

知识点1【第一个c语言程序】（了解）

知识点2【C语言的关键字】（了解）

1、数据类型相关的关键字12

2、存储相关关键字5

3、控制语句相关的关键字11

4、其他关键字3

知识点3【数据类型关键字】（了解）

1、常量和变量

2、整型

整型常量：10

整型变量的定义：

变量的初始化：

变量的使用：读（取值） 写（赋值）

变量的声明：先使用 后定义 必须事先对变量进行声明

获取键盘输入：scanf获取键盘输入函数

键盘输入两个数

键盘输入两个数求出最大值：

3、字符类型 char

字符变量：

字符的大小写转换：

3、实型（浮点数）

1、实型常量 3.14 3.14f

2、实型变量

知识点4【有符号数和无符号数】（了解）

有符号数：

无符号数：

定义有符号数的方式

输出有符号数：

定义无符号数的方式

输出无符号数：

知识点5【二进制、八进制、十进制、十六进制】（了解推展）

进制概述

十进制 转二进制、八进制、十六进制（短除法）

案例1:123转成二进制

案例2:123转成八进制--->0173

案例3:123转成十六进制--->0x7b

二进制、八进制、十六进制转十进制（位次幂）

案例1：将二进制数 1100 0011 转换成十进制

案例2：将 0123 转换成十进制 --->83

案例3：将 0x12 转换成十进制 --->18

二进制转八进制：

二进制转十六进制：

八进制 转 二进制

十六进制 转 二进制

八进制 转 十六进制（没有直接方式）

十六进制 转 八进制（没有直接方式）

注意：不同的进制 仅仅是数据的不同表现形式而已

知识点6【原码、反码、补码】（了解推展）

1、原码、反码、补码概述

2、补码意义：

1、统一了0的编码

2、将减法运算变加法运算

知识点7【计算机对数据的存储】（了解推展）

知识点8【计算机对数据的取】（了解推展）

总结：

有符号数、并有符号提取、且内存最高位为1 将内存数据 求补码 输出 ， 其他数据提取都是内存原样输出。

知识点9【其他关键字】（了解推展）

1、const修饰变量为只读

2、register修饰寄存器变量

2、volatile关键字

1、强制访问内存

2、防止编译器优化

3、sizeof测量类型的大小

4、typedef给已有的类型重新取个别名

案例1：给int arr[5]取个别名

案例2：给int *取个别名

知识点10【转义字符】（了解推展）

1、\和某些字符 结合 产生新的字符含义 就叫转义字符

2、八进制转义

知识点1 【第一个c语言程序】（了解）

```
//这是行注释
/*
这是块注释
块注释不能嵌套
*/

//头文件 是对 库函数 进行声明
#include <stdio.h>//标准输入输出头文件

//main 就是主函数 一个工程有且只能有一个主函数
//main 是程序调用的入口
//main左边的int 为函数返回值类型 （）为函数的形参
//形参：函数外部数据 传递到 函数内部 的桥梁
int main(int argc, char *argv[])
{
    //函数体
    //printf将()里面的字符串输出到 终端上
    printf("hello world\n");//分号;是结束语句

    //return 第一：返回值函数值 第二：结束当前函数
    return 0;
}
```

其他非main函数 可以多个

知识点2 【C语言的关键字】（了解）

1、数据类型相关的关键字

```
1 char 、 short、 int 、 long 、 float、 double、
2 struct、 union、 enum 、 signed、 unsigned、 void
```

计算机最小的存储单位为 二进制

计算机最小的分配单位为 字节

二进制：没位只能存放0或1 b

8位二进制位 == 1字节 B

```
1 0000 0000 ~ 1111 1111
2 0 ~ 255
```

1B == 8b

1KB = 1024B
1MB = 1024KB
1GB = 1024MB
1TB = 1024GB
1PB = 1024TB
1EB = 1024PB

-----在32位平台-----

char 字符类型 1字节
short 短整型 2字节
int 整型 4字节
long 长整型 4字节
float 单精度浮点数 4字节
double 双精度浮点数 8字节
struct 结构体 union 共用体 enum枚举 signed有符号数 unsigned无符号数
void 空类型

2、存储相关关键字5

register、static、const、auto、extern
register寄存器变量
static 静态变量
const只读变量
auto自动变量
extern声明变量外部可用（告知编译器 该变量在其他源文件定义 此处请通过编译）

3、控制语句相关的关键字11

if、else、break、continue、for、while、do、switch case goto、default
if else switch case default条件控制语句
for while break continue goto 循环控制语句

4、其他关键字3

sizeof、typedef、volatile
sizeof测量类型的大小
typedef给类型取别名
volatile防止编译器优化

知识点3 【数据类型关键字】（了解）

```
1 char、short、int、long、float、double、
2 struct、union、enum、signed、unsigned、void
```

