23041C 基础课

时间：8 到 10 天

学习方法：

1.主次分明，把大部分时间放在重点内容。

2. 多练习，多敲代码。

3. 及时复习，整理笔记。

4. 心态，保持一个良好的心态去学习。

上课要求：

1. 早自习分享

2. 整理笔记，用思维导图的方式。

3. 课堂纪律

4. 课堂气氛活跃，思维活跃。

基础知识

1. linux 基本命令

1.1 打开终端

直接点击图标

点击右键，open terminal

快捷键 ctrl alt t 打开家目录

快捷键 ctrl shift n 当前目录

1.2 关闭终端

点叉号

ctrl d

exit 命令

1.3 进入终端后基本操作

hq@Ubuntu:~$

用户名@主机名：当前路径$

进行用户切换命令：su 用户

切换到管理员 su root

允许使用 root 权限执行命令：sudo 命令

查看用户名：whoami

查看主机名：hostname

查看当前路径：pwd(绝对路径)

绝对路径：从根目录开始的路径

相对路径：从当前目录开始的路径

1.4 常用快捷键

放大终端：ctrl shift +

缩小：ctrl -

历史命令：上下键

history

清屏：ctrl l

自动补齐：tab

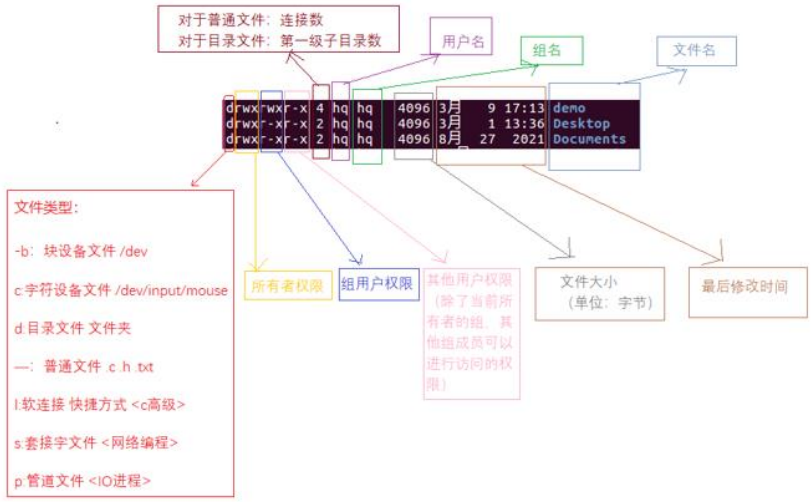
1.5 查看当前目录下内容命令：ls

查看当前目录内容：ls(查看其他路径目录：ls 路径/目录名)

-a 查看当前路径下包含隐藏文件的所有文件

-l 查看当前路径文件的详细信息

看下图：



r: 可读

w：可写

x: 可执行

1.6 修改权限命令：chmod

用法：chmod -R 权限 路径/文件名

-R 代表递归给目录下文件修改权限

权限对应数字：

r	4
w	2
x	1
-	0

chmod 后面跟的权限数字为 3 个，分别表示个人的权限、组员的权限、及其他人的权限

如果没有权限修改前面加 sudo:

例如：

sudo chmod -R 777 ~/demo

1.7 切换路径命令：cd

切换到指定路径：cd 路径

切换到家目录：cd ~

切换到根目录：cd /

切换到上一级目录：cd ../

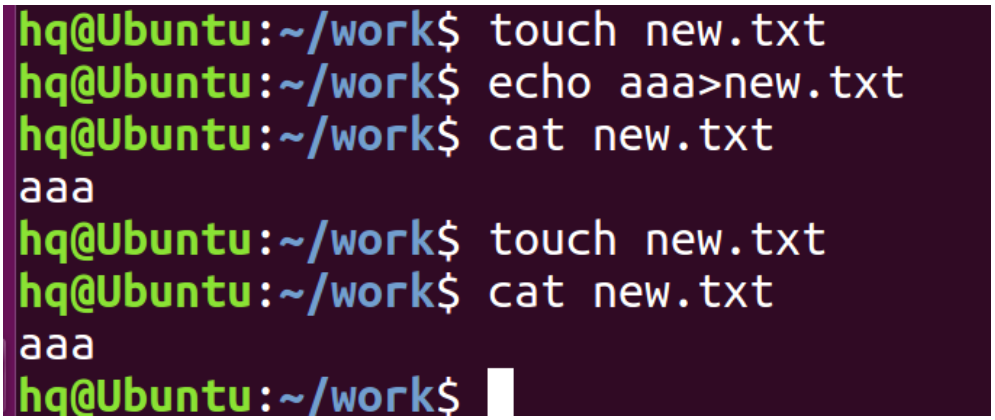
切换到当前目录：cd ./

切换到上次的目录：cd -

1.8 新建文件命令：touch

touch 文件名.后缀

touch 同名文件:会更新时间戳



1.9 新建目录命令：mkdir

mkdir 目录名

-p 创建多级目录

1.10 删除命令：rm

rm 文件名.后缀

-r 目录名

-i 删除前逐一确认

-f 即使原档文件设为只读，也直接删除，无需逐一确认

rm * -r 删除当前目录下所有文件和目录

1.11 复制：cp

cp 普通文件名.后缀 目标路径

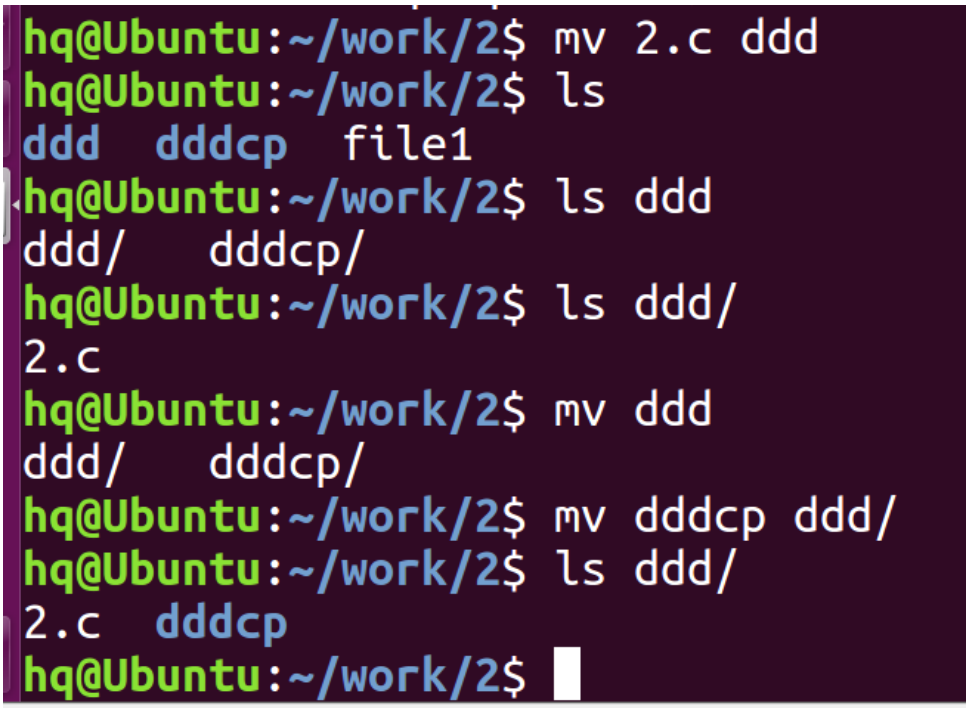
cp -r 目录文件名 目标路径

另存为：cp 文件名 路径/新文件名

1.12 移动：mv

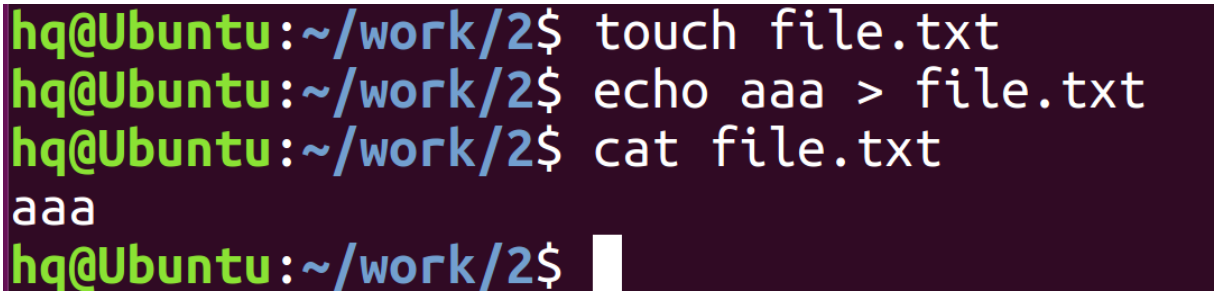
mv 文件名.后缀 目标路径

mv 目录名 目标路径



1.13 打印文件内容到终端：cat

cat 文件名.后缀



2. 常用的编辑工具 vi

vi 是 linux 中常用的文本编辑工具，vim 是其改进版。

2.1 插入模式

进入插入模式：先按 esc 键，然后按以下任意键：a i o A I O

新增 (append)

-- a 从光标所在位置后面开始新增资料，光标后的资料随新增资料向后移动。

-- A 从光标所在列最后面的地方开始新增资料

插入 (insert)

-- i 从光标所在位置前面开始插入资料，光标后的资料随新增资料向后移动。

-- I 从光标列的第一个非空白字符前面开始插入资料 。

打开 (open)

-- o 在光标所在列下新增一列并进入输入模式。

-- O 在光标所在列上方新增一列并进入输入模式。

2.2 命令模式

如何进入命令模式：按 esc 键

复制：yy nyy(n：行数)

删除(剪切): dd ndd

粘贴：p

撤销：u

反撤：ctrl r

光标移动首行： gg

光标移动末行： G

光标移动行尾：$

搜索：/或者？（搜索下一个用 n）

调整代码格式：gg=G

2.1 底行模式

进入底行模式：先按 esc 进入命令模式，然后按空格或者冒号。

指定第几行到第几行复制：5,10y

指定第几行到第几行删除: 5,10d

保存：w

退出：q

保存并退出：wq

强制：!

竖着分屏：vsp

横着分屏：split

取消分屏：on

查找：/str

取消高亮：noh

替换：s/str1/str2 光标所在行第一个 str1 替换成 str2

s/str1/str2/g 光标所在行所有 str1 替换成 str2

%s/str1/str2/g 每一行中所有 str1 来替换成 str2

n,$s/str1/str2/g 替换第 n 行到最后一行中所有 str1 为 str2

设置行号：set nu

取消：set nonu

3. 计算机结构

计算机五大结构：运算器、控制器、存储器、输入设备和输出设备。

3.1 运算器

计算机中完成二进制编码的数学运算和逻辑运算的设备。

3.2 控制器

计算机的控制中心，协调机器各部件运算。

3.3 存储器

计算机中用于存放数据和信息的程序部件，并且都是用二进制进行表示的。

3.4 输入设备

用于向计算机输入信息的设备。例如鼠标键盘。

3.5 输出设备

用于计算机向外输出信息的设备。例如打印机、音响和显示器。

4. 程序

对于计算机系统：程序就是系统可以识别的一组有序的指令。这组指令指挥这计算机系统工作。

计算机取指令分为两个阶段：

（1）取指：CPU 从存储器中取出指令码。

（2）执行：将指令翻译成要表达的功能，发出有关信号来实现这个功能。

5. 程序语言设计

5.1 机器语言

直接使用机器指令（也就是 0 和 1）来设计程序，可以被计算机系统直接识别。和自然语言完全不同，难于记忆和学习。工作量大效率低。

机器语言是二进制，例如：

00011000

00110001

00011001

5.2 汇编语言

把机器指令符号化,通过一组简单的符号来代表机器指令，更接近于自然语言，更容易理解。但是不能被计算机系统直接识别，所以要转换成机器语言让系统识别才能执行。

如：

MOV A, 1000

MOV 1010, A

5.3 高级语言

和特定的计算机系统无关，更接近于自然语言。一条语句对应多条指令，工作量小效率高。同样最终需要转换成机器语言让计算机系统识别、执行。例如高级语言有 C 语言、C++等。

6. 计算机数据表示和进制转换

送入计算机的数字、字母和符号等信息必须转换成 0、1 组合的数据形式才能被计算机所接收、存储和运算。

能够进行计算的数据并且能得出一个明确的数值叫数值数据，其余信息是非数值数据。

6.1 数值数据的表示

数值数据的计数方式是进位制。凡是按进位的方式计数的数制叫做进位计数制，简称进位制。用有限的数码表示。

十进制、二进制、八进制和十六进制。

6.1.1 十进制

例如：

123

=100+20+3

=1*10^2+2*10^1+3*10^0

基数：是指该进位制中所允许选用的数码的个数。例如十进制是 10。

权：每一位上的数值。

6.1.2 二进制

基数为 2 的进制叫二进制，只有 0 和 1 两种数码，逢二进一。英文缩写 BIN(前缀：0b)

数据的基本单位：字节 Byte

数据的最小单位：位 Bit

1KB=1024Byte

1MB=1024KB

1GB=1024MB

1TB=1024GB

(1)二进制转换成十进制：

例如：0b1010=1*2^3+0*2^2+1*2^1+0*2^0=8+2=10

0b1111=1*2^3+1*2^2+1*2^1+1*2^0=8+4+2+1=15

(2)十进制转换成二进制：

【1】短除法：除以 2 取余数，逆序排列。

例如：100 转换成二进制

100/2=50...0

50/2=25...0

25/2=12...1

12/2=6...0

6/2=3...0

3/2=1...1

1/2=0...1

得出二进制：0b01100100

【2】拆分法：例如 74=64+8+2=2^6+2^3+2^1 ==>0b1001010

【3】计算器

练习：把十进制 200 和 66 转换成二进制，把 0b0011110 转换成十进制。

6.1.3 八进制

基数为 8 的进制数叫八进制，码数有：0 1 2 3 4 5 6 7，逢八进一，英文缩写 OCT。（前缀：0）

（1）八进制转换成十进制

算法同十进制，例如：

0177=1*8^2+7*8^1+7*8^0=64+56+7=127

十进制转换成八进制也可以用短除法，或者先转换成二进制再转换成八进制。

（2）二进制转换成八进制

每三位二进制代表一位八进制，因为 8=2^3

比如：

二进制：0b111 ==>八进制：07

二进制：0b1000 ==> 八进制：010

0b110 111 010 ==> 0672

(3) 八进制转二进制

每一位八进制代表三位二进制，从低位也就是右边开始，如果不满三位则再左边补零。>例如： 026 ==> 0b 00010110

练习：04561 转换成二进制和十进制

6.1.3 十六进制

十六进制码数有：0 1 2 3 4 5 6 7 8 9 A B C D E F,一共有 16 个数码，逢十六进一，英文缩写 HEX。（前缀:0x）

(1) 十六进制转换成十进制

0x4A=4*16^1+10*16*0

=64+10

=74

(2) 二进制转换成十六进制

每四位二进制代表一位十六进制，因为 16=2^4

例如：

0b1111==>0xF

0b10000==>0x10

练习：0b1100 1000 1010 ==>0xC8A

(3) 十六进制转换二进制

每一位十六进制代表四位二进制，从低位也就是右边开始转换，如果不满四位就在左边补零。

例如：0xA5E1==>0b1010010111100001

十六进制转换成八进制就可以通过二进制转换

练习:

1.二进制 0111 1110 转换成八进制,十进制和十六进制数。

2. 十进制 1000 对应的二进制、八进制和十六进制数。

6.2 非数值数据的表示

除了数值数据以外的都是非数值数据，例如文字、符号和图像等，最终也要转换成二进制数让机器识别。

6.2.1 字符数据

字符数据主要是指数字、字母、通用符号和控制符号等，在计算机内最终也要转换成机器可以识别的二进制编码的形式。国际上被普遍采用的一种编码是美国国家信息交换标准代码，简称为 ASCII。英文全称：American Standard Code for Information Interchange。字符数据用单引号括起来表示。

ASCII值			控制字符	ASCII值			控制字符	ASCII值			控制字符	ASCII值			控制字符
二	十	十六		二	十	十六		二	十	十六		二	十	十六	
0000 0000	0	00	NUL(空字符)	0010 0000	32	20	SPACE(空格)	0100 0000	64	40	@	0110 0000	96	60	`
0000 0001	1	01	SOH(标题开始)	0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0000 0010	2	02	STX(正文开始)	0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0000 0011	3	03	ETX(正文结束)	0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0000 0100	4	04	EOT(传输结束)	0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0000 0101	5	05	ENQ(询问请求)	0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0000 0110	6	06	ACK(收到通知)	0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0000 0111	7	07	BEL(响铃)	0010 0111	39	27	,	0100 0111	71	47	G	0110 0111	103	67	g
0000 1000	8	08	BS(退格)	0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0000 1001	9	09	HT(水平制表)	0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0000 1010	10	0A	LF(换行)	0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0000 1011	11	0B	VT(垂直制表)	0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0000 1100	12	0C	FF(换页)	0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0000 1101	13	0D	CR(回车)	0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0000 1110	14	0E	SO(移位输出)	0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0000 1111	15	0F	SI(移位输入)	0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0001 0000	16	10	DLE(数据链路转义)	0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0001 0001	17	11	DC1(设备控制1)	0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0001 0010	18	12	DC2(设备控制2)	0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0001 0011	19	13	DC3(设备控制3)	0011 0011	51	33	3	0101 0011	83	53	X	0111 0011	115	73	s
0001 0100	20	14	DC4(设备控制4)	0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0001 0101	21	15	NAK(拒绝接收)	0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0001 0110	22	16	SYN(同步空闲)	0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0001 0111	23	17	ETB(传输块结束)	0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0001 1000	24	18	CAN(取消)	0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0001 1001	25	19	EM(介质中断)	0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0001 1010	26	1A	SUB(换置)	0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0001 1011	27	1B	ESC(退出)	0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0001 1100	28	1C	FS(文件分割符)	0011 1100	60	3C	<	0101 1100	92	5C	/	0111 1100	124	7C	|
0001 1101	29	1D	GS(分组符)	0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0001 1110	30	1E	RS(记录分隔符)	0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0001 1111	31	1F	US(单元分隔符)	0011 1111	63	3F	?	0101 1111	95	5F	_	0111 1111	127	7F	DEL(删除)

（1） ‘0’ 、0、 “0” 和'\0'

‘0’	字符 0
0	数字 0
“0”	字符串 0
'\0'	空字符

（2）常用字符

'\0' 0 空字符，字符串结束的标志

'\n' 10 换行

'0' 48 字符 0

‘9’ 57 字符 9

‘A’ 65 字符大写字母 A

‘Z’ 90 字符大写字母 Z

'a' 97 字符小写字母 a

'z' 122 字符小写字母 z

(3) 转换

‘0’ -48=数字 0

大写字母+32=小写字母

小写字母-32=大写字母

7. 简单变成步骤

(1) 创建一个.c 文件：touch hello.c

(2) vi hello.c

(3) 编写 C 语言程序：

```
#include <stdio.h>     //头文件
int  main()                    //主函数,程序入口
{
  printf("Hello world!\n"); //打印 Hello world
     return 0;                              //返回 0
}
```

