C语言的陌生点

陈坤出品，必属精品！

1.变量命名

可以用小写字母、大写字母、数字和下划线（_）来命名

注意：名称的第一个字符必须是字母或下划线，不能是数字！

有效的名称	无效的名称
cat2	Hot-Yab
_kcab	2cat

2.*（了解）位、字节和字

- 位（bit）：最小的存储单元，可以存储0或1.
- 字节：常用的计算机存储单位，**一字节均为8位（bit）**.

> 8位字节有256即2^8种不同的组合，通过二进制编码便可表示0~255的整数或一组字符。

- 字（word）：设计计算机时给定的自然存储单位。

> 对于8位微型计算机，一个字长只有8位。目前个人计算机字长增长到64位。计算机的字长越大，数据转移越快，允许的内存访问就更多。

3.基本数据类型及相关知识

int类型

a.存储范围

一般而言，存储一个int类型的数据需要占用一个机器字长(至少占16位)。而对于32位及以上的计算机而言，int类型数据的取值数有2^{32}个。

> 因为对于32位计算机，一字长有32位，这32位有2的32次方种变化。64位计算机中的int类型数据依然按照32位计算机计算。

因此，对于一个有符号的int数据，它的存储范围是 -2^{31}~2^{31}-1，而ISO C规定的int类型的最小范围是以16位字长为单位的存储方式，即-32768~32767

> -1的原因是0的存在。

b.进制的使用和显示

使用进制：

- 0x/0X表示十六进制的前缀。eg：0X1001 == 9
- 0表示八进制前缀。eg：020 == 16

显示进制：

- %d表示以十进制显示数字；
- %o表示以八进制显示数字；
- %x表示以十六进制显示数字；

> 注意：要显示各进制的前缀0，0x，0X，必须要分别使用%#o，%#x，%#X.

c.其他整数类型及其打印

- short int （或简写成short）：%hd...(至少占16位)
- long int (long): %ld...(至少占32位)
- long long int(long long): %lld(至少占64位)
- unsigned int(unsigned): %u...

> - unsigned long int: %lu...

*溢出：溢出相应类型会重新从起点开始，如下面的例子

```c
#include <stdio.h>

int main (void)
{
    short int i = 32767;
    unsigned short int j = 65535;
    printf("i = %hd,i + 1 = %hd,i + 2 = %hd\n",i,i+1,i+2);
    printf("j = %hu,j + 1 = %hu,j + 2 = %hu\n",j,j+1,j+2);
    return 0;
}
/*运行结果:
i = 32767,i + 1 = -32768,i + 2 = -32767
j = 65535,j + 1 = 0,j + 2 = 1
*/
```

float 类型

a.浮点数的表示方式

数字	科学计数法	指数计数法
100000	1.0$ \times $$10^5$	1.0e9
322.56	3.2256$\times$$10^2$	3.2256e2

b.其它浮点型类型及其打印

- float :%f
- double :%f

！！！这里一定要注意，对double类型输入用%lf，输出用%f

- long double:%Lf

> 若想用科学计数法表示则需要用%e，或者%Le

c.浮点数的上溢和下溢

浮点数的上溢和下溢

char类型

> C语言把1字节定义为char类型占用的位（bit）数

a.字符常量的初始化

字符常量以整形数字的形式储存，每个字符对于ASCII码上的码值，如果要把一个字符常量初始化，不必背下ASCII码表：

```
char ch;
ch = 'A'//这里的单引号理解为获取这个字符的ASCII码值
```

b.转义序列

C语言转义序列

4.const限定符

const 关键字：用于限定一个变量为只读

```
const int MONTHS = 12;//这意味着MONTHS这一变量的值将不会被改变
```

5.printf()

a.转换说明修饰符

修饰符	含义
标记	五种标记见附表（-、+、空格、#、0）
数字	最小字段宽度 eg : "%4d" **注意！！** 字段宽度会自动根据所占位数而调整
.数字	保留小数的位数/保留字符串长度 eg： "%5.2f"