1、常量和变量

常量：既见既所得 它的值不能修改

```
1 'a' '1' 10 "hello" 3.14f 3.14
```

```
edu@edu:~/work/c/day01$ gcc 01_code.c
edu@edu:~/work/c/day01$ ./a.out
a
10
10
10
3.140000
3.140000
hello world
edu@edu:~/work/c/day01$
```

```
#include <stdio.h>
void test01()
{
    /* '\n' 换行 刷新缓冲区 */
    printf("%c\n", 'a');
    printf("%d\n", 10);
    printf("%hd\n", 10);
    printf("%ld\n", 10L);
    printf("%f\n", 3.14f);
    printf("%lf\n", 3.14);
    printf("%s\n", "hello world");
}
int main(int argc, char *argv[])
{
    test01();
    return 0;
}
```

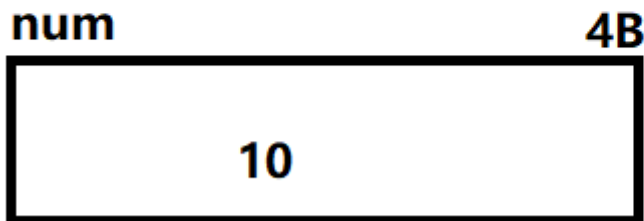
变量：系统会自动根据变量的类型大小 为变量开辟内存空间

变量的值 可以被修改

```
1 int num = 10;
2 void num=10; //系统无法给num开辟空间（无法判断void的大小） 所以定义失败
```

```
int num = 10;
```

变量名 代表 空间内容



记住：变量名的 命名规则

变量由字母、数值、下划线组成 不能以数值开头。不能是关键字

- 1 以下错误选项是：C D
- 2 A:int num; B:int _num2; C:int 2_num; D:int auto;

2、整型

整型常量：10

整型变量的定义：

```
int data; //在函数外定义的变量 全局变量
void test02()
{
    // {} 复合语句 在复合语句中定义的变量为 局部变量
    // 如果局部变量不初始化 内容不确定
    int num;

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

edu@edu: ~/work/c/day01

```
edu@edu:~/work/c/day01$ gcc 01_code.c
edu@edu:~/work/c/day01$ ./a.out
num = 0
edu@edu:~/work/c/day01$ _
```

- 1 以下代码循环几次__不确定____
- 2 void test01()
- 3 {
- 4 int i;
- 5 for(; i<10; i++)
- 6 {
- 7 ;
- 8 }
- 9 }

变量的初始化：

定义变量的时候 给变量赋值 叫初始化。

- 1 int num = 10; //是初始化
- 2 int data;
- 3 data=10; //不是初始化

一般变量都是初始化为0

```
1 int num = 0;
```

变量的使用：读（取值） 写（赋值）

```
int data; //在函数外定义的变量 全局变量
void test02()
{
    // {} 复合语句 在复合语句中定义的变量为 局部变量
    // 如果局部变量不初始化 内容不确定
    int num = 0;

    printf("num = %d\n", num); //读操作

    int data = num; //num读 data写

    data = 100; //data 写
    if(data > 100); //data 读

    data++; //data = data+1
}
```

变量的声明：先使用 后定义 必须事先对变量进行声明

```
//变量声明 extern显示声明 不会给变量开辟空间 声明的时候不要给变量赋值
extern int data2;
void test03()
{
    //变量的使用
    printf("data2 = %d\n", data2);
}

//变量定义 给data2开辟4字节空间 data2代表这4字节空间的内容
int data2 = 0;
```

```
edu@edu: ~/work/c/day01
edu@edu: ~/work/c/day01$ gcc 01_code.c
edu@edu: ~/work/c/day01$ ./a.out
data2 = 0
edu@edu: ~/work/c/day01$
```

大家思考：变量的定义、变量的声明、变量的使用 三者关系

变量

获取键盘输入：scanf获取键盘输入函数

```
1 #include <stdio.h>
2 int scanf(const char *format, ...);
3 format: 获取数据的格式
```

```
void test04()
{
    int num = 0;

