

前言

2021年12月19日 15:29

本笔记由Amos_Phoenix创建，默认观看者有一定基础。

QQ:1297289742

如有问题，欢迎加QQ进行指正，本人感激不尽。

本笔记由Amos_Phoenix创建

QQ:1297289742

如有问题，欢迎加QQ进行指正，本人感激不尽。

AC: Accept 通过

出现这个图标的时候表示在本测试点中你的输出是正确的，拿到了本测试点的满分

WA: Wrong Answer 答案错误

出现这个图标的时候表示在本测试点中你的输出是错误的，在本测试点中未拿到任何分数

RE: Runtime Error 运行时错误

这表明你的程序在运行过程中因为出锅而崩溃了，通常可能是访问非法内存等问题，出现这个提示但你还能过样例的话，大概率是数组没开够，仔细检查一下。

CE: Complie Error 编译错误

这表明你的程序没有通过编译。如果在本地编译可以通过的话，检查一下提交语言是不是选对了或是有没有引用一些不该引用的东西

TLE: Time Limit Exceeded 超出时间限制

这表明你的程序运行所用的时间超过了测试点的规定时间。出现这个提示时一般表明你的算法的时间复杂度不够优秀，需要优化；但也有可能程序深入死循环无♂法♂自♂拔♂

MLE: Memory Limit Exceeded 超出内存限制

这表明你的程序所调用的内存大小超出了测试点的内存限制，一般表明你的算法的空间复杂度不是很优秀，比如像NOIP铺地毯中什么百万级二维数组的失智做法什么的，好好优化一下~

PC: Partially Correct 部分正确

这个提示信息一般会在题目启用Special Judge的时候可能会返回，代表着你得到了本测试点的部分分数。一般情况有两个：

- 1.你的答案正确，格式错误，Special Judge给你送了点良心分
- 2.你的答案中间有一部分是正确的，根据题目要求，Special Judge 给你返回了你这一部分答案的分数

这个时候一定要仔细查看Special Judge 给你返回的信息，仔细检查，距离AC就不远啦！

OLE: Output Limit Exceeded 超出输出限制

表示你的程序出现了 **大量** 的输出，一般还是程序在输出过程中深陷死循环无♂法♂自♂拔♂，好好检查一下代码逻辑~

UKE: Unknown Error 未知错误

这个时候一般是评测姬锅了，如果多次提交都是出现这个提示的话，赶快联系网站的管理员鸭~

程序员计算器使用

2021年9月19日 10:25

本笔记由Amos_Phoenix创建

QQ:1297289742

如有问题，欢迎加QQ进行指正，本人感激不尽。

HEX：十六进制

DEC：十进制

OCT：八进制

BIN：二进制，注意这里的二进制为二进制补码形式

每输入一个数，进制转换会自动进行

QWORD：四字，64位 DWORD：双字，32位 WORD：字，16位 BYTE：字节，8位

Lsh：左移 Rsh：右移 点击按钮，变作，分别为循环左移和循环右移

Or：或 Xor：异或 Not：非 And：与 Mod：模运算（求余）

CE：清除本次输入 C：清除所有输入，结束计算 MS：内存存储

ASCII码对照表

2021年9月11日 20:22

本笔记由Amos_Phoenix创建

QQ:1297289742

如有问题，欢迎加QQ进行指正，本人感激不尽。

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
0	NUT	32	(space)	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(72	H	104	h
9	HT	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	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	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	/	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	`
31	US	63	?	95	_	127	DEL

特殊字符解释

NUL空	VT 垂直制表	SYN 空转同步
STX 正文开始	CR 回车	CAN 作废
ETX 正文结束	SO 移位输出	EM 纸尽
EOY 传输结束	SI 移位输入	SUB 换置
ENQ 询问字符	DLE 空格	ESC 换码
ACK 承认	DC1 设备控制1	FS 文字分隔符
BEL 报警	DC2 设备控制2	GS 组分分隔符
BS 退一格	DC3 设备控制3	RS 记录分隔符
HT 横向列表	DC4 设备控制4	US 单元分隔符
LF 换行	NAK 否定	DEL 删除

优先级表

2021年9月11日 20:24

本笔记由Amos_Phoenix创建

QQ:1297289742

如有问题，欢迎加QQ进行指正，本人感激不尽。

运算符优先级和结合性一览表

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

7	==	等于	表达式==表达式	左到右	双目运算符
	!=	不等于	表达式!= 表达式		双目运算符
8	&	按位与	表达式&表达式	左到右	双目运算符
9	^	按位异或	表达式^表达式	左到右	双目运算符
10		按位或	表达式 表达式	左到右	双目运算符
11	&&	逻辑与	表达式&&表达式	左到右	双目运算符
12		逻辑或	表达式 表达式	左到右	双目运算符
13	?:	条件运算符	表达式1? 表达式2: 表达式3	右到左	三目运算符
14	=	赋值运算符	变量=表达式	右到左	
	/=	除后赋值	变量/=表达式		
	=	乘后赋值	变量=表达式		
	%=	取模后赋值	变量%=表达式		
	+=	加后赋值	变量+=表达式		
	-=	减后赋值	变量-=表达式		
	<<=	左移后赋值	变量<<=表达式		
	>>=	右移后赋值	变量>>=表达式		
	&=	按位与后赋值	变量&=表达式		
	^=	按位异或后赋值	变量^=表达式		
	=	按位或后赋值	变量 =表达式		
15	,	逗号运算符	表达式,表达式,...	左到右	

对于优先级，只有与变量连在一起才有优先级

例子：

p++:++的优先级大于，所以先自增，后取直线地址中的数据

++*p:由于++未和p连在一起，所以是对*p加一。

上表中可以总结出如下规律：

1. 结合方向只有三个是从右往左，其余都是从左往右。
2. 所有双目运算符中只有赋值运算符的结合方向是从右往左。
3. 另外两个从右往左结合的运算符也很好记，因为它们很特殊：一个是单目运算符，一个是三目运算符。
4. C语言中有且只有一个三目运算符。
5. 逗号运算符的优先级最低，要记住。
6. 此外要记住，对于优先级：算术运算符 > 关系运算符 > 逻辑运算符 > 赋值运算符。逻辑运算符中“逻辑非！”除外。

函数

2021年9月11日 20:26

本笔记由Amos_Phoenix创建

QQ:1297289742

如有问题，欢迎加QQ进行指正，本人感激不尽。

变量

一个局部变量的作用域是包含他的块语句

块语句内部若有与外部同名的变量,则屏蔽外部的

全局变量生存周期很长，一直占用内存，除非程序退出很容易被其他函数修改（或被黑客修改）或因重名而被屏蔽，失效

避免占内存较大的变量使用全局变量

字符串的获取与输出

char str[10] = "notepad" ; //定义一个字符串，并存储字符最大为定义[]内的数。

gets(str); //输入一个字符串到字符变量。

puts(str); //输出一个字符串。

system(str); //在控制面版执行指令字符串。

system("notepad") //直接通过在system中输入字符来执行命令

补充:

函数可变参数

1.针对数

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

```
#include<stdlib.h>
```

```
#include<stdarg.h> //里面有很多宏，我们可以处理可变参数的函数
```

```
double add(int num,...) //...代表可变的参数
```

```
{
    int i = 0;
    double last = 0;
    va_list argp; //创建一个char类型的指针
    va_start(argp, num); //从这里开始读取参数，读取num个，并把地址放在argp
    for (;i<num;i++)
    {
        double temp = va_arg(argp, double); //读取参数，挨个 读取
        printf("\n % f", temp);
        last += temp;
    }
    va_end(argp); //结束读取
    return last;
}
```

```
int main(void)
```

```
{
    int x = 4;
    double jieguo = 0;
    jieguo = add(x, 1.2, 2.3, 3.4, 4.5);
    printf("\n % f", jieguo);
}
```



```
return 0;
}
```

2.针对字符串

```
#include <stdio.h>
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<stdarg.h> //里面有很多宏，我们可以处理可变参数的函数
void open(int num, ...) //...代表可变的参数
{
    va_list argp; //创建一个char类型的指针
    va_start(argp, num); //从这里开始读取参数，读取num个，并把地址放在argp
    for (int i = 0; i < num; i++)
    {
        char str[50];
        strcpy(str, va_arg(argp, char *)); //读取参数，挨个读取
        printf("%s\n", str);
        system(str);
    }
    va_end(argp); //结束读取
    return;
}
int main(void)
{
    int x = 4;
    open(x, "notepad", "calc", "tasklist", "mspaint");
    return 0;
}
```

va_list ap ; 定义一个va_list变量ap

va_start(ap,v) ; 执行ap = (va_list)&v + _INTSIZEOF(v), ap指向参数v之后的那个参数的地址，即 ap指向第一个可变参数在堆栈的地址。
va_arg(ap,t) , ((t)((ap += _INTSIZEOF(t)) - _INTSIZEOF(t)))取出当前ap指针所指的值，并使ap指向下一个参数。 ap += sizeof(t类型)，让ap指向下一个参数的地址。然后返回ap-sizeof(t类型)的t类型指针，这正是第一个可变参数在堆栈里的地址。然后用取得这个地址的内容。

va_end(ap) ; 清空va_list ap。

PS: strcpy 函数用于拷贝字符串，包含最后的结束符 '\0'。

strcpy(char *dest, const char *src);

dest	指向用于存放字符串的目标数组
src	指向待拷贝的源字符串

使用VA_LIST应该注意的问题：

- (1) 因为va_start, va_arg, va_end等定义成宏,所以它显得很愚蠢,可变参数的类型和个数完全在该函数中由程序代码控制,它并不能智能地识别不同参数的个数和类型. 也就是说,你想实现智能识别可变参数的话是要通过在自己的程序里作判断来实现的.
- (2) 另外有一个问题,因为编译器对可变参数的函数的原型检查不够严格,对编程查错不利.不利于我们写出高质量的代码。
- (3) 由于参数的地址用于VA_START宏，所以参数不能声明为寄存器变量，或作为函数或数组类型。

函数和printf求参顺序是从右向左的无论是在windows还是linux (vc, gcc)

函数返回值生命周期

函数内部定义的变量，返回以后，变量就会被销毁，内存会被回收

但是函数返回值有副本机制，返回的时候另外在保存一份，我们打印的是副本，原来的内存数据已经被销毁了
如果返回全局变量，全局变量会一直存在
返回临时变量，临时变量在cpu的寄存器会马上销毁

递归

递归不能无限循环，造成栈溢出

实例：腾讯天梯

```
#include <stdio.h>
#include <stdlib.h>
int ladder(int n);
int main(void)
{
    int n = 0;
    printf("请输入天梯层数: ");
    scanf("%d", &n);
    printf("有%d种可能\n", ladder(n));
    system("pause");
    return 0;
}
int ladder(int n)
{
    if (n == 1)
    {
        return 1;
    }
    else if (n == 2)
    {
        return 2;
    }
    else
    {
        return ladder(n - 1) + ladder(n - 2);
    }
}
```

数组

2021年9月11日 20:26

本笔记由Amos_Phoenix创建

QQ:1297289742

如有问题，欢迎加QQ进行指正，本人感激不尽。

数组

数组在内存中是连续排列的

通过printf("%d" ,sizeof(a)/sizeof(int));来求数组中元素的个数

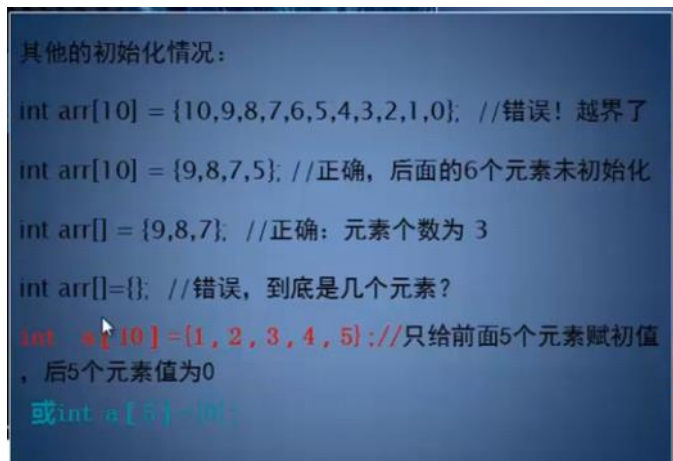
数组越界不报错，但是注意，以为数组外部的内存空间不确定是否有程序使用它的权限，如果越界访问，程序有可能会崩溃，记住：能编译不一定能运行

#define和const区别

#define N 20; //编译器放在常量区，没有地址，无从修改,如果下跟int b[N];则程序不会报错，可以执行

const int num = 10; //虽然const限定代码不能在编程时修改，但他还是是可变的常量，可以在内存中（手动/指针）修改，可以强制去掉常量的属性。如果下跟int b[num];则程序会报错

数组元素大小已经确定的时候，下标可以省略



关于#include<time.h>和srand/rand随机数的生成

一，srand/rand函数：

在实际编程中，我们经常需要生成随机数，例如，贪吃蛇游戏中在随机的位置出现食物，扑克牌游戏中随机发牌

1. 我们可以通过srand()函数来重新“播种”，这样种子就会发生改变。srand()的用法为：

void srand(unsigned int seed);它需要一个unsigned int类型的参数。在实际开发中，我们可以用时间作为参数，只要每次播种的时间不同，那么生成的种子就不同，最终的随机数也就不同。