附表：

标记	含义
-	左对齐：从字段的左侧开始打印"%-d"
+	有符号若为正则添加+号，为负则添加-号
空格	为正则向前添加空格
#	"%#o"、"%#x"
0	前导0代替空格填充字段宽度

b.一个大于255的数转化为字符型会发生什么？

答：该数以256为模得到的值对应ASCII码的字符。

> 对于其他超限问题，可以类似这种方式进行强制缩小转换得到对应值

- 有关这方面的内容请参考《C prime plus》P88~P89

c.printf()返回值

- 若输出正确，则返回printf()参数个数；
- 若输出错误，则返回负值；

d.打印较长的字符串

可以采用三种方法：

- 方法一：用多个printf()语句

```
    printf("Here's one way to print a ");
    printf("long string.\n");
```

- 方法二：在""中插入\

```
    printf("Here's one way to print a \
long string");//注意这里不能让long string缩进
```

- 方法三：采用多个""

```
    printf("Here's one way to print a"
           "long string");
```

6.scanf()

a.从scanf角度看输入

- %d :scanf跳过空白字符，直至遇到第一个非空白字符。因为要读取整数，所以scanf希望遇到一个+ -或者数字字符，它将一直保存这类字符直至遇到第一个非数字字符。然后scanf会将此非数字字符返回输入端。

> 注意：如果scanf遇到的第一个非空白字符是非数字、+、-字符，scanf会将此字符返回输入段，并不会将其赋给变量，程序继续进行！

> 注意：如果一个scanf含有多个参数变量时，当前一个读取失败后，不会继续向下读取

- %o,%x，%f :与%d相似，不同的是他们或许有更多的可读取字符。

> 对于%f：“.”也是可读取字符

> 对于%x：“a~f”也是合法字符

- %s :scanf会跳过空白字符开始读取第一个非空白字符，直至再次遇到空白为止。需要注意的是它会在字符序列的末尾添上'\0'

b.scanf的返回值

- 如果scanf读取数据成功，那么返回读取项数；
- 如果需要读取一个数字而用户却输入一个非数字字符串则返回0；
- 当检测到文件末尾时，会返回 "EOF"

7.printf与scanf的*修饰符

*a.printf中的*修饰符*

提供两个额外的参数：

- “.”之前的参数为数字宽度
- “.”之后的参数为保留小数位数

请看下面的例子：

```c
#include <stdio.h>
int main (void)
{
    float i = 666.666;
    printf("i = %*.*f",10,4,i);
    return 0;
}
/*运行结果:
i =   666.6660
*/
```

*b.scanf的*修饰符*

把*放在%和转换字母之间，会让scanf跳过对应的数字，请看下面的例子：

```c
#include <stdio.h>
int main (void)
{
    int a;
    scanf("%*d %*d %d",&a);
    printf("a = %d",a);
    return 0;
}
/*运行结果：
输入：123 456 789
输出：a = 789
*/
```

8.负数求模

C99规定“趋零截断”：(int)3.6 = 3; (int)-2.7 = -2.有了此条规则后，负数求模有如下结果：
第一个运算对象是负数，那么求模结果是负数，第一个运算对象是正数，那么求模结果是正数

请看下面的例子：

- 11 % 5 = 1
- 11 % -5 = 1
- -11 % -5 = -1
- -11 % 5 = -1

9.类型转换

- char和short类型数据出现在表达式时，会被自动转换成int类型
- 在函数参数传递过程中：char和short会被自动转换成int类型，float会被自动转换成double类型
- 赋值表达式语句中，计算的最终结果会被转换成被赋值变量的类型
- 在混合类型的运算中，较小类型会被转换成较大类型

