

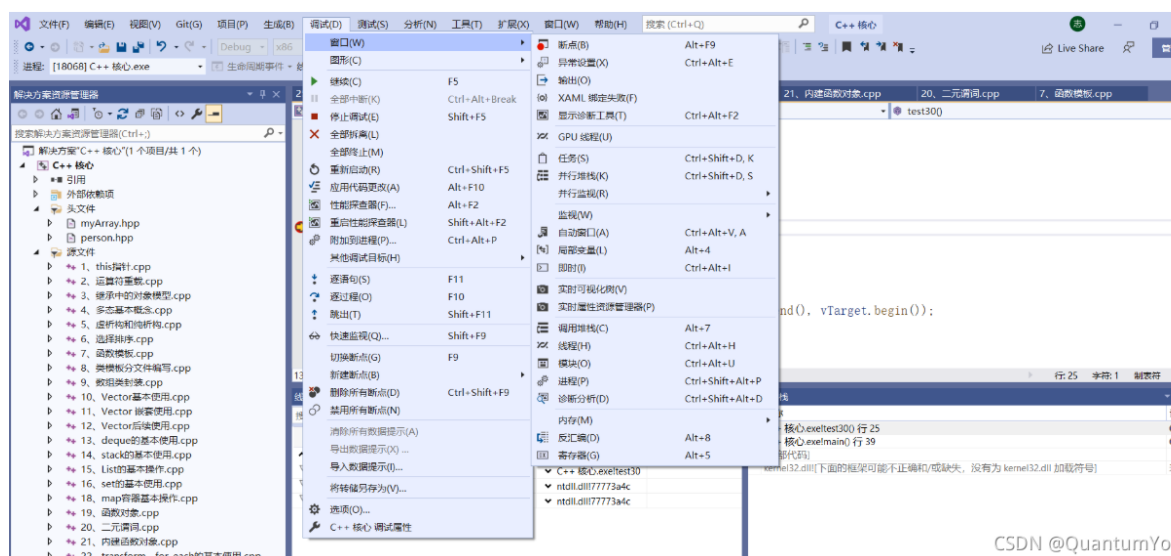
> 宽广 深刻 精确 自律 自信 自强

@[toc]