1. 使用<time.h>头文件中的time()函数即可得到当前的时间（精确到秒），就像下面这样：

```
srand((unsigned)time(NULL));
```

1. 在C语言中，我们一般使用<stdlib.h>头文件中的rand()函数来生成随机数，它的用法为：

```
int rand(void);
```

void表示不需要传递参数。

1. 在实际开发中，我们往往需要一定范围内的随机数，过大或者过小都不符合要求，那么，如何产生一定范围的随机数呢？我们可以利用取模的方法：

```
int a = rand() % 10; //产生0~9的随机数，注意10会被整除
```

如果要规定上下限：

```
int a = rand() % 51 + 13; //产生13~63的随机数
```

分析：取模即取余，rand()%51+13我们可以看成两部分：rand()%51是产生0~50的随机数，后面+13保证a最小只能是13，最大就是50+13=63。

PS 生成随机数并求平均值实战

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main(void)
{
    time_t ts;//设置时间变量
    srand((unsigned int)time(&ts));//设置时间的随机数种子
    int a[10];//定义一个随机数数组
    int i = 0;
    for (i = 0; i < 10; i++)
    {
        a[i] = rand() % 100;//循环赋100以内的值
        printf("%d,%x\n", a[i], &a[i]);
    }
    int num = 0;
    double total = 0;
    for (i=0; i < 10; i++)
    {
        num = num + a[i];
    }
    total = num;
    total /= 10;//求平均数
    printf("%f", total);
    return 0;
}
```

二， 获取时间的函数

time.h

有人总结成这么几句，的确是经典，自己好好编程试试效果吧，
两个类型：

time_t: 表示距离 UTC 时间 1970-01-01 00:00:00 的秒数。也叫做日历时，类型是 long

clock_t: 只用于程序计时，貌似其他的没它什么事。

struct tm: 通常用于存储本地时。

1. 函数原型: **time_t time(time_t *timer)**

函数功能: 得到机器的日历时间或者设置日历时间

函数返回: 机器日历时间

1. 函数原型: **struct tm *localtime(const time_t *timer)**

函数功能: 返回一个以tm结构表达的机器时间信息

函数返回: 以tm结构表达的时间

PS:生成当地时间的实战

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#include <time.h>
int main()
{
    struct tm *p;
    time_t t;
    time(&t);
    p = localtime(&t);
    // 北京时间
    printf("%d/%d/%d\n", p->tm_mon + 1, p->tm_mday, p->tm_year + 1900);
}
```

```

printf("%d:%d:%d\n", p->tm_hour, p->tm_min, p->tm_sec);
return 0;
}

```

思维拓展：用一维数组遍历的思想遍历二维和三维数组

1. 二维

```

#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    int n[3][4];
    int i = 0;
    for (; i < 3 * 4; i++)
    {
        n[i / 4][i % 4] = i; //精髓是抓住第二个下标的数用于除或模
        printf("n[%d][%d]=%d\n", i / 4, i % 4, n[i / 4][i % 4]);
    }
    system("pause");
    return 0;
}

```

2. 三维

```

#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    int a[3][4][5];
    int i = 0;
    int n = 0;
    for (; i < 3 * 4 * 5; i++)
    {
        a[i / 20][(i / 5) % 4][i % 5] = ++n;
        printf("a[%d][%d][%d]=%d ", i / 20, (i / 5) % 4, i % 5, a[i / 20][(i / 5) % 4][i % 5]);
        if ((i + 1) % 5 == 0)
        {
            printf("\n");
        }
    }
    getchar();
    return 0;
}

```

数组零碎但又重要的知识点

```

#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    int a[3][4] = { 0 }; //表示数组里每个数都赋值为零
    int b[3][4] = { {1,2,3,4},{5,6},{7,8,9,10} }; //数组第二行仅有5和6，往下两个都默认为零
    int c[][4] = { {1,2,3,4},{5,6,7,8} }; //正确，有大括号初始化了之后，行数可以省略
    printf("%d", sizeof(c)); //c的大小由赋初值个数决定
    int d[3][] = { {1,2,3} }; /*error,二维数组相当于一维数组中每一个元素都是一个数组

```

//只确定大元素无法确定小元素的个数*/
 /*推广：有n维数组的情况下，在有花括号赋值的情况下
 只有第一个方括号可以省略值的写入*/
 //n维数组和一维数组一样，没有越界检查，注意不要越界！！

```
int e[2][4] = { {1,2,3,4},{5,6,7,8} };
int i = 0;
for (; i < 8; i++)
{
    printf("e[%d][%d]=%x ", i / 4, i % 4, &e[i / 4][i % 4]);
}
printf("\n\n e[0]=%x,e[1]=%x", &e[0], &e[1]); //这里的e[0]和e[1]存放的是紧挨着e的方括号的地址
printf("%x", e[0] + 2); //e[0]+2等价于&e[0][2]
system("pause");
return 0;
}
```

矩阵转置

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
    int i = 0;
    int j = 0;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            printf("a[%d][%d]=%d ", i, j, a[i][j]);
        }
        printf("\n");
    }
    printf("\n\n");
    int b[4][3];
    for (int i=0; i < 4; i++)
    {
        for (int j=0; j < 3; j++)
        {
            b[i][j] = a[j][i];
            printf("b[%d][%d]=%d ", i, j, b[i][j]);
        }
        printf("\n");
    }
    getchar();
    return 0;
}
```

```
a[0][0]=1  a[0][1]=2  a[0][2]=3  a[0][3]=4
a[1][0]=5  a[1][1]=6  a[1][2]=7  a[1][3]=8
a[2][0]=9  a[2][1]=10 a[2][2]=11 a[2][3]=12
```

```
b[0][0]=1  b[0][1]=5  b[0][2]=9
b[1][0]=2  b[1][1]=6  b[1][2]=10
b[2][0]=3  b[2][1]=7  b[2][2]=11
b[3][0]=4  b[3][1]=8  b[3][2]=12
```

数组输出杨辉三角

```
#include<stdio.h>
int main(void)
{
    int a[10][10] = {0};
    int i = 0;
    int j = 0;
    for (i = 0; i < 10; i++)
    {
        for (j = 0; j <= i; j++) //限定了第i行只有i个数
        {
            if (j == 0 || i == j)
            {
                a[i][j] = 1;
            }
            else
            {
                a[i][j] = a[i - 1][j - 1] + a[i - 1][j]; //杨辉三角公式
            }
            printf("%-5d", a[i][j]);
        }
        printf("\n");
    }
    getchar();
    return 0;
}
```

```
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
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

奇怪的知识增加了：鬼火少年

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<windows.h>
int main(void)
{
    char c[100];
    int i = 0;
    while (1)
    {
        for (i = 0; i < 16; i++)
        {
            sprintf(c, "color %x%x", i, 16 - i);
            system(c);
            Sleep(50);
        }
    }
}
```

```

getchar();
return 0;
}

```

代码量最少的排序：冒泡排序

```

#include <stdio.h>
int main(void)
{
    int a[] = { 900, 2, 3, -58, 34, 76, 32, 43, 56, -70, 35, -234, 532, 543, 2500 };
    int n; //存放数组a中元素的个数
    int i; //比较的轮数
    int j; //每轮比较的次数
    int buf; //交换数据时用于存放中间数据
    n = sizeof(a) / sizeof(a[0]); /*a[0]是int型, 占4字节, 所以总的字节数除以4等于元素的个数*/
    for (i = 0; i < n - 1; ++i) //比较n-1轮
    {
        for (j = 0; j < n - 1 - i; ++j) //每轮比较n-1-i次
        {
            if (a[j] < a[j + 1]) //若要升序排列把"<"换成">"
            {
                buf = a[j];
                a[j] = a[j + 1];
                a[j + 1] = buf;
            }
        }
    }
    for (i = 0; i < n; ++i)
    {
        printf("%d\n", a[i]);
    }
    printf("\n");
    return 0;
}

```

程序中，为什么每轮比较的次数是 $j < n - 1 - i$ ，而不是 $j < n - 1$ ？

因为冒泡排序有一个特点，这个程序是从大到小排序，所以第一轮排序以后，最小的数就会浮到最右面；第二轮排序以后，第二小的数会浮到倒数第二个位置；第三轮排序以后，第三小的数会浮到倒数第三个位置……也就是说，排序多少轮，就有多少个数字已经按排序要求排好了，它们不需要再比较。

折半查找

原理



代码：包含随机数生成，排序，查找

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
    //生成随机数
    time_t t;
    srand((unsigned int)time(&t));
    int a[666] = {0};
    int i = 0;
    for (i = 0; i < 666; i++)
    {
        a[i] = rand() % 1000;
    }
    //按冒泡排序将随机数按降序排列
    int j = 0;
    int k = 0;
    for (i = 0; i < 666 - 1; i++)
    {
        for (j = 0; j < 666 - 1 - i; j++)
        {
            if (a[j] < a[j + 1])
            {
                k = a[j];
                a[j] = a[j + 1];
                a[j + 1] = k;
            }
        }
    }
    for (i = 0; i < 666; ++i)
    {
        printf("a[%d]=%d\n", i, a[i]);
    }
    //按折半查找查找想要的数
    int n = 233; //想要的数
    int shang = 0;
    int xia = 665;
    int zhong;
    int ok = 0;
    int r = 0;
    for (r = 0; r < 10; r++)
    {
        zhong = (shang + xia) / 2;
        if (n == a[zhong])
        {
            printf("找到了! ");
            ok = 1;
            break;
        }
        else if (n > a[zhong])
        {

```

```

        xia = zhong + 1;
    }
    else
    {
        shang = zhong - 1;
    }
}
if (ok == 0)
{
    printf("未生成你想要随机数!");
}
getchar();
return 0;
}

```

选择排序

```

#include<stdio.h>
#include<time.h>
#include<stdlib.h>
int main(void)
{
    time_t t;
    srand((unsigned int)time(&t));
    int a[10];
    for (int i = 0; i < 10; i++)
    {
        a[i] = rand() % 100;
        printf("%d ", a[i]);
    }
    printf("\n\n");
    int max = 0;
    for (int i = 0; i < 10-1; i++) //因为倒数第二个和最后一个已经比过了，所以无需i<10
    {
        max = i; //每循环一轮使默认最大值往下走一个
        for (int j = i + 1; j < 10; j++) //a[max]只用往下比就行了
        {
            if (a[max] < a[j])
            {
                max = j;
            }
        }
        if (max != i) //比完之后如果a[max]依旧等于a[i]，那就不用换了
        {
            int k = 0;
            k = a[max];
            a[max] = a[i];
            a[i] = k;
        }
    }
    for (int i = 0; i < 10; i++)
    {
        printf("%d\n", a[i]);
    }
}

```

```
}  
getchar();  
return 0;  
}
```

指针

2021年9月11日 20:27

指针

前情提要

任何程序在数内存，代码数据都有地址，函数名也有地址

32位计算机最大内存为 $4G=2^{32}$ 字节，也就是内存地址由32个二进制位来表示如图：

0000	0000	0000	0000	0000	0000	0000	0000
0	0	0	0	0	0	0	0
1111	1111	1111	1111	1111	1111	1111	1111
F	F	F	F	F	F	F	F

```
num=*(&num)
```

(int *)是一个指向int类型的指针变量，容纳int变量的地址

*p与int对称，*p就是int类型的数据

指针在使用时一定要初始化，不初始化就是野指针，不然会起冲突或触犯权限，引起程序崩溃

空指针指向0x00000000，比较安全，但是不能打印其内容，直接触犯操作系统大哥

p仅仅是起始地址具体多大还得看它前面的声明

指针存储的只是首地址，至于到哪结束由类型决定，类型决定长度和解析方式

不是同一类型的指针不能随便赋值：不同的类型大小和解析方式都不一样

指针的类型必须要与指针指向的类型一致，不一致的话大小（int和double）和解析方式（int和float）都不一样

用地址时必须强制类型转换

```
1 char * p1 = (char *) 0x1ffe4b;
   *p1 = 'M';
   int *p2 = (int *)0x1ffe3c;
   *p2 = 30;
   double *p3 = (double *)0x1ffe2c;
   *p3 = 19.8;
```