(3) 保存 wq

(4) 编译：gcc hello.c

(5) 执行程序 ./a.out

作业：

1. 整理笔记
2. 建立 test1、test2 文件夹;在 test1 文件夹中建立 1.txt 文件，在 1.txt 中写上 50 行 “hello farsight!” ,复制 1.txt 成为 2.txt，并把 2.txt 中所有的 “farsight” 改成 “world” ;移动 test1 文件夹到 test2 文件夹下;删除 test1
3. 编写一个 C 程序输出以下信息

****

Welcome to the game

****

4.把十六进制 0x1d5 转换成二进制、八进制、十进制

把十进制 102 传换成二进制、八进制、十六进制

把二进制 0b1001 1101 转换成八进制、十进制、十六进制

8. GCC 编译器

gcc(GNU CCompiler)是 GNU 推出的功能强大，性能优越的多平台编译器，gcc 编译器能将 C,C++语言源程序编译连接成可执行文件。预处理、编译、汇编和链接。

8.4 链接

gcc hello.o -o hello 是将各种代码和数据片段收集并组合成一个可执行文件的过程，这个文件可以被加载到内存执行。

```
hq@Ubuntu:~/work/23041$ gcc game.c
hq@Ubuntu:~/work/23041$ vi game.c
hq@Ubuntu:~/work/23041$ gcc -E game.c -o game.i
hq@Ubuntu:~/work/23041$ ls
a.out  game.c  game.i
hq@Ubuntu:~/work/23041$ vi game.i
hq@Ubuntu:~/work/23041$ vi game.c
hq@Ubuntu:~/work/23041$ gcc -E game.c -o game.i
hq@Ubuntu:~/work/23041$ vi game.i
hq@Ubuntu:~/work/23041$ gcc -S game.i -o game.s
hq@Ubuntu:~/work/23041$ ls
a.out  game.c  game.i  game.s
hq@Ubuntu:~/work/23041$ vi game.s
hq@Ubuntu:~/work/23041$ gcc -c game.s -o game.o
hq@Ubuntu:~/work/23041$ ls
a.out  game.c  game.i  game.o  game.s
hq@Ubuntu:~/work/23041$ vi game.o
hq@Ubuntu:~/work/23041$ gcc game.o -o game
hq@Ubuntu:~/work/23041$ ./game
**********
welcome to the game!
**********
hq@Ubuntu:~/work/23041$
```

9. 词法符号

词法符号是若干个字符组成的有意义的最小语法单位。按照在程序中的作用，可以分为：关键字、标识符、运算符、分隔符和标点符号。

9.1 关键字

由系统与定义好的词法符号，有特殊的含义，不允许用户重新定义。

（1）**存储类型**：auto(自动型) static(静态) extern(外部) register(寄存器)

（2）**数据类型**：char(字符型) short(短整型) int(整型) long(长整型) float(浮点型) double(双精度浮点型) signed(有符号) unsigned(无符号) struct(结构体) union(共用体) enum(枚举) void(空类型)

（3）**控制语句**：if else while do for switch case default break continue goto return

(4) **其他**：sizeof(计算数据所占空间大小) const(只读) typedef(重命名) volatile(防止编译器被优化)

9.2 标识符

由程序员按照命名规则自定义的词法符号，用于定义宏定义名、变量名、函数名和自定义类型名等。

C 语言标识符的命名规则：

（1）标识符由字母、数字和下划线组成

（2）标识符第一个字符必须是字母或下划线

（3）不能和关键字相同

练习：x　y　　_A7b_3x　3’　a　x*y　@　b.8　while　sum　a100

9.3 运算符

运算符是表示运算的词法符号，按功能分为：算术运算、逻辑运算、关系运算、赋值运算、位运算和其他运算符。

(1) 算术运算符：+ - * / % ++ --

(2) 赋值运算符：= += -= *= /= %=

(3) 关系运算符： < <= > >= == !=

(4) 逻辑运算符：&& || ！

(5) 位运算符：& | ~ ^ << >>

(6) 其他运算符：三目运算符 sizeof()

9.4 分隔符

用来分隔开其他的词法符号，主要包括：空格、制表符、换行符号和注释。

通过对分隔符的恰当运用，使得代码的外观格式更为清晰易读，还可以帮助分析程序中的语法错误

9.5 标点符号

C 语言中标点符号有逗号、分号、冒号、花括号、圆括号和方括号。标点符号的作用与分隔符相似，但是用法非常严格，有着明确的语法规定，如果写错了就会报错。

数据类型

1. 变量和基本数据类型

1.1 变量的概念

变量就是在程序中可以发生变化的量，变量有类型。

变量的类型决定了变量存储占用的空间，以及如何解释存储的位模式。

1 字节=8 位

1.2 定义格式

存储类型 数据类型 变量名；

例如：(auto) int a;

变量名是标识符，要符合标识符命名规则。

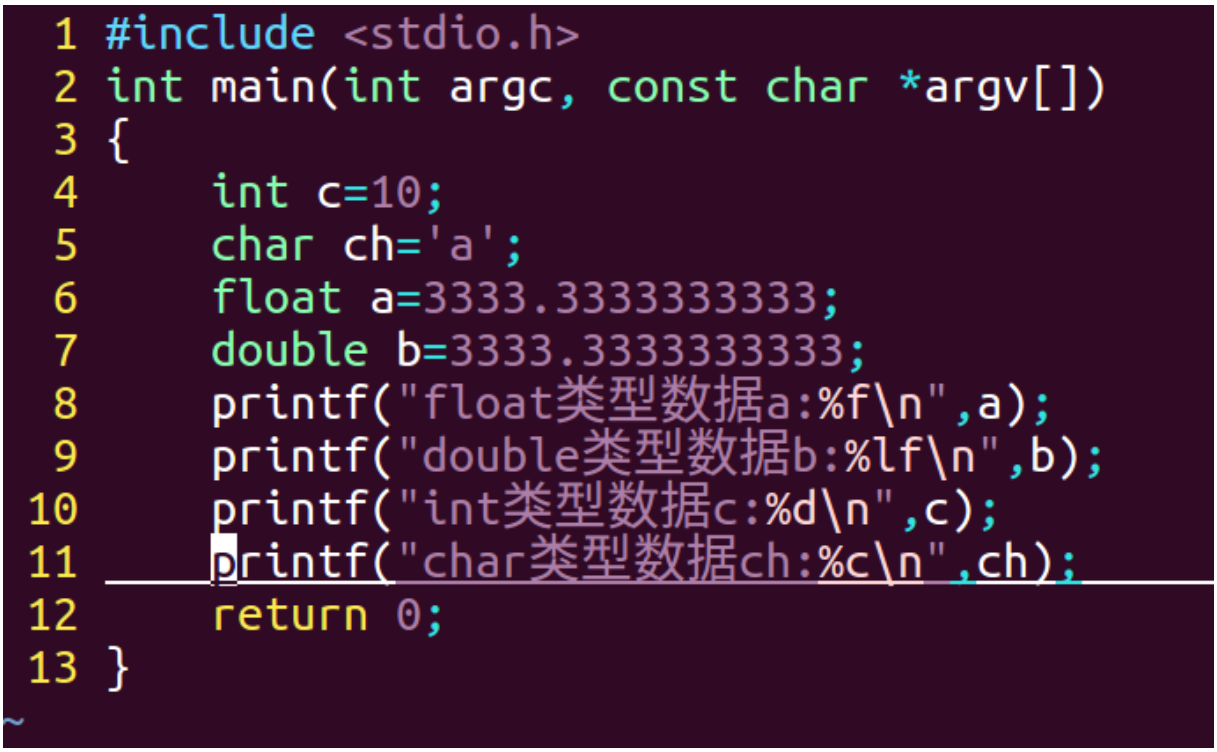
存储类型：变量存储的位置。

数据类型包含：名字 字节 取值范围

类型		存储大小	值范围
char	字符类型	1 字节	-2^7 到 (2^7-1)
unsigned char	无符号字符类型	1 字节	0 到 (2^8-1)
int	整数类型	4 字节	-2^31 到 (2^31-1)

unsigned int	无符号整数类型	4 字节	0 到 (2^32-1)
short	短整型	2 字节	-2^15　到 (2^15-1)
unsigned short	无符号短整形	2 字节	0 到 (2^16-1)
long	长整型	4 字节	-2^31 到 (2^31-1)
unsigned long	无符号长整形	4 字节	0 到 (2^32-1)
float	单精度浮点型	4 字节	有效数据 6-7 位
double	双精度浮点型	8 字节	有效数据 15-16 位

例子：



```
hq@Ubuntu:~/work/23041$ gcc float.c
hq@Ubuntu:~/work/23041$ ./a.out
float类型数据a:3333.333252
double类型数据b:3333.333333
int类型数据c:10
char类型数据ch:a
```

1.3 初始化格式

(1) 可以定义的时候初始化

int a=0;

(2) 先定义后初始化

int a;

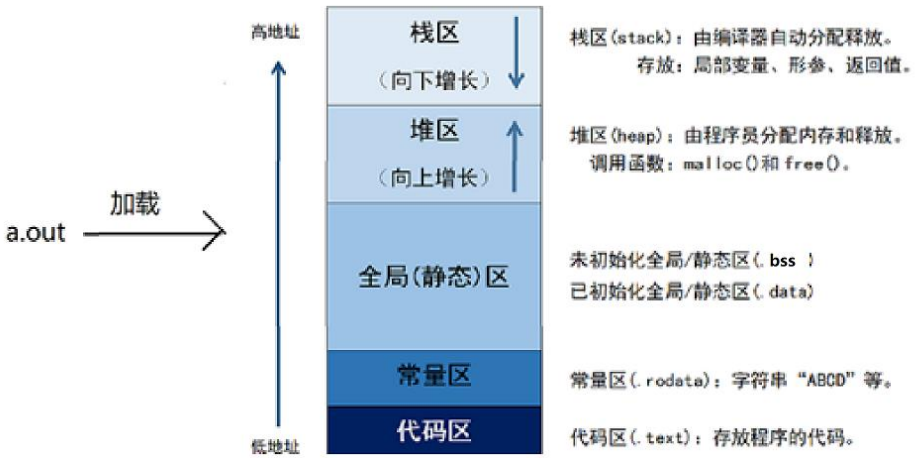
a=0;

1.4 局部变量和全局变量

（1）生命周期：变量的生命周期指的是变量的创建到变量的销毁之间的一个时间段。也就是存活的周期时间。时间维度。

（2）作用域：变量可以起作用的范围。空间维度。

（3）内存的五个区域：



局部变量和全部变量的区别：

	局部变量	全局变量
定义位置	函数体内部	函数体外部
存储位置	栈区	全局区
生命周期	同函数体共存亡	同整个程序共存亡
作用域	作用于函数体内部	作用于整个程序
初值	未初始化时，是一个随机值	未初始化时，值为 0

例子：

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

2. 常量

2.1 概念

程序运行中不会发生变化的量叫常量，存放在常量区。

2.2 分类

2.2.1 字符型常量

类型为 char，从 ascii 表中找到的字符都是字符型常量，不可以改变。用单引号括起来的就是字符常量，例如 ‘A’ 。

类型		存储大小	值范围
char	字符类型	1 字节	-128 到 127

unsigned char	无符号字符类型	1 字节	0 到　255
signed char	带符号字符类型	1 字节	-128 到　127

原码、反码和补码：

（1）最高位表示符号位，正数时 0，负数是 1。

（2）正数的原码、反码和补码都一样。

（3）负数要用补码来计算：

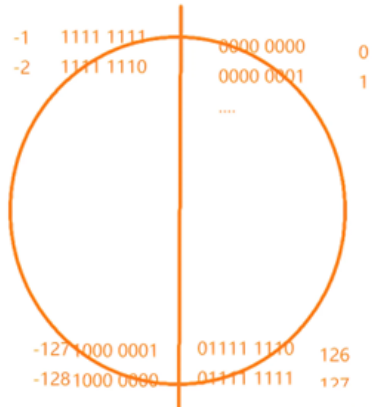
- 原码就是直接在数值前面加上符号表示
- 反码等于原码的符号位不变，其他位按位取反
- 补码等于反码加一

例如：-5

原码：1000 0101

反码：1111 1010

补码：1111 1011



用''括起来的就是字符常量：

'a' 字符 a

'\0' 空字符

'\n' 换行

用法：

```
        char c='b';
        char a='\101';
        printf("%c\n",'A');
        printf("%c\n",65);
        printf("%c\n",'\x42');
        printf("%c\n",c);
        printf("%c\n",a);
        printf("%c\n",a+32);
```

因为 C 规定转义字符'\x41'中\是转义字符引导符，后跟一个 x 表示 x 后面的数字是十六进制表示法，用' '括起来表示一字节 ASCII 码。\转义符后面加数字代表转义成八进制的字符，后面的数字是八进制。

2.2.2 字符串常量

用 “” 括起来的就是字符串，例如 “hello” ,字符串最后跟一个'\0'。

例如：

```
    printf("%s\n","hello");
```

2.2.3 整型常量

整型常量就是类型为整数的常量，包括从负数到零到正数的所有整数。

可以用二进制、八进制、十进制和十六进制表示。

```
    #include <stdio.h>
    int main(int argc, const char *argv[])
    {
        int a=10;//把整型常量赋值给整型变量
        printf("%d\n",a);
        printf("%d\n",10); //十进制
        printf("%d\n",0b1111); //二进制
        printf("%d\n",010); //八进制
        printf("%d\n",0xF); //十六进制
        return 0;
    }
```

2.2.4 浮点型常量

浮点型常量就是类型为浮点数的常量，包括从负数到零到正数的所有浮点数。

float double

2.2.5 指数常量

科学计数法表示

300000000=3*10^8==>3e8

0.0000012=1.2*10^-6==>1.2e-6

例子：

```
        float f=80000;
        printf("%e %e\n",0.0000012,f);
```

2.2.6 标识常量（宏定义）

宏定义：起到标识作用

（1）只是单纯的文本替换，在预处理的时候进行的

（2）遵循标识符命名规则

（3）一般用大写标识

格式：

#define 宏名 常量或表达式

特点：单纯的文本替换，不能手动运算（原样替换）

例子：

```
#define N 2
#define M N+3 //M 2+3
#define NUM N+M/2+1 //2+2+3/2+1=6
void main()
{
   int a = NUM;
   printf("%d\n",a);
}
```

作业：

1. 整理笔记，用思维导图的方式
2. 一个水分子的质量约为 3.0*10-23g，1 夸脱水大约有 950g，编写一个程序，要求给出水的夸脱数，然后显示这么多水中包含说少水分子。
3. 背过关键字

运算符和表达式

所谓表达式就是指由运算符、运算量和标点符号组成的有效序列，其目的是说明一个计算过程。表达式可以独立成语句：

表达式;

运算符按功能分为：算术运算、赋值运算、关系运算、逻辑运算、位运算以及其他运算符

1.算术运算符：+ - * / % ++ --

（1）/:整数相除，向下取整。

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    int n=12345,g=0,s=0,b=0,q=0,w=0;
    g=n%10;
    s=n/10%10;
    b=n/100%10;
    q=n/1000%10;
    w=n/10000;
    printf("%d %d %d %d %d \n",w,q,b,s,g);
    return 0;
}
```

(3) ++:自增

int a=0;

a++;或者++a;

相当于 a=a+1;

```
        int a=1,b=1;
        a++;//a=a+1
        ++b;//b=b+1
        printf("%d %d\n",a,b);//2 2
```

(4) --:自减

a--;

--a;

a=a-1;

```
    #include <stdio.h>
    int main(int argc, char const *argv[])
    {
        int a=1,b=1;
        a--;//a=a-1
        --b;//b=b-1
        printf("%d %d\n",a,b);//0 0
        return 0;
    }
```

自加自减和打印结合：++在前先++再打印，++在后先打印在++。

```
        int a=1,b=0;
        //b=++a;//结果会得 2
        b=a++;
        printf("%d",b);//结果得 1
```

例如：

z=++x+y++;

相当于：

x=x+1;

z=x+y;

y=y+1;

(3)a+=2;

相当于：a=a+2;

练习：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int a = 10;
    int b = ++a;  //b=11 a=11
    int c = a + (b++); //a=11 b=12 c=22
    int d = b + c;    //d=12+22=34
    printf("%d\n", d);
    return 0;
}
```

[单选题] *

A. d = 31

B. d = 32

C. d = 33

D. D. d = 34

2. 关系运算：> >= < <= == !=

用来判断两者得关系，返回真或者假。

运算符	名称	示例	功能	缩写
<	小于	a<b	a 小于 b 时返回真，否则返回假。	LT
<=	小于等于	a<=b	a 小于等于 b 时返回真，否则返回假。	LE
>	大于	a>b	a 大于 b 时返回真，否则返回假。	GT

>=	大于等于	a>=b	a 大于等于 b 时返回真，否则返回假。	GE
==	等于	a==b	a 等于 b 时返回真，否则返回假。	EQ
!=	不等于	a!=b	a 不等于 b 时返回真，否则返回假。	NE

例子：

```
#include <stdio.h>
int main()
{
    int a=2,b=3;
    printf("%d\n",a>b);
    printf("%d\n",100%10!=0);
    return 0;
}
```

3.逻辑运算符：&& || ！

3.1 &&（逻辑与）

全真则真，一假则假。

全都是真的才是真，只要有假就是假。

3.2 ||（逻辑或）

一真则真，全假则假。

只要有真就是真，全都是假才是假。

3.3 ！(逻辑非）

非真为假，非假为真。

例子：

```
#include <stdio.h>
int main()
{
    int a=5,b=6,c=7,d=8,m=2,n=2;
    int r=(m=a<b)||(n=c>d);
    printf("%d  %d %d",m,n,r);//1 2 1
}
```

```
#include <stdio.h>
int main()
{
    int a=5,b=6,c=7,d=8,m=2,n=2;
    int r=(m=a>b)&&(n=c>d);
    printf("%d  %d  %d",m,n,r);//0 2 0
}
```

4 位运算符：& | ~ ^ << >>

含义	C 语言
按位与	a&b
按位或	a|b
按位异或	a^b
按位取反	~a
左移	a<<b
右移	a>>b
无符号右移	/

负数都是用反码进行运算的

4.1 位与 &

全 1 则 1，有 0 则 0.