# Day01 开发环境搭建及调试窗口设置

## VS2019 编译器使用小技巧

- 关于后缀名 `.cpp` 代表C++ 文件(向下兼容C) ； `.c` 代表C文件
- `system("pause")` ； 暂留小黑框



- obj 链接程序

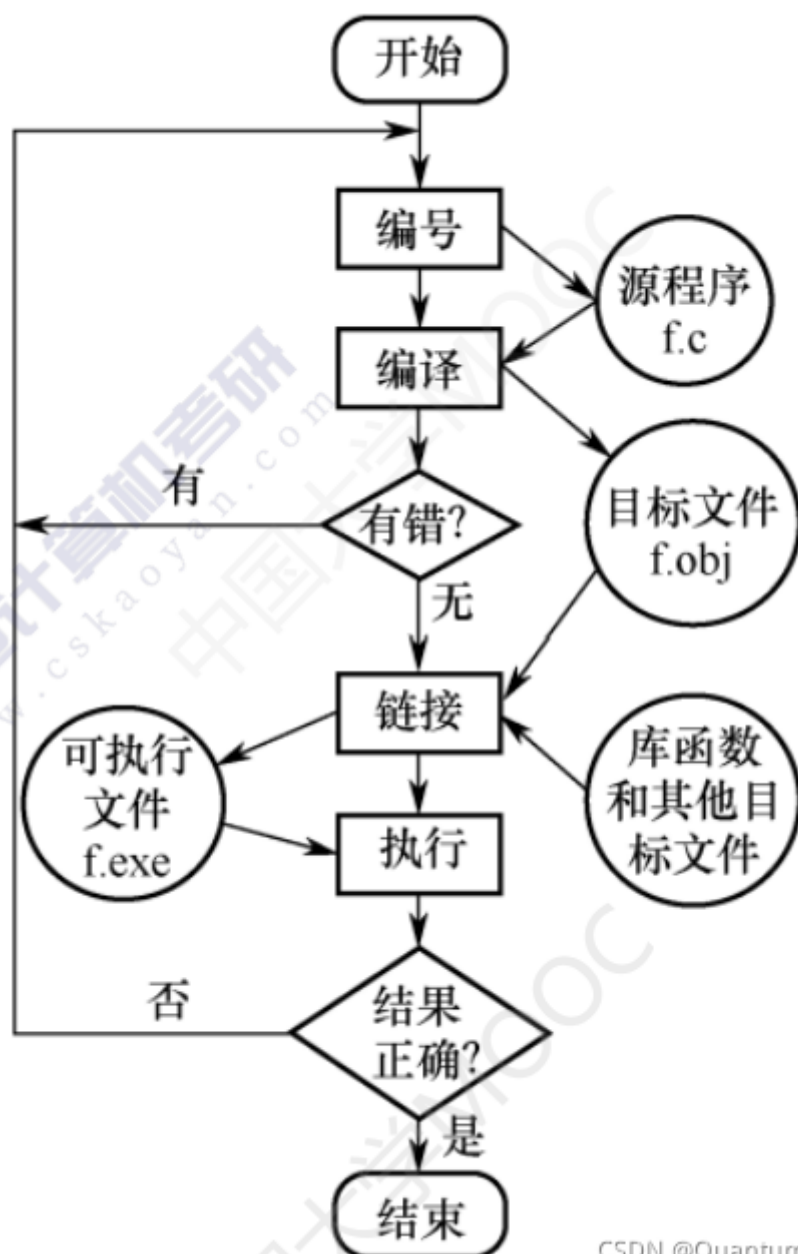
名称	修改日期	类型	大小
C++ 核心.tlog	2021/9/22 20:00	文件夹	
1、Hello.obj	2021/8/1 9:05	3D Object	52 KB
1、this指针.obj	2021/8/1 19:41	3D Object	55 KB
2、运算符重载.obj	2021/8/5 10:57	3D Object	41 KB
3、继承中的对象模型.obj	2021/8/6 10:50	3D Object	53 KB
4、多态基本概念.obj	2021/8/8 16:28	3D Object	62 KB
5、虚析构和纯析构.obj	2021/8/6 21:53	3D Object	167 KB
5、虚析构和纯析构测试.obj	2021/8/6 21:40	3D Object	170 KB
6、选择排序.obj	2021/8/9 8:55	3D Object	66 KB
7、函数模板.obj	2021/9/22 20:00	3D Object	156 KB
8、类模板分文件编写.obj	2021/8/9 21:21	3D Object	161 KB

## 关于解决方案与多项目

- 解决方案 (sln) 下面可以有多个项目
- 要运行哪个项目，要右键对应项目，把其设置为启动项目，如下图所示：谁是启动项对应项目名字是加粗的

## 程序的编译过程

- 程序的编译过程如下图所示。首先编写源程序f.c.编写完毕后，通过编译器进行编译，这里的编译包括预处理、编译、汇编，详细过程将在Linux系统编程中讲解。读者如果有兴趣，可以参阅关于编译原理的书籍，f.c经过编译后，得到f.obj文件，f.obj文件中均是0/1类型的机器码，即CPU能够识别的微指令（英特尔的机器指令）。f.obj文件并不能执行，因为我们调用的标准库函数的代码并不在f.obj文件中。例如，上面main.c中的printf函数，其代码并不在main.obj中，这时经过链接就得到可执行文件f.exe.了解这个编译过程后，后面在编写程序遇到编译错误时，就可以分析错误，进而区分是编译错误还是链接错误

