果
\’	一个单引号	输出：’
\”	一个双引号	输出：”
\\	一条反斜杠	输出：\
\b	退格符（backspace）	退回一格
\n	换行符	换行
\r	回车键	
\t	制表符（tab）	
\000		
\xhh		

输出格式：

`printf()` 可以定制占位符的输出格式。

（1）限定宽度

`printf()` 允许限定占位符的最小宽度。

```
printf("%5d\n", 123); // 输出为 "  123"
```

上面示例中，`%5d` 表示这个占位符的宽度至少为 5 位。如果不满 5 位，对应的值的前面会添加空格。

输出的值默认是右对齐，即输出内容前面会有空格；如果希望改成左对齐，在输出内容后面添加空格，可以在占位符的 `%` 的后面插入一个 `-` 号。

```
printf("%-5d\n", 123); // 输出为 "123  "
```

上面示例中，输出内容 `123` 的后面添加了空格。

对于小数，这个限定符会限制所有数字的最小显示宽度。

```
// 输出 "  123.450000"
printf("%12f\n", 123.45);
```

上面示例中，`%12f` 表示输出的浮点数最少要占据 12 位。由于小数的默认显示精度是小数点后 6 位，所以 `123.45` 输出结果的头部会添加 2 个空格。

（2）总是显示正负号

默认情况下，`printf()` 不对正数显示 `+` 号，只对负数显示 `-` 号。如果想让正数也输出 `+` 号，可以在占位符的 `%` 后面加一个 `+` 。

```
printf("%+d\n", 12); // 输出 +12
printf("%+d\n", -12); // 输出 -12
```

上面示例中，`%+d` 可以确保输出的数值，总是带有正负号。

（3）限定小数位数

输出小数时，有时希望限定小数的位数。举例来说，希望小数点后面只保留两位，占位符可以写成 `%.2f` 。

```
// 输出 Number is 0.50
printf("Number is %.2f\n", 0.5);
```

上面示例中，如果希望小数点后面输出 3 位（`0.500`），占位符就要写成 `%.3f` 。

这种写法可以与限定宽度占位符，结合使用。

```
// 输出为 "  0.50"
printf("%6.2f\n", 0.5);
```

上面示例中，`%6.2f` 表示输出字符串最小宽度为 6，小数位数为 2。所以，输出字符串的头部有两个空格。

最小宽度和小数位数这两个限定值，都可以用 `*` 代替，通过 `printf()` 的参数传入。

```
printf("%*.*f\n", 6, 2, 0.5);

// 等同于
printf("%6.2f\n", 0.5);
```

上面示例中，`%*.*f` 的两个星号通过 `printf()` 的两个参数 `6` 和 `2` 传入。

（4）输出部分字符串

`%s` 占位符用来输出字符串，默认是全部输出。如果只想输出开头的部分，可以用 `%.[m]s` 指定输出的长度，其中 `[m]` 代表一个数字，表示所要输出的长度。

```
// 输出 hello
printf("%.5s\n", "hello world");
```

上面示例中，占位符 `%.5s` 表示只输出字符串“hello world”的前 5 个字符，即“hello”。

2、scanf 函数

scanf 函数用于读取用户的输入。

`scanf()` 的语法跟 `printf()` 类似。

```
scanf("%d", &i);
```

它的第一个参数是一个格式字符串，里面会放置占位符（与 `printf()` 的占位符基本一致），告诉编译器如何解读用户的输入，需要提取的数据是什么类型。这是因为 C 语言的数据都是有类型的， `scanf()` 必须提前知道用户输入的数据类型，才能处理数据。它的其余参数就是存放用户输入的变量，格式字符串里面有多少个占位符，就有多少个变量。 注意，变量前面必须加上 `&` 运算符（指针变量除外），因为 `scanf()` 传递的不是值，而是地址，即将变量 `i` 的地址指向用户输入的值。如果这里的变量是指针变量（比如字符串变量），那就不用加 `&` 运算符。