指针与指针变量

指针：一个变量的地址

指针变量：专门存放地址的变量

直接调用与间接调用

直接调用就是直接对变量赋值

间接调用就是通过地址来修改变量

指针调换

```
#define _CRT_SECURE_NO_WARNINGS
```

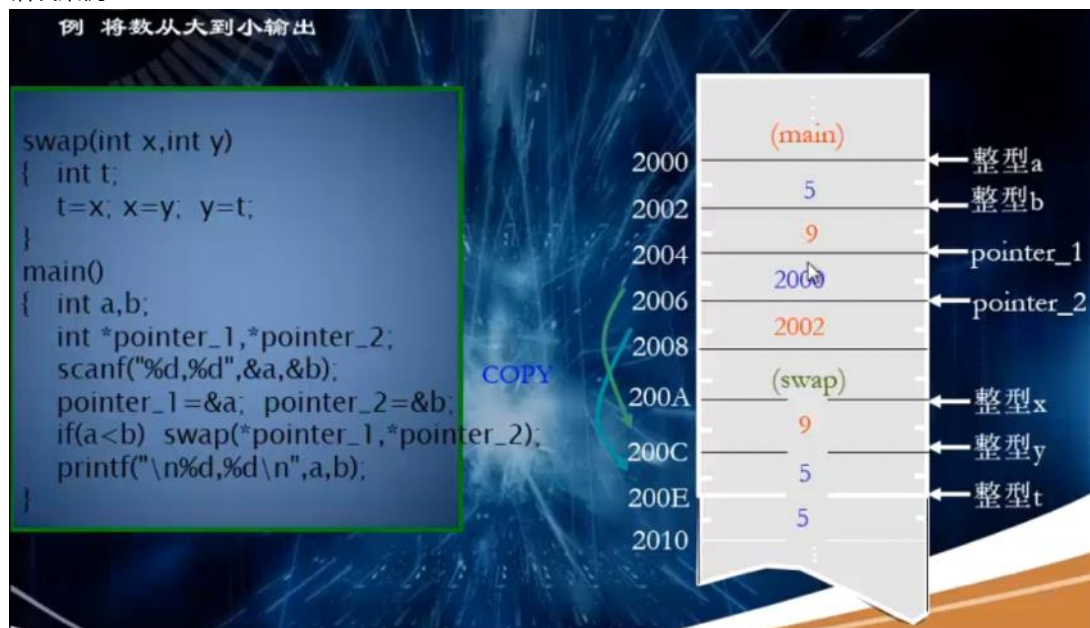
```
#include<stdio.h>
```

```

int main(void)
{
    int a, b;
    int * Pa, * Pb;
    Pa = &a; //指针变量在使用前必须赋值!!
    Pb = &b;
    scanf("%d,%d", Pa, Pb); //这里的Pa和Pb都是地址, 等价于输入
    &a和&b
    printf("*Pa=%d,*Pb=%d\n", *Pa, *Pb);
    getchar(); //输入时有逗号一定要输入逗号
    if (*Pa > *Pb)
    {
        int* p = Pa; //指针的交换, 一般必须要中间变量
        Pa = Pb;
        Pb = p;
    }
    printf("*Pa=%d,*Pb=%d", *Pa, *Pb);
    getchar();
}

```

错误案例1



必须传入指针, 通过地址修改变量!!

错误案例2:

例 将数从大到小输出

```
swap(int *p1, int *p2)
{
    int *p;
    p=p1;
    p1=p2;
    p2=p;
}
main()
{
    int a,b;
    int *pointer_1,*pointer_2;
    scanf("%d,%d",&a,&b);
    pointer_1=&a; pointer_2=&b;
    if(a<b) swap(pointer_1,pointer_2);
    printf("%d,%d",*pointer_1,*pointer_2)
}
```

swap中的int*p1和int*p2是新建的，p1和p2的修改，不会返回到main函数，因此必须修改指针指向的值

//函数中传递的数据，除了数组外，都会新建一个变量接收传入的变量的值，不影响原来的变量

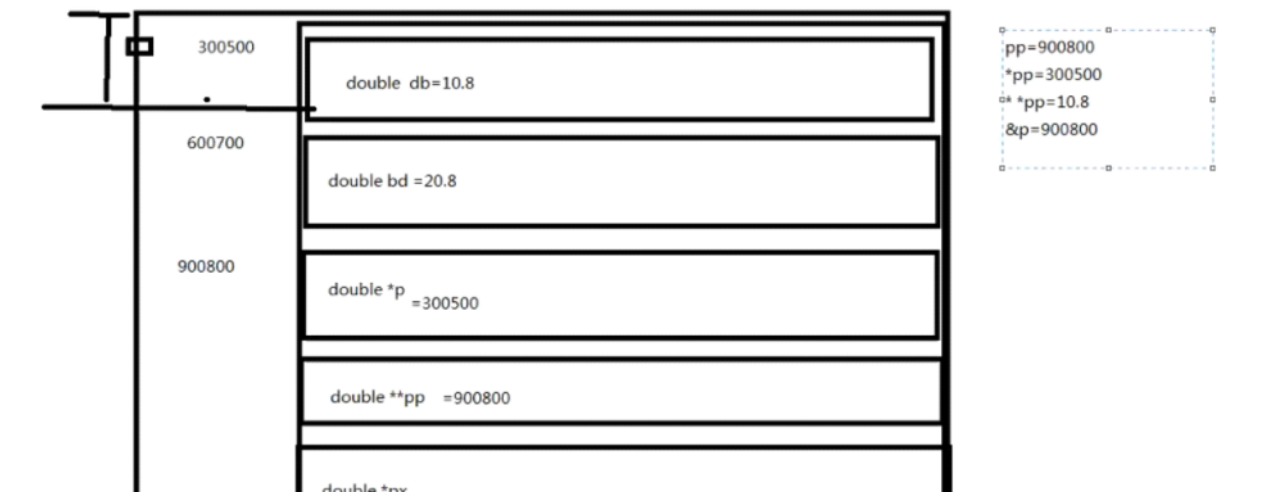
正确案例：

例 将数从大到小输出

```
swap(int *p1, int *p2)
{
    int p;
    p=*p1;
    *p1=*p2;
    *p2=p;
}
main()
{
    int a,b;
    int *pointer_1,*pointer_2;
    scanf("%d,%d",&a,&b);
    pointer_1=&a; pointer_2=&b;
    if(a<b)swap(pointer_1,pointer_2);
    printf("\n%d,%d\n",a,b);
}
```

COPY

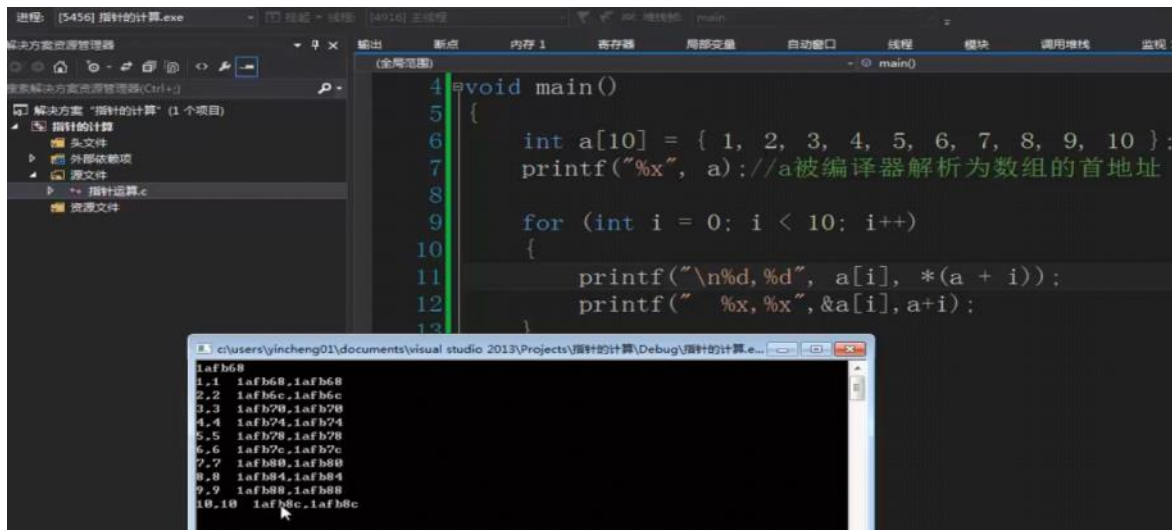
二级指针的理解



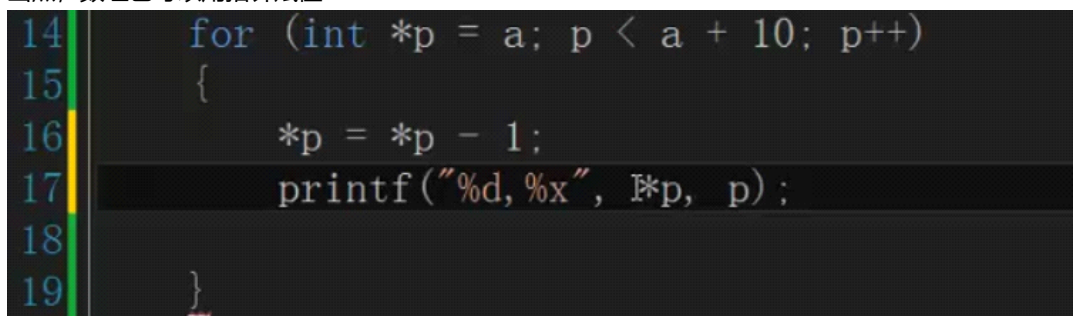
指针的运算

整数与指针最好不要直接运算

在数组中，数组名和下标就是指针



当然，数组也可以用指针赋值



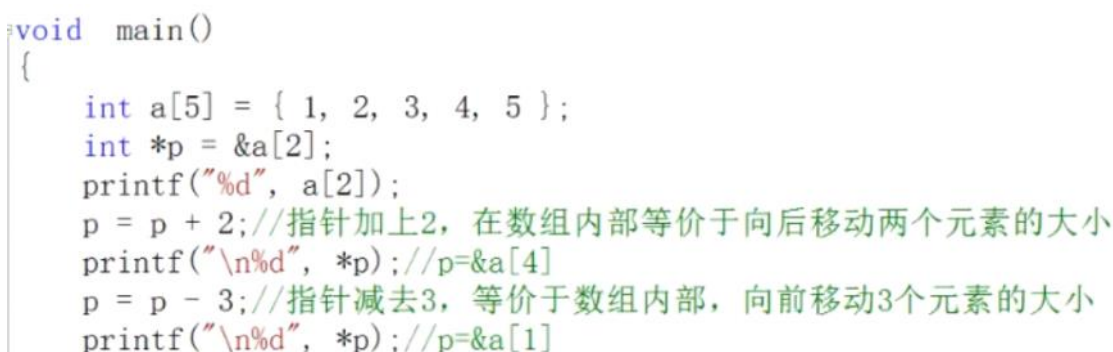
使得a[i]中的值每一个都减少1

指针赋值数组时的++和--都要在数组的范围之内，不要越界

指针p++，就是按照指针对应声明的类型的大小，int前进4个字节

double 8个，只有在数组内部操作才有意义

例：



指针与数组

若有 $p=a$ (p 指向数组 a)，则：

- $p++$ (或 $p+=1$)，表示 p 指向下一元素。
- $*p++$ 与 $*(p++)$ 等价。同样优先级，结合方向为自右向左。
- $*(p++)$ 与 $*(++p)$ 。
前者是先取 $*p$ 的值，后使 p 值加1，相当于 $a[i++]$ ；后者是先使 p 加1，再取 $*p$ ，相当于 $a[++i]$ 。
- $(*p)++$ 表示 p 所指向的元素值加1，而非指针值加1。

(++的优先级比*的优先级高)

指针相减的意义

```
int a = 10;
int b = 20;
int *p1 = &a;
int *p2 = &b;
int num = p1 - p2;
//指针相减，如果值为正，p1在p2后面，值为负，p1在p2前面
//具体之差就意味着指针之间相隔几个元素的大小
//具体只差不是地址之差，而是地址之差除以指向元素的大小
printf("%p, %p, %d", p1, p2, num); //num=有符号的整数
```

但是这两个指针所指向的元素没有结构上的关系，不在同一片内存，求差没啥用。。。

重要区分

1.

$\text{int } *pa[5]$ ，首先 $[]$ 优先级比 $*$ 高，所以 pa 与 $[]$ 先结合， $pa[5]$ 表明 pa 是一个数组，大小是5，既然知道 pa 是数组了，接下来就是确认数组元素了， $\text{int } *$ 表明数组元素是指针；
 $\text{int } (*p)[5]$ ，首先 $()$ 优先级比 $[]$ 高，所以 pa 先与 $*$ 结合， $*pa$ 表明 pa 是一个指针，既然知道 pa 是指针，接下来确认指针指向的数据类型， $\text{int } [5]$ 表明指针指向大小为5的 int 型数组。

```
int *p = a; //指向元素的指针
int (*pa)[5] = &a; //指向数组的指针
```

2.

```
int a[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
printf("%p, %p, %p", a, &a, *a);
//a是一个行指针，I 指向一个有四个元素的数组，16
//&a是一个指向二维数组的指针，二维数组有12个元素，48
//*a是一个指向int类型数据的指针， 4

printf("\n%d, %d, %d", sizeof(*a), sizeof(&a), sizeof(**a));
```

备注： $*\&a==a==((a+0)+0)$ 是个代表数组全部的指针；

$*a==(*(a+0)+0)$ 是个行指针；

$**a==**(*(a+0)+0)$ 是个列指针。

3.