例如：

1111 0101

| 0101 0011

1111 0111

4.3 异或^

不同为 1，相同为 0。

例如：

1111 0101

^ 1111 1101

0000 1000

4.4 取反 ~

按位取反，0 变成 1，1 变成 0。

	原码	反码	补码
正数：	本身	本身	本身
负数：	本身	除了符号位不变其他位按位取反	反码+1

~15

正数 15：0000 1111

~15:11110000

~-1：

原码： 1000 0001

反码： 1111 1110

补码=反码+1：1111 1111

~（-1）：0000 0000

~ -5：

原码：1000 0101

反码：1111 1010

补码：1111 1011

~（-5）：0000 0100

4.5 << 左移

左移几位，右边补几个零。

8<<2: 0000 1000<<2==>0010 0000

-5<<3:

原码：1000 0101

反码：1111 1010

补码：1111 1011

<<3: 1101 1000

1101 0111

1010 1000 = -40

负数补码求原码：先减一，再符号位不变然后按位取反。

4.6 >> 右移

右移几位，左边补几个符号位，正数补 0 负数补 1。

8>>2 :

0000 1000>>2 :0000 0010=2

4.7 置一公式和置零公式

置一公式：a|(1<<n)

置零公式：a&(~(1<<n))

例子:

```
#include <stdio.h>
int main()
{
    int a=0b0111;
    printf("%d\n",a|(1<<3));
    printf("%d\n",a&(~(1<<2)));
    return 0;
}
```

5 特殊运算符

5.1 三目运算符

表达式：表达式 1？表达式 2：表达式 3

先求解表达式 1，若其值为真则执行表达式 2，表达式 2 的返回值作为整体的取值。否则将表达式 3 的值作为整体的取值。

```
#include <stdio.h>
int main()
{
    int a=10,b=6,max=0;
    max=(a>b)?a:b;
    printf("%d\n",max);
    return 0;
}
```

练习：判断以下打印结果

```
int num1 = 10, num2 = 20;
int res = num1 > num2 ? num1++ : num2++;
printf("num1=%d  num2=%d   res=%d\n", num1, num2, res);
```

10 21 20

5.2 sizeof()

用来计算数据所占空间大小的运算符。

格式：sizeof(数据类型或者变量名)；

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    char ch = getchar();
    if (ch >= 'A' && ch <= 'Z')
        ch += 32;
    else if (ch >= 'a' && ch <= 'z')
        ch -= 32;
    else
        printf("input error:\n");
    putchar(ch);
    putchar(10);
    return 0;
}
```

3. 结合宏定义三目运算符实现求两个数的差

```
#include <stdio.h>
/*方式1.三目运算符结合宏 */
#define M 30
#define N 50
#define DIFF M>N?M-N:N-M
/*方式2.三目运算符结合宏函数 */
#define DIFF_FUN(A,B) A>B?A-B:B-A

int main(int argc, char const *argv[])
{
    printf("%d\n",DIFF);
    printf("%d\n",DIFF_FUN(7,1));//调用宏函数
}
```

4.给出一个年份，判断是平年还是闰年。如果是润年打印 1，平年打印 0。

(提示：判断闰年还是平年，闰年且闰年二月份以上比平年多一天。普通年份除以 4，有余数是平年，对于整百的年份，比如 2100 年，要除以 400，有余数就是平年，整除就是闰年。)

思路，可以设置一个标志 flag 来接收平年还是闰年。然后用逻辑运算去判断平年还是闰年。最后打印 flag。

4.

有以下程序：

```
int main()
{
    unsigned char a,b,c;
    a=0x3; b=a|0x8; c=b<<1;
    printf("%d %d\n" ,b,c);
    return 0;
}
```

程序运行后的输出结果是（ ）。

A. -11 12　B. -6 -13　C. 12 24　D. 11 22

5.

设 char 型变量 x 中的值为 10100111，则表达式(2 + x) ^ (~3)的值是（ ）。

A. 10101001　B. 10101000　C. 11111101　D. 01010101

4. b=0b 0011 | 0b 1000

= 0b 1011

c=0b1011<<1 =0b 0001 0110 =22

D

5.0000 0010

+1010 0111

= 1010 1001

^(~3)= 0101 0101

D

8.

有以下程序：

```
int main()
{
    unsigned int a;
    int b=-1;
    a=b;
    printf("%u",a);
    return 0;
}
```

程序运行后的输出结果是（ ）。 //假设 int 占 2 byte 。

A. -1　B. 65535　C. 32767　D. -32768

9.

设有定义语句： char c1=92,c2=92;，则以下表达式中值为零的是（ ）。

A. c1^c2　B. c1&c2　C. ~c2　D. c1|c2

10.

若a=1,b=2; 则a|b的值是（）

A. 0　B. 1　C. 2　D. 3

8. 涉及到隐式转换：

同样的字长有符号数和无符号数之间的转换规则是：数值可能会变，但是位模式不变。

如果转换的无符号数太大以至于超过了补码能够表示的范围，可能会得到该范围的最大值。

-1： 原码： 1000 0001

反码：1111 1110

补码：1111 1111

超过了最大值 2^16-1=65535,所以得到最大值 65535。

B

9. A

10. 0001

| 0010

= 0011 ==>3

D

输入输出

函数三要素：功能、参数和返回值。

1. 按字符输入输出

1.1 按字符输出 putchar()

1.1.1 介绍

#include <stdio.h>

int putchar(int c);

功能：向终端输出一个字符

参数：c:要输出字符的 ASCII 值

返回值：要输出字符的 ASCII 值。当输出错误的时候，返回 EOF(end of file)文件结束符号。

1.1.2 用法

```
        int a=10;
        putchar('A');
        putchar(66);
        putchar('\n');
        putchar(10);
        putchar('a'-32);
        putchar('A'+32);
        putchar(a);
```

输出结果：

AB

Aa

1.2 按字符输入 getchar()

1.2.1 介绍

#include <stdio.h>

int getchar(void);

功能：向终端输入一个字符

参数：无

返回值：输入字符的 ASCII 值。如果发生错误返回 EOF。

1.2.2 用法

例子：

```
        int a = getchar();
        putchar(a - 32);
        putchar(10);
```

2. 按格式输入输出

2.1 按格式输出 ptrintf()

2.1.1 介绍

#include <stdio.h>

int printf(const char *format, ...);

功能：按指定格式向终端输出

参数：format:用双引号括起来的格式控制串和输出表。

返回值：输出字符的个数（不常用）

2.1.2 格式

printf("格式控制串",输出表)；

格式控制串：包含两种信息。

1. 普通字符：原样输出。
2. 格式说明：修饰符%后面加格式字符，用于指定输出格式。

   输出表：要输出的数据（可以没用，如果多个要用","分隔)。

例子：

```
    int a=10,b=20;
    printf("a=%d b=%d\n",a,b);
```

格式字符：

%d int

%ld long

%f float

%lf double

%c char

%s 字符串

%x 十六进制

%o 八进制

%u 十进制无符号正数

%p 地址

%e 指数

%m.nf 含义：

m:位宽

.n:打印小数点后几位

-：左对齐，默认右对齐。

0：指定左边不使用的空位自动填充 0

```
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d %d\n",a,b);
```

注意：用 scanf 时 “” 中不要写上除了格式控制字符以外的东西，不然输入的时候要按原样输入才可以。

2.2.3 输入分隔符的指定

一般以空格、TAB 或者回车作为分隔符

要想用其他字符作为分隔符：格式串中添加要设置的分隔字符

例如：中间用逗号分隔开：

```
    int a, b;
    scanf("%d,%d", &a, &b);
    printf("%d\n%d",a,b);
```

练习：一个水分子的质量约为 3.0*10^-23g，1 夸脱水大约有 950g，编写一个程序，要求输入水的夸脱数，然后显示这么多水中包含多少水分子。表示：3.0e-23 打印格式：%f 或%e 。

```
    int num;
    scanf("%d",&num);
    printf("%e\n",num*950/3.0e-23);
```

注意：

scanf 函数输入回车符的问题,当用%c 方式输入字符时，回车符会被下次的%c 接收到。其他给格式不会出现这种问题。

例 1：连续输入两个 char 类型数据：

```
    char a,b;
    scanf("%c",&a);
    printf("%c\n",a);
    scanf("%c",&b);
    printf("%c\n",b);
```

第二个 scanf 会读到回车

例 2：连续输入两个 int 类型数据：

```
    scanf("%c",&c1);
    printf("%c\n",c1);
    scanf(" %c",&c2);
    printf("%c\n",c2);
```

(2) %*c

只能回收任意一个字符

```
    scanf("%c",&c1);
    printf("%c\n",c1);
    scanf("%*c%c",&c2);
    printf("%c\n",c2);
```

（3）getchar()

只能回收任意一个字符

执行顺序：如果条件成立，则执行语句块 1，如果调节不成立，则执行语句块 2。

例子：

if(下雨)

{

    带伞；

}

else

{

    不带伞；

}

```
#include <stdio.h>
#define RAIN 1
int main(int argc, char const *argv[])
{
    if(RAIN)
        printf("with unbrella\n");
    else
        printf("no unbrella\n");
    return 0;
}
```

练习：输入 a 和 b 两个整数，用 if 语句实现两数相减

```
#include <stdio.h>
int main()
{
    int a=0,b=0;
    scanf("%d%d",&a,&b);
    if(a>b)
        printf("%d\n",a-b);
    else
        printf("%d\n",b-a);

}
```

1.1.2 分层结构

if(表达式 1)

{

语句块 1；

}

else if(表达式 2)

{

语句块 2；

}

else

{

语句块 3；

}

执行顺序：如果满足 if 中的表达式则执行 if 后面的语句块 1，如果不满足则进行 else if 后面表达式 2 的判断，如果满足表达式则会执行语句块 2。如果前面条件都不满足则会执行 else 后面的语句块 3。

例子：

```
#include <stdio.h>
int main()
{
    int a=0,b=0;
    scanf("%d%d",&a,&b);
    if(a>b)
        printf("a>b");
    else if(a<b)
        printf("a<b");
    else
        printf("a=b");
}
```

练习：从终端输入一个学生的成绩，判断学生成绩，打印成绩级别。

【90 - 100】 A

【80 - 89】 B

【70 - 79】 C

【60 - 69】 D

< 60 sorry you lost

}

else

{

    语句块 3；

}

举例子：如果今天下雨，我就带伞，并且如果下小雨我就骑车，否则我就坐车，不下雨我就不带伞。

```
#include <stdio.h>
#define RAIN 0
#define LIANG 0
int main()
{
   if(RAIN)
   {
        printf("with umbrella!\n");
        if(LIANG>50)
            printf("by car\n");
        else
            printf("walk\n");
   }
   else
       printf("no umbrella!\n");
    return 0;
}
```

练习：输入一个数，如果大于 0 打印大于 0 并且再进行判断是否大于 10，如果大于等于 10 打印大于等于 10，小于 10 打印小于。否则如果小于 0 打印不想要这个数。

```
#include <stdio.h>
int main()
{
    int a;
    scanf("%d", &a);
    if (a > 0)
    {
        printf(">0\n");
        if (a >= 10)
            printf(">=10\n");
        else
            printf("<10\n");
    }
    else
    {
        printf("i don't want this number\n");
    }

    return 0;
}
```

总结：

（1）首先判断表达式是否成立，如果成立就执行语句块 1，否则执行语句块 2。

（2）if 后面可以没有 else,但是 else 前面必须有 if。

（3）if 和 else 后面如果只有一行语句可以省略花括号{}。

1.2 switch case 语句

基本结构：

swtich(变量或表达式)

{

case 常量 1: 语句块 1;break;

case 常量 2: 语句块 2;break;

case 常量 3: 语句块 3;break;

...

case 常量 n: 语句块 n;break;

default:语句块 n+1;

}

执行顺序：

当变量或表达式的量与其中一个 case 语句中的常量相符时，就执行此 case 后面的语句，并依次向下执行后面所有 case 中的语句，除非遇到 break 语句就会跳出 swtich 语句为止。如果变量或者表达式都不相符，就会执行 default 后面的语句。

注意：

1. 表达式不能是浮点型或者字符串。
2. case 后面的 break 可以省略，但是省略时会顺序执行，指导遇到 break 就会结束。
3. 如果 case 的语句块和下面 case 中的语句块相同，则可以省略。形式如下：

case 常量 1：

case 常量 2：语句块 1；break;

练习：从终端输入一个学生的成绩，判断学生成绩，打印成绩级别用 switch 写

【90 - 100】 A

【80 - 89】 B

【70 - 79】 C

【60 - 69】 D

< 60 sorry you lost

```
#include <stdio.h>
int main()
{
    int score;
    scanf("%d", &score);
    switch (score / 10)
    {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        printf("you lost\n");
        break;
    case 6:
        printf("D\n");
        break;
    case 7:
        printf("C\n");
        break;
    case 8:
        printf("B\n");
        break;
    case 9:
    case 10:
        printf("A\n");
        break;
    default:
        printf("error!\n");
        break;
    }
    return 0;
}
```

作业：

1. 整理笔记，用思维导图的方式。

2.从终端输入一个日期

scanf ("%d %d %d ", &year , &mon , &day)，判断这是这一年的第几天

(提示：判断闰年还是平年，闰年且闰年二月份以上比平年多一天。普通年份除以 4，有余数是平年，对于整百的年份，比如 2100 年，要除以 400，有余数就是平年，整除就是闰年。)

方式 1：

```
#include<stdio.h>
int main(int argc, char const *argv[])
{
    int year,month,day,days=0,leap=0;
    scanf("%d %d %d",&year,&month,&day);
    if(year % 100==0)
    {
       if(year%400==0)
          leap=1;
       else
          leap=0;
    }
    else
    {
       if(year%4==0)
          leap=1;
       else
          leap=0;
    }
   // leap = (year % 4 == 0 && year % 100 != 0 || year % 400 ==
0);
    switch (month)
    {
    case 1:days = day;break;
    case 2:days = 31+day;break;
    case 3:days = 31+28+leap+day;break;
    case 4:days = 31+28+leap+31+day;break;
    case 5:days = 31+28+leap+31+30+day;break;
    case 6:days = 31+28+leap+31+30+31+day;break;
    case 7:days = 31+28+leap+31+30+31+30+day;break;
    case 8:days = 31+28+leap+31+30+31+30+31+day;break;
    case 9:days = 31+28+leap+31+30+31+30+31+31+day;break;
    case 10:days= 31+28+leap+31+30+31+30+31+31+30+day;break;
    case 11:days = 31+28+leap+31+30+31+30+31+31+30+31+day;break;
    case 12:days
=31+28+leap+31+30+31+30+31+31+30+31+30+day;break;
    default:
        printf("month err\n");
        return -1;
    }
    printf("这是这一年的第%d 天\n",days);
    return 0;
}
```

方式 2：

```
#include <stdio.h>
int main()
{
    int year = 0, month = 0, day = 0, days = 0, leap = 1;
    scanf("%d %d %d", &year, &month, &day);
    leap = (year % 4 == 0 && year % 100 != 0 || year % 400 == 0);
    switch (month)
    {
    case 12:
        days += 30;
    case 11:
        days += 31;
    case 10:
        days += 30;
    case 9:
        days += 31;
    case 8:
        days += 31;
    case 7:
        days += 30;
    case 6:
        days += 31;
    case 5:
        days += 30;
    case 4:
        days += 31;
    case 3:
        days += 28 + leap;
    case 2:
        days += 31;
    case 1:
        days += day;
        break;
    default:
        printf("error!\n");
        break;
    }
    printf("%04d-%02d-%02d:这一年的第%d 天\n", year, month, day,
days);
}
```

2. 循环语句

2.1 for 循环

2.1.1 基本结构

格式：

for(表达式 1;表达式 2;表达式 3)

{

　　语句块；

}

表达式 1：赋初值

表达式 2：判断条件

表达式 3：增值或减值语句

执行顺序：首先执行表达式 1 进行赋值，然后判断表达式 2 是否成立，如果成立就进入循环执行语句块，最后再执行表达式 3 增值或减值语句，然后再判断表达式 2 是否成立如果成立再进入循环，指导表达式 2 不成立的时候退出循环。

例子：循环打印 1 2 3 4 5

```
 for(int i=1;i<=5;i++)
        printf("%d\n",i);
```

例子：求 5！

```
#include<stdio.h>
int main(int argc, char const *argv[])
{
    int sum =1;
    for(int i=1;i<=5;i++)
        sum *=i;     //sum=sum*i;
    printf("%d\n",sum);
    return 0;
}
```

2.1.2 嵌套结构

格式：

for(表达式 1;表达式 2;表达式 3)

{

for(表达式 4;表达式 5;表达式 6)

{

语句块;

}

}

练习：用 for 循环打印 99 乘法表

1*1=1

1*2=2 2*2=4

1*3=3 2*3=6 3*3=9

1*4=4 2*4=8 3*4=12 4*4=16

1*5=5 2*5=10 3*5=15 4*5=20 5*5=25

1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36

1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49

1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64

1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81

```
#include<stdio.h>
int main(int argc, char const *argv[])
{
    for(int i=1;i<=9;i++)
    {     for(int j=1;j<=i;j++)
             printf("%d*%d=%d ",j,i,j*i);
         printf("\n");
    }
    return 0;
}
```

2.1.3 变形

(1)变形一：

表达式 1 ;

for(;表达式 2;表达式 3)

{

语句块;

}

(2)变形二：

表达式 1;

for(;表达式 2;)

{

语句块;

表达式 3;

}

(3) 变形三：

表达式 1;

for(;;) //死循环

{

if(表达式 2)

{

语句块;

```
#include<stdio.h>
int main(int argc, char const *argv[])
{
    int g=0,s=0,b=0,sum=0;
    for(int i=100;i<1000;i++)
    {
        g=i%10;
        s=i/10%10;
        b=i/100;
        sum=g*g*g+s*s*s+b*b*b;
        if(sum==i)
            printf("%d ",sum);
    }
    printf("\n");
    return 0;
}
```

练习 1.打印以下图案：

要求行数从终端输入。

输入：5

输出以下：

```
*
**
***
****
*****
```

```
#include<stdio.h>
int main(int argc, char const *argv[])
{
    int n=0;
    scanf("%d",&n);
    for (int i = 1; i <=n; i++)
    {
        for (int j = 1; j <=i ; j++)
        {
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

练习 2：

要求行数从终端输入。

输入：5

输出以下：

```
*
**
***
****
*****
*****
 ****
  ***
   **
    *
```

行数 i	空格数=i-1	星星数=5-i+1
1	0=1-1	5=5-1+1
2	1=2-1	4=5-2+1
3	2=3-1	3=5-3+1
4	3=4-1	2=5-4+1
5	4=5-1	1=5-5+1

```c
#include <stdio.h>
int main(int argc, char const *argv[])
{
    int n = 0;
    scanf("%d", &n);
    /*1. 实现上半部分打印*/
    for (int i = 1; i <= n; i++)     //行数
    {
        for (int j = 1; j <= i; j++) //打印每行星星数
        {
            printf("*");
        }
        printf("\n");                  //每行打印结束后换行
    }

    /*2. 实现下半部分打印*/
    for (int i = 1; i <= n; i++)             //行数
    {
        for (int k = 1; k < i; k++)          //实现打印每行空格数
            printf(" ");
        for (int j = 1; j <= n - i + 1; j++)//实现打印每行星星数
            printf("*");
        printf("\n");                          //每行打印结束后换行
    }
    return 0;
}
```

2.2 while 循环

格式：

定义循环变量并赋值;

while(判断条件)

{

    语句块;

增值或减值语句;