占位符：

`scanf()` 常用的占位符如下，与 `printf()` 的占位符基本一致。

- `%c` ： 字符。
- `%d` ： 整数。
- `%f` ： `float` 类型浮点数。
- `%lf` ： `double` 类型浮点数。
- `%Lf` ： `long double` 类型浮点数。
- `%s` ： 字符串。
- `%[]` ： 在方括号中指定一组匹配的字符（比如 `%[0-9]` ），遇到不在集合之中的字符，匹配将会停止。

三、数据类型

1、字符类型

字符类型指的是单个字符，类型声明使用 `char` 关键字。

```
char c = 'B';
```

C 语言规定，字符常量必须放在单引号里面。在计算机内部，字符类型使用一个字节（8 位）存储。C 语言将其当作整数处理，所以字符类型就是宽度为一个字节的整数。每个字符对应一个整数（由 ASCII 码确定），比如 `B` 对应整数 `66` 。

2、整数类型

整数类型用来表示较大的整数，类型声明使用 `int` 关键字。

```
int a;
```

符号类型：

C 语言使用 `signed` 关键字，表示一个类型带有正负号，包含负值；使用 `unsigned` 关键字，表示该类型不带有正负号，只能表示零和正整数。

对于 `int` 类型，默认是带有正负号的，也就是说 `int` 等同于 `signed int` 。由于这是默认情况，关键字 `signed` 一般都省略不写，但是写了也不算错。

```
signed int a;
// 等同于
```

```
int a;
```

`int` 类型也可以不带正负号，只表示非负整数。这时就必须使用关键字 `unsigned` 声明变量。

```
unsigned int a;
```

整数变量声明为 `unsigned` 的好处是，同样长度的内存能够表示的最大整数值，增大了一倍。比如，16 位的 `signed int` 最大值为 32,767，而 `unsigned int` 的最大值增大到了 65,535。

`unsigned int` 里面的 `int` 可以省略，所以上面的变量声明也可以写成下面这样。

```
unsigned a;
```

整数子类型：

如果 `int` 类型使用 4 个或 8 个字节表示一个整数，对于小整数，这样做很浪费空间。另一方面，某些场合需要更大的整数，8 个字节还不够。为了解决这些问题，C 语言在 `int` 类型之外，又提供了三个整数的子类型。

- `short int`（简写为 `short`）：占用空间不多于 `int`，一般占用 2 个字节（整数范围为-32768～32767）。
- `long int`（简写为 `long`）：占用空间不少于 `int`，至少为 4 个字节。
- `long long int`（简写为 `long long`）：占用空间多于 `long`，至少为 8 个字节。

整数极限值：

有时候需要查看，当前系统不同整数类型的最大值和最小值，C 语言的头文件 `limits.h` 提供了相应的常量，比如 `SCHAR_MIN` 代表 signed char 类型的最小值 `-128`，`SCHAR_MAX` 代表 signed char 类型的最大值 `127`。

为了代码的可移植性，需要知道某种整数类型的极限值时，应该尽量使用这些常量。

- `SCHAR_MIN`，`SCHAR_MAX`：signed char 的最小值和最大值。
- `SHRT_MIN`，`SHRT_MAX`：short 的最小值和最大值。
- `INT_MIN`，`INT_MAX`：int 的最小值和最大值。
- `LONG_MIN`，`LONG_MAX`：long 的最小值和最大值。
- `LLONG_MIN`，`LLONG_MAX`：long long 的最小值和最大值。
- `UCHAR_MAX`：unsigned char 的最大值。
- `USHRT_MAX`：unsigned short 的最大值。
- `UINT_MAX`：unsigned int 的最大值。
- `ULONG_MAX`：unsigned long 的最大值。
- `ULLONG_MAX`：unsigned long long 的最大值。

进制：

C 语言的整数默认都是十进制数，如果要表示八进制数和十六进制数，必须使用专门的表示法。

八进制使用 `0` 作为前缀，比如 `017`、`0377`。

```
int a = 012; // 八进制，相当于十进制的 10
```

十六进制使用 `0x` 或 `0X` 作为前缀，比如 `0xf`、`0X10`。

```
int a = 0x1A2B; // 十六进制，相当于十进制的 6699
```

