第 1 章 c 数据类型及语句

c 语言特点

我的第一个 c 语言程序

```
#include<stdio.h>
int main()//这个我的第一个 c 语言程序
{
    printf("hello world\n");    //printf 是输出打印的函数
    return 0;
}
```

1.#include<stdio.h> 头文件包含，一定要有
2.每一个 c 语言的程序有且只有一个 main 函数，这是整个程序的开始位置
3.C 语言中()、[]、{}、“”、”、都必须成对出现,必须是英文符号
4.C 语言中语句要以分号结束。
5.//为注释

```
/*
有志者，事竟成，破釜沉舟，百二秦关终属楚；
苦心人，天不负，卧薪尝胆，三千越甲可吞吴
*/
```

1.1 关键字

1.1.1 数据类型相关的关键字

用于定义变量或者类型
类型 变量名;
char 、short、int 、long 、 float、double、
struct、union、enum 、signed、unsigned、void

1、 char 字符型 ，用 char 定义的变量是字符型变量，占 1 个字节
    char ch='a'; =为赋值号

```
char ch1=  '1' ;   正确
char ch2 =  '1234'  ;  错误的
```

2、 short 短整型 ,使用 short 定义的变量是短整型变量，占 2 个字节

```
short int a=11;    -32768 - ---32767
```

3、 int 整型 ，用 int 定义的变量是整型变量，在 32 位系统下占 4 个字节，在 16 平台下占 2 个字节

```
int a=44;       -20 亿---20 亿
```

4、 long 长整型 用 long 定义的变量是长整型的，在 32 位系统下占 4 个字节

```
long int a=66;
```

5、 float 单浮点型 （实数），用 float 定义的变量是单浮点型的实数，占 4 个字节

```
float b=3.8f;
```

6、 double 双浮点型 （实数），用 double 定义的变量是双浮点型的实数，占 8 个字节

```
double b=3.8;
```

7、 struct 这个关键字是与结构体类型相关的关键字，可以用它来定义结构体类型，以后讲结构体的时候再讲
8、 union 这个关键字是与共用体（联合体）相关的关键字，以后再讲
9、 enum 与枚举类型相关的关键字 以后再讲
10、signed 有符号(正负)的意思

在定义 char 、整型（short 、int、long） 数据的时候用 signed 修饰，代表咱们定义的数据是有符号的，可以保存正数，也可以保存负数

例 ：signed int a=10;

```
signed  int  b=-6;
```

注意：默认情况下 signed 可以省略 即 int a=-10;//默认 a 就是有符号类型的数据

11、unsigned 无符号的意思

在定义 char 、整型（short 、int、long） 数据的时候用 unsigned 修饰，代表咱们定义的数据是无符号类型的数据

只能保存正数和 0。

```
unsigned int a=101;
unsigned int a=-101;  //错误
```

12、void 空类型的关键字

char、int 、float 都可以定义变量

void 不能定义变量，没有 void 类型的变量

void 是用来修饰函数的参数或者返回值，代表函数没有参数或没有返回值

例：

```
void fun(void)
{
}
```

代表 fun 函数没有返回值，fun 函数没有参数

例 2 :

```
#include <stdio.h>
int main()
{
    char a = 'a';
    short int b = 10;
    int c;
```

```
    long int d;
    float e;
    double f;
    printf("%d\n",sizeof(a));
    printf("%d\n",sizeof(b));
    printf("%d\n",sizeof(c));
    printf("%d\n",sizeof(d));
    printf("%d\n",sizeof(e));
    printf("%d\n",sizeof(f));
    return 0;
}
```

1.1.2 存储相关关键字

register、static、const、auto、extern

1、register 是 寄存器的意思，用 register 修饰的变量是寄存器变量，

即：在编译的时候告诉编译器这个变量是寄存器变量，尽量将其存储空间分配在寄存器中。

注意：

(1):定义的变量不一定真的存放在寄存器中。

(2)：cpu 取数据的时候去寄存器中拿数据比去内存中拿数据要快

(3)：因为寄存器比较宝贵，所以不能定义寄存器数组

(4)：register 只能修饰 字符型及整型的，不能修饰浮点型

```
register char ch;
register short int b;
register int c;
register float d;//错误的
```

(5)：因为 register 修饰的变量可能存放在寄存器中不存放在内存中，所以不能对寄存器变量取地址。因为只有存放在内存中的数据才有地址

```
register int a;
int *p;
p=&a;//错误的，a 可能没有地址
```