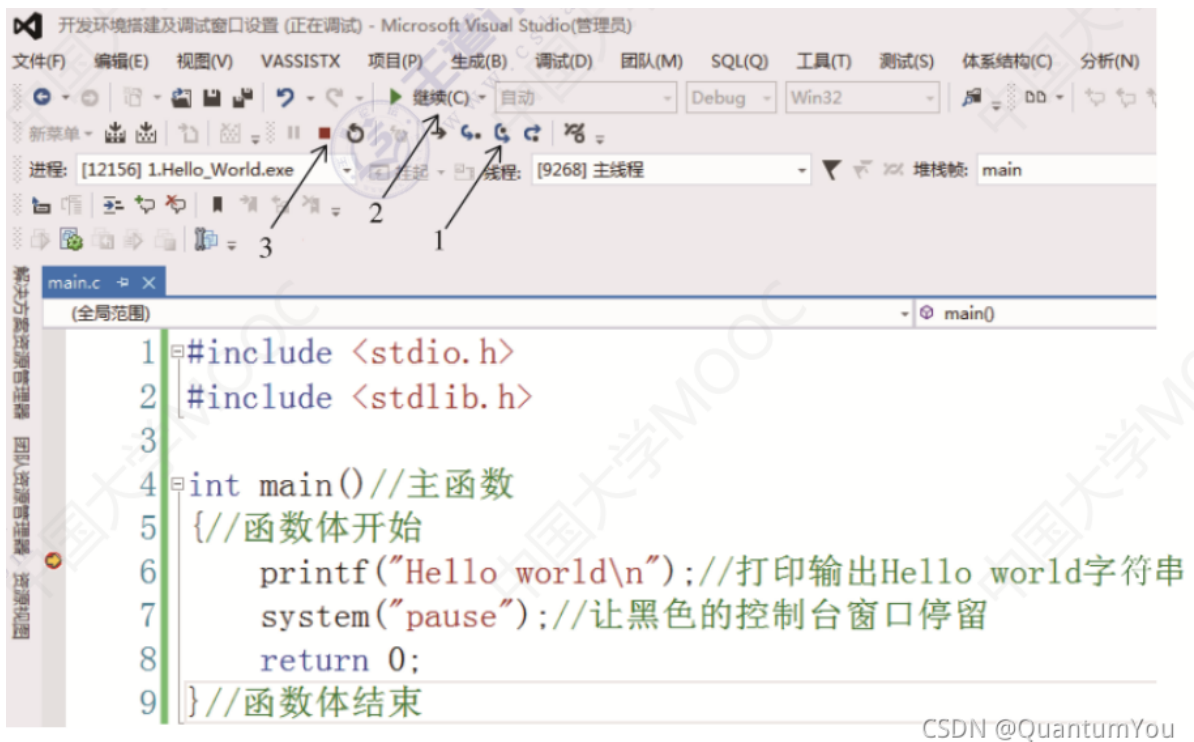


CSDN @QuantumYou

- 预处理阶段 输出 `main.i`

## 窗口调试

- 在如下图所示的窗口中，于第6行代码的左侧灰色区域单击，打上断点，然后单击“执行”按钮（绿色三角形，汉字注明的本地 Windows 调试器即为“执行”按钮），运行后的显示效果如图所示，其中按钮1是“单步执行”按钮，其快捷键是F10，单击该按钮一次，程序会向下执行一步；按钮2是“继续执行”按钮，单击后程序会执行到最后，或执行到下一个断点；按钮3是“停止执行”按钮，单击后程序直接停止运行（左键单击断点可以取消断点）



- 调试窗口设置 (常用的调式窗口: **内存与监视与调用堆栈**)

## Day02 03 04 数据类型

### 数据类型与关键字



表 2.1.1 C 语言中的关键字

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

### 常量与变量

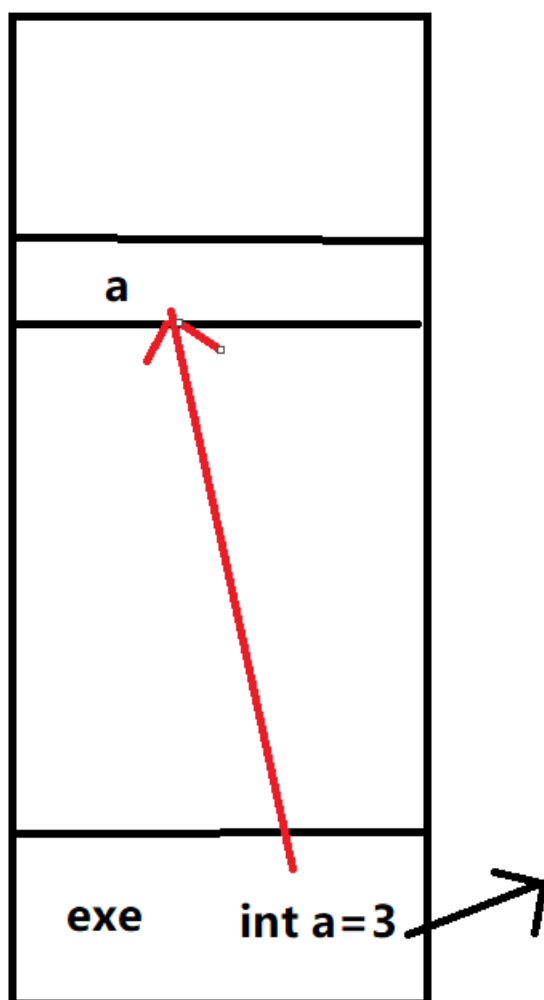
#### 常量

常量是指在程序运行过程中, 其值不发生变化的量。常量又可分为整型、实型 (也称浮点型), 常量的分类如下图所示,

整型	100, 125, -100, 0
实型	3.14, 0.125, -3.789
字符型	'a', 'b', '2'
字符串型	"a", "ab", "1c34"