}

执行顺序：

首先定义循环变量并且赋值，然后判断条件是否满足，如果满足就进入循环执行语句块，以及执行增值或减值语句。然后再进行判断，如果满足再次进入循环，直到不满足之后退出循环。相当于 for 循环的变形。

例如: 求从 1 加到 100

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    int i=1,sum=0;
    while (i<=100)
    {
        sum+=i;
        i++;
    }
    printf("%d\n",sum);
    return 0;
}
```

2.3 do while 循环

格式：

do

{

语句块;

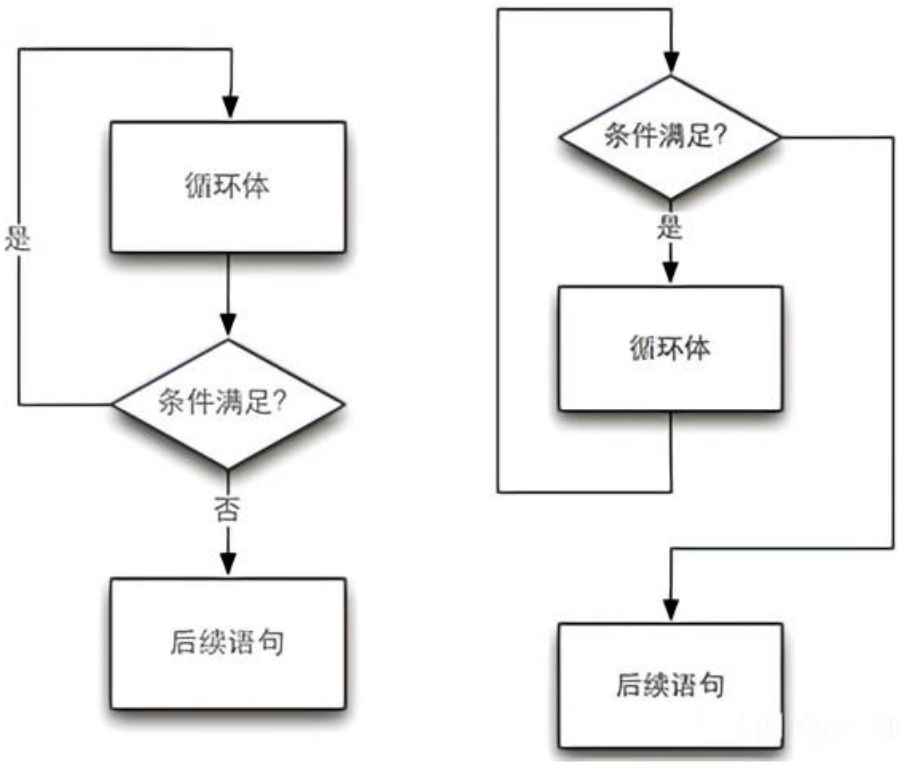
增值减值语句;

}while(判断条件);

执行顺序：先执行后判断

例如：实现 1-10 求和？

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    int i=1,sum=0;
    do
    {
        sum+=i;
        i++;
    } while (i<=10);

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



左边 do-while 右边 while

两者区别：

1. 执行顺序不同：do while 是先执行后判断，while 先判断后执行。
2. 执行次数不同：do while 至少会执行一次。

2.5 死循环

for(;;){}

while(1){}

while(1);

2.6 循环控制语句

break continue goto

（1）break：直接结束循环

（2）continue:结束本次循环，继续下一次循环

（3）goto:强制跳转到标签位置,可以应用于多重循环嵌套。

例如：

```
for(...)
{
    for(...)
    {
        for(...)
        {
            if(错误条件)
                goto error;
        }
    }
}
...
error:
    ...//处理情况
```

练习：打印 100 到 200 中能被三整除的数

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    char ch;
    while(1)
    {
        scanf("%c",&ch);
        //getchar();
        if(ch=='\n')
            continue;
        if(ch == 'q')
            break;
        printf("%c\n",ch);
    }
    return 0;
}
```

使用场景：

使用在循环语句中，做为结束循环的应用。使用时需要有判断条件。

练习：循环输入一个 5 位数，判断它是不是回文数。当输入 0 时循环结束，是回文数循环也结束。

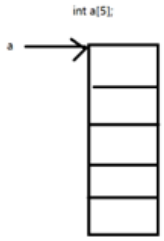
即 12321 是回文数，个位与万位相同，十位与千位相同。

1.2 定义格式

存储类型 数据类型 数组名[元素个数];

例如：(auto) int a[5];

数组名：代表数组的首地址，a 是地址常量，不能为左值，不能被赋值。



数组定义方法：

（1）数组名定义规则跟变量名相同，遵循标识符命名规则。

（2）数组名后使用方括号括起来的表达式常量，表示元素的个数。

（3）常量表达式中可以包含常量和符号常量，不能包含变量。

1.3 访问元素

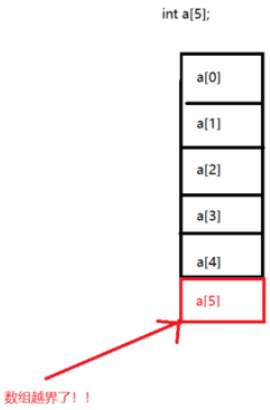
数组名[下标];

注意：下标从 0 开始

例如：

访问第一个元素: a[0];

访问第 n 个元素: a[n-1];



特点：

数据类型相同

在内存中连续

注意：

（1）数组的数据类型就是其元素的数据类型

（2）数组名要符合标识符命名规则

（3）在同一个函数中，数组名不要与变量名相同。

例如：int a[5];

int a;//错误

（4）数组下标从 0 开始，到 n-1 结束。

2. 一维数组

一维数组、二维数组

2.1 一维数组的概念

是有一个下标的数组

格式：存储类型 数据类型 数组名[元素个数];

访问元素: 数组名[下标]；

下标从 0 开始

数组名：数组的首地址

2.2 初始化

【1】定义时全部初始化

```
        int a[5]={1,2,3,4,5};
        printf("%d\n",a[0]);
        printf("%d\n",a[1]);
        printf("%d\n",a[2]);
        printf("%d\n",a[3]);
        printf("%d\n",a[4]);
```

【2】定义时部分初始化：

未初始化元素值为 0

```
        int a[5]={1,2};
        printf("%d\n",a[0]);
        printf("%d\n",a[1]);
        printf("%d\n",a[2]);
        printf("%d\n",a[3]);
        printf("%d\n",a[4]);
```

【3】先定义，后初始化。需要单独对每个元素赋值。

```
#include <stdio.h>
int a,b,c,sum;
int main(int argc, char const *argv[])
{
    scanf("%d %d",&a, &b);
    if(a>=b)              //用于两数交换，把小的数放前面。
    {
        c=a;
        a=b;
        b=c;
    }
    while(a<=b)          //循环包含两个数在内的所有数
    {
        if(a%2==0)        //   进行判断，如果是偶数就累加。
            sum+=a;
        a++;
    }
    printf("%d\n",sum);
}
```

2.3 定义空数组

【1】 int a[5]={0,0,0,0,0};

【2】int a[5]={0};

【3】int a[5]={};

2.4 数组引用

（1）先定义后引用

（2）每次只能引用一个数组元素 a[i], 如果想引用所有元素可以通过循环遍历数组。

（3）引用时防止数组越界

（4）打印数组元素地址用%p 的格式

例如：

int a[10];

printf("%p",a);

printf("%d",a); //错误，数组名是数组的首地址。

如果想遍历数组可以通过循环：

```
#include<stdio.h>
int main(int argc, char const *argv[])
{
    int a[5]={1,2,3,4,5};
    for(int i=0;i<5;i++)
        printf("%d ",a[i]);
    return 0;
}
```

练习：输入 5 个数组元素，然后依次打印出来。

```
    int a[5]={};
    for(int i=0;i<5;i++)
        scanf("%d",&a[i]);

    for(int i=0;i<5;i++)
        printf("%d ",a[i]);
```

2.5 数组的大小

例如：

```
    int a[5];//20
    double b[2];//16
    char c[32];//32
    printf("%d %d %d\n", sizeof(a), sizeof(b), sizeof(c));
```

计算方式：

（1）数组元素的个数 * 数据类型的大小

（2）sizeof(数组名)

数组元素的个数=sizeof(数组名)/sizeof(数组名[下标])

例如： printf("%d\n",sizeof(a)/sizeof(a[0]));

练习：计算斐波那契数列前 15 项并逆序输出

1 1 2 3 5 8 13 21 .....

```
#include <stdio.h>
#include <strings.h>
int main(int argc, char const *argv[])
{
    int a[10] = {1, 2, 3};
    for (int i = 0; i < 10; i++)
        printf("%d ", a[i]);
    printf("\n");
    bzero(a, sizeof(a));
    for (int i = 0; i < 10; i++)
        printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

打印结果：

1 2 3 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

2.6.2 memset()函数

```
#include <string.h>
void *memset(void *s, int c, size_t n);
```

功能：将指定内存空间设为 0
参数：
s:指定空间的首地址
c:要设置的数，一般为 0
n:要设置的内存大小
返回值：无

案例：

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    int a[10] = {1, 2, 3};
    for (int i = 0; i < 10; i++)
        printf("%d ", a[i]);
    printf("\n");
    memset(a,0,sizeof(a));
    for (int i = 0; i < 10; i++)
        printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

3. 字符数组

因为语言中没有字符串类型，所以可以用字符数组的形式去表示字符串。

概念：类型为字符的数组，也就是说数组中每个元素是字符类型。可以组成字符串。

3.1 表示形式

(1)单个字符表示：

char s[]={ ‘h’ ,’ e’ ,’ l’ ,’ l’ ,’ o’ }; //sizeof(s)=5

char s1[5]={ ‘h’ ,’ e’ ,’ l’ ,’ l’ ,’ o’ }; //sizeof(s1)=5

(2)用字符串表示,结尾会自动加上 ‘\0 ‘

char s2[]=” hello” ; //sizeof(s2)=6

char s3[]={ “hello” }; //sizeof(s3)=6

char s4[6]={ “hello “} //sizeof(s4)=6

char s5[]={};//0 错误

/*验证字符数组大小*/

printf("%d %d\n",sizeof(s),sizeof(s1));

printf("%d %d %d\n",sizeof(s2),sizeof(s3),sizeof(s4));

初始化和数组规则相同

3.2 输入和输出

3.2.1 输入

（1）用 scanf:

```
char s[32]={};
scanf("%s",s);
```

不能输入空格，如果输入空格，空格后面内容不会存入数组。例如输入 hello word, 只能存入 hello。

如需要保存带空格的字符串如下：

```
char s[32]={};
scanf("%[^\n]",s); //可以输入除了\n 以外的字符，遇到\n 结束。
printf("%s\n",s);
```

注意：输入字符串会在结尾自动加“\0”。

(2) 用循环遍历输入

例如：

```
char s[6]={};
for(int i=0;i<6;i++)
    scanf("%c",&s[i]);
printf("%s",s);
```

(3) gets()

```
#include <stdio.h>
char *gets(char *s);
```

功能：从终端获取字符串输入
参数：s:目标字符串的首地址
返回值：目标字符串的首地址
没有数组越界检查，使用时会报警告。

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    char s[32] = {};
    int sum = 0;
    gets(s);
    int i=0;
    while (s[i]!='\0')
    {
        if(s[i] == ' ')
            sum++;
        i++;
    }
    printf("%d\n", sum);
    return 0;
}
```

3.2.2 输出

（1）用 printf():

```
    char s[32]="hello";
    printf("%s",s);
```

(2) for 循环

```
    for(int i=0;i<32;i++)
        printf("%c",s[i]);
```

(3) puts

#include <stdio.h>
int puts(const char *s);
功能：向终端输出字符串
参数：要输出字符串的首地址
返回值：字符串长度

例子：

3.3 计算字符串长度

3.3.1 用循环统计

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    char string[15]="hello world";
    int i=0;
    while(string[i]!='\0')
        i++;
    printf("%d\n",i);
    return 0;
}
```

3.3.2 用 strlen()

#include <string.h>
size_t strlen(const char *s);
功能：计算字符串长度
参数：字符串的首地址
返回值：返回字符串实际长度，不包括'\0'在内。
例如：

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    char s[32]={};
    int len=0;
    scanf("%[^\n]",s);
    len=strlen(s);
    for(int i=0;i<len;i++)
    {
        if(s[i]>='a' && s[i]<='z')
            s[i]-=32;
        else if(s[i]>='A' && s[i]<='Z')
            s[i]+=32;
    }
    printf("%s\n",s);
    return 0;
}
```

4. 排序

4.1 冒泡排序

4.1.1 排序过程

（1）比较第一个数和第二个数，如果前面的数比后面的大，则交换。然后比较第二个和第三个数，也是如果前面的数比后面的大则交换。以此类推，直到比较完第 n-1 个和第 n 个数为止，第一趟冒泡排序结束。排序结果是最大的数被放在最后一个元素的位置上。

（2）把前 n-1 个数进行第二趟冒泡排序，结果是次大的数排在第 n-1 个的元素的位置上。

（3）重复上述过程，一共经过 n-1 趟冒泡排序，排序结束。

第一轮：4=5-1 次排序

第二轮：3=5-2 次排序

第三轮：2=5-3 次排序

第四轮：1=5-4 次排序

设元素个数为 N：

总轮数=元素个数-1=N-1

每轮排序次数=元素个数-轮数=N-i

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    int a[5]={5,4,3,2,1},temp=0;
    for(int i=1;i<5;i++)
    {
        for(int j=0;j<5-i;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
    for(int i=0;i<5;i++)
        printf("%d ",a[i]);
    printf("\n");
   return 0;
}
```

练习：一组数进行冒泡排序：100，34，56，78，99，2

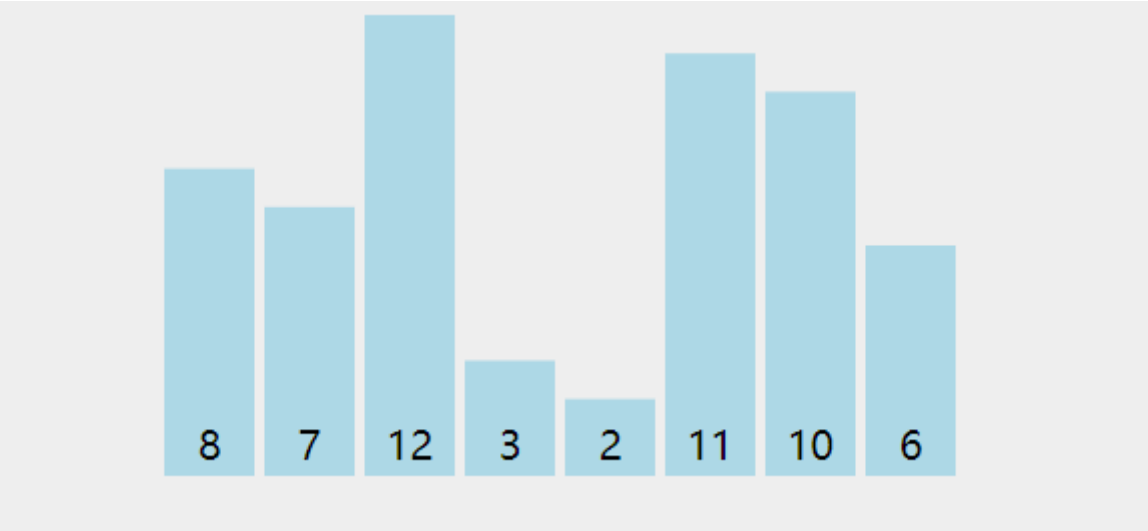
4.2 选择排序

4.2.1 排序过程

（1）首先通过 n-1 次比较，从 n 个数中找到最小的，将它与第一个数进行交换，这是第一趟选择排序。结果是最小的数被安置到第一个元素的位置上。

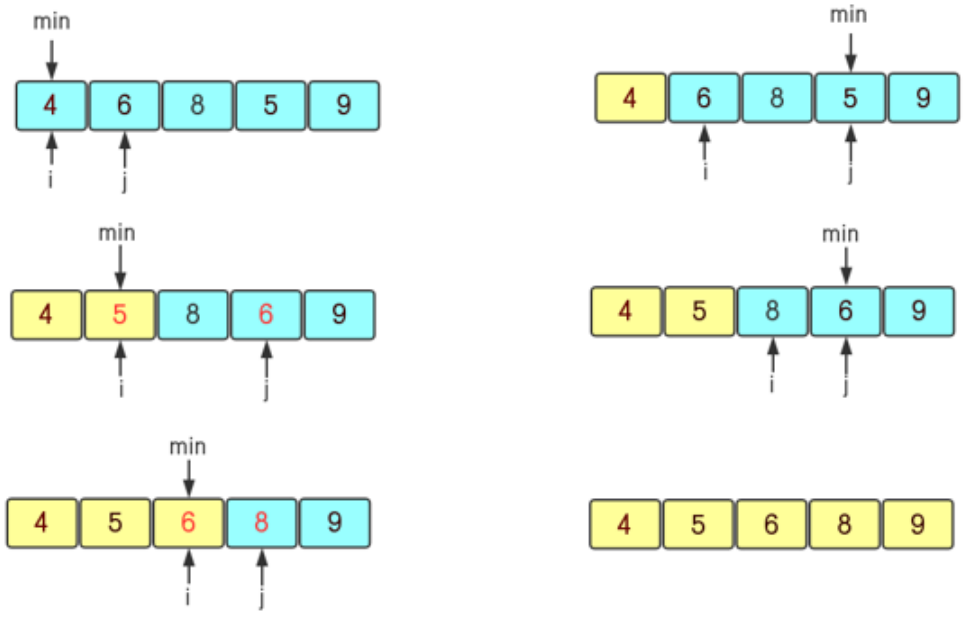
（2）再通过 n-2 次比较，从剩余的 n-1 个数中找到次小的数，将它与第二个数进行交换，这是第二趟选择排序。结果是次小的数排在第二个元素的位置上。

（3）重复上述过程，总共经历 n-1 次排序，排序结束。



4.2.2 原理

第一次排序存取最小值，然后与第一个数进行交换，以此类推。



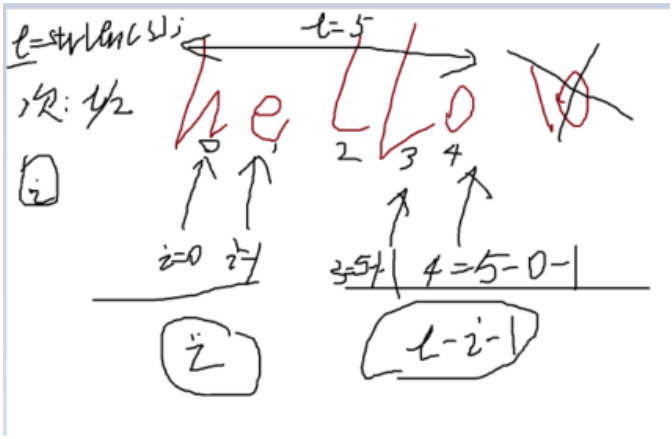
4.2.3 编程

把 5,4,3,2,1,56,7,8,9,34 进行选择排序

