C语言折腾日记

C语言的环境配置

C语言的环境设置主要由文本编译器和C编译器组成

文本编译器:

windows上可以使用notepad

linux上可以使用vim/vi

C语言编译器:

常用的编译器为GCC

win上安装C语言环境

windows上要安装**MinGW**,因为上面会有GCC的安装。在选择安装时要选择gcc-core、gcc-g++、binutils 和 MinGW runtime，但是一般情况下都会安装更多其他的项。

win环境变量设置:
添加您安装的 MinGW 的 bin 子目录到您的 **PATH** 环境变量中，这样您就可以在命令行中通过简单的名称来指定这些工具。

linux上安装C语言环境

linux上安装先检查是否装有GCC

```
$ gcc -v
```

会给出以下反馈

```
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/11/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none:amdgcn-amdhsa
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 11.2.0 (Ubuntu 11.2.0-7ubuntu2)
```

搭建C语言环境

```
$ sudo apt upgrade
$ sudo apt-get install vim
$ sudo apt-get install gcc
$ sudo apt-get install build-essential      #编译程序必须软件包的列表信息
```

第一个C语言程序

```c
#include <stdio.h>                //这是函数头//
int main()                       //main是程序的入口//
{
    float num;
    num = 1;
    printf("Do you like van you see?.\n");
    printf("I just like %f.\n",num);
    getchar();
    num = 2;
    printf("Do you like van you see?.\n");
    printf("I just like %f.\n",num);
    getchar();
    return 0;
    /*我是注释，我是注释adiwhiadkasn*/

}
```

在编辑好之后使用下面命令创造一个可运行的C程序

```
$ gcc -Wall text1.c -o text1
#-o text1 是创建一个名字为text1的C程序

$ ./text1                    #运行该程序
```

一个C程序由函数头和函数组成

这是一个最简单的C程序

```c
#include <stdio.h>
int main()
/*123 */     // 123 //
{
    return 0
}
```

备注一下这个最简单的小程序

1. 程序的第一行 *#include <stdio.h>* 是预处理器指令，告诉 C 编译器在实际编译之前要包含 stdio.h 文件。
2. 下一行 *int main()* 是主函数，程序从这里开始执行。
3. 下一行 */.../* 将会被编译器忽略，这里放置程序的注释内容。它们被称为程序的注释。
4. 下一行 **return 0;** 终止 main() 函数，并返回值 0。

C语言里面的数据类型

最基本的四种类型

序号	类型与描述
1	**基本类型：** 它们是算术类型，包括五种类型：整数类型、浮点类型、字符类型、布尔类型和枚举类型。
2	**指针类型：** 它们也是算术类型，被用来定义在程序中只能赋予其一定的离散整数值的变量。
3	**void 类型（空类型）：** 类型说明符 *void* 表明没有可用的值。
4	**构造类型：** 它们包括：数组类型、结构类型、共用体类型和函数类型。

整型: 原码, 反码, 补码的基础概念和计算方法.

在探求为何机器要使用补码之前, 让我们先了解原码, 反码和补码的概念.对于一个数, 计算机要使用一定的编码方式进行存储. 原码, 反码, 补码是机器存储一个具体数字的编码方式.

1. 原码

原码就是符号位加上真值的绝对值, 即用第一位表示符号, 其余位表示值. 比如如果是8位二进制:

> [+1]原 = 0000 0001
>
> [-1]原 = 1000 0001

第一位是符号位. 因为第一位是符号位, 所以8位二进制数的取值范围就是:

> [1111 1111 , 0111 1111]

即

> [-127 , 127]

原码是人脑最容易理解和计算的表示方式.

2. 反码

反码的表示方法是:

正数的反码是其本身

负数的反码是在其原码的基础上, 符号位不变，其余各个位取反.

> [+1] = [00000001]原 = [00000001]反
>
> [-1] = [10000001]原 = [11111110]反

可见如果一个反码表示的是负数, 人脑无法直观的看出来它的数值. 通常要将其转换成原码再计算.

3. 补码

补码的表示方法是:

正数的补码就是其本身

负数的补码是在其原码的基础上, 符号位不变, 其余各位取反, 最后+1. (即在反码的基础上+1)

> [+1] = [00000001]原 = [00000001]反 = [00000001]补

> [-1] = [10000001]原 = [11111110]反 = [11111111]补

对于负数, 补码表示方式也是人脑无法直观看出其数值的. 通常也需要转换成原码在计算其数值.