有些编译器使用 `0b` 前缀，表示二进制数，但不是标准。

```
int x = 0b101010;
```

`printf()` 的进制相关占位符如下。

- `%d`：十进制整数。
- `%o`：八进制整数。
- `%x`：十六进制整数。
- `%#o`：显示前缀 `0` 的八进制整数。
- `%#x`：显示前缀 `0x` 的十六进制整数。
- `%#X`：显示前缀 `0X` 的十六进制整数。

3、浮点数类型

float：单精度

浮点数的类型声明使用 `float` 关键字，可以用来声明浮点数变量。

```
float c = 10.5;
```

`float` 类型占用 4 个字节（32 位），其中 8 位存放指数的值和符号，剩下 24 位存放小数的值和符号。`float` 类型至少能够提供（十进制的）6 位有效数字，指数部分的范围为（十进制的）`-37` 到 `37`，即数值范围为 10−37 到 1037。

double：双精度

- `double`：占用 8 个字节（64 位），至少提供 13 位有效数字。
- `long double`：通常占用 16 个字节。

注意，由于存在精度限制，浮点数只是一个近似值，它的计算是不精确的

C 语言允许使用科学计数法表示浮点数，使用字母 `e` 来分隔小数部分和指数部分。

```
double x = 123.456e+3; // 123.456 x 10^3
// 等同于
double x = 123.456e3;
```

4、布尔类型

C 语言原来并没有为布尔值单独设置一个类型，而是使用整数 `0` 表示伪，所有非零值表示真。

```
int x = 1;
if (x) {
  printf("x is true!\n");
}
```

上面示例中，变量 `x` 等于 `1`，C 语言就认为这个值代表真，从而会执行判断体内部的代码。

C99 标准添加了类型 `_Bool`，表示布尔值。但是，这个类型其实只是整数类型的别名，还是使用 `0` 表示伪，`1` 表示真，下面是一个示例。

```
_Bool isNormal;

isNormal = 1;
if (isNormal)
  printf("Everything is OK.\n");
```

头文件 `stdbool.h` 定义了另一个类型别名 `bool`，并且定义了 `true` 代表 `1`、`false` 代表 `0`。只要加载这个头文件，就可以使用这几个关键字。

```
#include <stdbool.h>

bool flag = false;
```

上面示例中，加载头文件 `stdbool.h` 以后，就可以使用 `bool` 定义布尔值类型，以及 `false` 和 `true` 表示真伪。

5、sizeof 运算符

`sizeof` 是 C 语言提供的一个运算符，返回某种数据类型或某个值占用的字节数量。它的参数可以是数据类型的关键字，也可以是变量名或某个具体的值。

```
// 参数为数据类型
int x = sizeof(int);

// 参数为变量
int i;
sizeof(i);

// 参数为数值
sizeof(3.14);
```

6、类型隐式转换

赋值运算

赋值运算符会自动将右边的值，转成左边变量的类型。

（1）浮点数赋值给整数变量

浮点数赋予整数变量时，C 语言直接丢弃小数部分，而不是四舍五入。

```
int x = 3.14;
```

上面示例中，变量 `x` 是整数类型，赋给它的值是一个浮点数。编译器会自动把 `3.14` 先转为 `int` 类型，丢弃小数部分，再赋值给 `x`，因此 `x` 的值是 `3`。

注意，舍弃小数部分时，不是四舍五入，而是整个舍弃。

```
int x = 12.99;
```

上面示例中，`x` 等于 `12`，而不是四舍五入的 `13`。

（2）整数赋值给浮点数变量

整数赋值给浮点数变量时，会自动转为浮点数。

```
float y = 12 * 2;
```

上面示例中，变量 `y` 的值不是 `24`，而是 `24.0`，因为等号右边的整数自动转为了浮点数。

（3）窄类型赋值给宽类型

字节宽度较小的整数类型，赋值给字节宽度较大的整数变量时，会发生类型提升，即窄类型自动转为宽类型。

比如，`char` 或 `short` 类型赋值给 `int` 类型，会自动提升为 `int`。

```
char x = 10;
int i = x + y;
```