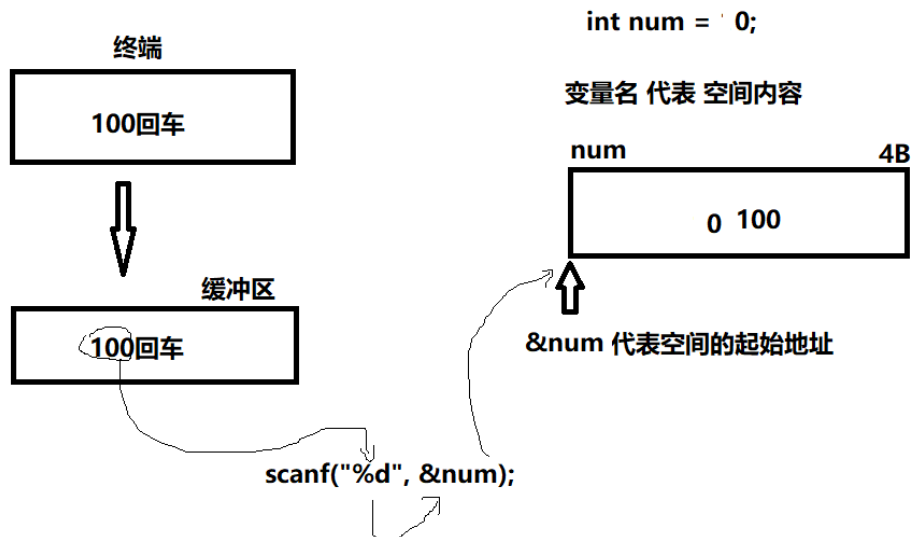
    printf("请输入一个int数据:");
    scanf("%d", &num);

    printf("num = %d\n", num);
}

int main(int argc, char *argv[])
{
    test04();
    return 0;
}
```

```
edu@edu: ~/work/c/day01
edu@edu: ~/work/c/day01$ gcc 01_code.c
edu@edu: ~/work/c/day01$ ./a.out
请输入一个int数据:100
num = 100
edu@edu: ~/work/c/day01$
```

scanf和%d结合 只能提取数据 非数值 立即停止提取



键盘输入两个数

```
void test05()
{
    //int data1=0;
    //int data2=0;
    int data1=0, data2=0;

    printf("请输入两个int数据:");
    scanf("%d %d", &data1, &data2);
    printf("data1=%d, data2=%d\n", data1, data2);
}

int main(int argc, char *argv[])
{
    test05();

    return 0;
}
```

```
edu@edu: ~/work/c/day01
edu@edu: ~/work/c/day01$ gcc 01_code.c
edu@edu: ~/work/c/day01$ ./a.out
请输入两个int数据:100 200
data1=100, data2=200
edu@edu: ~/work/c/day01$
```

键盘输入两个数求出最大值:

```
void test05()
{
    //int data1=0;
    //int data2=0;
    int data1=0, data2=0;

    printf("请输入两个int数据:");
    scanf("%d %d", &data1, &data2);

    if(data1 > data2)
    {
        printf("最大值为:%d\n", data1);
    }
    else
    {
        printf("最大值为:%d\n", data2);
    }
}
```

```
edu@edu: ~/work/c/day01
edu@edu: ~/work/c/day01$ gcc 01_code.c
edu@edu: ~/work/c/day01$ ./a.out
请输入两个int数据:100 200
最大值为:200
edu@edu: ~/work/c/day01$
```

过抄仿改调看练创悟

3、字符类型 char

字符串常量 'a' '0' '2'

单引号 只能作用一个字符，转义字符除外。

1 A: 'a' B: '#' C: '12' 错误 D: '2'

单引号:

第一个作用: 'a' 描述a为字符

第二个作用: 取字符的ASCII值

ASCII表

编码	字符	编码	字符	编码	字符	编码	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	TAB	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	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	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	ETB	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	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

printf %c 输出的是字符

printf %d输出的是字符的ASCII值

```
#include <stdio.h>
void test01()
{
    printf("%c\n", 'a');
    printf("%d\n", 'a');
}
int main(int argc, char *argv[])
{
    test01();
    return 0;
}
```

```
edu@edu: ~/work/c/day01
edu@edu: ~/work/c/day01$ gcc 02_code.c
edu@edu: ~/work/c/day01$ ./a.out
a
97
edu@edu: ~/work/c/day01$
```

字符变量:

```
1 char ch;
2 char ch='a';
3 char ch='\0';// '\0' 的ASCII 0
```

```
#include <stdio.h>
void test01()
{
    printf("%c\n", '\0');
    printf("%d\n", '\0');
}
int main(int argc, char *argv[])
{
    test01();
    return 0;
}
```

```
edu@edu: ~/work/c/day01
edu@edu: ~/work/c/day01$ gcc 02_code.c
edu@edu: ~/work/c/day01$ ./a.out
0
edu@edu: ~/work/c/day01$
```

谈谈'0'和'\0'的区别?

'0'字符串0 计算机存的ASCII 48

'\0' 计算机存的ASCII 0

```
void test02()
{
    char ch='\0';

    printf("请输入第一个字符:");
    //获取字符的第一种方式: scanf %c
    //%c只能提取一个字符
    scanf("%c", &ch);
    //去掉输入的回车
    getchar();

    printf("ch = %c\n", ch);
    printf("ch = %d\n", ch);