浮点数类型的储存原理

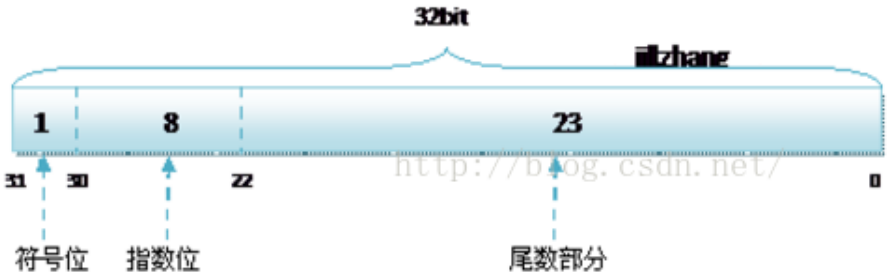
首先c编译系统把浮点型常量都是按双精度处理,分配8个字节(64位)

对于浮点类型的数据采用单精度类型(float)和双精度类型(double)来存储，float数据占用 32bit,double数据占用 64bit.其实不论是float类型还是double类型，在计算机内存中的存储方式都是遵从IEEE的规范的，float 遵从的是IEEE R32.24 ,而double 遵从的是R64.53。

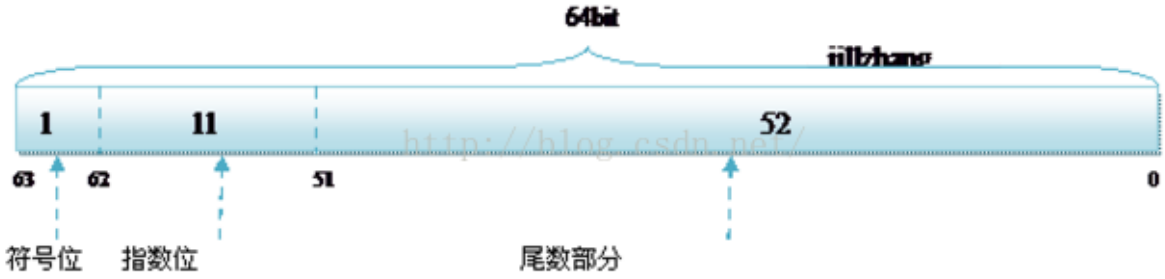
无论是单精度还是双精度，在内存存储中都分为3个部分：

1. 符号位(Sign)：0代表正，1代表为负；
2. 指数位(Exponent)：用于存储科学计数法中的指数数据，并且采用移位存储；
3. 尾数部分(Mantissa)：尾数部分

float数据类型:



double数据类型:



下面就看看8.25和120.5在内存中真正的存储方式：

8.25用二进制的科学计数法表示为:1.0001*2^3 按照上面的存储方式，符号位为0，表示为正；

指数位为3+127=130，位数部分为 1.00001

故8.25的存储方式如下： 0xbffff380: 01000001000001000000000000000000

分解如下：0--10000010--00001000000000000000000

符号位为0，指数部分为10000010，位数部分为 00001000000000000000000

同理，120.5在内存中的存储格式如下： 0xbffff384: 01000010111100010000000000000000

分解如下：0--10000101--11100010000000000000000

总结:浮点数的储存是以指数的方式储存的

基本类型

基本类型包括整数和浮点数,具体的讨论如下:

一个字节储存八位无符号数:范围0~255

类型	存储大小	值范围
char	1 字节	-128 到 127 或 0 到 255
unsigned char	1 字节	0 到 255
signed char	1 字节	-128 到 127
int	2 或 4 字节	-32,768 到 32,767 或 -2,147,483,648 到 2,147,483,647
unsigned int	2 或 4 字节	0 到 65,535 或 0 到 4,294,967,295
short	2 字节	-32,768 到 32,767
unsigned short	2 字节	0 到 65,535
long	4 字节	-2,147,483,648 到 2,147,483,647
unsigned long	4 字节	0 到 4,294,967,295

整数类型	浮点数类型	字符类型	布尔类型	枚举类型
short int	float	char	_Bool	enum
int	double			
long int	long double			
long long int				

不同类型所占的空间不同,根据合适的去选择是最好的

格式说明符号

%d 有符号10进制整数

%i 有符号10进制整数

%o 无符号8进制整数

%u 无符号10进制整数

%x 无符号的16进制数字，并以小写abcdef表示

%X 无符号的16进制数字，并以大写ABCDEF表示

%F/f 浮点数

%E/e 用科学表示格式的浮点数

%g 使用%f和%e表示中的总的位数表示最短的来表示浮点数 G 同g格式，但表示为指数

%c 单个字符

%s 字符串

```
#include <stdio.h>
int main()
{
    char number1 = 1;                                   输出的是asc码里的1
    char number2 = '1';                                 输出的是字符1
    printf("%d, %d \n", number1, number2);
    // unsigned int num = 2,147,483,647//
    printf("%d, %u \n", -123, -123);                    输出的结果根据%d和%u输出的解析方式
不一样,结果不一样
    return 0;
}


mhy@mhy-dxz:~/Learning$ ./a.out
1, 49
-123, 4294967173
```