10.逗号运算符

逗号运算符把两个表达式连接成一个表达式，并保证左边的表达式最先求值。整个逗号表达式的值是逗号右侧的表达式。

下面我们看一个逗号表达式的例子（这并不是个优秀程序员敲出来的代码）

```c
#include <stdio.h>
int main ()
{
    int x,y,z;
    x = (y = 3, (z = ++y + 2) + 5);
    printf("x = %d",x);
    return 0;
}
/*输出结果:
x = 11
*/
```

但是逗号表达式常常被用于for循环中:

```c
for (step = 2,fargo = 0;fargo < 1000;step *=2)
    fargo += step;
```

11.条件运算符 “？ ： ”

C语言中的唯一三元运算符。条件表达式的通用格式如下:

expression1 ? expression2 :expression3

> 它的含义是：如果expression1为真，则执行expression2，并将expression2的值赋给整个表达式；否则将执行expression3，并将其值赋给整个表达式；

```c
//max等于a与b中的最大值
max = (a > b) ? a : b;
```

12.switch的多重标签

示例:

```c
switch( expression )
{
    case label1:
    case label2:statement1 ;
                break;
    case label3:statement2;
                break;
}
```

在本示例中，

- 如果expression的值与label1或label2的值相等，则执行statement1；
- 如果expression的值与label3的值相等，则执行statement2；

13.！！文件结尾EOF

这里有疑惑《C prime plus》p220与测试结果不符合

14.数组的初始化

a.int类型的整形数组

未初始化之前，数组内的每一个元素被赋予了垃圾值，初始化后，若初始化数组元素数小于数组全体元素数，则其余元素自动被赋予0；

b.指定初始化器（C99）

可以在初始化列表中使用带方括号的下标指明待初始化元素，请见下例：

```
int Arr[6] = {[5] = 212}//将212赋值给Arr[5]
```

15.sizeof运算符

sizeof操作符以字节形式给出了其操作数的存储大小。操作数可以是一个表达式或括在括号内的类型名。操作数的存储大小由操作数的类型决定。

- 用于数据类型：sizeof(type)
- 用于变量：sizeof(var_name) or sizeof var_name

> 需要注意以下几点要求：
>
> - 当操作数具有数组类型时，其结果是数组的总字节数。
> - 联合类型操作数的sizeof是其最大字节成员的字节数。

16.函数的形式参数与实际参数

- **形参**出现在函数定义中，在整个函数体内都可以使用，离开该函数则不能使用。
- **实参**出现在主调函数中，进入被调函数后，实参变量也不能使用。

> 形参和实参的功能是作数据传送。发生函数调用时，主调函数把实参的值传送给被调函数的形参从而实现主调函数向被调函数的数据传送。

1. 形参变量只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只有在函数内部有效。函数调用结束返回
   主调函数后则不能再使用该形参变量
   。
2. 实参可以是常量、变量、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值，以便把这些值传送给形
   参。 因此应预先用赋值，输入等办法使实参获得。
3. 实参和形参在数量上，类型上，顺序上应严格一致，否则会发生“类型不匹配”的错误
4. 函数调用中发生的数据传送是单向的。 即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。因此在函数调用过程中，形参的值发生改变，而实参中的值不会变化。
5. 当形参和实参不是指针类型时，在该函数运行时，形参和实参是不同的变量，他们在内存中位于不同的位置，形参将实参的内容复制一份，在该函数运行结束的时候形参被释放，而实参内容不会改变。而如果函数的参数是指

针类型变量,在调用该函数的过程中，传给函数的是实参的地址，在函数体内部使用的也是实参的地址，即使用的就是实参本身。所以在函数体内部可以改变实参的值。

17.字符串疑难陌生点

a.字符串数组初始化

字符串的初始化方式：

```
char ch1[40]={'H','e','l','l','o',',','W','o','r','l','d','!','\0'};
char ch2[] = "Hello,World!";
char *p = "Hello,World!";
```