    //getchar 获取字符
    printf("请输入第二个字符:");
    ch = getchar();
    printf("ch = %c\n", ch);
    printf("ch = %d\n", ch);
}
int main(int argc, char *argv[])
{
    test02();
    return 0;
}
```

```
edu@edu: ~/work/c/day01
edu@edu: ~/work/c/day01$ gcc 02_code.c
edu@edu: ~/work/c/day01$ ./a.out
请输入第一个字符:A
ch = A
ch = 65
请输入第二个字符:N
ch = N
ch = 78
edu@edu: ~/work/c/day01$
```

字符的大小写转换:

键盘输入一个字符, 如果是大写 就转换成小写 如果是小写就转换成大写

```
1 'a' 97 'b' 98 'c' 99 ..... 'z' 122
2 'A' 65 'B' 66 'c' 67 ..... 'z' 90
3
4 char ch = 'a';
5 ch = ch - ('a' - 'A');
6
7 char ch = 'A';
8 ch = ch + ('a' - 'A');
```

```
void test03()
{
    char ch = '\0';

    printf("请输入一个字符:");
    ch = getchar();

    if(ch >= 'a' && ch <= 'z')
    {
        //ch = ch - ('a' - 'A');
        ch -= ('a' - 'A');
    }
    else if(ch >= 'A' && ch <= 'Z')
    {
        //ch = ch + ('a' - 'A');
        ch += ('a' - 'A');
    }

    printf("转换后的结果:%c\n", ch);
}

int main(int argc, char *argv[])
{
    test03();
    return 0;
}
```

```
edu@edu: ~/work/c/day01
edu@edu: ~/work/c/day01$ gcc 02_code.c
edu@edu: ~/work/c/day01$ ./a.out
请输入一个字符:1
转换后的结果:1
edu@edu: ~/work/c/day01$ ./a.out
请输入一个字符:A
转换后的结果:a
edu@edu: ~/work/c/day01$ ./a.out
请输入一个字符:h
转换后的结果:H
edu@edu: ~/work/c/day01$
```

3、实型 (浮点数)

单精度浮点数 (float) 双精度浮点数 double

1、实型常量 3.14 3.14f

不以结尾的实型常量为 double 类型

以结尾的实型常量为 float 类型

```
2
3 #include <stdio.h>
4 void test01()
5 {
6     printf("%lu\n", sizeof(3.14));
7     printf("%lu\n", sizeof(3.14f));
8 }
9 int main(int argc, char *argv[])
10 {
11     test01();
12     return 0;
13 }
14
```

```
edu@edu: ~/work/c/day01
edu@edu: ~/work/c/day01$ gcc 03_code.c
edu@edu: ~/work/c/day01$ ./a.out
8
4
edu@edu: ~/work/c/day01$
```

指数形式: 123e3 代表 123×10^3 123e-3

2、实型变量

```
void test02()
{
    //号两边类型必须匹配
    //float f=3.14; //3.14为double类型 而f为float类型 所以类型不匹配
    float f = 0.0f;

    printf("请输入一个浮点数:");
    scanf("%f", &f);

    printf("f = %f\n", f);

    double d = 0.0;
    printf("请输入一个浮点数:");
    scanf("%lf", &d);
    printf("d = %lf\n", d);
}
```

```
edu@edu: ~/work/c/day01
edu@edu: ~/work/c/day01$ gcc 03_code.c
edu@edu: ~/work/c/day01$ ./a.out
请输入一个浮点数:3.14
f = 3.140000
请输入一个浮点数:5.12
d = 5.120000
edu@edu: ~/work/c/day01$
```

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

知识点4【有符号数和无符号数】（了解）

有符号数:

数据二进制的最高位为符号位 其他位为数据位。

最高位为1 表示负数

最高位为0 表示正数

以一字节为例: xddd dddd

1111 1111~1000 0000~0000 0000~0111 1111

-127 ~-0 ~+0 ~+127

将-0看成-128

-128~127

无符号数:

没有符号位 所有二进制位都是数据位

0000 0000 ~1111 1111

0 ~ 255

定义有符号数的方式

- 1 //方式一: 默认方式(推荐)
- 2 int num; //num为有符号数
- 3 //方式二: 使用关键字signed显示说明
- 4 signed int num;

输出有符号数:

```
1 %d输出有符号int
2 %hd输出有符号short
3 %ld输出有符号long
```

定义无号数的方式

```
1 unsigned int num;
```

输出无符号数：

```
1 %u输出有符号unsigned int
2 %hu输出有符号unsigned short
3 %lu输出有符号unsigned long
```

知识点5【二进制、八进制、十进制、十六进制】 (了解推展)

进制概述

二进制：0~1

c语言 不能直接输出二进制

八进制：0~7

八进制：以0开头 0123 %o输出八进制 不区分正负数

十进制：0~9

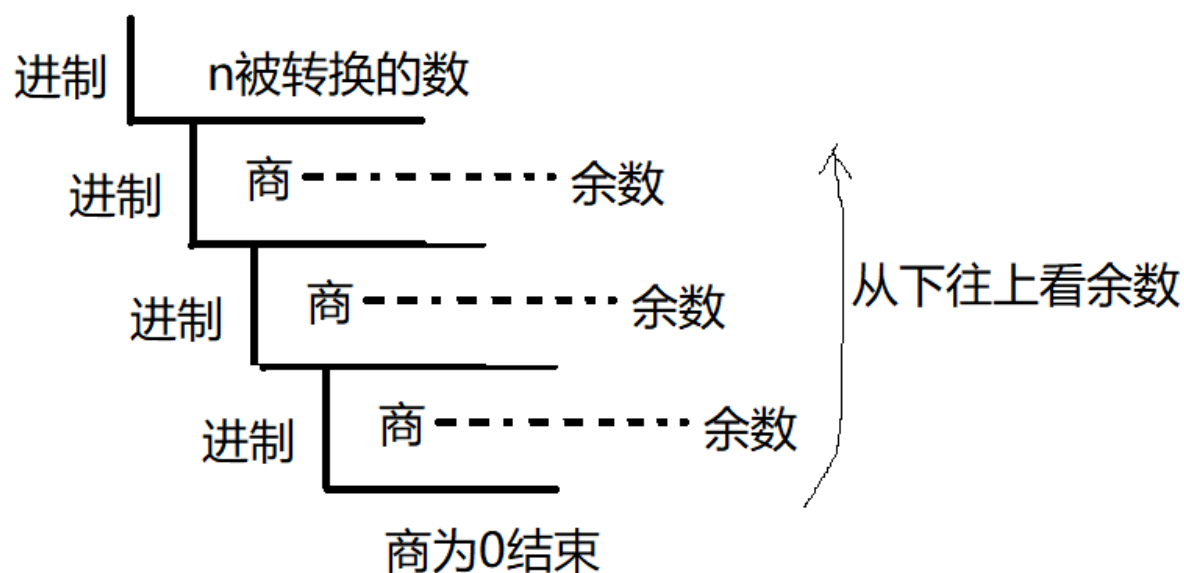
十进制：123 %d %ld %hd %u %lu %hu

十六进制：0~9 a~f

十六进制：以0x12 %x输出十六进制 不区分正负数

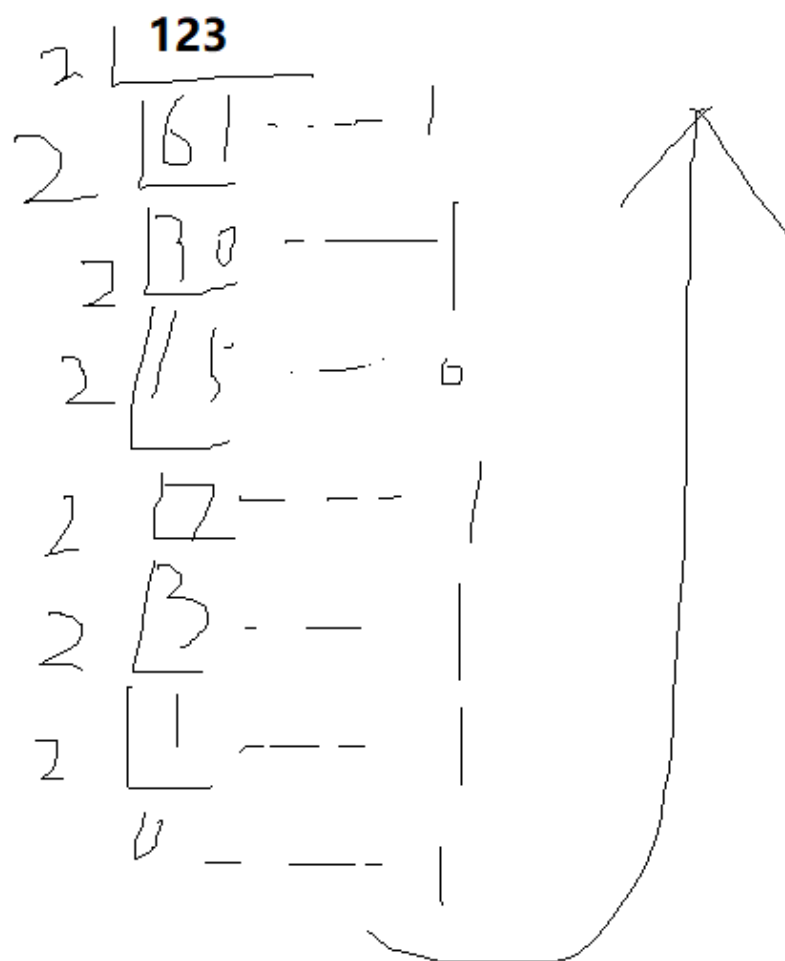
n进制：0~n-1

十进制 转二进制、八进制、十六进制（短除法）



案例1:123转成二进制

123--->0111 1011

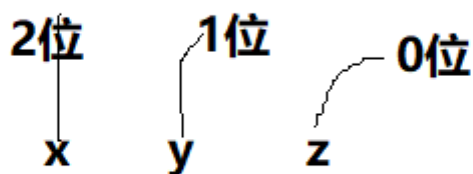


案例2:123转成八进制--->0173

案例3:123转成十六进制--->0x7b

二进制、八进制、十六进制转十进制（位次幂）

将n进制数xyz转换成 十进制



$$x*n^2+y*n^1+z*n^0$$

案例1：将二进制数 1100 0011 转换成十进制

$$1*2^7+1*2^6+1*2^1+1*2^0=195$$

案例2：将 0123 转换成十进制 --->83

案例3：将 0x12 转换成十进制 --->18

二进制转八进制：

从右往左：每3位二进制 对应 1位八进制

1101 1010 ---->0332

11 011 010

3 3 2

二进制转十六进制：

从右往左：每4位二进制 对应 1位十六进制

101 1010 ---->0xda

1101 1010

d a

0x123456-->占3字节

八进制 转 二进制

1位八进制 对应 3位二进制

0123---> 001 010 011

十六进制 转 二进制

1位十六进制 对应 4位二进制

0x1d3c--->0001 1101 0011 1100

八进制 转 十六进制（没有直接方式）

八进制 ----> 二进制 ----> 十六进制

十六进制 转 八进制（没有直接方式）

十六进制 ----> 二进制 ----> 八进制

案例：0x123---->(八进制)0443

注意：不同的进制 仅仅是数据的不同表现形式而已