由调试结果可知, *a, *a+1, *a+2等均为列指针, 等价于*(a+0),*(a+0)+1),*(a+0)+2)

用指针线性访问二维数组

```
void main()
{
    int a[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    printf("%p\n", a);

    for (int *p = &a[0][0]; p < &a[0][0] + 12; p++)//&a[0][0]    int*
    {
        printf("%d,%p\n", *p, p);
    }

    system("pause");
}
```

用指针输出

```
int a[3][4] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
//int **p=a;
printf("%d", sizeof(*a)); //a是一个行指针, 16个字节, 每一行有四个元素
int (*p)[4]=a; //二维数组的指针就是一个指向一维数组的指针, 元素是确定的
int i = 0;
int j = 0;
scanf("%d%d", &i, &j);
printf("%d,%d,%d,%d", i, j, p[i][j], (*(p + i) + j));

//int a[10]数组作为函数参数, 传递的是地址, 地址就是指针占4个字节,
//函数的参数对于数组没有副本机制, 为了节约内存, 拷贝数组浪费空间与CPU
void test(int a[10])
{
    printf("\ntest=%d", sizeof(a)); //
}

void main()
{
    int a[10];
    printf("\nmain=%d", sizeof(a));
    test(a);
}
```

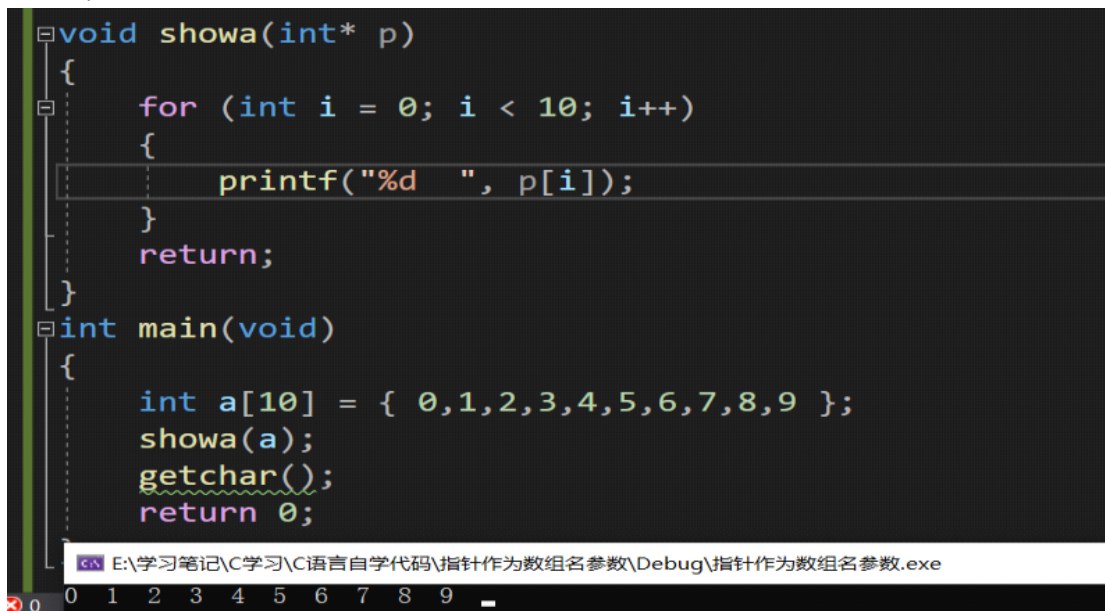
数组没有副本机制, 在函数中操作相当于直接操作原生数组

test中的a其实是个地址

所以, test=4, main=40.

所以我们可以用指针作为数组名参数

1. 一维



```
void showa(int* p)
{
    for (int i = 0; i < 10; i++)
    {
        printf("%d ", p[i]);
    }
    return;
}

int main(void)
{
    int a[10] = { 0,1,2,3,4,5,6,7,8,9 };
    showa(a);
    getchar();
    return 0;
}
```

E:\学习笔记\C语言\C语言自学代码\指针作为数组名参数\Debug\指针作为数组名参数.exe

1. 二维

```
指针作为数组名参数 (全局范围) showb(int(*p)[4])
16 void showb(int (*p)[4]) //注意加括号, 星号结合能力没有方括号大
17 {
18     for (int i = 0; i < 3; i++)
19     {
20         for (int j = 0; j < 4; j++)
21         {
22             printf("%-3d ", p[i][j]);
23         }
24         printf("\n");
25     }
26     return 0;
27 }
28 int main(void)
29 {
30     int b[3][4] = { {0,1,2,3},{4,5,6,7},{8,9,10,11} };
31     showb(b);
32     getchar();
33     return 0;
34 }
```

0	1	2	3
4	5	6	7
8	9	10	11

指向函数的指针

- 定义指向函数的指针变量的一般形式为
数据类型 (*指针变量名)(函数参数表列);
如 int (*p)(int,int);
p=max; 对
p=max(a,b); 错
p+n,p++,p--等运算无意义

```
指向函数的指针 (全局范围) main(void)
1 #include<stdio.h>
2 #include<windows.h>
3 void showb(int(*p)[4]);
4 int main(void)
5 {
6     int b[3][4] = { {0,1,2,3},{4,5,6,7},{8,9,10,11} };
7     int (*p)(int (*p)[4]) = showb;
8     p(b);
9     getchar();
10    return 0;
11 }
12 void showb(int(*p)[4]) //注意加括号, 星号结合能力没有方括号大
13 {
14     for (int i = 0; i < 3; i++)
15     {
16         for (int j = 0; j < 4; j++)
17         {
18             printf("%-3d ", p[i][j]);
19         }
20         printf("\n");
21     }
22     return;
23 }
```

0	1	2	3
4	5	6	7
8	9	10	11

(函数指针的参数名也可以省略)

在函数里用函数指针接收另一个函数的返回值充当桥梁返回给main

函数

```
#include<stdio.h>
int sort(int, int, int* (int, int));
int max(int, int);
```

```

int min(int , int );
int main(void)
{
    int a = 10;
    int b = 20;
    int c = 0;
    int d = 0;
    scanf_s("%d", &c);
    getchar();
    switch (c)
    {
        case 1:
        {
            d = sort(a, b, max);
            break;
        }
        case 2:
        {
            d = sort(a, b, min);
            break;
        }
        default:
        {
            break;
        }
    }

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

int sort(int a,int b, int* p(int, int)) //函数指针不仅仅是地址，必须明确
函数指针类型，并且知道输入的参数是什么类型，有几个
//返回值是什么类型，类型不匹配，
仅仅知道地址，也无法调用

{
    return p(a, b);
}

int max(int a, int b)
{
    if (a > b)
    {
        return a;
    }
    else
    {
        return b;
    }
}

```

```

int min(int a, int b)
{
    if (a > b)
    {
        return b;
    }
    else
    {
        return a;
    }
}

```

函数返回指针，并输出指针中存放的数

```

#include<stdio.h>
#include<time.h>
int* pmax(int a[], int );
int main(void)
{
    time_t t;
    srand((unsigned int)time(&t));
    int a[10];
    for (int i = 0; i < 10; i++)
    {
        a[i] = rand() % 100;
    }
    int* pMax = pmax(a, 10);
    printf("最大值是%d", *pMax);
    getchar();
    return 0;
}

int* pmax(int a[],int n)
{
    int max = a[0];
    int* p = &a[0];
    for (int i = 1; i < n; i++)
    {
        if (max < a[i])
        {
            max = a[i];
            p = &a[i];
        }
    }
    return 0;
}

```

指向为空的指针（void指针）


```

int num = 10;
double db = 10.8;
int*p2 = &num;
double *p3 = &db;

void *p1 = p2;//void类型的指针可以传递地址，
//p1 = p3;
//printf("%d", *p1);void类型的指针，由于指向不明确，大小不确定，所以无法取出内
printf("%d", *((int *)p1));

//用于参数还有返回值，不明确指针类型的情况传递地址需要用到空类型的指针
//要把它用于某种类型的指针，需要强制转换。I

```

任何指针都可以赋值给空类型的指针，用于保存地址

malloc与free

malloc函数返回的是空指针(void*)，所以一般要在其前面加强制类型转换

malloc使用完内存后必须用free释放

malloc和calloc的区别

```

int num;
scanf("%d", &num);
printf("num=%d", num);
int *p = (int *)calloc(num, sizeof(int));//动态分配，
//第一个参数，多少个，第二个参数元素占多大 I
//calloc会自动将内存初始化为0，malloc就不会I

```

除传入参数和是否对分配的内存进行初始化外，二者没有任何区别，都要有free释放

realloc函数

现在我有17个内存，我想保留原来不变，加8个内存，就需要realloc函数

■ realloc()函数有两个参数

- 已分配的内存地址
- 重新分配的字节数

```

void *realloc( void *ptr, size_t size );

```

```

#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>

```

```

int main(void)
{
    int num = 0;
    scanf("%d", &num);
    getchar();
    int* p = (int*)malloc(num * sizeof(int));
    if (p == NULL)
    {
        printf("内存分配错误");
        exit(1);
    }
}

```

```

}
else
{
    for (int i = 0; i < num; i++)
    {
        p[i] = i + 1;
        printf("%d\n", p[i]);
    }
}
printf("%p\n", p);
printf("请输入扩大后的总数\n");
int add = 0;
scanf("%d", &add);
getchar();
//重新分配内存，如果可以拓展就拓展，否则就重新分配
//拓展就是原来地址后面增加内存
//不够的情况下，就回收原来的内存，并且在回收之前，将原来的内
容拷贝一份
int* px = (int*)realloc(p, (add + num) * sizeof(int));
if (px == NULL)
{
    printf("内存分配错误");
    exit(1);
}
else
{
    for (int i = num; i < num + add; i++)
    {
        px[i] = i + 1;
        printf("%d\n", px[i]);
    }
}
printf("%p\n", px);

free(px);
//软件工程规范，内存释放之后，指针应该赋值为空，就可以避免再
次引用以及反复释放的问题
px = NULL;
free(p);
p = NULL;
getchar();
return 0;
}

```

内存泄漏实例

```
while (1)
{
    void *p= malloc(100 * 1024 * 1024); //内存泄露
    p = NULL; //指针消亡或者指针发生变化，对应的内存不会被释放，就会引起内存泄露
    Sleep(2000);
    free(p);
    Sleep(2000);
}
```

了解：32位开发和64位开发区别

寻址能力会变得非常强，可以访问海量内存

64位系统可以同时运行32和64位应用程序，32位只能运行32位

了解：Debug和Release的区别

debug主要是供你调试用的，他生成的程序会比较大

release是供发行用的，他会对代码有所精简，省去了你调试和输出信息，生成的程序小，运行的快

字符串

2021年9月11日 20:27

字符串概览

1. 字符串与字符数组的关系

C语言中没有字符串这种数据类型，可以通过char的数组来代替

字符串一定是一个char的数组，但char的数组未必是字符串

数字0(和字符'\0'等价)结尾的char数组就是一个字符串，但如果char数组没有以数字0结尾，那就不是一个字符串，只是普通字符数组，所以字符串是一种特殊的char数组。

2. 字符数组元素的初始化

```
//数组元素初始化的时候，如果没有指定后面的元素，  
//一般会默认为0。  
//编号为0的字符就是'\0'
```

3. 字符串常量与字符串变量

(1) 字符串常量

```
#include<stdio.h>  
int main(void)  
{  
    char* ch = "china is great"; //这是一个指向字符串常量的指针  
    printf("%c\n", *ch);  
    printf("%d\n", sizeof(ch)); //ch是字符串常量的首地址，占4字节  
    printf("%d\n", sizeof("china is great")); //加上'\0'，占15个字节  
    for (int i = 0; i < 14; i++)  
    {  
        printf("%c", ch[i]); //输出前14个，依然正确输出  
    }  
    /*ch='A'; //常量不可写入，不可修改，赋值  
    getchar();  
    return 0;  
}
```

(2) 字符串变量

```
#include<stdio.h>  
int main(void)  
{  
    char ch[10] = "balc"; //将字符串赋给字符数组ch  
    //从头到尾开始填充，没赋值到的赋值为0，0是'\0'的ASCII码  
    char* pch = ch; //指针存储字符数组的地址  
    pch[0] = 'c'; //指针修改字符数组第一个字符，等价于ch[0]='c'  
    system(ch);  
    system(pch); //弹出两个计算器  
    getchar();  
    return 0;  
}
```

4. 作为一种特殊的数组，字符数组所能享受的“特殊待遇”就是允许整体输入和输出，数组操作的其他限制对字符数组同样成立，如不允许使用一个数组给另一个数组赋值，不允许整体比较两个数组，不允许对两个进行整体的算术操作，下列用法都是错误的：

5. ``` //char str[10]-c a l c \0 \0 \0 \0 \0 \0-10，数组长度就是多少个元素 //char str[10]-c a l c 4 字符串呢，遇到/0之前有多少个元素 ```

字符数组的初始化

```

void main()
{
    //char str[100] = { "color 4f" };
    char str[100] = "color 4f" ;
    //字符数组可以用字符串初始化
    // 大括号往往可以省略
    system(str);
    system("pause");
}

```

字符数组的输入输出

```

#define _CRT_SECURE_NO_WARNINGS //去掉安全检查
#include<stdio.h>
#include<stdlib.h>