以上三种方式中需要注意：

- 要确保数组元素个数比字符串长度至少多1，系统在字符串末尾添加空字符
- 初始化时方式一末尾一定添加字符串空字符'\0',添加了后是字符串，否则是字符数组；
- 初始化后数组中的其他元素被自动置0；

b.数组和指针对字符串存储方式的不同

- **数组形式**字符串储存在静态存储区，当程序开始运行时，才为数组分配内存，此后才将字符串拷贝到数组中。注意此时字符串有两个副本，一个在静态存储区，另外一个是存储在数组中的字符串。
- **指针形式**系统为指针变量分配一个指针变量的存储空间后，让指针变量指向字符串首字符的地址。

因此，用指针形式的字符串更加高效！

c.字符串的输入比较

1. scanf()函数

错误的形式：

```
char *a;
scanf("%s",a);
```

出错的原因是a指针并没有指向任何地址，它是个野指针。
正确的形式:

```
char a[100];
scanf("%s",a);
```

a指针指向一个分配了100内存的数组的首地址。

> 用scanf函数输入字符串具有很大的局限性：

> scanf只能读取到空格处，只能读取到一个单词，面对整行输入时，他显得无能为力

2. 不幸的gets()函数
使用形式：

```
char word[100];
gets(word);
```

需要注意的是，gets()函数简单易用，它读取整行输入，直至遇到换行符，然后丢弃换行符，存储其余字符，并在字符结尾添加空字符。

与之对应的是puts()函数，用于显示字符串，并且会自动在末尾添加换行符。

> gets()函数也具有一定的局限性：
> 它无法保证字符串长度合适地保存到数组中，也就是说存在缓冲区溢出的情况，并占用其他内存，具有一定的风险。

3. gets()的替代品fgets()函数
使用形式：

```
char a[100];
fgets(a,100,stdin);
```

- fgsts()函数的第二个参数为n，那么函数将会读入n-1个字符；
- fgets()函数读到换行符时会将其保存到字符串中；
- 第三个参数代表要读入的文件，stdin为标准输入端；
- fgets()函数与fputs函数相对应，fputs()函数不在语句后自动添加换行符。

```
fputs(a,stdout)//将字符串输出到标准输出端
```

18.常见的字符串函数

注意：本节所有函数存储在 <string.h> 头文件中

a.strlen()

功能：测量字符串长度

输入参数：字符串指针

返回值：字符串长度（不包括'\n'）

b.strcat()与strncat

```
c = strcat(a,b);
```

功能： 将 b 字符串接到 a 字符串后

输入参数：两字符串指针

返回值： a 字符串指针

> 注意必须保证 a 字符串拥有足够的空间存储这两个字符串。

> 对于 strncat() 函数，与 strcat() 函数不同的是，提供第三个参数指定了最大添加字符数

c.strcmp()与strncmp()

```
cmp = strcmp(ch_1,ch_b);
```

功能： 依次比较字符串 ch_1 与 ch_2 的每一个字符

输入参数： ch_1 与 ch_2 是字符串指针

返回值：若两字符串相同则返回 0 ， 若不相同返回非 0

> 对于 strncmp() 函数，与 strcmp() 函数不同的是，提供第三个参数指定了比较字符数。

d.strcpy()与strncpy()

```
char target[20];
int x;
x = 5;//数字赋值
strcpy(target,"Hello,World!");//字符串赋值
/*错误的表达方式：
target = "Hello,World!";
*/
```