上面示例中，变量 `x` 的类型是 `char`，由于赋值给 `int` 类型，所以会自动提升为 `int`。

（4）宽类型赋值给窄类型

字节宽度较大的类型，赋值给字节宽度较小的变量时，会发生类型降级，自动转为后者的类型。这时可能会发生截值（truncation），系统会自动截去多余的二进制位，导致难以预料的结果。

```
int i = 321;
char ch = i; // ch 的值是 65（321 - 256）
```

上面例子中，变量 `ch` 是 `char` 类型，宽度是 8 个二进制位。变量 `i` 是 `int` 类型，将 `i` 赋值给 `ch`，后者只能容纳 `i`（二进制形式为 `101000001`，共 9 位）的后八位，前面多出来的二进制位被丢弃，保留后八位就变成了 `01000001`（十进制的 65，相当于字符 `A`）。

浮点数赋值给整数类型的值，也会发生截值，浮点数的小数部分会被截去。

```
double pi = 3.14159;
int i = pi; // i 的值为 3
```

混合运算

不同类型的值进行混合计算时，必须先转成同一个类型，才能进行计算。转换规则如下：

（1）整数与浮点数混合运算时，整数转为浮点数类型，与另一个运算数类型相同。

```
3 + 1.2 // 4.2
```

上面示例是 `int` 类型与 `float` 类型的混合计算，`int` 类型的 `3` 会先转成 `float` 的 `3.0`，再进行计算，得到 `4.2`。

（2）不同的浮点数类型混合运算时，宽度较小的类型转为宽度较大的类型，比如 `float` 转为 `double`，`double` 转为 `long double`。

（3）不同的整数类型混合运算时，宽度较小的类型会提升为宽度较大的类型。比如 `short` 转为 `int`，`int` 转为 `long` 等，有时还会将带符号的类型 `signed` 转为无符号 `unsigned`。

函数参数返回值类型

函数的参数和返回值，会自动转成函数定义里指定的类型。

```
int dostuff(int, unsigned char);

char m = 42;
unsigned short n = 43;
long long int c = dostuff(m, n);
```

上面示例中，参数变量 `m` 和 `n` 不管原来的类型是什么，都会转成函数 `dostuff()` 定义的参数类型。

下面是返回值自动转换类型的例子。

```
char func(void) {
  int a = 42;
  return a;
}
```

上面示例中，函数内部的变量 `a` 是 `int` 类型，但是返回的值是 `char` 类型，因为函数定义中返回的是这个类型。

7、类型显示转换

C 语言提供了类型的显式转换，允许手动转换类型。

只要在一个值或变量的前面，使用圆括号指定类型 `(type)`，就可以将这个值或变量转为指定的类型，这叫做“类型指定”（casting）。

```
(unsigned char) ch
```

上面示例将变量 `ch` 转成无符号的字符类型。

```
long int y = (long int) 10 + 12;
```

上面示例中，`(long int)` 将 `10` 显式转为 `long int` 类型。这里的显示转换其实是不必要的，因为赋值运算符会自动将右边的值，转为左边变量的类型。

四、运算符

1、算术运算符

- `+`：正值运算符（一元运算符）
- `-`：负值运算符（一元运算符）
- `+`：加法运算符（二元运算符）
- `-`：减法运算符（二元运算符）
- `*`：乘法运算符
- `/`：除法运算符
- `%`：余值运算符
- `=`：赋值运算符

  复合赋值：

  - `+=`
  - `-=`
  - `*=`
  - `/=`
  - `%=`

2、自增、自减运算符

C 语言提供两个运算符，对变量自身进行 `+ 1` 和 `- 1` 的操作。

- `++`：自增运算符
- `--`：自减运算符

```
 i++; // 等同于 i = i + 1
i--; // 等同于 i = i - 1
```

这两个运算符放在变量的前面或后面，结果是不一样的。`++var` 和 `--var` 是先执行自增或自减操作，再返回操作后 `var` 的值；`var++` 和 `var--` 则是先返回操作前 `var` 的值，再执行自增或自减操作。

```
int i = 42;
int j;

j = (i++ + 10);
// i: 43
// j: 52

j = (++i + 10)
// i: 44
// j: 54
```