```
#include<stdio.h>
#define N 10
int main(int argc, char const *argv[])
{
    int arr[N]={5,4,3,2,1,56,7,8,9,34},min=0,temp=0;
    for (int i = 0; i < N-1; i++)            //循环轮数
    {
        min=i;                               //设最小值下标初值为还未
排序的第一个元素下标
        for(int j=i+1; j<N;j++)              //把要比较的元素和其后面
所有元素通过循环进行比较
        {
            if(arr[min]>arr[j])              //通过每次的比较，找出最
小元素的下标。
                min=j;
        }

        if(arr[min]!=arr[i])                 ///如果记录的最小值元素和
用来比较的元素不一样则交换
        {
            temp=arr[min];
            arr[min]=arr[i];
            arr[i]=temp;
        }

    }

    for(int k=0;k<N;k++)
    {
        printf("%d ",arr[k]);
    }
    printf("\n");

    return 0;
}
```

作业：

1. 用思维导图的方式整理复习笔记
2. 写选择排序的程序
3. 将一串字符串进行倒置 char buf[32]=” hello” ;//olleh



```c
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    char str[32] = "hello",c=0;
    int l=strlen(str); //l=5

    for(int i=0;i<l/2;i++)
    {
        c=str[i];
        str[i]=str[l-i-1];
        str[l-i-1]=c;
    }
    printf("%s\n",str);

    return 0;
}
```

4. 在终端输入一个字符串，计算大写字母、小写字母、空格、数字个数，分别在终端输出他们的个数。

a[1][2]:最后一行最后一列的元素。

设 n 行 m 列：a[n-1][m-1]就是最后一行最后一列的元素。

注意：

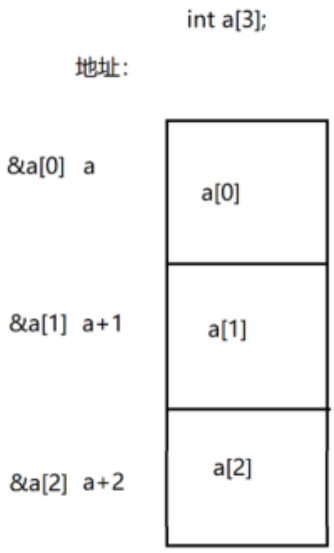
（1）行下标和列下标都不能越界。

（2）行数可以省略，列数不能省略。

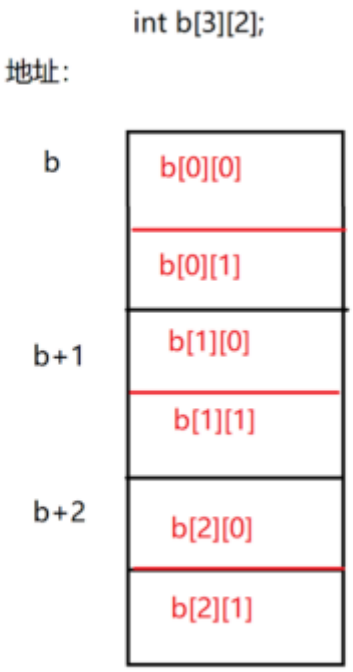
（3）数组名代表的是第一行的首地址

例如：一维数组和二维数组对比

一维数组：



二维数组：



5.3 二维数组元素的个数和二维数组的大小

元素个数：行数*列数

二维数组大小：

（1）数据类型大小*行数*列数

（2）sizeof(数组名)

5.4 二维数组数组名

二维数组数组名代表第一行首地址

例如：int a[2][3]={1,2,3,4,5,6};

a: 第一行的首地址

a+1:第二行的首地址

设 n 行,最后一行首地址是：a+n-1

5.5 初始化

【1】全部初始化

int a[2][3]={1,2,3,4,5,6}; //顺序赋值

int b[2][3]={{1,2,3},{4,5,6}}//按行赋值

【2】部分初始化

未初始化的元素值为 0

int a[2][3]={1,2,3,4,}; //顺序赋值 1 2 3 4 0 0

int b[2][3]={{1,2},{4,5}//按行赋值 1 2 0 4 5 0

【3】未初始化

随机值，需要单独赋值。

5.6 遍历二维数组

for 循环嵌套，外层行数，内层列数。

```
int a[n][m]={};
for(int i=0;i<n;i++)//行下标
{
	for(int j=0;j<m;j++)
	{
		//scanf()/printf()
	}
}
```

练习：设一个二维数组，每个数等于行和列数的乘积。

```
#include <stdio.h>
int a[3][3]={0};
int main(int argc, char const *argv[])
{
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            a[i][j]=i*j;
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

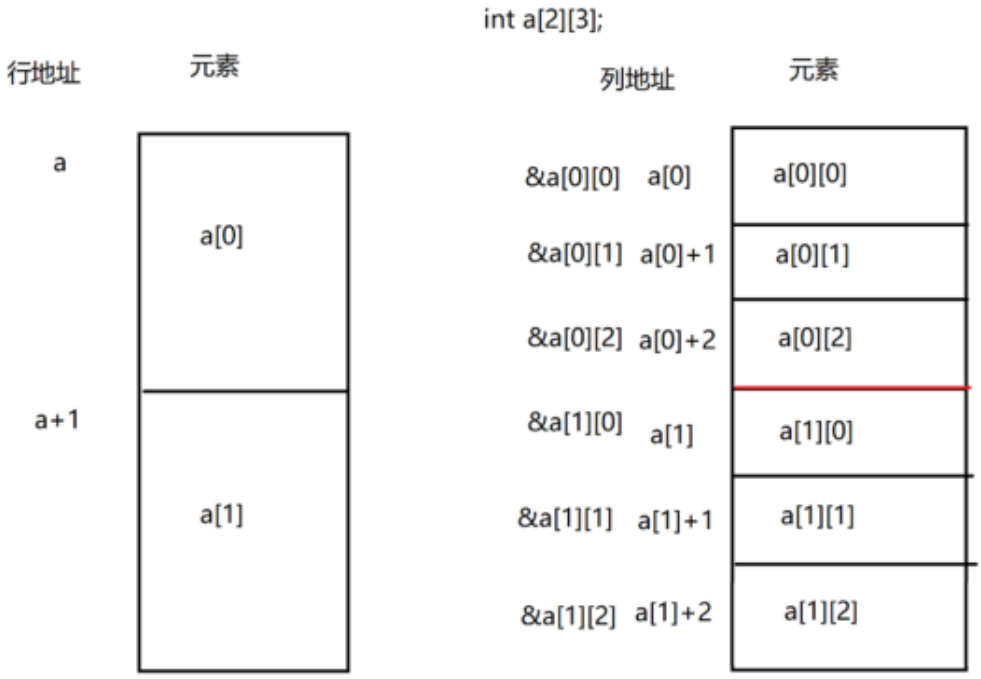
a[0][0]=0; a[0][1]=0;a[0][2]=0

a[1][0]=0; a[1][1]=1;a[1][2]=2

a[2][0]=0; a[2][1]=2;a[]2[2]=4

5.7 内存分配

int a[2][3];



(1) 行地址

1. 在二维数组中，数组名 a 的值，表示的实际含义是第一行首地址。这个数值还等于整个数组的首地址，还等于第一个元素的地址（&a[0][0]）。
2. a+1 代表第二的首地址。这个数值还等于第二行第一个元素的地址（&a[1][0]）。

（2）列地址

1. 二维数组中，a[0]的值是第一行第一列的地址，也就是该数组中首个元素的地址(&a[0][0])
2. a[0]+1 的值，是数组元素第一行第二列的地址(&a[0][1])

```
        int a[2][3]={1,2,3,4,5,6};
        printf("%p %p\n",a,a+1);
        printf("%p %p\n",a[0],a[1]);
        printf("%p %p\n",&a[0][0],&a[1][0]);

        printf("%p %p\n",a[0]+1,a[1]+1);
        printf("%p %p\n",&a[0][1],&a[1][1]);
```

打印结果：

0xbf858184 0xbf858190
0xbf858184 0xbf858190
0xbf858184 0xbf858190
0xbf858188 0xbf858194
0xbf858188 0xbf858194

3、在执行语句： int a[ ][3]={1,2,3,4,5,6}; 后，
a[1][0]的值是_____。
A) 4　　B) 1　　C) 2　　D) 5

A

4、在定义 int a[5][6];
后，数组 a 中的第 10 个元素是_____。（设 a[0][0]为第一个元素）
A) a[2][5]　　B) a[2][4]　　C) a[1][3]　　D) a[1][5]

C

5、下列程序执行后的输出结果是_____。

```
main()
{ int i,j,a[3][3];
  for(i=0;i<3;i++)
    for(j=0;j<=i;j++)  a[i][j]=i*j;
  printf("%d,%d\n",a[1][2],a[2][1]);
}
```

A) 2,2　　B) 不定值,2　　C) 2　　D) 2,0

B

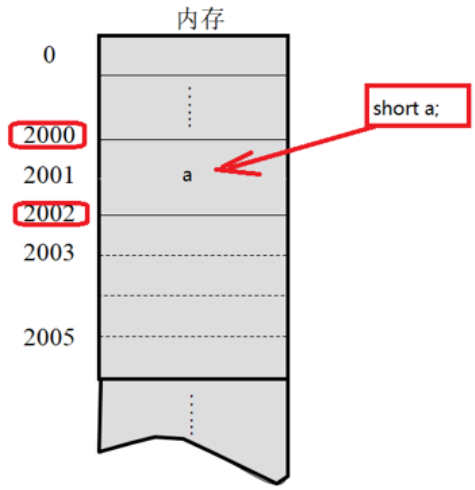
1. 指针的基本用法

1.1 指针的概念

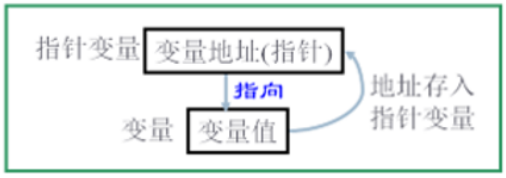
内存地址：内存中每个字节单位都有一个编号（一般用十六进制表示）

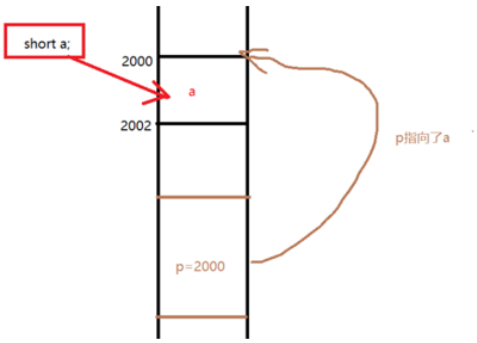
指针：指针就是内存地址

指针变量：用于存放地址的变量就叫做指针变量



指针变量画图展示：





1.2 格式

存储类型 数据类型 *指针变量名；

int *p; //定义了一个指针变量 p,指向的数据是 int 类型的。

```
        int a = 5;
        int *p = &a;
        char c='v';
        char *q=&c;
        printf("%p %p\n", p, &a);
        printf("%p %p\n",q,&c);
        printf("%d %d\n",a,*p);
        printf("%c %c\n",c,*q);
```

访问指针所指向空间的内容用取内容运算符*

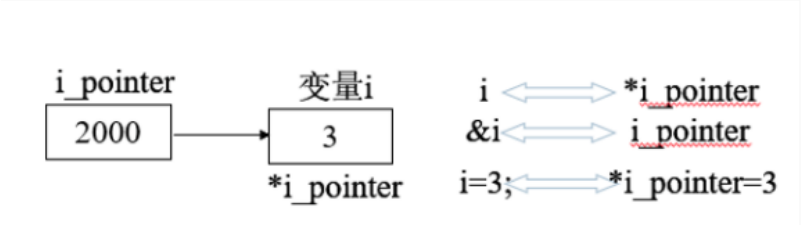
那么 p 变量存放的就是 a 的地址，q 变量存放的是 c 的地址。

符号*可以访问地址里面的内容。

指针与所指变量之间的关系如下图：

int i=3;

int *i_potinte=&i;



2.3 指针操作符

&：取地址符：取变量的地址

*：取内容符：取地址里面存放的内容

*&a=a;//*和&是互逆运算

&*a//错误（因为运算符优先级）

2.4 指针变量初始化和赋值

指针变量使用前不仅要定义，还要初始化。未初始化的指针变量不能随便使用，不然可能会产生野指针。

（1）将普通变量的地址赋值给指针变量

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    int a=10;
    int *p = &a;//定义的同时赋值
    int *q=NULL;//可以初始化为空指针
    int *p2;
    p2=p;//先定义后赋值

    printf("%d %d %d\n",a,*p,*p2);//打印 a 的值
    printf("%p %p %p\n",&a,p,p2); //打印 a 的地址
    *p=20;                        //同过指向 a 的指针改变 a 的值
    printf("%d %d %d\n",a,*p,*p2);//打印 a 的值
    printf("%p %p %p\n",&a,p,p2); //打印 a 的地址
    return 0;
}
```

(2) 将数组的首地址赋值给指针变量

```
        char s[10]="hello";
        char *p = s;
        int arr[5]={1,2,3,4,5};
        int *q=arr;
        printf("%c\n",*p);//h
        printf("%d\n",*q);//1
```

(3) 将指针变量里面保存的地址赋值给另一个指针变量

```
        int a=10;
        int *p=&a;
        int *q=p;
        printf("%d %d %d\n",a,*p,*q);
        *q=20;//通过指针改变变量 a 的值
        printf("%d %d %d\n",a,*p,*q);
```

p 和 q 都指向了变量 a

2. 指针运算

2.1 算术运算- +

对指针加减操作其实就是让指针向前向后移动

```
        char s[10]="hello";
        char *p1=s;
        char *p2=&s[2];
        if(p2>p1)
            printf("p2>p1");
        else
            printf("p2<p1");
```

练习：以下程序打印出来什么

```
    char s[32]="hello";
    char *p=s;
    p++;
    char y=(*--p)++;
    printf("%c",y);//h
```

练习：以下程序打印什么结果

3. 指针的大小和段错误

3.1 指针的大小

计算指针大小用：sizeof(指针变量名)

```
        int a=5;
        short b=6;
        char c='a';
        int *p1=&a;
        short *p2=&b;
        char *p3=&c;
        printf("%d %d %d\n",sizeof(p1),sizeof(p2),sizeof(p3));//4 4 4
    4
```

总结：

1.32 位操作系统指针大小是 4 字节。64 位操作系统，指针大小是 8 字节。

2. 内存地址是固定的，但是变量的地址不是固定的。
3. 指针类型根据指针指向的空间的数据类型来匹配

作业：

1. 整理笔记，用思维导图的方式。
2. 利用指针判断输入的字符串是否为回文

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
   char a[32]={};
   char *p1,*p2;
   int flag=1;
   gets(a);
   p1=a;
   p2=a+strlen(a)-1;
   while(p2>p1)
   {
       if(*p1!=*p2)
            flag=0;
       p1++;
       p2--;
   }
    if(flag==0)
        printf("no!\n");
    else
        printf("yes!\n");
    return 0;
}
```

3. 编写一个程序实现功能：将字符串 “Computer Science” 赋值给一个字符数组，然后从第一个字母开始间隔的输出该字符串，用指针完成。结果：Cmue cec

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    char string[32] = "Computer Science";
    char *p = string;
    while (*p != '\0')
    {
        printf("%c", *p);
        p += 2;
    }
    printf("\n");
    return 0;
}
```

3.2 使用指针时报段错误

段错误： Segmentation fault (core dumped)

（1）野指针，没有规定指向的指针会在内存中乱指。

野指针产生的原因，主要有两种：

【1】指针变量没有初始化

【2】指针被 free 之后，没有置 NULL，会让人认为时合法指针。

这两种情况都是没有给指针规定指向。

例如：int *p;

scanf("%d", p);

printf("%d\n",*p);

修改：

```
        int a=10;
        int *p=&a;
        scanf("%d",p);
        printf("%d\n",*p);
```

(2)内存泄露，对非法空间进行赋值。

练习 1.将字符串转换成整型数字输出。用指针实现

```
    #include <stdio.h>
    #include <string.h>
    int main(int argc, char const *argv[])
    {
        char a[32]="123";
        char *p=a;
        int num=0;
        while(*p != '\0')
        {
            num = num*10+(*p-48);
            p++;
        }
        printf("num=%d\n",num);
        return 0;
    }
```

4.指针修饰

4.1 const 常量化

（1）修饰普通变量

const int a=10;

int const b=10;

此时 a 和 b 只读，不可以修改。但是可以通过指针修改。

```
const int a=10;
int const b=20;
int *p=&a,*q=&b;
printf("%d %d\n",a,b);//10 20
*p=1;
*q=2;
//a=1;//error: assignment of read-only variable ‘a’
//b=2;//error: assignment of read-only variable ‘b’
printf("%d %d\n",a,b);//1 2
```

(2) 修饰指针指向的内容

也就是修饰*p,此时指针指向的内容不能改变，但是指向可以改变。

const int *p;

int const *p;

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    int a=10;
    int b=20;
    int const *p=&a;
    const int *q=&b;
    printf("%d %d\n",a,b);//10 20
    //*p=1;//*p 只读不可以被更改
    //*q=2;//*q 只读不可以被更改
    p=q;
    printf("%d %d\n",*p,*q);//20 20
    return 0;
}
```

(3) 修饰指针指向

int *const p;

此时 const 修饰 p,指针指向不能被更改，但是指向的内容可以被更改。

```
int a=100;
void *p=&a;
int *q=(int *)p;
printf("%d %d\n",*(int *)p,*q);
```

现用现转，或者转换之后赋值给另外一个变量然后应用另外一个变量。

4.3 大小端

在计算机进行超过 1 字节数据进行存储时，会出现存储数据顺序不同的情况即大小端存储

大端：数据的低位存储在高地址位，数据的高位存储在低地址位,大端字节序称为 MSB

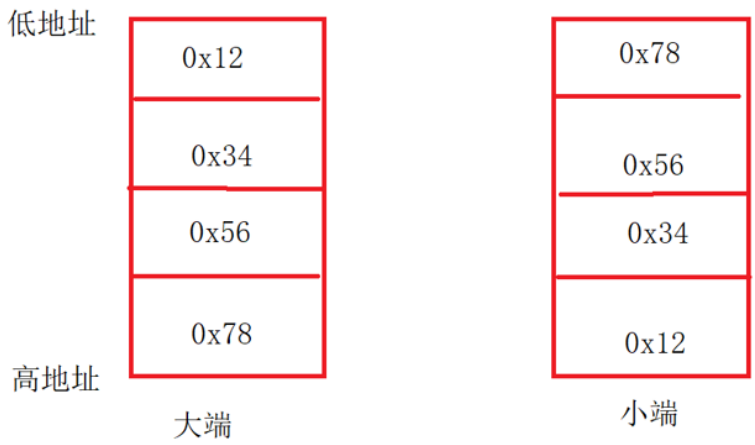
小端：据的低位存储在低地址位，数据的高位存储在高地址位,小端字节序称为 LSB

举例：存储数据：0x12345678, 假设起始地址是 0x4000

地址：0x4000 0x4001 0x4002 0x4003

小端：0x78　0x56　0x34　0x12

大端：0x12　0x34　0x56　0x78



```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    int a=0x12345678;
    char *p=(char *)&a;

    if(*p==0x78)
        printf("LSB\n");
    else if(*p==0x12)
        printf("MSB\n");

    return 0;
}
```