功能： strcpy() 函数就像是字符串的赋值函数，能够实现从一个字符串复制到另外一个字符串，值得注意的是赋值前，目标字符串会被清空。

输入参数：目标字符串和源字符串指针

返回值：第一个参数

> 对于 strncpy() 函数，与 strcpy() 函数不同的是，提供第三个参数指定了可拷贝的最大字符数。

其他常见字符串操作头文件<ctype.h>

	函 数	功 能	说 明
1	**int isalnum(int ch);**	判断字符变量ch是否为字母或数字	当ch为数字0-9或字母a-z及A-Z时，返回非零值，否则返回零.
2	**int isalpha(int ch);**	判断字符变量ch是否为字母	当ch为字母a-z及A-Z时，返回非零值，否则返回零。
3	**int islower(int ch);**	判断字符变量ch是否为小写字母	当ch为字母a-z时，返回非零值，否则返回零。
4	**int isupper(int ch);**	判断字符变量ch是否为大写字母	当ch为字母A-Z时，返回非零值，否则返回零。
5	**int isdigit(int ch);**	判断字符变量ch是否为十进制数字	当ch为数字0-9时，返回非零值，否则返回零。
6	**intisxdigit(int ch);**	判断字符变量ch是否为十六进制数字	当ch为数字0-9 a-f A-F时，返回非零值，否则返回零。
7	**int iscntrl(int ch);**	判断ch是否控制字符(其ASCII码 0-31之间).	当ch为控制符时，返回非零值，否则返回零。
8	**int isgraph(int ch);**	判断ch是否为可显示的图形字符	当ch为图形字符时，返回非零值，否则返回零。

	函数	功能	说明
9	**int isspace(int ch);**	判断ch是否为空格符或跳格符或换行符	当ch为符合时，返回非零值，否则返回零。
10	**int isblank(int ch);**	判断ch是否为空格符或跳格符	当ch为符合时，返回非零值，否则返回零。
11	**int isprint(intch);**	与isgraph类似，增加了空格	
12	**int ispunct(int ch);**	判断ch是否为标点字符	
13	**int tolower(int ch);**	转换ch（A-Z）对应的小写字母	非（A-Z）字符保持不变
14	**int toupper(int ch);**	转换ch对应的大写字母	非（a-z）字符保持不变

19.多维数组（二维数组）的指针表示法

二维数组的数组理解

对于一个二维数组Arr[m][n]，可以理解为存在m个一维数组Arr[0]~Arr[m-1]，使每一个这样数组都包括n个元素。

二维数组的指针理解

对于一个二位数组Arr[m][n]，*（Arr+i）代表每一个一维数组的首地址，而 *（ *(Arr+i）+j）代表每一个数组元素，他们有下面的对应方式：

```
Arr[m][n]== *(*(Arr+m)+n);//这两种表达方式是等价的
```

下面我们看一个例子：用指针实现三维矩阵的倒置：

```
    int m[3][3]={1,2,3,4,5,6,7,8,9};
    int i,j,tem;
    for(i = 0;i < 3;i++)
    {
        for(j = i;j < 3;j++)
             {
                tem = *(*(m+i)+j);
                *(*(m+i)+j) = *(*(m+j)+i);
                *(*(m+j)+i) = tem;
             }
    }
```

上述表达用数组形式体现得到下面的结果：

```
    int m[3][3]={1,2,3,4,5,6,7,8,9};
    int i,j,tem;
    for(i = 0;i < 3;i++)
    {
        for(j = i;j < 3;j++)
             {
                tem = m[i][j];
                m[i][j] = m[j][i];
                m[j][i] = tem;
             }
    }
```

熟练掌握指针与数组间的关系是需要加强的地方。

20.重点：存储类别

a.作用域

块作用域

作用域描述程序中可访问标识符的区域。块作用域是最常见的C语言作用域，在这里一般指的是一对花括号内括起来的代码区域。

- 整个函数体是一个块，函数中的任意复合语句也是一个块。
- 定义在块中的变量具有块作用域，块作用域的变量的可用范围仅在块内。
- 尽管函数头部的形参声明在花括号外部，我们也称该变量属于函数的作用域。
- 我们看下面的这个例子：

```
for(int i = 0;i < 10;i++)
{
    getchar(Arr);
    printf("Hello,World!");
}
```

在这个例子中，对于i变量只属于for循环这个块作用域，离开for循环后，i变量失效。

文件作用域

定义在函数外面的变量，具有文件作用域。具有文件作用域的变量，从它的定义处到文件末尾处均可用。

文件作用域变量，即我们所说的全局变量。

b.链接

- 无链接：具有块作用域的变量都属于无链接变量。这意味着这些变量属于定义他们的块、函数或原型私有。
- 外部链接和内部链接：具有文件作用域的变量具有外部或内部链接，其中可以在多个文件中空用的是外部链接变量，内部链接变量只能在一个翻译单元中使用。

```
int a = 6;
static int b = 7;
int main ()
{
...
}
```