2、static 是静态的意思

static 可以修饰全局变量、局部变量、函数

这个以后的课程中重点讲解

3、const

const 是常量的意思

用 const 修饰的变量是只读的，不能修改它的值

```
const int a=101;//在定义 a 的时候用 const 修饰，并赋初值为 101
```

从此以后，就不能再给 a 赋值了

a=111;//错误的

const 可以修饰指针，这个在以后课程中重点讲解

4、auto int a;和 int a 是等价的，auto 关键字现在基本不用

5、extern 是外部的意思，一般用于函数和全局变量的声明，这个在后面的课程中，会用到

1.1.3 控制语句相关的关键字

if 、else 、break、continue、for 、while、do、switch case
goto、default

1.1.4 其他关键字

sizeof、typedef、volatile

1、sizeof

使用来测变量、数组的占用存储空间的大小（字节数）

```
例 3：
int a=10;
int num;
num=sizeof(a);
```

2、typedef 重命名相关的关键字

unsigned short int a = 10;
U16
关键字 ，作用是给一个已有的类型，重新起个类型名，并没有创造一个新的类型
以前大家看程序的时候见过类似的变量定义方法
INT16 a;
U8 ch;
INT32 b;
大家知道，在 c 语言中没有 INT16 U8 这些关键字

INT16 U8 是用 typedef 定义出来的新的类型名，其实就是 short int 及 unsigned char 的别名

typedef 起别名的方法：

1、用想起名的类型定义一个变量

```
short int a;
```

2、用新的类型名替代变量名

```
short int INT16;
```

3、在最前面加 typedef

```
typedef short int INT16;
```

4：就可以用新的类型名定义变量了

```
INT16 b;和 short int b;//是一个效果
```

```
例 4：
```

```
#include <stdio.h>
//short int b;
//short int INT16;
typedef   short int   INT16;
int main(int argc, char *argv[])
{
      short int a=101;
      INT16 c=111;
      printf("a=%d\n",a);
      printf("c=%d\n",c);
      return 0;
}
```

3、volatile 易改变的意思

用 volatile 定义的变量，是易改变的，即告诉 cpu 每次用 volatile 变量的时候，重新去内存中取保证用的是最新的值,而不是寄存器中的备份。

volatile 关键字现在较少适用

```
volatile int a=10;
```

1.2 数据类型

1.2.1 基本类型

char 、short int 、int、long int、float、double

1.2.2 构造类型

概念：由若干个相同或不同类型数据构成的集合，这种数据类型被称为构造类型

例：int a[10];

数组、结构体、共用体、枚举

扩展：常量和变量

常量：在程序运行过程中，其值不可以改变的量

例：100 ‘a’ “hello”

- 整型 100，125，-100，0
- 实型 3.14 ， 0.125f，-3.789
- 字符型 ‘a’，‘b’，‘2’
- 字符串 “a”,“ab”， “1232”