声明(变量)

声明就是去定义一个变量，这在python中是一样的。

```
        a = 123
        type a
out[1]: int
```

这是在python中去定义一个变量，在python中可以直接给予变量名，赋值和数据类型。

但是这在c/c++中是不可以的

```c
int num;
num = 123;
int num = 123;
```

在c里命名变量要先定义变量的数据类型，再去给变量定义内容。但是变量是可以重新赋值的，同python一样(python同C一样)。

当要进行多个变量声明的时候用','隔开。

```c
int number1, number2;
int number1 = 1, number2 = 2;
```

但是注意变量的声明是将变量值放到内存中的一个地址当中。

C语言中的标识符号:可以用数字,字母,下划线,但是不能以数字为开头。区分大小写

一些标记符号

`signed`

一般默认都会带有符号(就是有正号和负号)

`unsigned`

加上之后就是没有符号

`const`

C99常变量,不能改变其值,但是常变量占用内存

`#define 变量名 内容`

但是宏变量没有地址,只是一种替换

我们将其放在函数头中

一些基本的运算符

杂项运算符 ↦ sizeof & 三元

下表列出了 C 语言支持的其他一些重要的运算符，包括 **sizeof** 和 **? :**。

运算符	描述	实例
sizeof()	返回变量的大小。	sizeof(a) 将返回 4，其中 a 是整数。
&	返回变量的地址。	&a; 将给出变量的实际地址。
*	指向一个变量。	*a; 将指向一个变量。
? :	条件表达式	如果条件为真 ? 则值为 X : 否则值为 Y

算术运算符

下表显示了 C 语言支持的所有算术运算符。假设变量 **A** 的值为 10，变量 **B** 的值为 20，则：

运算符	描述	实例
+	把两个操作数相加	A + B 将得到 30
-	从第一个操作数中减去第二个操作数	A - B 将得到 -10
*	把两个操作数相乘	A * B 将得到 200
/	分子除以分母	B / A 将得到 2
%	取模运算符，整除后的余数	B % A 将得到 0
++	自增运算符，整数值增加 1	A++ 将得到 11
--	自减运算符，整数值减少 1	A-- 将得到 9

关系运算符

下表显示了 C 语言支持的所有关系运算符。假设变量 **A** 的值为 10，变量 **B** 的值为 20，则：

运算符	描述	实例
==	检查两个操作数的值是否相等，如果相等则条件为真。	(A == B) 为假。
!=	检查两个操作数的值是否相等，如果不相等则条件为真。	(A != B) 为真。
>	检查左操作数的值是否大于右操作数的值，如果是则条件为真。	(A > B) 为假。
<	检查左操作数的值是否小于右操作数的值，如果是则条件为真。	(A < B) 为真。
>=	检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。	(A >= B) 为假。
<=	检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。	(A <= B) 为真。

逻辑运算符

下表显示了 C 语言支持的所有关系逻辑运算符。假设变量 **A** 的值为 1，变量 **B** 的值为 0，则：

运算符	描述	实例
&&	称为逻辑与运算符。如果两个操作数都非零，则条件为真。	(A && B) 为假。
||	称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。	(A || B) 为真。
!	称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。	!(A && B) 为真。

位运算符

位运算符作用于位，并逐位执行操作。&、 | 和 ^ 的真值表如下所示：

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

运算符	描述	实例
&	按位与操作，按二进制位进行"与"运算。运算规则：`0&0=0; 0&1=0; 1&0=0; 1&1=1;`	(A & B) 将得到 12，即为 0000 1100
|	按位或运算符，按二进制位进行"或"运算。运算规则：`0|0=0; 0|1=1; 1|0=1; 1|1=1;`	(A | B) 将得到 61，即为 0011 1101
^	异或运算符，按二进制位进行"异或"运算。运算规则：`0^0=0; 0^1=1; 1^0=1; 1^1=0;`	(A ^ B) 将得到 49，即为 0011 0001
~	取反运算符，按二进制位进行"取反"运算。运算规则：`~1=-2; ~0=-1;`	(~A) 将得到 -61，即为 1100 0011，一个有符号二进制数的补码形式。
<<	二进制左移运算符。将一个运算对象的各二进制位全部左移若干位（左边的二进制位丢弃，右边补0）。	A << 2 将得到 240，即为 1111 0000
>>	二进制右移运算符。将一个数的各二进制位全部右移若干位，正数左补0，负数左补1，右边丢弃。	A >> 2 将得到 15，即为 0000 1111