在上面的这个例子中，a 这个变量属于外部链接变量，可以供多个文件使用，而 static 标识符的使用让b这个变量成为了内部链接变量，只能在该文件内部使用。

c.存储期

- 静态存储期：文件作用域变量具有静态存储期，静态存储内存中的变量从程序开始到程序结束的这段时间内都存在。
- 自动存储期：块作用域的变量通常具有自动存储期。

d.存储类别！！！

5 种存储类别:

存储类别	存储期	作用域	链接	声明方式
自动	自动	块	无	块内
寄存器	自动	块	无	块内，使用关键字register
静态外部链接	静态	文件	外部	所有函数外
静态内部链接	静态	文件	内部	所有函数外，使用关键字 static
静态无链接	静态	块	无	块内，使用关键字 static

自动变量（auto）

属于自动存储类别的变量具有自动存储期，块作用域，无链接。默认情况下声明在块或函数头中的任何变量都属于自动变量，可以强调显示使用 auto 存储类别说明符：

```
int main (void)
{
    auto int val;
    ...
}
```

寄存器变量（register）

在程序运行时，根据需要到内存中相应的存储单元中调用，如果一个变量在程序中频繁使用，例如循环变量，那么，系统就必须多次访问内存中的该单元，影响程序的执行效率。因此，C语言\C++语言还定义了一种变量，不是保存在内存上，而是直接存储在CPU中的寄存器中，这种变量称为寄存器变量。

- C编译程序会自动地将寄存器变量变为自动变量。
- 由于受硬件寄存器长度的限制，所以寄存器变量只能是char、int或指针型。寄存器说明符只能用于自动存储期变量，因此不允许将外部变量或静态变量说明为"register"。
- register变量使用的是硬件CPU中的寄存器，寄存器变量无地址，所以不能使用取地址运算符"&"求寄存器变量的地址。

块作用域的静态变量

可以创建具有静态存储期、块作用域的局部变量。这些变量和自动变量一样，具有相同的块作用域，但是程序离开它们所在的函数后，这些变量不会消失，不过也无法直接使用这些变量，但可以通过访问变量地址的方式使用变量。

```
int main (void)
{
    static int num;
    ...
}
```

在块内部使用 static 关键字声明块作用域的静态变量。

外部链接的静态变量、内部链接的静态变量

外部链接的静态变量，具有外部链接，静态存储期和文件作用域。放在块之外声明的变量属于该类型变量，可以加存储类别关键字 extern 再次声明加以强调：

> 注意 extern 的含义是表面声明的变量定义在别处，要求该变量一定具有外部链接。

```
#include <stdio.h>
char Coal// 外部链接的静态变量
int main (void)
{
    extern char Coal;//再次声明时可加 extern
    ...
}
```

如果一个源代码中的文件使用外部变量定义在另一个源文件中，则extern关键字不可省略。

内部链接的静态变量需要在快外部定义变量时加static关键字。

二十一、结构体、共用体与枚举

a.结构体

在相同的数据对象中存储多个不同类型数据项的办法是，使用标记标识一个具体的结构模板，并声明该类型的变量。通过点运算符(.)可以使用结构模板中的标签来访问结构的各个成员。指向结构的指针，可以用该指针和间接成员运算符(->)来访问各结构成员。

有关结构体的内容这里不再赘述。

b.联合简介

联合，即共用体，是一种数据类型，它能在同一个内存空间中存储不同的数据类型（并非同时存储）。下面是一个带标记的联合模板：

```
union hold
{
    int digit;
    double bigfl;
    char letter;
};
```