```

```

void main()
{
    char str[100] = { 0 }; //初始化字符数组
    scanf("%s", str); //根据键盘输入初始化
    printf("%s", str); //打印字符串
    system(str); //字符串按照指令来执行
}

```

一些案例

6. 错误

```

char str[100];
//str[100] = "tasklist"; //str[100]代表第101个元素，越界

```

7. 错误

```

//str = "tasklist"; str是数组名，是常量不可以修改

```

8. 正确

```

//char *p = "tasklist"; //p存储了字符串常量的首地址
char *p;
p = "tasklist";

```

字符串与字符数组的关系

9.

```

char str[]={"Hello!"};           (√)
char str[]="Hello!";             (√)
char str[]={'H','e','l','l','o','!'}; (√)
char *cp="Hello!";               (√)
int a[]={1,2,3,4,5};             (√)
int *p={1,2,3,4,5};              (×)

```

10.

```

char str[10],*cp;
int a[10],*p;
str="Hello";    (×)
cp="Hello!";    (√)
a={1,2,3,4,5}; (×)
p={1,2,3,4,5}; (×)

```

对于cp，cp存储常量字符串首地址，所以正确，但是注意hello此时是一个常量字符串，无法修改

而且cp只能在定义的时候初始化，不能用scanf函数进行初始化，但是由于cp是指针变量，可以改变地址所以可以直接更改

```

char *p = "calc"; //指针变量，可以改变地址
system(p);
p = "notepad";
system(p);

```

此时可以既打开计算器又打开记事本

对于p，p是指针，指针不可以用数组初始化

11. 字符串有两种输入和输出方式

```
scanf("%s",str);
printf("%s",str);
gets(str);
puts(str);
```

12. 字符指针特性

```
char *p = "123notepad";
p = p + 3; //字符指针, 改变地址, 可以决定字符串从哪里开始
printf("%s", p);
system(p);
getchar();
```

scanf, printf; gets, puts的区别

```
char str1[100];
char str2[100];
scanf("%s%s", str1, str2); //空格, tab都会当做一个字符串输入结束
printf("%s, %s", str1, str2); //打印字符串
system("pause");
```

```
char str1[100];
char str2[100];
gets(str1); //可以接受空格, tab等等按键, 以回车为结束
gets(str2);
puts(str1); //输出字符串并换行
puts(str2);
```

sprintf函数

sprintf()函数用于将格式化的数据写入字符串

sprintf()最常见的应用之一莫过于把整数打印到字符串中, 如:

```
sprintf(s, "%d", 123); //把整数123打印成一个字符串保存在s中
sprintf(s, "%8x", 4567); //小写16进制, 宽度占8个位置, 右对齐
```

sscanf函数

sscanf()会将参数str的字符串根据参数format字符串来转换并格式化数据。格式转换形式请参考scanf()。转换后的结果存于对应的参数内。

```
char str[100] = "我有1000元";
int num;
sscanf(str, "我有%d元", &num);
printf("%d", num);
```

将数字从字符串中分离出来给num

strcmp函数

用来比较字符串(区分大小写), 其原型为:

int strcmp(const char *s1, const char *s2); 【参数】s1, s2 为需要比较的两个字符串。

字符串大小的比较是以ASCII 码表上的顺序来决定, 此顺序亦为字符的值。

strcmp()首先将s1 第一个字符值减去s2 第一个字符值, 若差值为0 则再继续比较下个字符, 若差值不为0 则将差值返回。

例如字符串"Ac"和"ba"比较则会返回字符'A'(65)和'b'(98)的差值(- 33)。

即当比较到第一个不一样的字符的时候, 返回他俩ASCII码的差

注意: strcmp() 以二进制的方式进行比较, 不会考虑多字节或宽字节字符

返回值<0, 第一个字符串比较小; 返回值=0, 两个字符串一样大; 返回值>0, 第一个字符串比较大

备注: Windows排序忽略大小写, 而strcmp严格区分大小写

strncmp函数

strncmp (第一个字符串, 第二个字符串, 比较前几位)

```

char str1[30] = "notepad1";
char str2[30] = "notepadcalc";
if (strncmp(str1, str2, 7) == 0)
{
    printf("相等");
}
else
{
    printf("不相等");
}

```

有n顾名思义就是比较前n位，n为手动输入值

[_strupr函数](#)

上面说到，Windows排序忽略大小写，而strcmp区分大小写，若要模拟Windows下比大小，则需要_strupr函数

int _strupr(const char *s1); //将整个字符串的字符全部升级为大写

实战实现：

```

void tobig(char *p)
{
    while (*p != '\0') //判断字符串是否结束
    {
        if ((*p) >= 'a' && (*p) <= 'z') //判断是否小写字母
        {
            *p = *p - 32; //小写转大写
            //a 97, A65 -32
        }
        p++;
    }
}

```

[_strlwr函数](#)

int _strlwr(const char *s1); //将整个字符串的字符全部降级为小写

实战实现：

```

void tosmall(char *p)
{
    while (*p != '\0') //判断字符串是否结束
    {
        if ((*p) >= 'A' && (*p) <= 'Z') //判断一下字符是否大写字
        {
            *p = *p + 32; //将大写转换小写
            //a 97, A65 -32
        }
    }
}

```

[上方两个函数的综合实战](#)

```
#define _CRT_SECURE_NO_WARNINGS
```

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

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

```
#include<math.h>
```

```
int main(void)
```

```
{
```

```
char str1[10] = "leigenb";
```

```
char str2[10] = "leigehb";
```

```
_strupr(str1);
```

```
_strupr(str2);
```

```
printf("%s, %s\n", str1, str2);
```

```
int i = strcmp(str1, str2);
```

```
if (i == 0)
```

```
{
```

```
printf("一样大\n");
```

```

}
else if (i < 0)
{
printf("str2大\n");
}
else
{
printf("str1大\n");
}
int j = 0;
while (str1[j] != '\0' && str2[j] != '\0')
{
if (str1[j] != str2[j])
{
printf("在第%d个字符不同,相差%d个ASCII码大小\n", j + 1, abs(str1[j] - str2[j]));
}
j++;
}
getchar();
return 0;
}

```

E:\学习笔记\C学习\C语言自学代码\strcmp和_strupr\Debug\strcmp和_strupr.exe

```

LEIGENB, LEIGEB
str1大
在第6个字符不同,相差6个ASCII码大小

```

strchr函数

strchr(被查找的字符串, '想查找的字符')

```

#include<stdio.h>
#include<string.h>
int main(void)
{
char str1[10] = "leigenb";
char*p = strchr(str1, 'e'); //第一个变量不一定非得是一个字符串首地址
printf("%p, %c", p, *p); //还可以是如str1+3等地址
getchar();
return 0;
}

```

strncat函数

```

char str[30] = "yincheng";
char str1[20] = "8848.88";

```

```

strncat(str, str1, 4); //从str1中拷贝四个字节到str
printf("%s", str);

```

```

getchar();

```

对strncat的模拟实验

```

#include<stdio.h>
#include<string.h>
void mystrncat(char*, char*, int);

```

```

int main(void)

```

```

{
char str1[10] = "ca";
char str2[10] = "lc";
mystrncat(str1, str2, 2);
printf("%s\n", str1);
system(str1);
getchar();
return 0;
}

void mystrncat(char* p1, char* p2, int length)
{
if (p1 == NULL || p2 == NULL)
{
printf("出现错误");
return;
}
while (*p1 != 0)
{
p1++; //让p1的指针指向'\0'
}
for (int i = 0; i < length; i++)
{
*p1 = p2[i]; //在p1地址的结尾在确保不越界的情况下强行加上p2的字符
p1++;
}
}

```

和strcmp一样，strcat也有加n的版本（strncat，strncmp）可以控制比较或者粘贴的字符个数

atoi函数

```

char str[10] = "e8848";
int num = atoi(str);
//转换的时候，传递字符串的首地址，地址不要求是首地址，
//字符串的任何地址都可以，num起到接受赋值的作用，
//转换成功就是整数，失败就是0，出现非数字字符都会转换失败

printf("%d", num);

```

strset函数

```

char str[20] = "yincheng8848";
printf("str原来%s", str);
//_strset(str, '8'); //_strset标准C语言
_strset(str, '\0');
printf("str后来%s", str);

```

```
getchar(); //等待
```

可以用来将一个字符数组内字符原先非'\0'的部分全变成一个字符

当然也可以全部赋值为'\0'来初始化字符数组

strrev函数

```
char str[20] = "yincheng8848";
printf("str原来%s", str);
//_strset(str, '8'); //_strset标准C语言
_strset(str, '\0');
printf("str后来%s", str);
```

```
getchar(); //等待
```

将字符串逆转

```
void mystrev(char *p)
{
    int length = strlen(p); //获取字符串长度
    for (int i = 0; i < (length / 2); i++) //循环一半，交换字符
    {
        //对调字符
        char ch = p[i];
        p[i] = p[length - i - 1];
        p[length - i - 1] = ch;
    }
}
```

strstr函数

strstr()函数用来检索子串在字符串中首次出现的位置，其原型为：

```
char *strstr( char *str, char * substr );
```

【参数说明】str为要检索的字符串，substr为要检索的子串。

【返回值】返回字符串str中第一次出现子串substr的地址；如果没有检索到子串，则返回NULL。

【函数示例】strstr()函数的使用。

手写strstr函数

```
#include<stdio.h>
char* mystrstr(char* , char* );
int main(void)
{
    char str1[50];
    char str2[30];
    gets(str1);
    gets(str2);
    char*p = mystrstr(str1, str2);
    printf("地址是%p", p);
    getchar();
    return 0;
}
char* mystrstr(char* mstr, char* sstr)
{
    int mnum = 0;
    int snum = 0;
    for (int i = 0; mstr[i] != '\0'; i++)
    {
        mnum++;
    }
    for (int i = 0; sstr[i] != '\0'; i++)
    {
        snum++;
    }
    printf("母字符串%d个单位, 子字符串%d个单位\n", mnum, snum);
    char* p = NULL;
    if (mstr == NULL || sstr == NULL)
    {
        printf("检索没有重合部分且其中一个为无效字符串\n");
    }
}
```



```

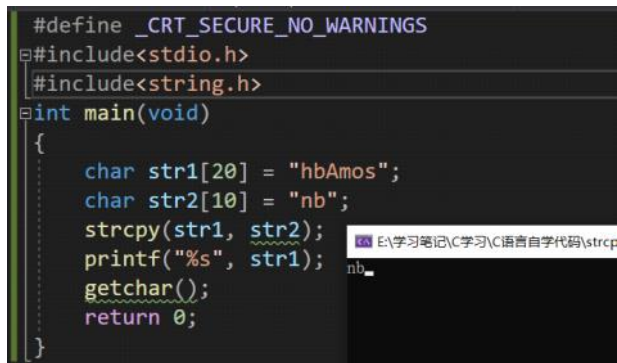
return p;
}
for (int i = 0; i < mnum - snum; i++) //联想RNA聚合酶在DNA上寻找结合位点的过程
{
    int flag = 1;
    for (int j = 0; j < snum; j++)
    {
        if (mstr[i + j] !=sstr[j])
        {
            flag = 0;
            break;
        }
        if (flag == 1)
        {
            printf("检索有重合部分,");
            p = mstr + i + j;
            return p;
        }
    }
}
}
printf("检索没有重合部分");
return p;
}

```

strcpy函数

C 库函数 `char *strcpy(char *dest, const char *src)` 把 `src` 所指向的字符串复制到 `dest`。

需要注意的是如果目标数组 `dest` 不够大，而源字符串的长度又太长，可能会造成缓冲溢出的情况。



```

#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<string.h>
int main(void)
{
    char str1[20] = "hbAmos";
    char str2[10] = "nb";
    strcpy(str1, str2);
    printf("%s", str1);
    getchar();
    return 0;
}

```

一定要注意，用于粘贴的字符的字符数组（`str1`）一点要比用于复制的字符数组（`str2`）大，以免造成溢出

str的综合运用面试题

13.

实现插入

//目标，在"hello world,hello Amos,hello c."中间第二个逗号后面插入"hello windows,"

```

#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<string.h>
int main(void)
{
    char mstr[60] = "hello world,hello Amos,hello c.";
    char s1str[16] = "hello windows,";
    char* p = strstr(mstr, "Amos,"); //检索到Amos, 返回A的地址给p
    char s2str[10]; //创建临时存储字符串, 存储hello c.

```



```
strcpy(s2str, p + 5); //将hello c.插入临时字符串
*(p + 5) = '\0'; //放弃mstr中hello Amos, 后面的内容
strcat(mstr, s1str); //缝合怪1
strcat(mstr, s2str); //缝合怪2
printf("%s", mstr); //打印字符串
getchar();
return 0;
}
```

```
E:\学习笔记\C学习\C语言自学代码\字符串插入面试题\Debug\字符串插入面试题.exe
hello world,hello Amos,hello windows,hello c.
```

14.