ASCII 码表

例 6：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    unsigned int i;
    for(i=0;i<=255;i++)
    {
        printf("%d %c    ",i,i);
        if(i%10==0)
            printf("\n");
    }
        return 0;
}
```

变量：其值可以改变的量被称为变量

int a=100;

a=101;

整型数据

- **整型常量:（按进制分）：**

  十进制：　以正常数字 1-9 开头，如 457 789

  八进制：　以数字 0 开头，如 0123

  十六进制：以 0x 开头，如 0x1e

- **整型变量:**
  - 有/无符号短整型(un/signed) short(int)　　2 个字节
  - 有/无符号基本整型(un/signed) int　　4 个字节
  - 有/无符号长整型(un/signed) long (int)　　4 个字节　　(32 位处理器)

实型数据(浮点型)

- **实型常量**
  - 实型常量也称为实数或者浮点数

    十进制形式: 由数字和小数点组成:0.0、0.12、5.0

    指数形式： 123e3 代表 123*10 的三次方

    123e-3
  - 不以 f 结尾的常量是 double 类型
  - 以 f 结尾的常量(如 3.14f)是 float 类型
- **实型变量**

  单精度(float)和双精度(double)3.1415926753456

  float 型:　占 4 字节，7 位有效数字,指数-37 到 38

  3333.333 33

  double 型: 占 8 字节，16 位有效数字,指数-307 到 308

字符数据

➢ **字符常量：**

直接常量：用单引号括起来，如：'a'、'b'、’0’等.

转义字符：以反斜杠“\”开头，后跟一个或几个字符、如'\n','\t'等，分别代表换行、横向跳格.

‘\\’表示的是\ ‘%%’ ‘\’’

➢ **字符变量：**

用 char 定义，每个字符变量被分配一个字节的内存空间

字符值以 ASCII 码的形式存放在变量的内存单元中;

注：char a;

a = 'x';

a 变量中存放的是字符'x'的 ASCII :120

即 a=120 跟 a='x'在本质上是一致的.

字符串常量

是由双引号括起来的字符序列，如“CHINA”、”哈哈哈”

“C program”，“$12.5”等都是合法的字符串常量.

字符串常量与字符常量的不同

‘a’为字符常量，”a”为字符串常量

每个字符串的结尾，编译器会自动的添加一个结束标志位'\0',

即“a”包含两个字符‘a’和’\0’

格式化输出字符：

%d 十进制有符号整数

%ld 十进制 long 有符号整数

%u 十进制无符号整数

%o 以八进制表示的整数

%x 以十六进制表示的整数

%f float 型浮点数

%lf double 型浮点数

%e 指数形式的浮点数

%c 单个字符

%s 字符串

%p 指针的值

特殊应用：

%3d %03d %-3d %5.2f

%3d 要求宽度为 3 位，如果不足 3 位，前面空格补齐;如果足够 3 位，此语句无效

%03d 要求宽度为 3 位，如果不足 3 位，前面 0 补齐;如果足够 3 位，此语句无效

%-3d 要求宽度为 3 位，如果不足 3 位，后面空格补齐;如果足够 3 位，此语句无效

%.2f 小数点后只保留 2 位

1.2.3 类型转换

数据有不同的类型，不同类型数据之间进行混合运算时必然涉及到类型的转换问题.

转换的方法有两种：

自动转换:

遵循一定的规则,由编译系统自动完成.

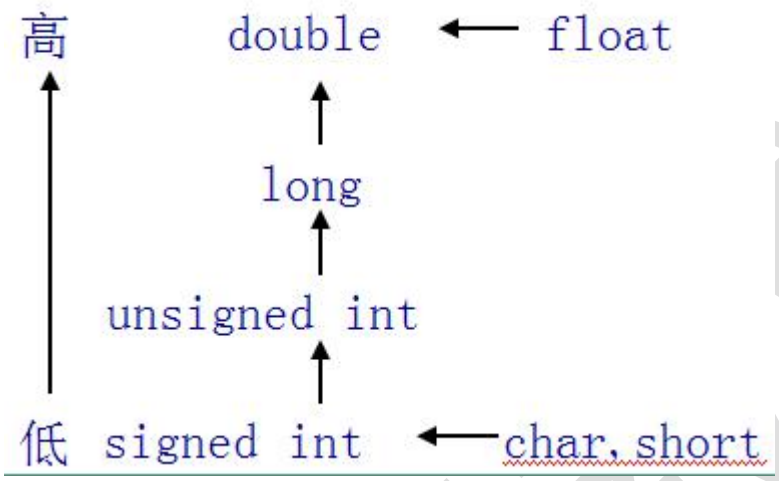
强制类型转换：

把表达式的运算结果强制转换成所需的数据类型

自动转换的原则：

1、 占用内存字节数少(值域小)的类型，向占用内存字节数多(值域大)的类型转换,以保证精度不降低.

2、 转换方向:



1) 当表达式中出现了 char 、short int 、int 类型中的一种或者多种，没有其他类型了参加运算的成员全部变成 int 类型的参加运算，结果也是 int 类型的‘

例 8：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("%d\n",5/2);
    return 0;
}
```

2) 当表达式中出现了带小数点的实数，参加运算的成员全部变成 double 类型的参加运算，结果也是 double 型。

例 9：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("%lf\n",5.0/2);
    return 0;
}
```

3) 当表达式中有有符号数 也有无符号数，参加运算的成员变成无符号数参加运算结果也是无符号数.(表达式中无实数)

例 10：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int a=-8;
    unsigned int b=7;
    if(a+b>0)
    {
        printf("a+b>0\n");
    }
    else
    {
        printf("a+b<=0\n");
    }
    printf("%x\n",(a+b));
    printf("%d\n",(a+b));
    return 0;
}
```