赋值运算符

下表列出了 C 语言支持的赋值运算符：

运算符	描述	实例
=	简单的赋值运算符，把右边操作数的值赋给左边操作数	C = A + B 将把 A + B 的值赋给 C
+=	加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数	C += A 相当于 C = C + A
-=	减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数	C -= A 相当于 C = C - A
*=	乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数	C *= A 相当于 C = C * A
/=	除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数	C /= A 相当于 C = C / A
%=	求模且赋值运算符，求两个操作数的模赋值给左边操作数	C %= A 相当于 C = C % A
<<=	左移且赋值运算符	C <<= 2 等同于 C = C << 2
>>=	右移且赋值运算符	C >>= 2 等同于 C = C >> 2
&=	按位与且赋值运算符	C &= 2 等同于 C = C & 2
^=	按位异或且赋值运算符	C ^= 2 等同于 C = C ^ 2
|=	按位或且赋值运算符	C |= 2 等同于 C = C | 2

C 中的运算符优先级

运算符的优先级确定表达式中项的组合。这会影响到一个表达式如何计算。某些运算符比其他运算符有更高的优先级，例如，乘除运算符具有比加减运算符更高的优先级。

例如 x = 7 + 3 * 2，在这里，x 被赋值为 13，而不是 20，因为运算符 * 具有比 + 更高的优先级，所以首先计算乘法 3*2，然后再加上 7。

下表将按运算符优先级从高到低列出各个运算符，具有较高优先级的运算符出现在表格的上面，具有较低优先级的运算符出现在表格的下面。在表达式中，较高优先级的运算符会优先被计算。

类别	运算符	结合性
后缀	() [] -> . ++ - -	从左到右
一元	+ - ! ~ ++ - - (type)* & sizeof	从右到左
乘除	* / %	从左到右
加减	+ -	从左到右
移位	<< >>	从左到右
关系	< <= > >=	从左到右
相等	== !=	从左到右
位与 AND	&	从左到右
位异或 XOR	^	从左到右
位或 OR	|	从左到右
逻辑与 AND	&&	从左到右
逻辑或 OR	||	从左到右
条件	?:	从右到左
赋值	= += -= *= /= %=>>= <<= &= ^= |=	从右到左
逗号	,	从左到右

一些基本函数

`printf()`

将文字打印在屏幕上,同python里面的print();

`scanf()`

用户输入信息,将信息传输给程序,同python里面的input();

`getchar()`

等待用户的操作

`main()`

主函数,程序的入口

C语言里面的杂项

注释

```
/* */
```

对的，这是一个注释，里面什么都可以输入，这种注释的好处是可以用到多行。

```
// //
```

这也是一种注释，但是这种注释只能用于单行。

debug

debug(调试)是找出并修正错误的过程。

debugger

debugger(调试器)是用来调试debug的程序。

32位和64位

从程序上说：32位与64位程序，是指经过语言编译后的可执行文件，比如 C 语言编写的程序就需要区分是32位的还是64位

从系统和硬件上讲：CPU一次处理数据的能力是32位还是64位，关系着系统需要安装32位还是64位的系统

32 位和 64 位中的“位”，也叫字长，是 CPU 通用寄存器的数据宽度，是数据传递和处理的基本单位。字长是 CPU 的主要技术指标之一，指的是 CPU 一次能并行处理的二进制位数，字长总是8的整数倍

ASCII

ASCII码表

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
0	NUL	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{undefined
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

ASCII 字符代码表 一

高四位 / 低四位		ASCII非打印控制字符 0000 0					ASCII非打印控制字符 0001 1					ASCII 打印字符 0010 2		ASCII 打印字符 0011 3		ASCII 打印字符 0100 4		ASCII 打印字符 0101 5		ASCII 打印字符 0110 6		ASCII 打印字符 0111 7		
		十进制	字符	ctrl	代码	字符解释	十进制	字符	ctrl	代码	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	ctrl
0000	0	0	BLANK NULL	^@	NUL	空	16	▶	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH	头标开始	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	反确认	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK	确认	22	▬	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO	移出	30	▲	^6	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	Δ	Back Space

注：表中的ASCII字符可以用：ALT + "小键盘上的数字键" 输入

ASCII 字符代码表 二

高四位 / 低四位		扩充ASCII码字符集															
		1000		1001		1010		1011		1100		1101		1110		1111	
		8		9		A/10		B/16		C/32		D/48		E/64		F/80	
		+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符
0000	0	128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
0001	1	129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
0010	2	130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
0011	3	131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
0100	4	132	ä	148	ö	164	ñ	180	┤	196	─	212	Ô	228	Σ	244	⌠
0101	5	133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
0110	6	134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
0111	7	135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
1000	8	136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
1001	9	137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	•
1010	A	138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
1011	B	139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
1100	C	140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	n
1101	D	141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
1110	E	142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
1111	F	143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	BLANK FF

注：表中的ASCII字符可以用：ALT + "小键盘上的数字键" 输入