3、关系运算符

C 语言用于比较的表达式，称为“关系表达式”（relational expression），里面使用的运算符就称为“关系运算符”（relational operator），主要有下面 6 个。

- `>` 大于运算符
- `<` 小于运算符
- `>=` 大于等于运算符
- `<=` 小于等于运算符
- `==` 相等运算符
- `!=` 不相等运算符

4、逻辑运算符

逻辑运算符提供逻辑判断功能，用于构建更复杂的表达式，主要有下面三个运算符。

- `!`：否运算符（改变单个表达式的真伪）。
- `&&`：与运算符（两侧的表达式都为真，则为真，否则为伪）。
- `||`：或运算符（两侧至少有一个表达式为真，则为真，否则为伪）。

5、位运算符

C 语言提供一些位运算符，用来操作二进制位（bit）。

（1）取反运算符 `~`

取反运算符 `~` 是一个一元运算符，用来将每一个二进制位变成相反值，即 `0` 变成 `1`，`1` 变成 `0`。

```
// 返回 01101100
```

```
~ 10010011
```

上面示例中，`~` 对每个二进制位取反，就得到了一个新的值。

注意，`~` 运算符不会改变变量的值，只是返回一个新的值。

（2）与运算符 `&`

与运算符 `&` 将两个值的每一个二进制位进行比较，返回一个新的值。当两个二进制位都为 `1`，就返回 `1`，否则返回 `0`。

```
// 返回 00010001
10010011 & 00111101
```

上面示例中，两个八位二进制数进行逐位比较，返回一个新的值。

与运算符 `&` 可以与赋值运算符 `=` 结合，简写成 `&=`。

```
int val = 3;
val = val & 0377;

// 简写成
val &= 0377;
```

（3）或运算符 `|`

或运算符 `|` 将两个值的每一个二进制位进行比较，返回一个新的值。两个二进制位只要有一个为 `1`（包含两个都为 `1` 的情况），就返回 `1`，否则返回 `0`。

```
// 返回 10111111
10010011 | 00111101
```

或运算符 `|` 可以与赋值运算符 `=` 结合，简写成 `|=`。

```
int val = 3;
val = val | 0377;

// 简写为
val |= 0377;
```

（4）异或运算符 `^`

异或运算符 `^` 将两个值的每一个二进制位进行比较，返回一个新的值。两个二进制位有且仅有一个为 `1`，就返回 `1`，否则返回 `0`。

```
// 返回 10101110
10010011 ^ 00111101
```

异或运算符 `^` 可以与赋值运算符 `=` 结合，简写成 `^=`。

```
int val = 3;
val = val ^ 0377;

// 简写为
val ^= 0377;
```

（5）左移运算符 `<<`

左移运算符 `<<` 将左侧运算数的每一位，向左移动指定的位数，尾部空出来的位置使用 `0` 填充。

```
// 1000101000
10001010 << 2
```

上面示例中， `10001010` 的每一个二进制位，都向左侧移动了两位。

左移运算符相当于将运算数乘以 2 的指定次方，比如左移 2 位相当于乘以 4（2 的 2 次方）。

左移运算符 `<<` 可以与赋值运算符 `=` 结合，简写成 `<<=` 。

```
int val = 1;
val = val << 2;

// 简写为
val <<= 2;
```

（6）右移运算符 `>>`

右移运算符 `>>` 将左侧运算数的每一位，向右移动指定的位数，尾部无法容纳的值将丢弃，头部空出来的位置使用 `0` 填充。

```
// 返回 00100010
10001010 >> 2
```

上面示例中， `10001010` 的每一个二进制位，都向右移动两位。最低的两位 `10` 被丢弃，头部多出来的两位补 `0` ，所以最后得到 `00100010` 。

注意，右移运算符最好只用于无符号整数，不要用于负数。因为不同系统对于右移后如何处理负数的符号位，有不同的做法，可能会得到不一样的结果。

右移运算符相当于将运算数除以 2 的指定次方，比如右移 2 位就相当于除以 4（2 的 2 次方）。

右移运算符 `>>` 可以与赋值运算符 `=` 结合，简写成 `>>=` 。

```
int val = 1;
val = val >> 2;

// 简写为
val >>= 2;
```