删除全部字符串中的某个字符

//目标: 在保留源字符串的字符不变的情况下, 生成一个字符串, 使其满足从中连续抠掉一个确定的字符

```
#include<stdio.h>
#include<string.h>
int main(void)
{
char mstr[50] = "hello world,hello Amos,hello c.";
char ch = 'l';
char sstr[50] = { 0 };
char* p = mstr;
int i = 0;
while (*p != '\0')
{
if (*p != ch)
{
sstr[i] = *p; //在*p!='\0'情况下, 遍历sstr
i++;
}
else {} //什么都不用做, 跳过即可
p++;
}
printf("%s", sstr);
getchar();
return 0;
}
```

```
E:\学习笔记\C学习\C语言自学代码\删除字符面试题\Debug\删除字符面试题.exe
heo word,heo Amos,heo c.
```

15.

字符数组与整数的互化

(1) 字符串转整数

```
#include<stdio.h>
#include<string.h>
void change1(char*, int*);
int main(void)
{
char str[10] = "12345678";
int a = 0;
int *p = &a; //让p存放a的地址, 用来在函数内部改变a
```

```

change1(str, p);
printf("%d\n", a); //输出整数a
getchar();
return 0;
}
void change1(char* ps, int* p)
{
while (*ps != '\0')
{
*p *= 10;
*p += (int)*ps - 48;
ps++;
}
return;
}

```

E:\学习笔记\C学习\C语言自学代码\整数和字符串互化\Debug\整数和字符串互化.exe

12345678

(2) 整数转字符串

```

#include<stdio.h>
#include<string.h>
void change2(char* , int , int );
int main(void)
{
int a = 12345678;
char str[10] = { 0 };
change2(str, a, 10);
_strrev(str); //使字符串倒置
printf("%s", str);
getchar();
return 0;
}
void change2(char* ps, int a,int strlen)
{
for (int i = 0; i < strlen-1; i++) //保证其还是字符串，结尾有'\0'，不越界
{
ps[i] = (char)(a % 10 + 48); //取余数
a /= 10;
if (a % 10 == 0) //余数等于零就停止
{
break;
}
}
}
return;
}

```

E:\学习笔记\C学习\C语言自学代码\整数和字符串互化\Debug\整数和字符串互化.exe

12345678_

C语言中Unicode问题

前情提要：

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

```

int main(void)
{
    //C语言是老外写的，只有英文（ASCII码表内的）才是窄字符，中文等文字占2个char单位
    //所以即便是一个汉字也不能用char a='我'来实现
    char str[20] = "我是个好鸟";
    //str占2*5=10个字节
    printf("%s\n", str); //输出中文字符串
    printf("%c%c", str[0], str[1]); //输出中文 '我'
    getchar();
    return 0;
}

```

16. 那么我们怎么才能让中文占一个单位（但是内存依然是两个字节）使操作更方便呢？

我们需要引入宽字符

```

1  #include<stdio.h>
2  #include<locale.h> //设置本地化
3  #include<string.h>
4
5  int main(void)
6  {
7      setlocale(LC_ALL, "chs"); //chinese simple的缩写,用来设置语言
8      wchar_t ch = L'好';
9      printf("%d\n", sizeof(ch));
10     wprintf(L"%wc\n", ch); //将汉字当作一个字符，打印'我'
11     wchar_t str[5] = L"我是好人";
12     wprintf(L"%ws\n", str); //打印字符串
13     wchar_t* p = strchr(str, ch); //寻找'好'并返回首地址
14     printf("%p\n", p);
15     wprintf(L"%wc", *p); //打印'好'
16
17     getchar();
18     return 0;

```

这样我们就可以用string.h头文件去操作宽字符串数组

17. 我们在windows下编程，存在多字符集和Unicode字符集，二者的编译是有区别的，比如MessageBox就会出现字符集问题

```

#include<stdio.h>
#include<windows.h>

MessageBoxA(0, "A我123", "A我123", 0);
//无论是多字节还有unicode，我就是多字节

MessageBoxW(0, L"我123", L"我123", 0);
//无论是多字节还有unicode，我就是unicode

MessageBox(0, L"我123", L"我123", 0);
//使用多字节字符集  MessageBox(0, "我123", "我123", 0);
// unicode  MessageBox(0, L"我123", L"我123", 0);

MessageBox(0, TEXT("AW我123"), TEXT("AW我123"), 0);
//自动适应unicode或者多字节

```

memset函数

```
void *memset(void *s, int ch, size_t n);
```

函数解释：将s中当前位置后面的n个字节（typedef unsigned int size_t）用ch替换并返回s。

memset：作用是在一段内存块中填充某个给定的值，它是对较大的结构体或数组进行清零操作的一种最快方法

```
char str[100] = "hello yincheng hello c hello cpp";
printf("%s", str);
memset(str, 'A', 10);
printf("\n%s", str);
memset(str, 'A', 11);
//第一个参数是内存的首地址, 第二个参数要设置的字符
//第三个参数是整数, 从首地址开始前进多少字节, 吧这一段设置
memset(str, '0', strlen(str)); //清空字符串的作用
printf("\n%s", str);
getchar();
```

memset和strset一样, 都可以将数组初始化, 但是memset可以掌握初始化数组元素的个数, 而且可以用于其他非字符类数组

注意:

```
#include<stdio.h>
#include<memory.h>
int main(void)
{
    int a[10] = { 1,2,3,4,5 };
    memset(a, 0, 8); //这里面第三个参数是应该是该数组数据类型size的倍数
    for (int i = 0; i < 10; i++)
    {
        printf("%d", a[i]);
    }
    getchar();
    return 0;
}
```

因为8是4的两倍, 所以更改了前两字符的数据

memcpy函数

```
#include<stdio.h>
#include<memory.h>
int main(void)
{
    char str1[20] = "hbAmos";
    char str2[10] = "nbleige";
    memcpy(str1, str2, 2); //从头开始覆盖str前2个字符
    printf("%s", str1);
    getchar();
    return 0;
}
```

注意:

```
void main()
{
    int a[] = { 1, 3, 5, 7, 9 };
    int b[] = { 0, 2, 4, 6, 8 };
    memcpy(a, b, 8); //每个元素四个字节, 两个元素8个字节
}
```

解释和memset函数的解释一样, 能处理多种类型的数组

和strcpy相比他还能控制更改的个数, 而strcpy就是一个先清空字符数组再粘贴的过程

_memccpy函数

和memcpy相比, 他能控制读取到某一个数据就停止

某些时候我们可以把某个字符设置为结束符, 代表结束, 后面的不再替换

```
#include<stdio.h>
#include<memory.h>
int main(void)
{
    int a[10] = { 1,2,3,4,5,6,7 };
    int b[5] = { 4,3,2,1,0 };
    _memccpy(a, b, 1, 20); //第三个变量为标记为结束的数据, 计算机读取到这里将停止读取
    //第四个变量是向下读取的字节数, 应该是数组变量类型的n倍
    //如果在指定的字节数中没有读到代表结束的字符, 那么他和memcpy没有什么区别
    for (int i = 0; i < 10; i++)
    {
        printf("%d", a[i]);
    }
    getchar();
    return 0;
}
```

复制到第四个结束读取, 没有读取到0

memchr函数

定义函数: void * memchr(const void *s, char c, size_t n);

函数说明: memchr()从头开始搜寻s 所指的内存内容前n 个字节, 直到发现第一个值为c 的字节, 则返回指向该字节的指针。

返回值：如果找到指定的字节则返回该字节的指针，否则返回NULL。

```
void main()
{
    char str[30] = "helloyincheng";
    char ch = 'c';
    char *p = memchr(str, ch, 30);
    //从地址str开始，前进30个字节检索
    //如果存在，返回地址，否则返回为空
    if (p==NULL)
    {
        printf("没有找到");
    }
    else
    {
        printf("找到字符%c，地址%p", *p, p);
    }
}
```

memcmp函数

memcmp(void *buf1, void *buf2, unsigned int count);

功能：比较内存区域buf1和buf2的前count个字节但不区分字母的大小写。

说明：memcmp同memcmp的唯一区别是memcmp不区分大小写字母。

```
void main()
{
    char *buf1 = "ABCDE123";
    char *buf2 = "abcde456";

    int num = memcmp(buf1, buf2, 5);
    if (num == 0)
    {
        printf("忽略大小写的情况相等");
    }
    else
    {
        printf("忽略大小写的情况不相等");
    }
}
```

不区分大小写的情况下，相等返回0，不相等返回非0

strcmp函数

参考上方，区分大小写比较，可以比较字符数组以外的东西

返回值和strcmp一样，分为大于，小于和等于三种情况

附件：stdlib.h下的字符串函数

一、atoi函数（将字符串转换成整型数）

定义函数 int atoi(const char * nptr);

函数说明 atoi()会扫描参数nptr字符串，跳过前面的空格字符，直到遇上数字或正负符号才开始做转换，而再遇到非数字或字符串结束时('\0')才结束转换，并将结果返回。

返回值：返回转换后的整型数。

附加说明 atoi()与使用strtol(nptr, (char**)NULL, 10); 结果相同。

例程序：

```
#include <ctype.h>
#include <stdio.h>
int atoi (char s[]);
int main(void )
{
    char s[100];
    gets(s);
    printf("integer=%d\n",atoi(s));
    return 0;
}
int atoi (char s[])
{
    int i;
```

```

int i,n,sign;
for(i=0;isspace(s[i]);i++) //跳过空白符;
    sign=(s[i]=='-')?-1:1;
if(s[i]=='+'||s[i]=='-') //跳过符号 i++;
for(n=0;isdigit(s[i]);i++)

n=10*n+(s[i]-'0');    //将数字字符转换成整形数字
return sign *n;
}

```

二、itoa (把一整数转换为字符串)

例程序:

```
#include <ctype.h>
```

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

```
void itoa (int n,char s[]);//atoi 函数: 将 s 转换为整形数
```

```
int main(void )
```

```
{   int n; char s[100];
```

```
printf("Input n:\n"); scanf("%d",&n);
```

```
    printf("thestring : \n");
```

```
    itoa (n,s); return 0;
```

```
}
```

```
void itoa (int n,char s[])
```

```
{   int i,j,sign;
```

```
if((sign=n)<0)    //记录符号
```

```
n=-n;           //使n成为正数
```

```
i=0;
```

```
do{
```

```
s[i++]=n%10+'0'; //取下一个数字
```

```
}while ((n/=10)>0); //删除该数字
```

```
if(sign<0)
```

```
    s[i++]='-';
```

```
s[i]='\0';
```

```
for(j=i;j>=0;j--)//生成的数字是逆序的，所以要逆序输出
```

```
printf("%c",s[j]);
```

```
}
```

注意，atoi是标准库函数，itoa不是，用到itoa的时候可以用sprintf()a函数代替。

三、atof（将字符串转换成浮点型数）

相关函数 atoi, atol, strtod, strtol, strtoul

表头文件 #include <stdlib.h>

定义函数 double atof(const char *nptr);

函数说明 atof()会扫描参数nptr字符串，跳过前面的空格字符，直到遇上数字或正负符号才开始做转换，而再遇到非数字或字符串结束时('\0')才结束转换，并将结果返回。参数nptr字符串可包含正负号、小数点或E(e)来表示指数部分，如123.456或123e-2。返回值 返回转换后的浮点型数。

结构体，共用体和typedef

2021年9月11日 20:27

本笔记由Amos_Phoenix创建

QQ:1297289742

如有问题，欢迎加QQ进行指正，本人感激不尽。

了解结构体



结构体的使用初级

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

struct customer //创建声明时本身不分配内存，只有在调用时才分配内存
//此声明必须放在main函数的上方，不能像函数一样放在下方
{
    char name[6];
    char phone[50];
    int num; //结构体变量在声明的时候不能赋值
}; //分号不能省略

int main(void)
{
    struct customer cus1; //创建名为cus1的struct customer型结构体
    cus1.num = 1;
    sprintf(cus1.phone, "1513713"); //字符串赋值不能用等号，这是第一种赋值方法，在stdio.h
    strcpy(cus1.name, "Amos"); //这是第二种赋值方法，在string.h
    printf("%d, %s, %s", cus1.num, cus1.name, cus1.phone); //结构体变量不能整体引用，只能引用成员
    getchar();
    return 0;
}
```

```
E:\学习笔记\C学习\C语言自学代码\无名结构体\Debug\无名结构体.exe
1, Amos
2, Leo_
```

无名结构体

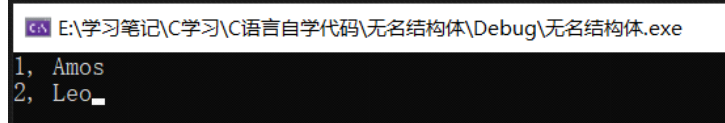
```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
```



```

#include<string.h>
struct //无名结构体，不能在main函数里定义，只能在声明中定义，是限量版
{
    char str[10];
    int num;
}M1,M2; //定义两个无名结构体变量
int main(void)
{
    strcpy(M1.str, "Amos");
    strcpy(M2.str, "Leo");
    M1.num = 1;
    M2.num = 2;
    printf("%d, %s\n%d, %s", M1.num, M1.str, M2.num, M2.str);
    getchar();
    return 0;
}

```