根据以上声明的结构可以存储一个int类型数据***或***一个double类型数据***或***一个char类型数据。

下面是联合的一种使用方法：

```
union hold fit;
fit.digit = 23;//把23存储在fit，占2字节
fit.bigfl = 2.3//清除23，存储2.3，占8字节
fit.letter = 'a'//清除2.3，存储a，占1字节
```

在联合中，一次只存储一个值。但值得注意的是，编译器会分配给共用体足够的空间以便它存储数据，例如本例中，编译器会给共用体hold分配8字节的存储空间（联合内各数据类型的所占最大存储量）。即使有足够的空间也不能同时存储一个char和一个int类型的数据。

c.枚举类型

使用enum关键字创建一个枚举类型，并指定它拥有的值。

> 笔者所理解的枚举类型，就是给不同的常量命名，以提高程序的可读性。

例如有下面的用法：

```
enum spectrum {red,orange,yellow,green,blue,white};
enum spectrum color;
```

在本例中创建了枚举类型，而它将 red 赋值 0，将 orange 赋值 1，……以此类推。

```
printf("red = %d,orange = %d\n",red,orange);
//输出：red = 0,orange = 1
```

默认情况下枚举给它所包含的标签赋值0、1、2……，但依然可以自行赋值，而部分自行赋值，部分未自行赋值的情况下，编译器会根据实际情况赋值，请看下例自行理解，笔者不再阐释：

```
enum spectrum{red,orange,yellow = 8,green,blue,white};
/*在本例中：
red = 0;
orange = 1;
yellow = 8;
green = 9;
blue = 10;
white = 11;
*/
```

二十二、运算符的优先级

说明：

同一优先级的运算符，运算次序由结合方向所决定。
简单记就是：！ > 算术运算符 > 关系运算符 > && > || > 赋值运算符

优先级	运算符	名称或含义	*使用形式	结合方向	说明
1	[]	数组下标	数组名[常量表达式]	左到右	--
	()	圆括号	(表达式)/ 函数名 (形参表)	左到右	--

优先级	运算符	名称或含义	*使用形式*	结合方向	说明
	.	成员选择 （对象）	对象.成员名	左到右	--
	->	成员选择 （指针）	对象指针->成员名	左到右	--
2	-	负号运算符	-表达式	右到左	单目
	~	按位取反运算符	~表达式	右到左	单目
	++	自增运算符	++变量名/变量名++	右到左	单目
	--	自减运算符	--变量名/变量名--	右到左	单目
	*	取值运算符	指针变量	右到左	单目
	&	取地址运算符	&变量名	右到左	单目
	!	逻辑非运算符	!表达式	右到左	单目
	（类型名）	强制类型转换	(数据类型)表达式	右到左	--
	sizeof	长度运算符	sizeof(表达式)	右到左	--
3	/	除	表达式/表达式	左到右	双目
	*	乘	表达式*表达式	左到右	双目
	%	余数（取模）	整型%整型	左到右	双目
4	+	加	表达式+表达式	左到右	双目
	-	减	表达式-表达式	左到右	双目
5	<<	左移	变量<<表达式	左到右	双目
	>>	右移	变量>>表达式	左到右	双目
6	*>*	大于	表达式>表达式	左到右	双目
	>=	大于等于	表达式>=表达式	左到右	双目
	<	小于	表达式<表达式	左到右	双目
	<=	小于等于	表达式<=表达式	左到右	双目
7	==	等于	表达式==表达式	左到右	双目