4) 在赋值语句中等号右边的类型自动转换为等号左边的类型

例 11：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int a;
    float b=5.8f;//5.8 后面加 f 代表 5.8 是 float 类型，不加的话，认为是 double 类型
    a=b;
    printf("a=%d\n",a);
    return 0;
}
```

5) 注意自动类型转换都是在运算的过程中进行临时性的转换，并不会影响自动类型转换的变量的值和其类型

例 12：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int a;
    float b=5.8f;//5.8 后面加 f 代表 5.8 是 float 类型，不加的话，认为是 double 类型
```

```
    a=b;
    printf("a=%d\n",a);
    printf("b=%f\n",b);//b 的类型依然是 float 类型的，它的值依然是 5.8
    return 0;
}
```

强制转换:通过类型转换运算来实现

(类型说明符) (表达式)

功能：

把表达式的运算结果强制转换成类型说明符所表示的类型

例如：

(float)a; // 把 a 的值转换为实型

(int)(x+y); // 把 x+y 的结果值转换为整型

注意:

类型说明符必须加括号

```
例 13：
#include <stdio.h>
int main(int argc, char *argv[])
{
    float x=0;
    int i=0;
    x=3.6f;
    i = x;
    i = (int)x;
    printf("x=%f,i=%d\n",x,i);
    return 0;
}
```

说明：

无论是强制转换或是自动转换，都只是为了本次运算的需要，而对变量的数据长度进行的临时性转换，而不改变数据**定义**的类型以及它的值

1.2.4 指针

1.3 运算符

1.3.1 运算符

用运算符将运算对象(也称操作数）连接起来的、符合C语法规则的式子，称为C表达式

运算对象包括常量、变量、函数等

例如： a * b / c－ 1 .5 ＋ 'a'

1.3.2 运算符的分类：

1、双目运算符：即参加运算的操作数有两个
例：+
a+b
2、单目运算符：参加运算的操作数只有一个
++自增运算符　给变量值+1
--自减运算符

int a=10;
a++;
3、三目运算符:即参加运算的操作数有 3 个
()?():()

1.3.3 算数运算符

+ - * / %　+=　-=　*=　/=　%=
10%3　表达式的结果为 1
复合运算符：
a += 3　相当于 a=a+3
a*=6+8　相当于 a=a*(6+8)

1.3.4 关系运算符

（>、<、==、>=、<=、!= ）
!=为不等于
一般用于判断条件是否满足或者循环语句

1.3.5 逻辑运算符

1、&& 逻辑与
两个条件都为真，则结果为真
if((a>b) && (a<c))
if(b<a<c)//这种表达方式是错误的
2、|| 逻辑或
两个条件至少有一个为真，则结果为真
if((a>b) || (a<c))
3、！逻辑非
if(!(a>b))
{
}

1.3.6 位运算符

1、**&**按位 与
任何值与 0 得 0，与 1 保持不变
使某位清 0
0101 1011　&

1011 0100

0001 0000

2、| 按位或

任何值或 1 得 1，或 0 保持不变

0101 0011 |

1011 0100

1111 0111

3、~ 按位取反

1 变 0，0 变 1

0101 1101 ~

1010 0010

4、^ 按位异或

相异得 1，相同得 0

1001 1100 ^

0101 1010

1100 0110

5、位移

>>右移

<< 左移

注意右移分：逻辑右移、算数右移

（1）、右移

逻辑右移 高位补 0，低位溢出

算数右移 高位补符号位，低位溢出 （有符号数）

-15

1000 1111

1111 0000

1111 11 00 -4

A）、逻辑右移

低位溢出、高位补 0

0101 1010 >>3

0000 1011

B）、算数右移：

对有符号数来说

低位溢出、高位补符号位。

1010 1101 >> 3

1111 010 1

0101 0011 >>3

0000 101 0

总结 右移：

1、逻辑右移 高位补 0，低位溢出

注：无论是有符号数还是无符号数都是高位补 0，低位溢出

2、算数右移 高位补符号位，低位溢出 （有符号数）

注：对无符号数来说，高位补 0，低位溢出
对有符号数来说，高位补符号位，低位溢出

在一个编译系统中到底是逻辑右移动，还是算数右移，取决于编译器

```
判断右移是逻辑右移还是算数右移
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("%d\n",-1>>3);
    return 0;
}
如果结果还是-1 证明是算数右移
```