6、逗号运算符

- 逗号运算符用于将多个表达式写在一起，从左到右依次运行每个表达式。
- 逗号运算符返回最后一个表达式的值，作为整个语句的值。

```
int x;
x = 1, 2, 3;
```

上面示例中，逗号的优先级低于赋值运算符，所以先执行赋值运算，再执行逗号运算，变量 `x` 等于 `1` 。

7、运算优先级

运算符的优先级顺序很复杂。下面是部分运算符的优先级顺序（按照优先级从高到低排列）。

- 圆括号（`()`）
- 自增运算符（`++`），自减运算符（`--`）
- 一元运算符（`+` 和 `-`）
- 乘法（`*`），除法（`/`）
- 加法（`+`），减法（`-`）
- 关系运算符（`<`、`>` 等）
- 赋值运算符（`=`）

由于圆括号的优先级最高，可以使用它改变其他运算符的优先级。

五、程序结构

C 语言的程序是顺序执行，即先执行前面的语句，再执行后面的语句。开发者如果想要控制程序执行的流程，就必须使用流程控制的语法结构，主要是条件执行和循环执行。

1、条件语句

if 语句:

单分支

```
if(条件表达式){
  语句;
}
```

双分支

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

多分支

```
if(条件表达式 1){
  语句 1;
}else if(条件表达式 2){
  语句 2;
}
...
else{
  语句 n;
}
```

嵌套

```
if(条件表达式 1){
  if(条件表达式 2){
    语句;
```

```
  }
}else{
  if(条件表达式 3){
    语句;
  }
}
```

三元运算符? :

条件运算符是由? 和: 组成的，是三元运算符。

```
表达式 1?表达式 2:表达式 3
```

上述表达式是一个 if...else 语句的简写形式

表示是如果表达式 1 成立，就执行表达式 2，否则执行表达式 3

switch 语句:

switch 语句是一种特殊形式的 if...else 结构，用于判断条件有多个结果的情况。它把多重的 `else if` 改成更易用、可读性更好的形式。

```
switch(表达式){
  case 常量表达式 : 语句;
  case 常量表达式 : 语句;
  ...
  default: 语句;
}
```

2、循环语句

while 循环

```
while(表达式){
  语句;
}
```

只要条件为真， `while` 会产生无限循环。下面是一种常见的无限循环的写法。

```
while (1) {
  // ...
}
```

do...while 循环

`do...while` 结构是 `while` 的变体，它会先执行一次循环体，然后再判断是否满足条件。如果满足的话，就继续执行循环体，否则跳出循环。

```
do{
  语句;
}
while (表达式){
  语句;
```

```
}
```

上面代码中，不管条件表达式是否成立，do 循环体至少会执行一次。

for 循环

```
for(表达式 1;表达式 2;表达式 3){
  语句;
}
-----------------------------------------
for (initialization; continuation; action)
  statement;
```

上面代码中，`for` 语句的条件部分（即圆括号里面的部分）有三个表达式。

- `initialization`：初始化表达式，用于初始化循环变量，只执行一次。
- `continuation`：判断表达式，只要为 `true`，就会不断执行循环体。
- `action`：循环变量处理表达式，每轮循环结束后执行，使得循环变量发生变化。

3、跳转语句

break 语句

break 语句用于跳出整个语句

`break` 语句有两种用法。一种是与 `switch` 语句配套使用，用来中断某个分支的执行。另一种用法是在循环体内部跳出循环，不再进行后面的循环了。

> 注意，`break` 命令只能跳出循环体和 `switch` 结构，不能跳出 `if` 结构。

continue 语句

`continue` 语句用于在循环体内部终止本轮循环，进入下一轮循环。只要遇到 `continue` 语句，循环体内部后面的语句就不执行了，回到循环体的头部，开始执行下一轮循环。

goto 语句

goto 语句用于跳到指定的标签名。这会破坏结构化编程，建议不要轻易使用。