```
void test03()
{
    //
    int num = 123;
    printf("%#o\n", num);
    printf("%d\n", num);
    printf("%#x\n", num);
}

int main(int argc, char *argv[])
{
    test03();
    return 0;
}
```

edu@edu: ~/work/c/day01

```
edu@edu:~/work/c/day01$ gcc 03_code.c
edu@edu:~/work/c/day01$ ./a.out
0173
123
0x7b
edu@edu:~/work/c/day01$ _
```

知识点6【原码、反码、补码】（了解推展）

1、原码、反码、补码概述

计算机存储的是数据的**补码**。

原码：数据的二进制形式

123:原码0111 1011

无符号数：

补码==反码==原码

123原码：0111 1011

123反码：0111 1011

123补码：0111 1011

有符号数：

正数：

补码==反码==原码

+123原码：0111 1011

+123反码：0111 1011

+123补码：0111 1011

负数：

反码==原码**符号位**不变 其他位按位**取反**。

补码==反码+1

-123原码: 1111 1011

-123反码: 1000 0100

-123补码: 1000 0101

负数在计算机以补码的方式存储

非负数在计算机以原码的方式存储

2、补码意义:

1、统一了0的编码

1 +0补码: 0000 0000

2 -0补码: 0000 0000

2、将减法运算变加法运算

假如: 没有补码 10-6

```
1 10: 0000 1010
2 -6: 1000 0110
3 -----
4 1001 0000---->-16结果有问题
```

有补码 10-6

```
1 10: 0000 1010
2 -6: 1111 1010
3 -----
4 0000 0100---->4
```

知识点7 【计算机对数据的存储】（了解推展）

负数在计算机以补码的方式存储

非负数在计算机以原码的方式存储

八进制数 以原码存储

十六进制 以原码存储

以十六进制查看内存数据存储情况。

```
void test04()
{
    char ch1=-10;
    printf("ch1=%#x\n", ch1);

    char ch2=6;
    printf("ch2=%#x\n", ch2);
}

int main(int argc, char* argv[])
{
    test04();
    return 0;
}
```

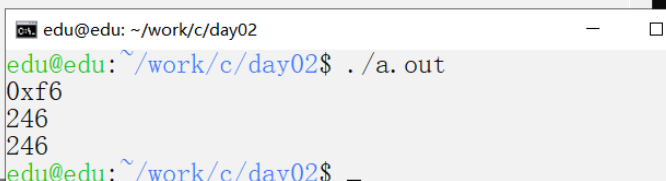
```
edu@edu: ~/work/c/day01
edu@edu:~/work/c/day01$ gcc 03_code.c
edu@edu:~/work/c/day01$ ./a.out
ch1=0xffffffff6
ch2=0x6
edu@edu:~/work/c/day01$ _
```

知识点8 【计算机对数据的取】（了解推展）

如果是对 **无符号变量** 进行取值：

不管是有符号提取(%d %hd %ld) 还是无符号提取(%u %hu %lu %o %x) 都是输出**内存的原样**数据。