5. 二级指针

一级指针：存放变量的地址

二级指针：存放一级指针的地址

5.1 格式

存储类型 数据类型 **指针变量名；

int a = 10;

int *p = &a;

int **q = &p;

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    int a=10;
    int *p=&a;
    int **q=&p;
    printf("%d %d %d\n",a,*p,**q);
    printf("%p %p %p\n",&a,p,*q);
    printf("%p %p\n",&p,q);
    return 0;
}
```

结果：
10 10 10
0xbf9839b0 0xbf9839b0 0xbf9839b0
0xbf9839b4 0xbf9839b4

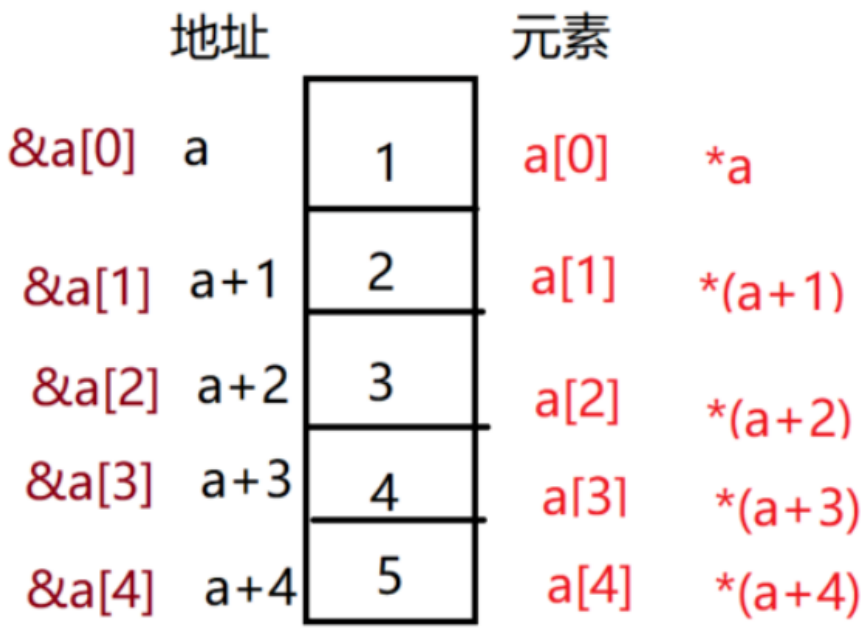
6 指针和一维数组

6.1 用法

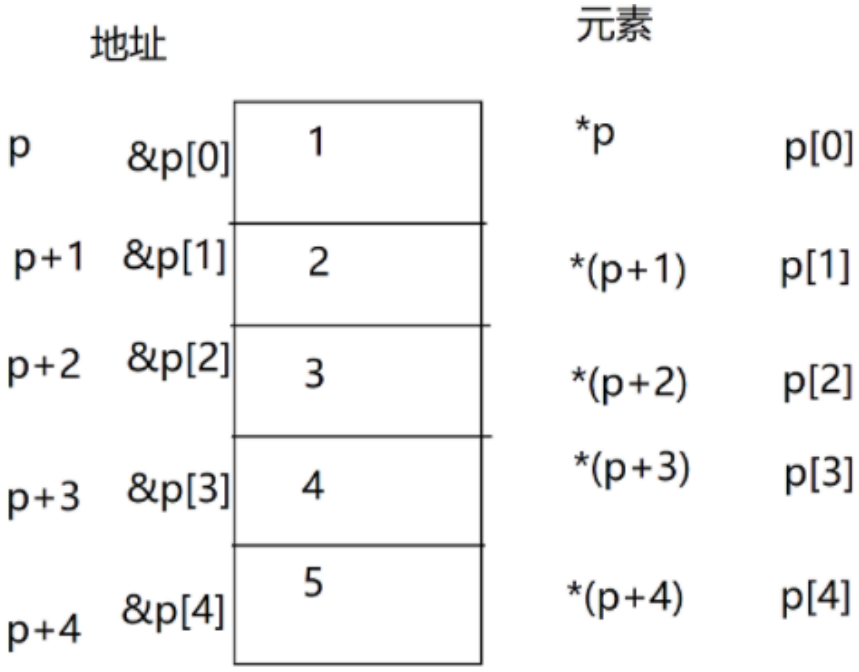
int a[5]={1,2,3,4,5}; //a 是数组名，也是数组首地址。

int *p=a;

直接访问：



通过地址间接访问：



访问数组元素 a[i]的地址：

直接访问：&a[i] a+i

间接访问：&p[i] p+i

访问数组元素 a[i]的内容：

直接访问：a[i] *(a+i)

间接访问：p[i] *(p+i)

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    int a[3]={1,2,3};
    int *p=a;
    printf("%p %p %p %p\n",a,p,&a[0],&p[0]);
    printf("%p %p %p %p\n",a+1,p+1,&a[1],&p[1]);
    printf("%d %d %d %d\n",a[0],p[0],*a,*p);
    printf("%d %d %d %d\n",a[1],p[1],*(a+1),*(p+1));
    return 0;
}
```

打印结果：

0xbfb27e90 0xbfb27e90 0xbfb27e90 0xbfb27e90
0xbfb27e94 0xbfb27e94 0xbfb27e94 0xbfb27e94
1 1 1 1
2 2 2 2

练习：用指针实现字符串 “hello” 的倒叙打印。

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    char str[32]="";
    char *p1=str,*p2=NULL;
    int temp=0;
    gets(str);
    p2=p1+strlen(str)-1;
    while (p2>p1)
    {
        temp=*p1;
        *p1=*p2;
        *p2=temp;
        p1++;
        p2--;
    }
    printf("%s\n",str);

    return 0;
}
```

题:打印杨辉三角形前十行

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

1 6 15 20 15 6 1

```
#include <stdio.h>
int main()
{
    int a[10][10], i, j;
    /*负责对角线和每行第一列*/
    for (i = 0; i < 10; i++)
    {
        a[i][i] = 1;
        a[i][0] = 1;
    }

    /*负责第三行起，每个数等于左上方和正上方的数相加*/
    for (i = 2; i < 10; i++)
    {
        for (j = 1; j <= i - 1; j++)
            a[i][j] = a[i - 1][j - 1] + a[i - 1][j];
    }

    for (i = 0; i < 10; i++)         //打印结果
    {
        for (j = 0; j <= i; j++)
            printf("%4d ", a[i][j]);
        printf("\n");
     }

    return 0;
}
```

6.2 运算方法

（1）++和*都是单目运算符

（2）单目运算符从右向左运算

（3）++在前是先移动再取值，++在后是先取值再移动。

int a[5]={1,2,3,4,5};

int *p=a;

//printf("%d\n",*(p++)); //1

//printf("%d\n",++*p); //2

//printf("%d\n",++(*p));//同上

//printf("%d\n",*++p);//2

printf("%d\n",*(++p));//同上

6.3 指针和二维数组

6.3.1 数组名表示

例如：

int a[2][3]={1,2,3,4,5,6};

a 是数组名，表示第一行的首地址。

a+1 表示第二行的首地址，以此类推。

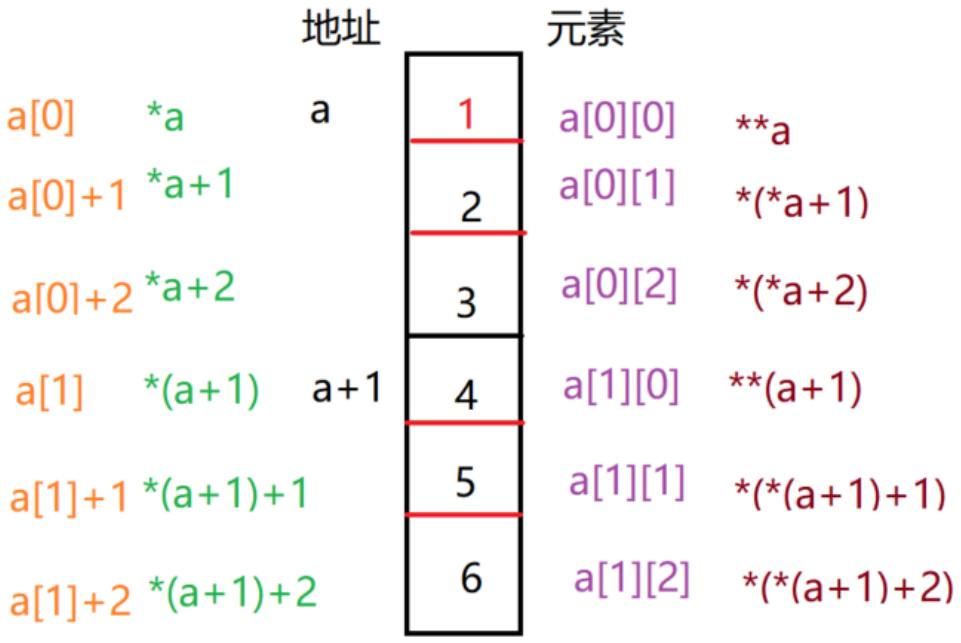
在 a 前面加*，表示将行地址降级称为列地址。

*a;//第一行第一列的地址

*a+1;//第一行第二列的地址

*(a+1);//第二行第一列的地址

*(a+1)+1;//第二行第二列的地址



访问数组元素地址（a[i][j]的地址）：

&a[i][j]

*(a+i)+j

a[i]+j

访问数组元素值：

a[i][j]

((a+i)+j)

*(a[i]+j)

代码验证：

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    int a[2][3]={1,2,3,4,5,6};
    for(int i=0;i<2;i++)
        for(int j=0;j<3;j++)
            printf("%d %d\n",*(a[i]+j),*(*(a+i)+j));

    return 0;
}
```

打印结果：

1 1

2 2

3 3

4 4

5 5

6 6

7. 数组指针

7.1 概念

本质还是指针，指向的是数组（又称为行指针）

7.2 定义格式

存储类型 数据类型(* 指针变量名)[列数];

例如：

int a[2][3]={1,2,3,4,5,6};

int (*p)[3]=a;

p 可以代替进行元素访问，但本质不同，p 是指针变量，a 是地址常量。

把 p 进行运算的时候，例题中情况要 3 个单位 3 个单位进行运算。

访问数组元素地址（a[i][j]的地址）：

*(p+i)+j

p[i]+j

访问数组元素值：

((p+i)+j)

*(p[i]+j)

例如：用数组指针遍历二维数组。

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    int a[2][3] = {1, 2, 3, 4, 5, 6};
    int(*p)[3] = a;

    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
            //printf("%d ", *(p[i] + j));
            printf("%d ",*(*(p+i)+j));
        printf("\n");
    }
    return 0;
}
```

练习：

有一个班，3 个学生，各学 4 门课，计算总平均分以及输出第 n 个学生的成绩。

int a[3][4] = {65,55,23,57,52,67,64,80,90,42,75,92};

思路：用数组指针的方式把每个学生分别当成一个数组，int (*p)[4]表示，这样每列的元素就代表每门课的成绩。然后循环嵌套算出总成绩，总成绩除以 12 算出平均成绩。然后通过循环，找到第几个学生的成绩输出。

```
#include <stdio.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    int a[3][4]={65,55,23,57,67,80,90,42,75,92};
    int (*p)[4]=a;
    int sum=0,ave=0,n=0;
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<4;j++)
            sum+=*(p[i]+j);
    }
    ave=sum/12;
    printf("sum=%d ave=%d\n",sum,ave);
    scanf("%d",&n);
    if(n<=3&&n>0)
    {
        for(int k=0;k<4;k++)
            printf("%d ",*(*(p+n-1)+k));
        printf("\n");
    }
    return 0;
}
```

7.3 大小

sizeof(p)=4

因为本质还是指针，所以大小都是四字节。

8. 指针数组

8.1 定义

所谓指针数组是指若干个具有相同存储类型和数据类型的指针变量构成的集合。

其本质是数组，里面存放的是指针。

8.2 格式

存储类型 数据类型 *数组名[元素个数];

指针数组名表示的是该指针数组的首地址。

例如：

int *arr[2];

8.3 应用

(1) 用于存放普通变量的地址

例如：

int a=10,b=20,c=30;

int *p[3]={&a,&b,&c};

访问 b 的值：

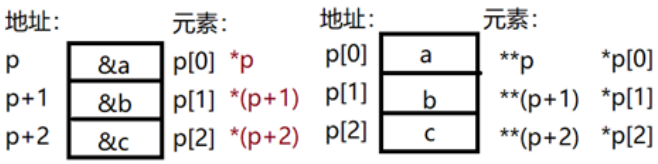
*p[1]

**(p+1)

访问 b 的地址：

p[1]

*(p+1)



```
        int a=10,b=20,c=30;
        int *p[3]={&a,&b,&c};
        printf("%p %p %p\n",p[0],p[1],p[2]);
        printf("%p %p %p\n",*p,*(p+1),*(p+2));

        printf("%d %d %d\n",*p[0],*p[1],*p[2]);
        printf("%d %d %d\n",**p,**(p+1),**(p+2));
```

(2) 用于存放二维数组的每行第一个元素的地址(列地址)

例如：

int a[2][3]={1,2,3,4,5,6}

int *p[2]={a[0],a[1]};

访问 a[1][2]的值：

*(p[1]+2)

((p+1)+2)

访问 a[1][2]的地址：

p[1]+2

*(p+1)+2

大小：sizeof(p);//8,因为包含两个指针，每个指针大小为 4。

访问地址：p[i]+j

*(p+i)+j

访问元素：*(*(p+i)+j)

*(p[i]+j)

(3) 用于存放字符串

char *p[3]={"hello","world","hqyj"};

练习：遍历字符串中每一个元素

```c
#include <stdio.h>
int main(int argc, char const *argv[])
{
    char *p[3] = {"hello", "world", "hqyj"};
    /*
    for(int i=0;i<3;i++)
    {
        int j=0;
        while(*(p[i]+j)!='\0')
        {
            //printf("%c",*(p[i]+j));
            printf("%c",*(*(p+i)+j));
            j++;
        }
        printf("\n");
    }
    */
    for (int i = 0; i < 3; i++)
        // printf("%s\n",*(p+i));
        printf("%s\n", p[i]);
    return 0;
}
```

练习：用行指针实现杨辉三角

```
#include<stdio.h>
int main(int argc, char const *argv[])
{
    int a[10][10]={};
    int (*p)[10]=a;
    for(int i=0;i<10;i++)
    {
        *p[i]=1;
        *(*(p+i)+i)=1;
    }
    for(int i=2;i<10;i++)
    {
        for(int j=1;j<10;j++)
        {
            *(p[i]+j)=*(p[i-1]+j)+*(p[i-1]+j-1);
        }
    }
    for(int i=0;i<10;i++)
    {
        for(int j=0;j<=i;j++)
            printf("%d ",*(p[i]+j));
        printf("\n");
    }
    return 0;
}
```

作业：

1. 整理笔记，用思维导图的方式。
2. 遍历 char *p[3]={"hello","world","hqyj"};中的每个元素
3. 用指针将整型组 s[8]={1,2,3,4,5,6,7,8}中的值逆序存放并打印

```
#include <stdio.h>
int main()
{
    int i;
    int s[8]={1,2,3,4,5,6,7,8};
    int *p=s,t=0;
    int *q=s+sizeof(s)/sizeof(s[0])-1;

    while (p<q)
    {
        t=*p;
        *p=*q;
        *q=t;
        p++;
        q--;
    }

    for(int i=0;i<8;i++)
        printf("%d ",s[i]);
    printf("\n");

    return 0;
}
```

4. 把一句话倒叙排放，比如"i love china"倒叙成 “china love i”

方法一：

```c
#include<stdio.h>
#include<string.h>
int main(int argc, char const *argv[])
{
    char a[32]="you love china";
    char *p = a,*q=NULL,*k=NULL,t;
    q = p+strlen(a)-1;              //指向整个字符串的结尾
    while(p<q)
    {
        t = *p;
        *p = *q;
        *q = t;
        p++;
        q--;
    }                               //全部倒置结束
    printf("%s\n",a);
    p = q = a;                      //重新指向字符串开头
    while(*p != '\0')               //遍历整个字符串
    {
        while(*p == ' ')            //让 p 一直指向开头
            p++;
        q=p;                          //p 和 q 指向单词开头
        while(*q != ' '&& *q != '\0')
            q++;
        k=q;                          //k 指向空格
        q--;                          //q 指向单词结尾
        while(p < q)
        {
            t = *p;
            *p = *q;
            *q = t;
            p++;
            q--;
        }                               //单词倒置结束
        p=k;                            //p 指向空格
    }
    printf("%s\n",a);
    return 0;
}
```

方法二：

```
#include <stdio.h>
#define MAX 30
int main(void)
{
    char buf[32];
    char str[MAX][15] = {0};
    int i, j, t;
    i = MAX - 1; //表示最后一行
    t = 0;
    gets(buf);

    while (buf[t] != '\0')
    {
        j = 0;
        while (buf[t] != '\0' && buf[t] != ' ')
        {
            str[i][j] = buf[t];      //二维数组的每一行记录字符串
            j++;
            t++;
        }
        if (buf[t] == ' ')
        {
            i--;                     //找上一行，把字符串塞到上一行里。
            t++;
        }
    }

    for (; i < MAX; i++)             //依次打印每个小字符串
        printf("%s ", str[i]);
    printf("\n");
    return 0;
}
```

方法三：

```
#include <stdio.h>
#define MAXN 32
#define MAXM 32
int main(void)
{
    char buf[32];
    char str[MAXN][MAXM] = {0};
    char(*p)[MAXM] = str + MAXM - 1;
    int j;
    char *q=buf;
    gets(buf);
    while (*q != '\0' )
    {
        j = 0;
        while (*q != '\0' && *q != ' ' )
        {
            *(*p + j) = *q;
            j++;
            q++;
        }
        if (*q == ' ')
        {
            p--;
            q++;
        }
    }
    for (int i = 0; i < MAXN; i++)
        if (*str[i])
            printf("%s ", str[i]);
    printf("\n");
    return 0;
}
```

(4) 命令行参数

```
int main(int argc, char const *argv[])
{
    printf("%s\n",argv[0]);
```

}

argv:是一个指针数组，里面存放了命令行传递的字符串。

argc:表示的是 argv 指针数组里面字符串的个数，也就是命令行传递字符串的个数。

例子：打印命令行除了开头名字的每一个参数字符串

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    for (int i = 1; i < argc; i++)
        printf("%s\n",argv[i]);
    return 0;
}
```

练习：将命令行输入的 3 个字符串转换成整型数字输出。

要求：字符串为 0-9 组成，输出数据为一个整形数

```
#include<stdio.h>
#include<string.h>
int main(int argc, char const *argv[])
{
    int len=0,arr[4]={};                        //用一个数组接收没一个转
换后的数字
    for(int i=1;i<4;i++)
    {
        len=strlen(argv[i]);                    //计算每个字符串长度
        arr[i-1]=*argv[i]-48;                   //转换每个字符串的第一个
数组
        for ( int j=1; j < len; j++)
        {
            arr[i-1]=arr[i-1]*10+(*(argv[i]+j)-48); //用于接收后面
的转换
        }

    }

    for(int k=0;k<3;k++)
        printf("%d\n",arr[k]);
    return 0;
}
```

函数

1. 函数基本用法

1.1 定义和三要素

函数是一个可以完成特定功能的代码模块，其程序代码是独立的，通常有返回值，也可以没

有。

函数的三要素：

功能：函数要实现的功能。

参数：在函数声明和调用时定义的遍历，它用于传递数据给函数。

【1】形参：就是在函数声明的时候定义的变量，用于传递信息给函数。此时它没有具体的数值。

【2】实参：就是在调用函数时传递给函数的实际数值。

返回值：函数调用后留下的右值。

1.2 函数的声明和定义

1.2.1 函数声明

存储类型 数据类型 函数名(数据类型 形参 1,数据类型 形参 2,...);

1.2.2 函数定义格式

存储类型 数据类型 函数名(数据类型 形参 1,数据类型 形参 2,...)