## 变量

- 变量的命名规定如下：C语言规定标识符只能由**字母、数字和下划线**三种字符组成，并且第一个字符必须为字母或下划线。（编译原理可知）
- 变量名实际上以一个名字代表一个对应的存储单元地址。编译、链接程序时，由编译系统为每个变量名分配对应的内存地址。从变量中取值实际上是通过变量名找到内存中存储单元的地址，并从该存储单元中读取数据，如下图所示



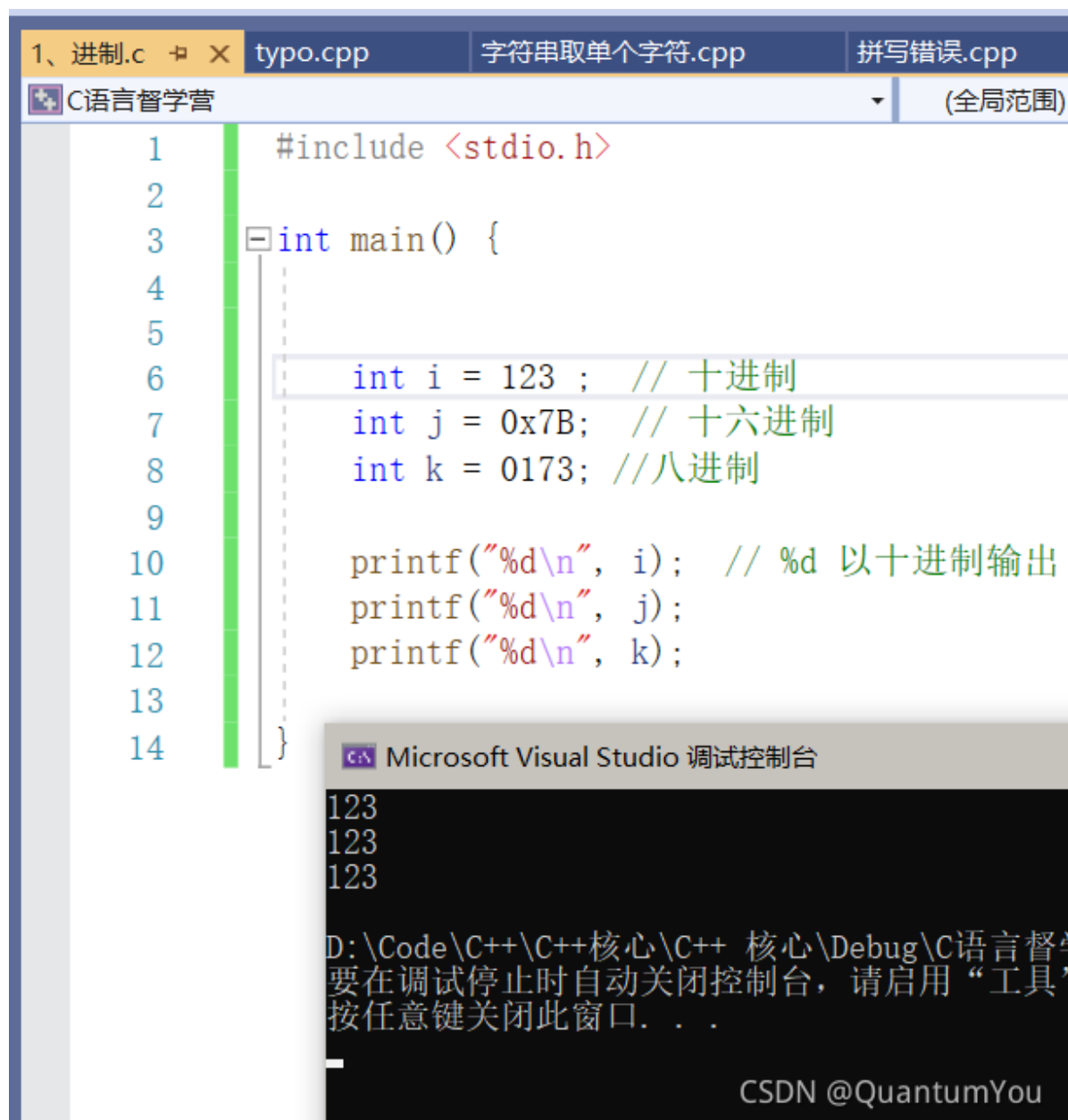
代码段（存放可执行性文件，从上至下依次执行，执行到 `int a = 3` 时，将3赋值到a的空间）