```

E:\学习笔记\C学习\C语言自学代码\无名结构体\Debug\无名结构体.exe
1, Amos
2, Leo_

```

结构体的嵌套

```

4 struct MyStruct
5 {
6     char str[200];
7     int num;
8 };
9
10 struct YouStruct
11 {
12     int data;
13     char you[200];
14     struct MyStruct my1; //结构体的嵌套
15 };
16
17 void main()
18 {
19     struct YouStruct you;
20     you.data = 100;
21     sprintf(you.you, "you are powerful");
22     you.my1.num = 99;
23     sprintf(you.my1.str, "ABCDE"); //嵌套结构体用多个. 访问
24     printf("data=%d,you=%s,my1.num=%d,my1.str=%s", you.data, you.you, you.my1.num, you.my1.str);
25
26     I
27     getchar();

```



```

data=100,you=you are powerful,my1.num=99,my1.str=ABCDE_

```

结构体整体赋值

```

void main()
{
    struct ours o1 = { 100, "hello china" };
    struct ours o2 = o1; //结构体直接赋值的时候，整体，即使字符串也可赋值
    printf("%d,%s", o2.num, o2.str);
}

```

//结构体只有在定义的时候才能赋值，否则只能一个一个挨个赋值
 //结构体在类型一致的情况下可以用等号给整个新结构体赋值

在结构体中声明另一个结构体

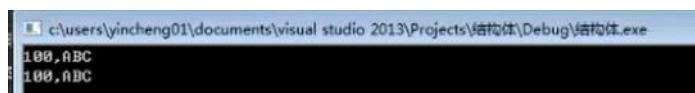
```
5 struct tianchao
6 {
7     int data;
8     char name[100];
9     struct beijing //结构体内部再次定义一个结构体，但是没有创建结构体的实例
10                  //再次定义的结构体内部的变量，会被当作母结构体的成员变量
11     {
12
13         char str[100];
14         int num;
15     };
16 };
17
```

//跟没有struct beijing一样

要想操作结构体内部声明的结构体，就要在母结构体中定义内部声明的子结构体变量

```
33
34 struct tianchaoA
35 {
36     int data;
37     char name[100];
38     struct beijingA //结构体内部再次定义一个结构体，创建结构体变量，该变量会直接作为一个成员
39     {
40
41         char str[100];
42         int num;
43     } b1; //内部定义的第一种方式
44     struct beijingA b2; //内部定义的第二种方式
45 };
46
47
```

```
1 void main()
2 {
3     struct tianchaoA t2;
4     t2.b1.num = 100;
5     sprintf(t2.b1.str, "ABC");
6     printf("%d,%s", t2.b1.num, t2.b1.str);
7
8
9     t2.b2.num = 100;
10    sprintf(t2.b2.str, "ABC");
11    printf("\n%d,%s", t2.b2.num, t2.b2.str);
12    getchar();
13
14 }
```



综上，结构体嵌套，用多个点(.)来访问

结构体数组

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

```
struct customer
```

```
{
```

```
char name[10];
```

```
int money;
```

```
}cus1[2] = { {"qqq",11}, {"www",22}}; //结构体数组的第一种创建和赋值方法
```

```
int main(void)
```

```
{
```

```
struct customer cus2[2] = { {"eee",33}, {"rrr",44}}; //结构体数组的第二种创建和赋值方法
```

```

printf("%d\n\n", sizeof(struct customer));
for (int i = 0; i < 2; i++)
{
printf("%s, %d\n", cus1[i].name, cus1[i].money);
printf("%p\n", &cus1[i]); //可以看出结构体数组也是按顺序挨个排列的
}
for (int i = 0; i < 2; i++)
{
printf("\n%s, %d\n", cus2[i].name, cus2[i].money);
printf("%p", &cus2[i]);
}
getchar();
return 0;
} //当然, 如果你想先声明再一个一个赋值也可以


```

结构体指针

```

#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
struct customer
{
char name[5];
int money;
};
int main(void)
{
struct customer cus; //创建结构体变量cus
sprintf(cus.name, "Amos");
cus.money = 55;
printf("%s, %d\n", cus.name, cus.money);
struct customer* p = &cus; //让p存放cus的首地址
printf("%d\n", sizeof(p)); //从中可以知道p作为指针依旧是占4字节
sprintf((*p).name, "Leo"); //指针修改结构体变量的第一种方式 (繁琐, 不推荐)
(*p).money = 66;
printf("%s, %d\n", (*p).name, (*p).money);
sprintf(p->name, "Marina"); //指针修改结构体变量的第二种方式 (简便, 推荐)
p->money = 77;
printf("%s, %d\n", p->name, p->money);
getchar();
return 0;
}

```

 E:\学习笔记\C学习\C语言自学代码\结构体指针\Debug\结构体指针.exe

```

Amos, 55
4
Leo, 66
Marina, 77

```

指向结构体数组的指针

```

#define _CRT_SECURE_NO_WARNINGS

```

```

#include<stdio.h>
struct customer
{
    char name[10];
    int money;
};
int main(void)
{
    struct customer cus[2];
    int i = 1;
    for (struct customer* p = cus; p < cus + 2; p++) //设置struct customer*类型的指针，存放cus首地址
    {
        sprintf(p->name, "Amos");
        p->money = 100;
        printf("%d, %s, %d\n", i, p->name, p->money); //打印存储的内容
        i++;
    }
    getchar();
    return 0;
}

```

E:\学习笔记\C学习\C语言自学代码\指向结构体数组的指针\Debug\指向结构体数组的指针.exe

```

1, Amos, 100
2, Amos, 100

```

用指向结构体的指针作为函数参数

- 用结构体变量的成员作参数----值传递
- 用指向结构体变量或数组的指针作参数----地址传递
- 用结构体变量作参数----多值传递，效率低

1. 用指向结构体的指针作为函数参数

```

#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<string.h>
void change(struct customer*);
struct customer
{
    char name[6];
    int money;
};
int main(void)
{
    struct customer cus1;
    cus1.money = 99;
    strcpy(cus1.name, "Amos");
    printf("%d, %s\n", cus1.money, cus1.name);
    change(&cus1); //注意，这不是数组，cus1是名称而不是指针变量，必须取地址才能接收
    printf("%d, %s\n", cus1.money, cus1.name);
    getchar();
}

```

```

return 0;
}
void change(struct customer* p) //通过地址改变cus1变量
{
printf("%d\n", sizeof(p)); //打印指针p的大小
p->money = 100;
sprintf(p->name, "Leo");
return;
}

```

CS E:\学习笔记\C学习\C语言自学代码\用指向结构体的指针作为函数参数\Debug\用指向结构体的指针作为函数参数.exe

```

99, Amos
4
100, Leo

```

2. 用指向结构体数组的指针作为函数参数

```

#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<string.h>
void change(struct customer cus[2]);
struct customer
{
char name[6];
int money;
};
int main(void)
{
struct customer cus[2];
change(cus); //注意，这里的cus是指针，可以直接传递
for (int i = 0; i < 2; i++)
{
printf("%s, %d\n", cus[i].name, cus[i].money);
}
getchar();
return 0;
}
void change(struct customer cus[2]) //对数组来说，参数传递的是地址
{
printf("%d\n", sizeof(cus)); //可以看到虽然是结构体数组指针，但是每一个依然只有4字节
cus[0].money = 88;
cus[1].money = 99;
sprintf(cus[0].name, "Amos");
sprintf(cus[1].name, "Leo"); //等价于(*(cus+1)).name或者(cus+1)->name (因为'.'的优先级最大，所以加括号)，是指针操作
return;
}

```

CS E:\学习笔记\C学习\C语言自学代码\用指向结构体的指针作为函数参数\Debug\用指向结构体的指针作为函数参数.exe

```

4
Amos, 88
Leo, 99

```

动态分配结构体

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
struct customer
{
    char name[6];
    int money;
};
int main(void)
{
    struct customer* p = (struct customer*)malloc(sizeof(struct customer));
    p->money = 99;
    sprintf(p->name, "Amos");
    printf("%s, %d", p->name, p->money);
    getchar();
    return 0;
}
```

CS E:\学习笔记\C学习\C语言自学代码\动态分配结构体\Debug\动态分配结构体.exe

Amos, 99

结构体字节对齐

- 出于效率的考虑，C语言引入了字节对齐机制，一般来说，不同的编译器字节对齐机制有所不同，但还是有以下3条通用准则：
- （1）结构体变量的大小能够被其最宽基本类型成员的大小所整除；
- （2）结构体每个成员相对于结构体首地址的偏移量（offset）都是成员大小的整数倍，如有需要编译器会在成员之间加上填充字节（internal adding）；
- （3）结构体的总大小为结构体最宽基本类型成员大小的整数倍，如有需要编译器会在最末一个成员之后加上填充字节（trailing padding）。
- 字节对齐第3条准则提及最宽基本类型的概念，所谓基本类型是指像char、short、int、float、double这样的内置数据类型。“数据宽度”就是指其sizeof的大小。诸如结构体、共用体和数组等都不是基本数据类型

注：第二行的成员大小指最宽基本类型成员大小

基本类型不包含数组

具体如下：

```
#include <stdio.h>
struct info1
{
    //最大基本类型是int
    char name[7]; //8
    int money; //4
    short judge; //4
};
struct info2
{
    //最大基本类型是double
    double d; //8
}
```

```

char ch; //1
char c[15]; //15
}; //当两个相同类型再相邻位置且长度加起来是最大基本类型长度整数倍时不再扩增长度
struct info3
{
    double db; //8
    int i; //4
    int j; //4
};
int main()
{
    printf("%d\n", sizeof(struct info1));
    printf("%d\n", sizeof(struct info2));
    printf("%d\n", sizeof(struct info3));
    getchar();
    return 0;
}

```

e:\learning notes\C_learning\vscode C language\byte alignment of struct\byte alignment.exe

```

16
24
16

```

typedef与结构体

```

#include <stdio.h>
#include <stdlib.h>
struct info1
{
    char ch[10];
    int i;
};
typedef struct info1 si1; //起别名的第一种方法
typedef struct info2
{
    char str[10];
    int j;
} si2, *psi2; //起别名的第二种方法，重定义了结构体及其指针的名字
int main()
{
    si2 s; //两种方法声明并赋值
    sprintf(s.str, "Amos");
    s.j = 99;
    psi2 ps;
    ps = (psi2)malloc(sizeof(si2));
    ps->j = 66;
    sprintf(ps->str, "phoenix");
    printf("%s, %d\n", s.str, s.j);
    printf("%s, %d\n", ps->str, ps->j);
}

```

```

    getchar();
    return 0;
}

```

e:\learning notes\C_learning\vscode C language\use typedef to give struct a nickname\typedef.exe

```

Amos, 99
phoenix, 66

```

共用体

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

union costume //共用体大小由最大变量大小决定和最小变量大小共同决定，会相互覆盖

```

{
    char name[10];
    int age;
    int money;
} data1, data2, *datap, datan[10]; //第二种定义方式，绝对不能在定义时就初始化

```

union costume dataA, datapA, datanA[10]; //第三种定义方式

```

union
{
    char name[10];
    int age;
    int money;
} data3, data4, *datapx, datanx[10]; //匿名共用体只能用第二种定义方式，限定数量

```

```

int main()
{
    union costume info; //共用体第一种定义方式
    printf("%d\n", sizeof(union costume));
    info.money = 10;
    sprintf(info.name, "Amos"); //定义Amos时，info.mooney被覆盖
    printf("%d, %s", info.money, info.name);
    getchar();
    return 0;
}

```

e:\learning notes\C_learning\vscode C language\defintion of union\defintion.exe

```

12
1936682305, Amos

```


这里是12而不是10的原因是共用体大小应该能被最小类型整除，所以发生填充现象（字节对齐）

枚举类型

- C语言提供了关键字Enum定义枚举类型，基本格式为：
- `enum` 枚举类型名 {枚举常量1[=整型常数], 枚举常量2[=整型常数],};
- 作为一条完整的C语句，不要忘记结尾的分号。
- 枚举类型的定义包括以下要素：
- 枚举类型名，有效的C语言标识符。
- 枚举表，即“{枚举常量1[=整型常数], 枚举常量2[=整型常数],}”部分，枚举表是枚举常量的集合，枚举表中每项后的“=整型常数”是给枚举常量赋初值，方括号代表赋初值的操作可以省略。
- 如果不给枚举常量赋初值，编译器会为每一个枚举常量赋一个不同的整型值，第一个为0，第二个为1，等等。当枚举表中某个常量赋值后，其后的成员则按依次加1的规则确定其值。
- 来看一个简单的例子：
- `enum day {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};`

```
//枚举类型的一般形式，限定作用范围在这个范围内
//如果没有一个赋初始值，就会从0循环到最后一个，每次加1
//如果第一个赋初值，后面就按照每次加1的规则，确保每个枚举常量都不同
//除非自己赋值，否则计算机赋值会让每个枚举常量都不同
```

```
#include <stdio.h>
enum level
{
    T1,
    T2,
    T3,
    T4,
    T5,
    T6 //不用加分号
};
enum status
{
    S1 = 10000,
    S2 = 1000,
    S3 = 100,
    S4 = 10,
    S5 = 1
};
int main()
{
    printf("%d,%d,%d,%d,%d,%d\n", T1, T2, T3, T4, T5, T6); //系统自动从0开始一次赋初值
    printf("%d,%d,%d,%d,%d\n", S1, S2, S3, S4, S5); //也可以直接在枚举类型中直接赋初值
    printf("%d", sizeof(enum status)); //枚举默认为一个整型 (int)，占4字节
    getchar();
    return 0;
}
```

 e:\learning notes\C_learning\vscode C language\operation of enum\enum.exe

```
0, 1, 2, 3, 4, 5
10000, 1000, 100, 10, 1
4
```