```
1 // 00_code.c
2
3 #include <stdio.h>
4 void test01()
5 {
6     //-10的无符号数 就是-10的补码
7     unsigned char ch = -10; //原码1000 1010 反码1111 0101 补码 1111 0110=0xf6
8
9     printf("%#x\n", ch); //0xf6
10    printf("%d\n", ch); //246=0xf6
11    printf("%u\n", ch); //246=0xf6
12 }
13
14 int main(int argc, char *argv[])
15 {
16     test01();
17 }
```



```
edu@edu: ~/work/c/day02
edu@edu: ~/work/c/day02$ ./a.out
0xf6
246
246
edu@edu: ~/work/c/day02$
```

如果是对 **有符号变量** 进行取值：

系统会去看内存的最高位，如果最高位为0 表明**正数**，（有符号或无符号输出）都是**内存原样** 输出

```
1 // 00_code.c
2
3 #include <stdio.h>
4 void test01()
5 {
6     //-10的无符号数 就是-10的补码
7     char ch = 10; //原码0000 1010
8
9     printf("%#x\n", ch); //0xa
10    printf("%d\n", ch); //10
11    printf("%u\n", ch); //10
12 }
13
14 int main(int argc, char *argv[])
15 {
16     test01();
17 }
```



```
edu@edu: ~/work/c/day02
edu@edu: ~/work/c/day02$ gcc 00_code.c
edu@edu: ~/work/c/day02$ ./a.out
0xa
10
10
edu@edu: ~/work/c/day02$
```

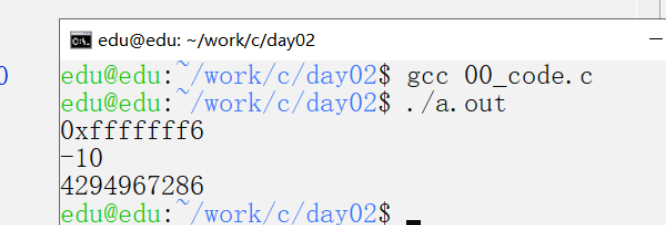
系统会去看内存的最高位，如果最高位为1 表明**负数**，无符号输出 (%u %lu %hu %o %x) 都是 **内存原样** 输出, 有符号输出 (%d %hd %ld) ,将**内存数据求补码**（得到原码）输出。

```
1 // 00_code.c
2
3 #include <stdio.h>
4 void test01()
5 {
6     //-10的无符号数 就是-10的补码
7     char ch = -10; //补码1111 0110
8
9     printf("%#x\n", ch); //0xffffffff6
10    printf("%d\n", ch); //-10
11    printf("%u\n", ch); //4294967286
12 }
13
14 int main(int argc, char *argv[])
15 {
16     test01();
17 }
```



```
edu@edu: ~/work/c/day02
edu@edu: ~/work/c/day02$ gcc 00_code.c
edu@edu: ~/work/c/day02$ ./a.out
0xffffffff6
-10
4294967286
edu@edu: ~/work/c/day02$
```

```
1 // 00_code.c
2
3 #include <stdio.h>
4 void test01()
5 {
6     //-10的无符号数 就是-10的补码
7     char ch = 0xf6; //内存数据 1111 0110
8
9     printf("%#x\n", ch); //0xffffffff6
10    printf("%d\n", ch); //-10
11    printf("%u\n", ch); //4294967286
12 }
13
14 int main(int argc, char *argv[])
15 {
16     test01();
17 }
```



```
edu@edu: ~/work/c/day02
edu@edu: ~/work/c/day02$ gcc 00_code.c
edu@edu: ~/work/c/day02$ ./a.out
0xffffffff6
-10
4294967286
edu@edu: ~/work/c/day02$
```

```
#include <stdio.h>
void test01()
{
    //-10的无符号数 就是-10的补码
    char ch = 0xf6; //内存数据 1111 0110

    printf("%#x\n", ch); //0xffffffff6
    printf("%d\n", ch); //
    printf("%u\n", ch); //
}
```