CSDN @QuantumYou

- 预处理时会消除所有的 `define`
- 左值 --> 可以修改的变量
- `# define CRT_SECURE_NO_WARNINGS` 解决 `scanf`编译报错问题

## 整型

## 整型数字在不同进制中的表示



HEX	7B
DEC	123
OCT	173
BIN	0111 1011

## 整型调试探究

- 如下图所示的结果。在监视窗口中输入&i（取地址i），得到i的地址，左键长按将其拖入右边的内存区域，就可以看到i的内存。我们的32位控制台应用程序的地址范围是从0到4G，即从0x00000000到0x FFFFFFFF
- 如图2.4.9所示，这称为进程（程序运行起来后称为进程）地址空间。程序编译完毕，开始执行时，会被放入进程地址空间的代码段区域。执行到哪条语句，PC指针就指向该条语句对应的地址。例如，目前我们执行到语句 `int i = 0x7b`，变量i会在栈空间上被分配空间，大小为4字节，起始地址为0x0013FAF8。按F10键，结果为其中i的值变为7b（我们以十六进制方式查看内存），其十进制值为7×16+11=123。i的值是0x0000007b为什么显示结果为7b000000呢？原因是英特尔的CPU采用了

## 小端方式进行数据存储

```

1 #include <stdio.h>
2
3 int main() {
4
5     int i = 123; // 十进制
6     int j = 0x7B; // 十六进制 已用时间 <= 1ms
7     int k = 0173; // 八进制
8
9     printf("%d\n", i); // %d 以十进制输出
10    printf("%d\n", j);
11    printf("%d\n", k);
12
13 }
14

```

内存 1

地址	值	类型
0x008FF8EC	7b 00 00 00	int
0x008FF8F0	cc cc cc cc	???
0x008FF8F4	14 f9 8f 00	???
0x008FF8F8	a3 1e 9c 00	???
0x008FF8FC	01 00 00 00	???
0x008FF900	40 6c a2 00	???
0x008FF904	00 ac a2 00	???
0x008FF908	01 00 00 00	???
0x008FF90C	40 6c a2 00	???
0x008FF910	00 ac a2 00	???
0x008FF914	70 f9 8f 00	???
0x008FF918	f7 1c 9c 00	???
0x008FF91C	a5 53 74 fc	???
0x008FF920	23 10 9c 00	???
0x008FF924	23 10 9c 00	???
0x008FF928	00 80 6c 00	???
0x008FF92C	00 00 00 00	???
0x008FF930	00 00 00 00	???
0x008FF934	00 00 00 00	???
0x008FF938	00 00 00 00	???
0x008FF93C	00 00 00 00	???
0x008FF940	00 00 00 00	???
0x008FF944	00 00 00 00	???
0x008FF948	00 00 00 00	???
0x008FF94C	7c a5 9c 00	???
0x008FF950	88 a5 9c 00	???
0x008FF954	00 00 00 00	???
0x008FF958	1c f9 8f 00	???
0x008FF95C	00 00 00 00	???
0x008FF960	dc f9 8f 00	???
0x008FF964	20 37 9c 00	???
0x008FF968	35 3a 67 fc	???
0x008FF96C	00 00 00 00	???
0x008FF970	78 f9 8f 00	???
0x008FF974	8d 1b 9c 00	???
0x008FF978	80 f9 8f 00	???
0x008FF97C	28 1f 9c 00	???
0x008FF980	90 f9 8f 00	???
0x008FF984	50 63 18 77	???