{

    函数体；

}

函数名：是一个标识符，要符合标识符命名规则。

数据类型：是整个函数的返回值类型，如果没有返回值为 void。

形式参数说明：是逗号分隔的多个变量的形式说明，简称形参。

形参就是函数声明或定义时括号中的变量，因为形式参数只有在函数调用时也被传递真实的数值。

大括号对{语句序列}，称为函数体，函数体由大于等于零个语句组成的。

函数的数据总结：

（1）没有参数：括号中的参数列表可以省略，也可以写 void。

（2）没有返回值：数据类型为 void，函数内没有 return 语句。

（3）有返回值：要跟根据返回值的数据类型定义函数的数据类型，可以用 return 接收返回值。

（4）定义子函数时可以直接在主函数上面，如果想在主函数下面定义需要在主函数上面事先声明。

1.3 函数调用

（1）没有返回值：直接调用

函数名(实参);

（2）有返回值：要在函数内定义一个与返回值类型相同的变量用 return 接收。如果不需要接收返回值，就直接调用函数。如果想拿到返回值来用，可以在调用函数前设一个同类型变量去接收。

实参：在调用有参数函数时，函数名后面括号中的参数称为“实参”，此时是我们传递给函数的真实数值。实参可以是：常量、变量、表达式、函数等。

代码示例：

```c
#include<stdio.h>
void fun()                  //实现打印功能
{
    printf("in fun\n");
}

void add(int a,int b)     //实现两数相加函数，无返回值。
{
    int c=a+b;
    printf("a+b=%d\n",c);
}
int add2(int a,int b)     //实现两数相加函数，有返回值。
{
    return a+b;
}

int dec(int a,int b);     //实现两数相减函数，子函数在主函数下面定义需要在上面事先声明。
int main(int argc, char const *argv[])
{
    int n1=2,n2=3;
    int rel=0,diff=0;
    fun();
    add(2,3);
    add(n1,n2);
    rel=add2(n1,n2);
    printf("in main add:%d\n",rel);
    diff=dec(6,5);
    printf("in main dec :%d\n",diff);
    return 0;
}

int dec(int a, int b)
{
    return a-b;
}
```

练习：定义求 x 的 n 次方值的函数（ x 是实数, n 为正整数）。

```
#include<stdio.h>
float mypow(float x, int n)
{
    float rel=1;
    if(n<0)
    {
        printf("error\n");
        return -1;
    }
    for(int i=0;i<n;i++)
        rel *=x;
    return rel;
}

int main(int argc, char const *argv[])
{
    float result=mypow(1.5,2);
    printf("result= %f\n",result);
    return 0;
}
```

练习：编写一个函数，函数的 2 个参数，第一个是一个字符，第二个是一个 char *，
返回字符串中该字符的个数。

```c
#include<stdio.h>

int str_fang(char c,char *p)
{
    int n=0;
    while (*p)
    {
        if(*p == c)
            n++;
        p++;
    }
    return n;
}

int main(int argc, char const *argv[])
{
    char s[100]="hellooooooo";
    int val=str_fang('o',s);
    printf("%d\n",val);

    return 0;
}
```

练习：编程实现 strlen 函数的功能，strlen 计算字符串实际长度，不包含’ \0’

```
#include<stdio.h>
int mystrlen(char *s)
{
    int n=0;
    while(*s != '\0')
    {
        n++;
        s++;
    }
    return n;
}

int main(int argc, char const *argv[])
{
    int len=mystrlen("hello");
    printf("%d\n",len);

    return 0;
}
```

1.4 函数传参

【1】值传递

单向传递，将实参传递给形参使用，改变形参实参不会受影响。

```
void fun(int a, int b)
{
    a++;
    b++;
    printf("in fun: a=%d b=%d\n", a, b);
}
int main(int argc, char const *argv[])
{
    int n1 = 10, n2 = 20;
    fun(n1, n2);
    printf("in main: a=%d b=%d\n", n1, n2);
    return 0;
}
```

【2】地址传递

双向传递，在函数中修改形参，实参也会随之变化。

```
void fun2(int *a, int *b)
{
    *a = 40;
    *b = 90;
    printf("in fun2: a=%d b=%d\n", *a, *b);
}

int main(int argc, char const *argv[])
{
    int n1 = 10, n2 = 20;
    fun2(&n1, &n2);
    printf("in main: a=%d b=%d\n", n1, n2);
    return 0;
}
```

【3】数组传递

和地址传递一样，参数中存在数组的定义，也认为是指针。

2. 开辟堆空间

2.1 堆的概念

申请的空间种分为五个区域栈区（堆栈区），堆区，全局区，字符常量区，代码区，我们之前讲的这些定义局部变量、数组都是在内存的栈区存储。

栈区和堆区的区别

栈区：是由 CPU 自动申请和释放的，不需要我们手动申请。

堆区：需要我们自己随时申请，由我们自己去释放的。随用随取，用完释放。

3. malloc 函数

3.1 定义

```
#include <stdlib.h>
void *malloc(size_t size);
```

功能：在堆区开辟大小为 size 的空间
参数：size:开辟空间的大小，单位为字节。
返回值：
成功：返回开辟空间的首地址
失败：空指针 NULL
malloc()要和 free()搭配使用

3.2 用法

malloc 内的参数是需要动态分配的字节数，而不是可以存储的元素个数！
代码格式：
type* var_name = (type *)malloc(sizeof(type)*n);

3.3 free()函数定义

```
#include <stdlib.h>
void free(void *ptr);
```

功能：释放之前用 malloc、calloc 和 realloc 所分配的内存空间。
参数：ptr:堆空间的首地址。
返回值：无
可以释放完堆空以后，把指针赋值为空指针：

```
free(p);
p=NULL;
```

malloc 和 free 搭配用法：

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char const *argv[])
{
    int *p=(int *)malloc(sizeof(int)*100);

    if(p==NULL)
        printf("lost!\n");
    else
        printf("success\n");
    free(p);
    p=NULL;
    return 0;
}
```

注意：

1.手动开辟堆区空间，要注意内存泄漏

当指针指向开辟堆区空间后，又对指针重新赋值，则没有指针指向开辟的堆区空间，就会造成内存泄漏。

2. 使用完堆区空间后及时释放空间

例如：

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char const *argv[])
{
    char *p=(char *)malloc(sizeof(char)*32);
   // p="hello"; //错误，指针指向别的地方，没有对堆空间进行操作
    scanf("%s",p);
    printf("%s\n",p);
    free(p);
    p=NULL;
    return 0;
}
```

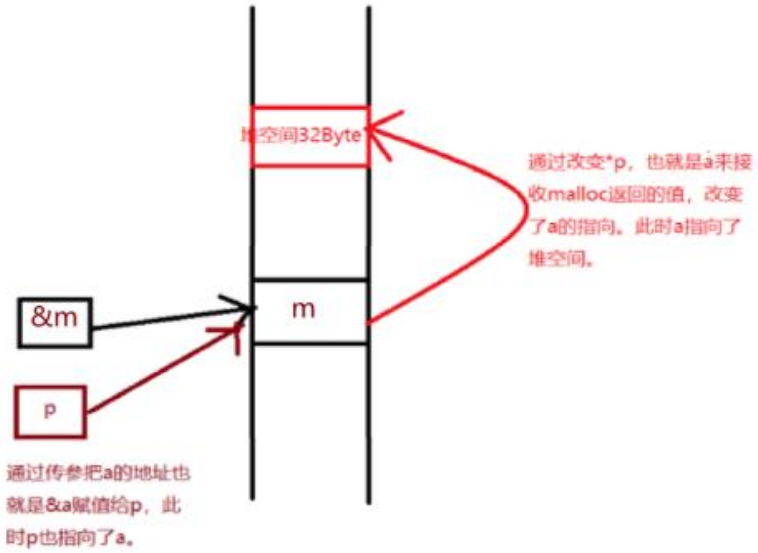
3.4 函数中开辟堆空间

思考：如下代码输出结果

```
#include <stdio.h>
#include <stdlib.h>
char *fun()
{
    char *p= (char *)malloc(32);
    scanf("%s",p);
    return p;
}

int main(int argc, char const *argv[])
{
    char *m=fun();
    char *n=fun();
    printf("%s\n",m);
    printf("%s\n",n);
    free(m);
    m=NULL;
    free(n);
    n=NULL;
    return 0;
}
```

解决方法 2：通过传参

```
#include <stdio.h>
#include <stdlib.h>
void fun(char **p)
{
    *p = (char *)malloc(32);
    scanf("%s", *p);
}

int main(int argc, char const *argv[])
{
    char *m=NULL;
    fun(&m);
    printf("%s\n",m);
    free(m);
    m=NULL;
    return 0;
}
```



4. string 函数族

4.1 strcpy

#include <string.h>
char *strcpy(char *dest, const char *src);

功能：实现字符串的复制，包括\0
参数：dest:目标字符串首地址
　　src:源字符串首地址
返回值：目标字符串首地址

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    char s1[32]="hello";
    char s2[32]="world";
    strcpy(s1,s2);
    printf("%s\n",s1);
    return 0;
}
```

4.2 strlen

#include <string.h>

size_t strlen(const char *s);

功能：计算字符串长度

参数：s:字符串的首地址

返回值：返回字符串实际长度，不包括 ‘\0’ 在内。

4.3 strcat

#include <string.h>
char *strcat(char *dest, const char *src);
功能：用于字符串拼接
参数：dest:目标字符串首地址
　　src:源字符串首地址
返回值：目标字符串首地址

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    char s1[32]="hello ";
    char s2[32]="world";
    strcat(s1,s2);
    printf("%s\n",s1);
    return 0;
}
```

char *strncat(char *dest, const char *src, size_t n);

拼接 src 的前 n 个字符

```
    char s1[32]="hello ";
    char s2[32]="world";
    strncat(s1,s2,4);
    printf("%s\n",s1);  //hello worl
```

4.4 strcmp

#include <string.h>
int strcmp(const char *s1, const char *s2);
功能：用于比较字符串
参数：s1 和 s2 是比较的字符串的首地址
返回值：
1：s1>s2
0: s1==s2
-1: s1<s2

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char const *argv[])
{
    int a=strncmp("hello world","hello",5);
    if(a==0)
        printf("相等! \n");
    else
        printf("不相等! \n");
    return 0;
}
```

作业：

1. 用思维导图的方式整理复习笔记
2. 封装函数实现两数交换

```
#include <stdio.h>
#include <string.h>
char *myStrcpy(char *dest, char *src)
{
    while(*src != '\0')
    {
        if(*dest!=*src)
            *dest=*src;
        dest++;
        src++;
    }
    *dest='\0';
}
int main()
{
    char s1[32]="hello";
    char s2[32]="world~~";

    myStrcpy(s1,s2);
    printf("%s\n",s1);
}
```

4. 封装函数实现冒泡排序

```
#include<stdio.h>

void *maopao(int *arr,int n)
{
    int c=0;
    for (int i=1;i<n;i++)
    {
        for(int j=0;j<n-i;j++)
        {
            if(arr[j]>arr[j+1])
            {
                c=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=c;
            }
        }
    }
}
int main(int argc, char const *argv[])
{
    int a[5]={5,4,3,2,1};
    maopao(a,5);
    for(int i=0;i<5;i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

5. 递归函数

5.1 概念

所谓递归函数是指一个函数体中直接或者间接调用了该函数本身，这里的直接调用是指一个函数的函数体中含有调用自身的语句，间接调用是指一个函数在函数体里有调用了其它函数，而其它函数又反过来调用了该函数的情况。

5.2 执行过程

递归函数调用的执行过程分为两个阶段：

（1）递归阶段：从原问题出发，按递归公式递推从未知到已知，最终达到递归终止条件。就是从最里层开始算，然后一层层算，直到终止。

（2）回归阶段：按递归终止条件求出结果，逆向逐步带入递归公式，回到原问题求解。

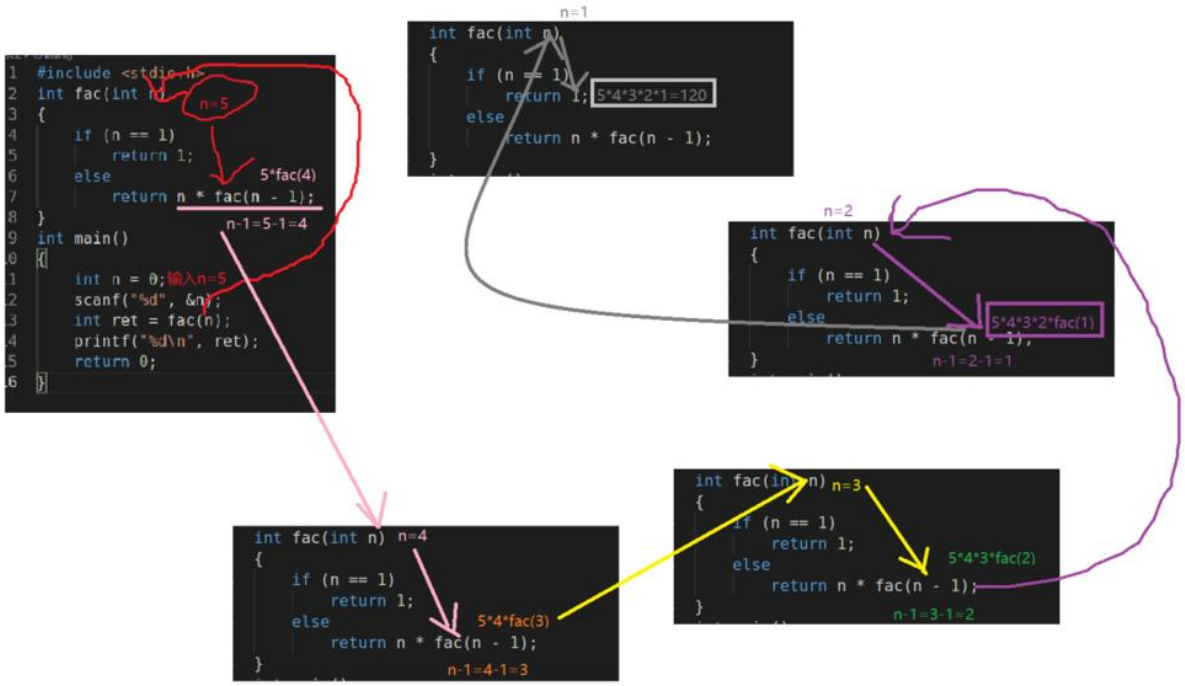
例子：求 5 的阶乘 5！

递归公式：

fac(n)= 1 n=1

n*fac(n-1) n>1

```
#include<stdio.h>

int fac(int n)
{
    if(n==1)
        return 1;
    else
        return n*fac(n-1);
}
int main(int argc, char const *argv[])
{
    int n=5,ret=fac(n);
    printf("ret=%d\n",ret);
    return 0;
}
```



练习：斐波那契数列：指的是这样一个数列：1、1、2、3、5、8、13、21、34、……在数学上，斐波纳契数列以如下被以递归的方法定义：F(1)=1，F(2)=1, F(n)=F(n-1)+F(n-2)（n>=2, n∈N*） 求第五个数。

F(n)： 1 n<=2

F(n-1)+F(n-2) n>2

```c
#include <stdio.h>
int fun(int n)
{
    if (n <= 2)
        return 1;
    else
        return fun(n - 1) + fun(n - 2);
}
int main()
{
    int rel = fun(5);
    printf("rel=%d\n", rel);
    return 0;
}
```

练习：封装函数实现 strcat 功能

1.2 定义格式

struct 结构体名

{

数据类型 成员变量 1;

数据类型 成员变量 2;

数据类型 成员变量 3;

...