（2）、左移<< 高位溢出，低位补 0
5<<1

1.3.7 条件运算符号

()?():()
A?B:C;
如果？前边的表达式成立，整个表达式的值，是？和：之间的表达式的结果
否则是：之后的表达式的结果

```
例 14：
#include <stdio.h>
int main(int argc, char *argv[])
{
    int a;
    a=(3<5)?(8):(9);
    printf("a=%d\n",a);
    return 0;
}
```

1.3.8 逗号运算符

(... , ... , ...)
例如：A = (B , C , D)

```
例 15：
#include <stdio.h>
int main(int argc, char *argv[])
{
    int num;
    num=(5,6);
```

```
    printf("%d\n",num);
    return 0;
}
```

1.3.9 自增自减运算符

++ --

i++ i--

运算符在变量的后面，在当前表达式中先用 i 的值，下条语句的时候 i 的值改变

例 16：

```
#include <stdio.h>
int main()
{
    int i=3;
    int num;
    num=i++;
    printf("num=%d,i=%d\n",num,i);//num=3 ,i=4
    return 0;
}
```

++i 先加 ，后用

例 17：

```
#include <stdio.h>
int main()
{
    int i=3;
    int num;
    num=++i;
    printf("num=%d,i=%d\n",num,i);//num=4,i=4
    return 0;
}
```

```
    return 0;
}
```

1.3.10 运算符优先级表

优先级别	运算符	运算形式	结合方向	名称或含义
1	()	(e)	自左至右	圆括号
	[]	a[e]		数组下标
	.	x.y		成员运算符
	->	p->x		用指针访问成员的指向运算符
2	- +	-e	自右至左	负号和正号
	++ --	++x 或 x++		自增运算和自减运算
	!	! e		逻辑非
	~	~e		按位取反
	(t)	(t)e		类型转换
	*	*p		指针运算，由地址求内容
	&	&x		求变量的地址
	sizeof	sizeof(t)		求某类型变量的长度
3	* / %	e1 * e2	自左至右	乘、除和求余
4	+ -	e1+e2	自左至右	加和减
5	<< >>	e1<<e2	自左至右	左移和右移
6	< <= > >=	e1<e2	自左至右	关系运算(比较)
7	== !=	e1==e2	自左至右	等于和不等于比较
8	&	e1&e2	自左至右	按位与
9	^	e1^e2	自左至右	按位异或
10	|	e1|e2	自左至右	按位或
11	&&	e1 && e2	自左至右	逻辑与(并且)
12	||	e1||e2	自左至右	逻辑或(或者)
13	? :	e1? e2: e3	自右至左	条件运算
14	=	x=e	自右至左	赋值运算
	+= -= *=	x+=e		复合赋值运算
	/= %= >>=			
	<<= &= ^=			
	|=			
15	,	e1, e2	自左至右	顺序求值运算

1.4 控制语句相关关键字讲解

1.4.1 选择控制语句相关的关键字

1、 if 语句

形式：

1) if(条件表达式)

{//复合语句，若干条语句的集合

语句 1;

语句 2;

}

如果条件成立执行大括号里的所有语句，不成立的话大括号里的语句不执行

例 20：

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

```
int main()
{
    int a=10;
    if(a>5)
    {
        printf("a>5\n");
    }
    return 0;
}
```

2) if(条件表达式)
{
语句块 1
}
else
{
语句块 2
}

if else 语句的作用是，如果 if 的条件成立，执行 if 后面{}内的语句，否则执行 else 后的语句

例 21：

```
#include<stdio.h>
int main()
{
    int a=10;
    if(a>5)
    {
        printf("a>5\n");
    }
    else
    {
        printf("a<=5\n");
    }
    return 0;
}
```

注意 if 和 else 之间只能有一条语句，或者有一个复合语句，否则编译会出错

例 22：

```
if()
    语句 1；
    语句 2；
else
```

```
        语句 3；
        语句 4；
```

错误：if 和 else 之间只能有一条语句,如果有多条语句的话加大括号

例 23：

```
if()
{
        语句 1；
        语句 2；
}
else
{
        语句 3；
        语句 4；
}
```