优先级	运算符	名称或含义	*使用形式	结合方向	说明
	!=	不等于	表达式!= 表达式	左到右	双目
8	&	按位与	表达式&表达式	左到右	双目
	^	按位异或	表达式^表达式	左到右	双目
	|	按位或	表达式|表达式	左到右	双目
	&&	逻辑与	表达式&&表达式	左到右	双目
	||	逻辑或	表达式||表达式	左到右	双目
	?:	条件运算符	表达式1?表达式2: 表达式3	右到左	三目
9	=	赋值运算符	变量=表达式	右到左	--
	/=	除后赋值	变量/=表达式	右到左	--
	*=	乘后赋值	变量*=表达式	右到左	--
	%=	取模后赋值	变量%=表达式	右到左	--
	+=	加后赋值	变量+=表达式	右到左	--
	-=	减后赋值	变量-=表达式	右到左	--
	<<=	左移后赋值	变量<<=表达式	右到左	--
	>>=	右移后赋值	变量>>=表达式	右到左	--
	&=	按位与后赋值	变量&=表达式	右到左	--
	^=	按位异或后赋值	变量^=表达式	右到左	--
	|=	按位或后赋值	变量|=表达式	右到左	--
10	,	逗号运算符	表达式,表达式,…	左到右	--

二十三、文件输入/输出

a.基本知识

C语言文件模式

C语言提供两种文件模式：文本模式和二进制模式。

C语言提供两种访问文件的途径：文本模式和二进制模式。

保存文件地址的指针

使用关键字 FILE 声明指针，并一般用fopen()函数对其初始化

例如可以用下面的方式定义一个文件指针变量：

```
FILE * fp;
fp = fopen("HelloWorld.txt","r");
```

指向标准文件的指针

标准文件	文件指针	通常使用的设备
标准输入	stdin	键盘
标准输出	stdout	显示器
错误输出	stderr	显示器

b.与文件操作有关的函数

fopen()

功能：打开文件

参数：1.待打开文件名称（字符串首地址）2.fopen（）模式字符串

返回值：成功返回文件时，返回文件指针

> fopen()模式字符串

fclose()

功能：关闭文件

参数：待关闭文件地址

返回值：如果成功关闭，返回0；否则返回EOF

getc()与putc()

```
char ch;
ch = getc(fp);
```

这条语句的意思是从fp指定的文件中获取一个字符。

```
putc(ch,fpout);
```

这条语句的意思是把字符 ch 放入 FILE 指针 fpout 指定的文件中。

fprintf()与fscanf()

文件I/O函数 fprintf() 与 fscanf() 与 printf() 与 scanf()函数工作方式类似，不同的是，前者需要用第一个参数指定待处理的文件。

fgets()与fputs()

```
fgets(ch,LEN,fp);
/*
目的：将从fp指向的文件中读取LEN大小的字符串存储在ch字符串数组中
第一个参数代表存储输入位置的字符串数组地址
第二个参数代表待输入字符串的大小
第三个参数是文件指针，指定待读取的文件
*/
```

```
fputs(ch,fp);
/*
目的：将ch字符串数组中存储的字符串写入fp指向的文件中
第一个参数是存储字符串数组的地址
第二个参数是待写入文件的文件指针
*/
```

fseek()与ftell()

fseek()与ftell()函数用法（了解）

ungetc() 亲测好用

把 ch 指定的字符放回输入流以供下次调用输入函数时将读取该字符：

例： c语言输入一行未知个数字符，获取其中的数字并存入数组

- 问题背景：

1.对于任意输入的一串字符串如何高效地取出指定的部分；

2.对于整形或浮点型数组而未知输入个数之时，如何及时输入与终止；

3.如何让录入的数据重新回到输入流

- 所需函数：

1. <ctybe.h>

isdigit：判断是否为十进制数，是返回非0，否则返回0；

2. <stdio.h>

ungetc(c,stdin)：将c中的数据返回输入流；

完整代码见下：

```
#include<stdio.h>
#include<ctype.h>
int main()
{
    int a[1000];
    char c;
    int i = 0, n = 0;
    while((c = getchar())!= '\n')
    {
        if(isdigit(c))
        {
            ungetc(c,stdin);//将c送回输入流
```

```
            scanf("%d",&a[n++]);
        }
    }
    for(i = 0;i < n;i++)
        printf("%d ",a[i]);
    return 0;
}
```

二进制 I/O

fread() 函数用于读取二进制文件

fwrite() 函数用于写入二进制文件