};

举例：构造一个新的数据类型叫 student,用来描述学生。

```
struct student
{
    char name[32];
    int age;
    float score;
};
```

1.3 结构体变量

1.3.1 概念

通过结构体数据类型定义的变量

1.3.2 格式

struct 结构体名 变量名;

1.3.3 定义结构体变量

(1) 先定义结构体，再定义结构体变量。

struct 结构体名

{

成员变量;

};

struct 结构体名 变量名;

例如：

```
    struct student
    {
        char name[32];
        int age;
        float score;
    };
    struct student s1;
```

(2) 定义结构体的同时，定义结构体变量。

struct 结构体名

{

    成员变量;

} 变量名;

例如：

```
struct person
{
    char name[32];
    int age;
} per1;
```

也可以缺省结构体名定义结构体变量：

struct

{

成员变量;

} 变量名;

1.3.4 结构体变量赋值

先定义一个结构体：

struct student

{

int id;

char name[32];

int age;

};

(1) 定义结构体变量时直接用花括号赋值：

```
struct student stu2 ={
    .id = 2,
    .name = "xiaoming",
    .age = 42
};
```

(3) 定义结构体变量时未初始化，需要挨个单独赋值:

```
struct student stu3;
stu3.id = 3;
stu3.age = 69;
strcpy(stu3.name,"laowang");
```

1.3.5 访问结构体成员变量

通过.访问： 结构体变量名.结构体成员变量名。

```
#include <stdio.h>
#include <string.h>
typedef struct flower
{
    char type[32];
    int size;
    char color[32];
} flo;

int main()
{
    flo f1 = {"rose",10,"red"};
    printf("%s %d %s\n",f1.type,f1.size,f1.color);
}
```

(2) 先定义结构体，再进行重定义。

```
struct animal
{
    char type[32];
    char class[32];
    int wight;
};
typedef struct animal ANIMAL;
int main()
{
    ANIMAL an1 = {"dog","buru",200};
    printf("%s %s %d\n",an1.type,an1.class,an1.wight);
}
```

练习：创建一个描述手机的结构体叫 phone, 包含品牌，型号，颜色，价格。从终端输入你自己手机的信息并打印。

```
    int id;
    int age;
    float score;
};
struct student stu[5];
```

2.3 初始化和赋值

（1）定义结构体数组的同时赋值

```
#include <stdio.h>
#include <string.h>
struct hero
{
    char name[32];
    char postion[32];
    char skill1[32];
    char skill2[32];
    char skill3[32];
    char skill4[32];
    int price;
};
int main()
{
    struct hero h[2] =
        {
            {"ali","fashi","1","2","3","4",4300},
            {"shitou","tanke","1","2","3","4",3150}
        };

    printf("%s %s %s %s %s %s %d\n", h[1].name, h[1].postion,
h[1].skill1, h[1].skill2, h[1].skill3, h[1].skill4, h[1].price);
    return 0;
}
```

(2) 先定义结构体数组，在对数组内每个元素分别赋值。

```
#include <stdio.h>
#include <string.h>
struct hero
{
    char name[32];
    char postion[32];
    char skill1[32];
    char skill2[32];
    char skill3[32];
    char skill4[32];
    int price;
};
int main()
{
    struct hero he[2];
    strcpy(he[0].name,"namei");
    strcpy(he[0].postion,"fashi");
    he[0].price = 6300;
    printf("%s %s %d\n", he[0].name, he[0].postion,he[0].price);
    return 0;
}
```

2.4 结构体数组输入输出（通过循环）

```
#include <stdio.h>
#include <string.h>
struct hero
{
    char name[32];
    char postion[32];
    int life;
    int magic;
    float speed;
    int price;
};
int main()
{
    struct hero h[5] =
        {
            {"ali", "magicer", 596, 481, 53.04, 6300},
            {"gailun", "tank", 700, 0, 55.04, 450},
            {"jee", "ad", 596, 200, 55.04, 6300},
            {"cat", "helfer", 400, 400, 45, 6300},
            {"ez", "adc", 550, 450, 53.04, 6300},
        };
    for (int i = 0; i < 5; i++)
    {

printf("%s %s %d %d %f %d\n",h[i].name,h[i].postion,h[i].life,h[i
].magic,h[i].speed
,h[i].price);
    }
    return 0;
}
```

2.5 结构体大小

（1）结构体类型大小*元素个数

（2）用 sizeof(struct 结构体名) 或者 sizeof(结构体变量名)

例如接着上面的练习：

```
printf("struct hero:%d %d\n",sizeof(struct hero),sizeof(h));//80
400
```

3. 结构体大小以及对齐原则

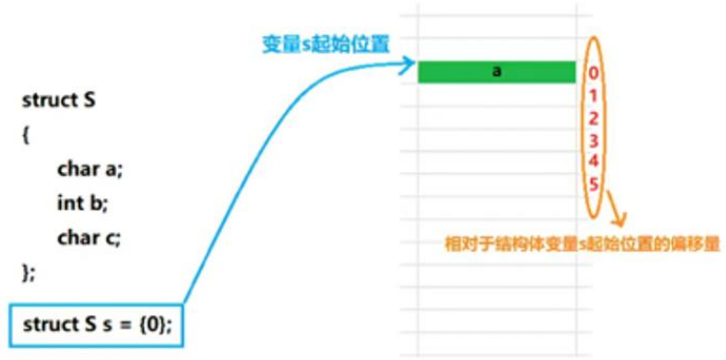
3.1 结构体大小用 sizeof 计算

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

struct st
{
    char a;
    char b;
    int c;
};

struct s
{
    char a;
    int b;
    char c;
};
int main()
{
    printf("%d\n",sizeof(struct st)); //8
    printf("%d\n",sizeof(struct s));  //12
    return 0;
}
```

3.2 字节对齐原则

（1）第一个成员在相对于结构体变量起始位置偏移量为 0 的地址处

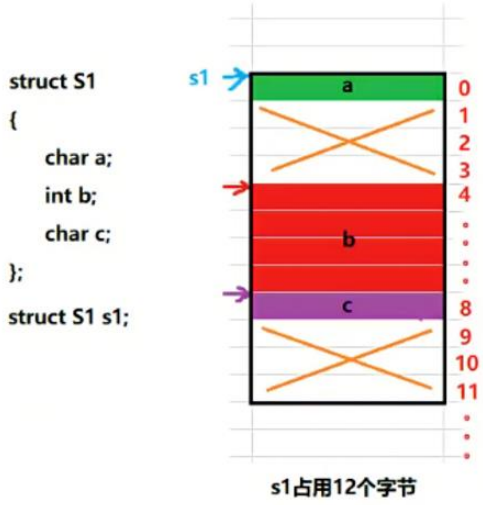
（通俗点来说，就是第一个成员变量的地址与结构体起始位置的地址是相同的）如下图所示：

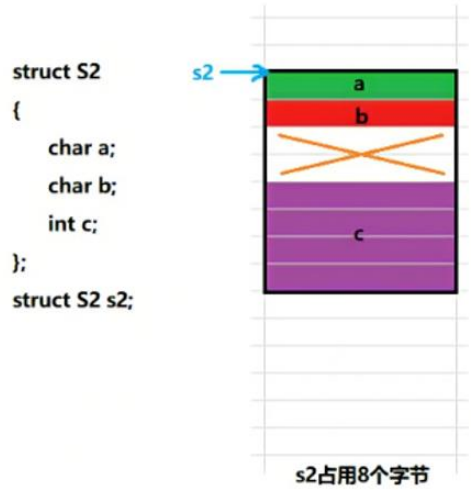


（2）用结构体成员里面最大的数据类型的大小和字节进行比较，然后按照字节数小的为单位开辟空间。例如上图例子，每个成员都要给他以 4 字节为单位开辟空间。

3.3 节省空间原则

为了减少空间浪费，把占用空间小的成员集中到一起。





4. 结构体指针

4.1 概念

指向结构体变量的指针

4.2 定义格式

struct 结构体名 *结构体指针名；

例如：

```
struct student
{
    int id;
    char name[32];
}s1,s2;

struct work
{
    int id;
    int age;
}w1,w2;

struct student *sp = &s1;
//struct student *p=&w1; //错误，结构体类型不同
```

4.3 结构体指针成员变量赋值

格式：指针变量名->成员变量名

```
#include <stdio.h>
#include <string.h>
struct student
{
    int id;
    char name[32];
}s1,s2;
int main()
{
    struct student *sp = &s1;
    strcpy(sp->name,"xiaoming");
    sp->id=123;
    printf("%s %d\n",s1.name,s1.id);
    printf("%s %d\n",sp->name,sp->id);

    return 0;
}
```

或者：(*指针变量名).成员变量名

结构体指针大小：4 字节，因为本质还是指针。

5 结构体内结构体

如果结构体内成员本身属于另一种结构体，得用若干个成员运算符一级级找到最低级的成员变量。

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

struct work
{
    int ip;
};

struct student
{
    int id;
    int age;
    struct work w1;
} stu1, stu2;

int main()
{
    stu1.w1.ip = 2;
    stu1.id = 1;
    stu1.age = 20;
    printf("%d %d %d\n", stu1.w1.ip, stu1.id, stu1.age);
    return 0;
}
```

结构体总结：

1. 不能把结构体类型变量作为整体引用，只能对结构体内变量中的各个成员分别引用。
2. 如果成员本身属于另一种结构体类型，要用若干个成员运算符.一级级找到最低级成员变量。
3. 可以把成员变量当成普通变量运算
4. 在数组中，成员之间不能彼此赋值，但是结构体变量可以互相赋值。

练习：创建一个结构体数组，数组名为 book，结构体成员包含编号，书名，售价（数据类型自己设定）。写一个函数，包含两个形参，分别接收结构体数组的首地址和一个指定的售价，函数的功能为打印结构体数组中售价大于指定售价的书的信息。

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

typedef struct book
{
    int number;
    char name[32];
    int price;
} BOOK;
BOOK b[5]={
    {1,"santi",25},
    {2,"hongloumeng",45},
    {3,"jinpingmei",50},
    {4,"xiyouji",87},
    {5,"sanguo",67}
};

void book_Pri(BOOK *p, int n)
{
    for(int i=0;i<5;i++)
    {
        if(p->price >= n)
            printf("%d %s %d\n",p->number,p->name,p->price);
        p++;
    }
}
int main()
{
    book_Pri(b,50);
    return 0;
}
```

作业：

1. 整理笔记，用思维导图的方式。
2. 创建一个名为 student 的结构体数组，包含学号，姓名，分数,(数据类型自己定义),从终端输入学生的信息并打印分数及格的学生信息（输入 3 人即可）。封装函数实现把其中第几个

学生信息清零。

```
#include <stdio.h>
#include <string.h>
typedef struct student
{
    int id;
   char name[32];
   float score;
} STU;
STU s[3];

void delStudent(STU *p, int n)
{
    n--;
    /*第一种清零方式 */
    // (p+n)->id=0;
    // strcpy((p+n)->name,"\0");
    // (p+n)->score=0;

    /*第二种清零方式 */
    //memset((p+n),0,sizeof(struct student));

    /*第三种清零方式*/
    STU stu2 ={0,"\0",0};
    p[n] =  stu2;
}

int main()
{
    printf("pls input student information:\n");
   for(int i=0;i<3;i++)
      scanf("%d %s %f",&s[i].id,s[i].name,&s[i].score);
    for(int i=0;i<3;i++)
    {
        if(s[i].score >= 60)
            printf("%d %s %f\n",s[i].id,s[i].name,s[i].score);
    }
    delStudent(s,2);
   printf("student information:\n");
    for(int j=0;j<3;j++)
      printf("%d %s %f\n",s[j].id,s[j].name,s[j].score);
   return 0;
}
```

3. 已知字符数组 a[10]和 b[10]中元素的值递增有序，封装函数并用指针实现将两个数组中元素按照递增顺序输出。

如：char a[10]=” acdgjmno” ; char b[10]=” befhil” ;->” abcdefghijlmno”

思路：利用指针比较两字符数组中的元素，若 a 中的小，打印 a 中字符，然后 a 的指针移动到下一个元素继续比较，反之打印 b 然后移动 b 指针到下一个元素继续比较。最后判断字符串是否结束，打印没有结束的字符串剩下部分

```
#include <stdio.h>
void fun(char *p1, char *p2)
{
    while (*p1 != '\0' && *p2 != '\0')   //判断是否到两个数组的结尾
    {
        if (*p1 < *p2)                   //进行比较，把小的数据打印出来
        {
            printf("%c", *p1);
            p1++;
        }
        else
        {
            printf("%c", *p2);
            p2++;
        }
    }
    if (*p1 == '\0')                     //把其中剩下的数据打印出来
        printf("%s\n", p2);
    else if (*p2 == '\0')
        printf("%s\n", p1);
}
int main(int argc, char const *argv[])
{
    char a[10] = "acegijkm";
    char b[10] = "bdfh";
    fun(a, b);
    return 0;
}
```

4. 预习共用体和枚举

共用体 union

共用体有时候也称联合体，用法类似结构体 struct,区别在于结构体所占内存大小等于所有成员占用内存总和。而共用体使用了内存覆盖技术，同一时刻只能保存一个成员的值，如果对新成员赋值，就会把原来成员的值覆盖掉。

1 格式

union 共用体名

{

    成员列表;

};

2 定义共用体 union 变量

```
#include <stdio.h>
union val
{
    int a;
    char ch;
};
union val v;

int main(int argc, char const *argv[])
{
    v.a=10;
    v.ch='c';

    printf("%d\n",v.a);
    return 0;
}
```

得出共用体变量共用一块空间

3 特性

(1) 共用体成员共同使用一块内存空间

(2) 赋值顺序以最后依次赋值为准

(3) 共用体的大小为成员中数据类型最大的数据为大小

```
#include <stdio.h>
union val
{
    int a;
    char ch;
};
union val v;

int main(int argc, char const *argv[])
{
    v.a=10;
    if(v.ch == 10)
        printf("small\n");
    else if(v.ch == 0)
        printf("big\n");

    printf("%d\n",v.ch);
    return 0;
}
```

枚举 enum

1 定义

用户自定义数据类型，可以用于声明一组常数。

在实际编程中，有些数据的取值往往是有限的，只能是非常少量的整数，并且最好为每个值都起一个名字。

例如以每周七天为例子，我们可以使用宏定义#define 来为每天命名：

#define MON 1

#define TUES 2

...

虽然能解决问题但是宏名过多导致代码看起来很松散，所以 C 语言提供了一种枚举 enum,能够列出所有可能的取值，并给它们起一个名字。

2 格式

enum 枚举名

{

value 1,

value 2,

value 3,

...

};

未赋初值时，第一个常数会默认为 0，然后下面的依次加一。每个元素都可以赋值，如果有元素没有赋值则默认值为上一个元素加一。

例子：

```
#include <stdio.h>
enum week
{
    MON,
    THES = 2,
    WED,
};
int main(int argc, char const *argv[])
{
    int day = 0;
    scanf("%d", &day);
    switch (day)
    {
    case MON:
        printf("%d\n", MON);
        break;
    case THES:
        printf("%d\n", THES);
        break;
    case WED:
        printf("%d\n", WED);
        break;
    default:
        break;
    }
    printf("MON:%d\n", MON);
    printf("THES:%d\n", THES);
    printf("WED:%d\n", WED);
    return 0;
}
```

数据类型总结

（1）**基本数据类型**：基本数据类型主要的特点是其值不可以再分解为其他类型，也就是说，基本数据类型是自我说明的。如: short int long float double signed unsigned

（2）**构造数据类型**：构造数据类型是根据已定义的一个或多个数据类型用构造的方法来定义

的。也就是说一个构造型的值可以分解成若干个“成员”或者“元素”。每个“成员”都是一个基本数据类型或者又是一个构造型。struct 结构体、union 联合体、enum 枚举和数组。

（3）**指针类型**：指针是一个特殊的数据类型。其值用来标识某个变量再内存中的地址。虽然指针变量的取值类似于整型，但这两个类型完全不同，不能混淆。定义时要加*。

（4）**空类型**：空类型 void 用来定义任意类型的指针和无返回值的函数。

存储类型

存储类型有：auto static extern register

1. auto 自动型

修饰变量，一般省略存储类型会默认为 auto。

2. static 静态

可以修饰变量或函数

2.1 修饰变量

1. 变量的存放位置在全局区（静态区）

如果静态变量有初值，存放.data 区，没有初值存放在.bss 区域

2. 生命周期为整个程序

3. 限制作用域：

修饰局部变量，和普通局部变量作用域没有区别，但是生命周期被延长为整个程序。

也就是在作用函数外有生命但是不能被操作，变成了植物人。

修饰全局变量，限制在本文件中使用。

4. 值初始化一次，初值赋值为 0。

举例对比用 static 和不用的局部变量在函数中自加操作

```
#include <stdio.h>
void fun()
{
    static int a;        //初值为 0
    int b=0;
    a++;
    b++;
    printf("in fun:%4d %4d\n",a,b);
}
int main(int argc, char const *argv[])
{
    fun();
    fun();
    fun();
    return 0;
}
```

打印结果：
in fun: 1 1
in fun: 2 1
in fun: 3 1
因为 a 用 static 修饰，所以会保留上一次调用函数后留下的值。此时 a 放在静态区。

2.2 修饰函数

static 修饰函数，限制在本文件中使用。

总的来说，使用 static 定义的变量存储在静态存储区中，具有静态持续性，即使函数返回，它的值也会保留。而且，static 变量只能在当前文件中访问，不能被其他文件访问。

配合 extern 举例

3. extern

外部引用:

通过 extern 可以引用其他文件中的全局变量或函数

举例对比引用外部变量和函数，如果加 static 和不加的区别。

main.c

23041 ▸ struct ▸ hw1 ▸ C main.c ▸ main(int, char

```c
#include <stdio.h>
extern int a;
//extern int b;
extern void fun1();
//extern void fun2();


int main(int argc, char cons
{

    //printf("b=%d\n",b);
    printf("a=%d\n",a);
    fun1();
    //fun2();
    return 0;
}

```

fun.c

23041 ▸ struct ▸ hw1 ▸ C fun.c ▸ fun2()

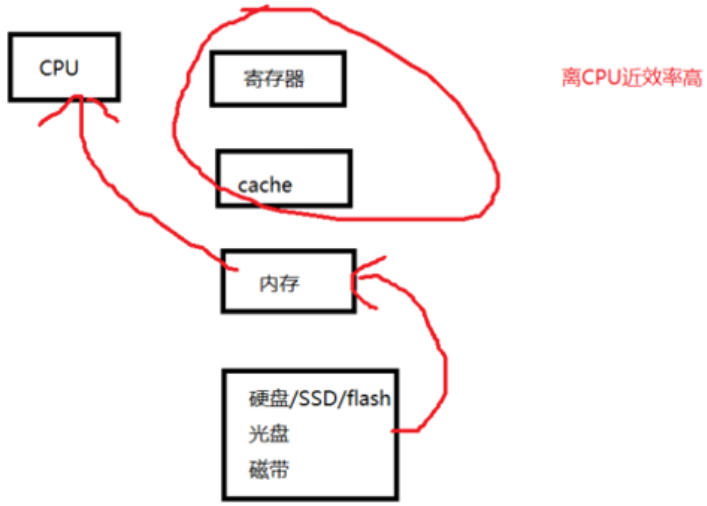
```c
#include <stdio.h>

int a=10;
static int b=20;
void fun1()
{
    printf("fun1\n");
}

static void fun2()
{
    printf("fun2\n");
}
```

解释：因为变量 b 和函数 fun2()被 static 修饰，所以限制在 fun.c 内不能被其他文件调用。

4. register 寄存器类型

register 是寄存器类型，顾名思义，是把修饰的变量放到寄存器中，目的是为了提高程序的运行效率。但要记住，不管是单片机也好，计算机也罢，任何 CPU 的寄存器资源都是有限的，如果寄存器满了，被修饰的变量就会默认回到 auto 类型。

寄存器相当于皇帝身边的小太监，离皇帝近。而内存就相当于大臣，离皇帝比较远，但是地位高。



练习：

创建一个结构体数组,数组名为 lolHero,成员包含名称, 位置, 血量和价格。给出每个 lol 英雄信息，封装函数实现按价格从低到高打印英雄信息。（用冒泡排序）

```
#include <stdio.h>
#define N 5
typedef struct hero
{
    char *name;
    char *position;
    int blood;
    int price;
} HERO;

void sort(HERO *p, int n)
{
    HERO temp;
    for (int i = 1; i < N ; i++)
    {
        for (int j = 0; j < N - i; j++)
        {
            if (p[j].price > p[j + 1].price)
            {
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

int main(void)
{
    HERO lolHero[N] = {
        {"ali", "ap", 596, 6300},
        {"gailun", "tank", 700, 450},
        {"jee", "ad", 596, 6000},
        {"cat", "helfer", 400, 6300},
        {"ez", "adc", 550, 6500}
    };

    sort(lolHero, N);
    for (int i = 0; i < N; i++)
        printf("%d. %8s %8s %8d %8d\n",i+1, lolHero[i].name,
lolHero[i].position, lolHero[i].blood, lolHero[i].price);
}
```