正确

3) if(条件表达式)

```
{
}
else if(条件表达式)
{
}
else if(条件表达式)
{
}
else
{
}
```

在判断的时候，从上往下判断，一旦有成立的表达式，执行对应的复合语句，
下边的就不再判断了，各个条件判断是互斥的

例 24：

```
#include <stdio.h>
int main(void)
{
        char ch;
        float score = 0;
        printf("请输入学生分数:\n");
        scanf("%f",&score);
        if(score<0 || score >100)
```

```
        {
            printf("你所输入的信息有错\n");
        return 0;
        }
        else if( score<60)
        {
            ch = 'E';
        }
        else if ( score < 70 )
        {
            ch = 'D';
        }
        else if ( score < 80 )
        {
            ch = 'C';
        }
        else if ( score < 90 )
        {
            ch = 'B';
        }
        else
        {
            ch = 'A';
        }

        printf("成绩评定为：%c\n",ch);
        return 0;
}
```

2、 switch 语句

```
switch（表达式）//表达式只能是字符型或整型的(short int    int    long int)
{
    case 常量表达式1：
        语句1；
        break；
    case 常量表达式2：
        语句2;
        break；
    default：
```

语句3；
break;
}

注意：break 的使用

例 25：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int n;
    printf("请输入一个 1~7 的数\n");
    scanf("%d",&n);
    switch(n)
    {
        case 1:
            printf("星期一\n");
            break;
        case 2:
            printf("星期二\n");
            break;
        case 3:
            printf("星期三\n");
            break;
        case 4:
            printf("星期四\n");
            break;
        case 5:
            printf("星期五\n");
            break;
        case 6:
            printf("星期六\n");
            break;
        case 7:
            printf("星期天\n");
            break;
        default:
            printf("您的输入有误，请输入 1~7 的数\n");
            break;
    }
    return 0;
```

```
}
```

1.4.2 循环控制语句相关的关键字

1、 for 循环

for(表达式 1;表达式 2;表达式 3)
{//复合语句，循环体

}

第一次进入循环的时候执行表达式 1，表达式 1 只干一次，

表达式 2，是循环的条件，只有表达式 2 为真了，才执行循环体，也就是说
每次进入循环体之前要判断表达式 2 是否为真。

每次执行完循环体后，首先执行表达式 3

例 25：for 循环求 0~100 的和

```
#include <stdio.h>
int main(void)
{
    int i;
    int sum=0;
    for(i=1;i<=100;i++)
    {
        sum = sum+i;
    }
    printf("sum=%d\n",sum);
    return 0;
}
```

例 26：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int i,j;
    for(i=1;i<=9;i++)
    {
        for(j=1;j<=i;j++)
        {
            printf("%d*%d=%d   ",i,j,i*j);
```

```
            }
            printf("\n");
        }
        return 0;
}
```

2、 while 循环

1) 形式 1：
while(条件表达式)
{//循环体，复合语句

}

进入 while 循环的时候，首先会判断条件表达式是否为真，为真进入循环体，否则退出循环

例 27：

```
#include <stdio.h>
int main(void)
{
```

2) 形式 1：
while(条件表达式)
{//循环体，复合语句

}

```
    int i=1;
    int sum=0;
    while(i<=100)
    {
        sum = sum+i;
        i++;
    }
    printf("sum=%d\n",sum);
    return 0;
}
```

3) 形式 2 ： do
do{//循环体

}while(条件表达式);

先执行循环体里的代码，然后去判断条件表达式是否为真，为真再次执行循环体，否则退出循环

例 28：

```
#include <stdio.h>
int main(void)
{
    int i=1;
    int sum=0;
     do
    {
        sum = sum+i;
        i++;
    }while(i<=100);
    printf("sum=%d\n",sum);
    return 0;
}
```

形式 1 和形式 2 的区别是，形式 1 先判断在执行循环体，形式 2 先执行循环体，再判断

break 跳出循环

continue 结束本次循环，进入下一次循环

例 29：

```
#include <stdio.h>
int main(void)
{
    int i;
    int sum=0;
    for(i=1;i<=100;i++)
    {
        if(i==10)
            break;//将 break 修改成 continue 看效果
        sum = sum+i;
    }
    printf("sum=%d\n",sum);
    return 0;
}
```

return 返回函数的意思。结束 return 所在的函数，

在普通函数中，返回到被调用处，在 main 函数中的话，结束程序

3、 goto

例 30：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
	printf("test000000000000000000\n");
	printf("test1111111111111111\n");
	goto xiutao;
	printf("test22222222222222222\n");
	printf("test3333333333333333\n");
	printf("test444444444444444444444\n");
	printf("test5555555555555555555\n");
xiutao:
	printf("test66666666666666666\n");
	return 0;
}
```

千锋教育