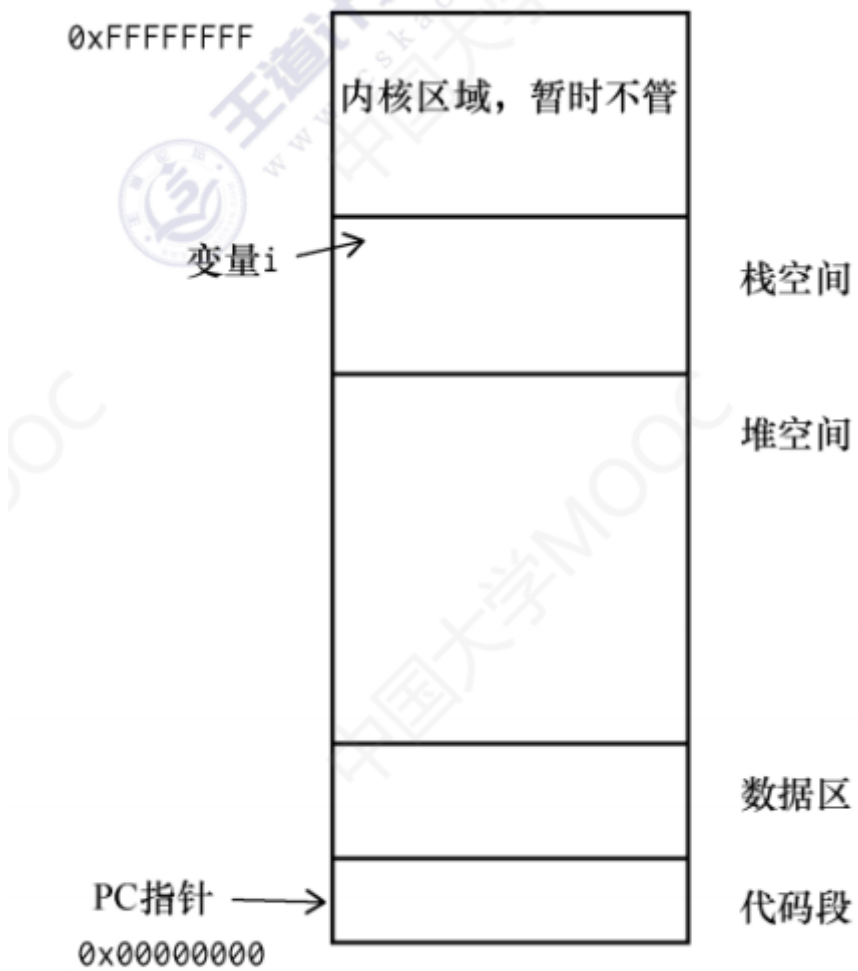


图 2.4.9 32 位控制台应用程序的地址范围

7b 00 00 00 x86架构是小端存储（英特尔和AMD）=> 低位在前，高位在后



多字节数据在内存里一定是占连续的几个字节

最高有效字节 (MSB)

最低有效字节 (LSB)

4字节 int: 01 23 45 67 H

19088743 D

0000 0001 0010 0011 0100 0101 0110 0111 B

便于人类阅读

大端方式

	0800H	0801H	0802H	0803H	
...	01H	23H	45H	67H	...

便于机器处理

小端方式

	0800H	0801H	0802H	0803H	
...	67H	45H	23H	01H	...

- **大端模式**: 先从最高位读数, 依次放于内存单元 0800H 0801H .... 官方解释 是指数据的高字节保存在内存的低地址中, 而数据的低字节保存在内存的高地址中
- 为什么内存数据要用十六进制去看, 就是非常高效, 两个字符就可以表示一个字节 ,0x 00 7D FD 04 地址是4个字节,1 个字节就是8位, 你们其为4\*8=32 位

编程:

1 字节 1 byte =8 bit

1Kb = 1024 字节

1Mb =1024 Kb

1Gb = 1024 Mb

实际生产:

磁盘 1G =1000 000 000 字节

## 进程地址空间分布

内存 1

0xFFFFFFFF	?? ?? ?? ??	....
0x00000003	?? ?? ?? ??	....
0x00000007	?? ?? ?? ??	....
0x0000000B	?? ?? ?? ??	....
0x0000000F	?? ?? ?? ??	....

程序员

FFFF FFFF

HEX FFFF FFFF

DEC -1

OCT 37 777 777 777

BIN 1111 1111 1111 1111 1111 1111 1111 1111

底层分析: 一个进程为4GB -->  $2^{32}$  ,0x ffff ffff

[推荐参考链接](#)

## 字符型

---

### 注意事项:

- 如果先用语句 `char c` 定义字符型变量`c`,后令 `c="a"` 或 `c="CHINA"` ,那么这样的赋值都是非法的,原因是不可以将字符串型常量赋值给字符型变量。
- C语言中没有定义字符串型变量的关键字,介绍字符数组时我们将详细讲解如何存放字符串
- C语言规定,在每个字符串型常量的结尾加一个字符串结束标志,以便系统据此判断字符串是否结束。C语言规定以字符'`\0`' 作为字符串结束标志

此为本人在2023王道C语言督学营第一期笔记,侵联删,装载请添加备注此原链