```
edu@edu: ~/work/c/day02
edu@edu: ~/work/c/day02$ gcc 00_code.c
edu@edu: ~/work/c/day02$ ./a.out
0xffffffff6
-10
4294967286
edu@edu: ~/work/c/day02$
```

总结:

有符号数、并有符号提取、且内存最高位为1 将内存数据 求补码 输出 , 其他数据提取都是内存原样输出。

对无符号数据 提取 都是内存原样输出

对有符号数据 无符号提取 是内存原样输出

对有符号数、内存最高位为0 有符号提取 是内存原样输出

知识点9【其他关键字】（了解推展）

1、const修饰变量为只读

```
void test02()
{
    //const修饰num为只读变量 本质是变量
    //只读变量 只能初始化不能被赋值
    const int num = 100;
    num = 200; //error
}
```

2、register修饰寄存器变量

如果变量 别高频繁使用 会自动将变量存储在寄存器中 目的: 提高访问效率

如果用户想将变量 直接放入寄存器中 可以加register修饰

```
1 register int num = 10; //num 将放入寄存器
```

```
void test03()
{
    //将num放入寄存器中 提高访问效率 num也叫寄存器变量
    register int num = 10;

    //不能对寄存器变量 取地址 &num是错误的
    &num; //&取的是内存地址 而num可能在寄存器中 所以取地址失败

    //register修饰的变量 只是尽量放入寄存器中
}
```

2、volatile关键字

1、强制访问内存

```
1 volatile int num = 10; //对num的访问 必须从内存访问
```

2、防止编译器优化

```
1 volatile int num=10;
2 num=11;
3 num=12;
4 num=13;
5 num=14;
6
7 printf("num = %d\n", num);
```

3、sizeof测量类型的大小

```
void test04()
{
    int num=0;
    printf("%lu\n", sizeof(num));
    printf("%lu\n", sizeof(int));

    printf("%lu\n", sizeof(short));
    printf("%lu\n", sizeof(long));
    char ch='a';
    printf("%lu\n", sizeof(ch));
    printf("%lu\n", sizeof('a'));
}
int main(int argc, char *argv[])
```

```
edu@edu: ~/work/c/day02
edu@edu: ~/work/c/day02$ gcc 00_code.c
edu@edu: ~/work/c/day02$ ./a.out
4
4
2
8
1
4
edu@edu: ~/work/c/day02$ _
```

4、typedef给已有的类型重新取个别名

不能创建新类型。

将长且复杂的类型名 取一个短小的名称。

typedef作用的步骤：

- 1、先用 已有的类型 定义一个普通的变量
- 2、用别名 替换 变量名
- 3、在整个表达式最前方 加typedef

```
void test05()
{
    //INT32 是int的别名
    typedef int INT32;
    int num; //已有的类型任然有效
    INT32 data; //int data
}
```

```
edu@edu: ~/work/c/day02
edu@edu: ~/work/c/day02$ gcc 00_code.c
edu@edu: ~/work/c/day02$ ./a.out
edu@edu: ~/work/c/day02$
```

案例1：给int arr[5]取个别名

```
1 typedef int MY_ARRAY[5];
2 MY_ARRAY arr;
```

```
void test06()
{
    typedef int MY_ARRAY[5];
    MY_ARRAY arr={1, 2, 3, 4, 5};
    int i=0;
    for(i=0;i<5;i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

```
edu@edu: ~/work/c/day02
edu@edu:~/work/c/day02$ gcc 00_code.c
edu@edu:~/work/c/day02$ ./a.out
1 2 3 4 5
edu@edu:~/work/c/day02$
```

案例2：给int *取个别名

```
1 typedef int *POINTER;//POINTER == int *
2 POINTER p;//int *p;
```

知识点10 【转义字符】（了解推展）

1、\和某些字符 结合 产生新的字符含义 就叫转义字符

```
1 '\0' == ASCII 为0
2 '\n' == 换行符
3 '\t' == tab缩进符
4 '\r' ==回到行首符号
5 '\a' ==发出警报
```

```
void test07()
{
    printf("###c###\n", '\n');
    printf("###c###\n", '\t');
    printf("###c###\n", '\r');
    printf("###c###\n", '\a');
}
```

```
edu@edu: ~/work/c/day02
edu@edu:~/work/c/day02$ gcc 00_code.c
edu@edu:~/work/c/day02$ ./a.out
###
###
###    ##
###
#####
```

2、八进制转义

'\ddd' 每个d的范围必须是0~7 3个d表示最多识别3位八进制数据

以下字符描述正确的是__A__

```
1 A: '\123' B: '\18' C: '\1234' D: '\183'
```

3、十六进制转义

'\xhh' 每个h的范围0~9 a~f 2个h表示最多识别2位十六进制

以下字符描述正确的是__BD__

```
1 A: '\af' B: '\123' C: '\x3df' D: '\xab'
```

mspaint calc