数组和枚举型变量

```
#include <stdio.h>
enum week
{
    monday,tuesday,wednesday,thursday,friday
};
int main()
{
    int a[5]={45,46,43,47,49};
    for(int i=monday;i<=friday;i++) //用整数来映射特定标识符
    {
        printf("周%d有%d人来上课\n",i+1,a[i]);
    }
    getchar();
    return 0;
}
```


❏ e:\learning notes\C_learning\vscode C language\arrays and enumerated constants\arrays and enumerated constants.exe

```
周1有45人来上课
周2有46人来上课
周3有43人来上课
周4有47人来上课
周5有49人来上课
```

结构体与共用体的相互嵌套

```
#include <stdio.h>
struct tele
{
    char str[16];
};
union type
{
    char BrandType[20];
    char AssembledType[20];
};
struct costumer //想要嵌套成功，里面的声明必须在主嵌套声明的前面
{
    char name[10];
    int money;
    struct tele tele;
    union type type;
};
int main()
{
    struct costumer cos1;
    cos1.money = 8848;
    sprintf(cos1.name, "Amos");
    sprintf(cos1.tele.str, "1861111");
}
```

```
    sprintf(cos1.type.BrandType, "dell");  
    printf("%s,%s,%d,%s", cos1.name, cos1.tele.str, cos1.money, cos1.type.BrandType);  
    getchar();  
    return 0;  
}
```

 e:\learning notes\C_learning\vscode C language\nesting between steuct and union\nesting.exe

Amos, 1861111, 8848, dell

位运算

2021年9月19日 10:26

本笔记由Amos_Phoenix创建

QQ:1297289742

如有问题，欢迎加QQ进行指正，本人感激不尽。

位运算符

1. 位取反

位取反的操作符为“~”，如果A为10101010，那么~A返回的结果为01010101，即每位都取反，0变成1，1变成0，需要注意的是，位取反运算并不改变操作数的值。

```
#include <stdio.h>

int main()
{
    unsigned char ch = 15; //0000 1111
    unsigned char rech = ~ch; //1111 0000, 取反运算符不改变操作数的值
    printf("%d, %d", ch, rech);
    getchar();
    return 0;
}
```

e:\learning notes\C_learning\vscode C language\bit operation\bit reverse.exe

15, 240

2. 按位与

位与运算的操作符为&，将对两个操作数的每一位进行与运算，位“与”运算的准则如下：

1 & 1 = 1 1 & 0 = 0 0 & 1 = 0 0 & 0 = 0

```
#include <stdio.h>

int main()
{
    unsigned char ch1 = 15; //0000 1111
    unsigned char ch2 = 240; //1111 0000
    unsigned char ch3 = 255; //1111 1111
    printf("%d\n", ch1 & ch2); //0000 0000(0) 全部清零
    printf("%d\n", ch1 & ch3); //0000 1111(15) 清零高位
    printf("%d\n", ch2 & ch3); //1111 0000(240) 清零低位
    getchar();
    return 0;
}
```

e:\learning notes\C_learning\vscode C language\bit operation\bit reverse.exe

15, 240

所以，按位与可以实现按位清零操作

按位与还能实现取出某一指定位

例如：

```
#include <stdio.h>

int main()
{
    //目标：取出中间四位。
    unsigned char ch1 = 169; //1010 1001
    unsigned char ch2 = 60; //0011 1100
    printf("%d\n", ch1 & ch2); //0010 1000(40)
    getchar();
    return 0;
}
```

e:\learning notes\C_learning\vscode C language\bit operation\extract.exe

40

综上，与0进行按位与，执行清零动作；与1进行按位与，执行保留动作

按位或

位或运算的操作符为|，将对两个操作数的每一位进行或运算，位“或”运算的准则如下：

1 | 1 = 1 1 | 0 = 1 0 | 1 = 1 0 | 0 = 0

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

```
int main()
```

```
{
```

```
    unsigned char ch1 = 169; //1010 1001
```

```
    unsigned char ch2 = 240; //1111 0000
```

```
    printf("%d", ch1 | ch2); //1111 1001(249)
```

```
    getchar();
```

```
    return 0;
```

```
}
```

e:\learning notes\C_learning\vscode C language\bit operation\OR.exe

249

所以，按位或运算符有与0或位不变，与1或位变1的性质

按位异或

位或运算的操作符为^，将对两个操作数的每一位进行异或运算。通俗地讲，如果位“异或”运算的两个位相同（同为0或同为1），结果为0，若两个位不同（一个为0，另一个为1），结果为1，对应的准则为：

1 ^ 1 = 0 1 ^ 0 = 1 0 ^ 1 = 1 0 ^ 0 = 0

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

```
int main()
```

```
{
```

```
    unsigned char ch1 = 169; //1010 1001
```

```
    unsigned char ch2 = 0;    //0000 0000
```

```
    unsigned char ch3 = 255; //1111 1111
```

```
    printf("%d\n", ch1 ^ ch2); //1010 1001(169)
```

```
    printf("%d\n", ch1 ^ ch3); //0101 0110(86)
```

```
    getchar();
```

```
    return 0;
```

```
}
```

e:\learning notes\C_learning\vscode C language\bit operation\XOR.exe

169

86

可以看出，与0异或，位值不变，与1异或，位值取反

按位异或能实现不引入中间变量的交换

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

```
int main()
```

```
{
```

```
    unsigned char ch1 = 10;
```

```
    unsigned char ch2 = 20;
```

```
    ch1 = ch1 ^ ch2;
```

```
    ch2 = ch1 ^ ch2;
```

```
    ch1 = ch1 ^ ch2;
```

```
    printf("%d, %d", ch1, ch2);
```

```
    getchar();
```

```
    return 0;
```

```
}
```

```
    //用于面试和嵌入式开发需要节约内存的时候
```

e:\learning notes\C_learning\vscode C language\bit operation\exchange without intermediate variable.exe

```
20, 10_
```

取反运算的高级用法

```
#include <stdio.h>
int main()
{
    unsigned char ch1 = 169; //1010 1001
    //我们想抹去最后一位的1
    ch1 = ch1 & ~1; //相当于1010 1001和1111 1110进行与运算，抹掉了最后一位
    printf("%d\n", ch1);
    getchar();
    return 0;
}
```

e:\learning notes\C_learning\vscode C language\bit operation\advanced usage of bit reverse.exe

```
168
```

关于printf的注意

```
#include <stdio.h>
int main()
{
    printf("%d, ", 10.3);
    printf("%d\n", (int)10.3);
    printf("%f, ", 10);
    printf("%f\n", (float)10);
    //我们可以看出，printf不会进行数据类型转换，必须手动进行数据类型转换
    getchar();
    return 0;
}
```

e:\learning notes\C_learning\vscode C language\bit operation and complement\things about printf.exe

```
-1717986918, 10
10.299995, 10.000000
```

补码

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

int main()
{
    //int类型的补码第一位表示符号，是符号位，1表示负，0表示正
    printf("%d, %d\n", INT_MAX, INT_MIN); //max=7FFF FFFF,min=FFFF FFFF因为带符号，所以只能显示unsigned类型的一半
    printf("%u, %u\n", UINT_MAX, 0); //max=FFFF FFFF因为不用考虑符号问题，所以可以全部用来显示正整数
    getchar();
    return 0;
}
```

E:\learning notes\C_learning\vscode C language\complement principle\Debug\complement principle.exe

```
2147483647, -2147483648
4294967295, 0
```

原码，反码与补码

计算机中只有补码，一切按照补码计算

原码，反码，补码是机器存储一个具体数字的编码方式。原码跟补码之间的关系是：正数的补码与原码相同，负数的补码为其原码除符号位外所有位取反（得到反码了），然后最低位加1

7FFF FFFF是最大正数2147483647，+1之后是8000 000是最大负数-2147483648

移位操作

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

```

int main()
{
    //移位运算是直接操纵二进制位的运算，分为左移"<<"和右移">>"
    //左移几就乘了该进制下的几次方，右移几就除了该进制下的几次方
    //1.左移
    //左移时不要忘记考虑到数字过大的溢出问题
    unsigned char chl=1; //0000 0001
    printf("%-2d, %-2d\n",chl,chl<<1); //0000 0010
    //2.右移
    unsigned char chs=32; //0010 0000
    printf("%2d, %2d\n",chs,chs>>1); //0001 0000
    getchar();
    return 0;
}

```

e:\learning notes\C_learning\vscode C language\shift operation\shiftoperation.exe

```

1, 2
32, 16

```

混合移位操作

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

```

int main()
{
    //ch <<= 1等价于ch = ch << 1
    unsigned char ch1 = 1;
    printf("%-3d\n", ch1 <<= 1);
    //ch >>= 1等价于ch = ch >> 1
    unsigned char ch2 = 4;
    printf("%-3d\n", ch2 >>= 1);
    //ch |= 1等价于ch = ch | 1
    unsigned char ch3 = 4; //0000 0100 | 0000 0001
    printf("%-3d\n", ch3 |= 1); //0000 0101
    //ch ^= 1等价于ch = ch ^ 1
    unsigned char ch4 = 4; //0000 0100 ^ 0000 0001
    printf("%-3d\n", ch4 ^= 1); //0000 0101
    //ch = ~ch,没有简便取法
    unsigned char ch5 = 4; //0000 0100
    printf("%-3d\n", ch5 = ~ch5); //1111 1011
    getchar();
    return 0;
}

```

e:\learning notes\C_learning\vscode C language\compound shift operation\compound shift operation.exe

```

0
3
5
251

```


综合实战

2021年12月19日 15:33

本笔记由Amos_Phoenix创建

QQ:1297289742

如有问题，欢迎加QQ进行指正，本人感激不尽。

一、学生管理系统（典中典好吧）

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

```
#include <stdlib.h>
```

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

```
typedef struct student
```

```
{
```

```
    char name[10];
```

```
    char sex[10];
```

```
    int age;
```

```
    int score;
```

```
    int visual;
```

```
}STUDENT,*PSTUDENT;
```

```
void welcome (void);
```

```
void input (PSTUDENT,int);
```

```
void sort (STUDENT *,int );
```

```
void output (STUDENT *,int );
```

```
void help (int*);
```

```
void revise (PSTUDENT,int);
```

```
void detail (int*);
```

```
void see (PSTUDENT);
```

```
void delet (PSTUDENT,int );
```

```
void judge (PSTUDENT,int);
```

```
int main()
```

```
{
```

```
    int a = 0;
```

```
    int b = 0;
```

```
    int c = 0;
```

```
    int d = 0;
```

```
    int e = 0;
```

```
    int f = 0;
```

```
    int g = 0;
```

```
    int h = 0;
```

```
    int i = 0;
```

```
    PSTUDENT pstu;
```

```
    //welcome
```

```
    welcome();
```

```

//加入并创建考生
printf ("请输入录入考生的人数: ");
scanf ("%d",&a);
getchar ();
pstu = (PSTUDENT)malloc(a * sizeof(STUDENT));
//对考生信息进行录入
input (pstu,a);
//对考生成绩按降序排列
sort (pstu,a);
//输出考生信息
output (pstu,a);
//询问
for (d=0;d<10000;d+=1)
{
    help (&c);
    if (c == 1)
    {
        judge (pstu,a);
        sort (pstu,a);
        printf ("数据系统已更新，以下是最新内容：\n");
        output (pstu,a);
    }else if (c==3)
    {
        break;
    }else if (c == 2)
    {
        delet (pstu,a);
        sort (pstu,a);
        output (pstu,a);
    }else
    {
        printf ("error,请重新输入");
        getchar ();
    }

}

//再次输出更改过的数据
sort (pstu,a);
output (pstu,a);
//询问是否要看具体学生数据
for (;g<10000;++g)
{
    detail (&f);
    if (f == 1)

```

```

{
    see (pstu);
}else if (f == 2)
{
    printf ("goodbye!");
    getchar();
    break;
}else
{
    printf ("error,请重新输入");
}

}

//释放动态内存
free (pstu);
getchar();
return 0;
}

void welcome (void)
{
    printf ("欢迎进入考生成绩管理系统! \n");
    printf ("按回车键继续\n");
    getchar();
}

void input (PSTUDENT pst,int j)
{
    int i = 0;
    int t = j;
    for (;i<j;i+=1)
    {
        printf("请输入第%d个考生的姓名: ",i+1);
        scanf ("%s",pst[i].name);
        getchar ();
        printf("请输入第%d个考生的性别: ",i+1);
        scanf ("%s",pst[i].sex);
        getchar ();
        printf("请输入第%d个考生的年龄: ",i+1);
        scanf ("%d",&pst[i].age);
        getchar ();
        printf("请输入第%d个考生的成绩: ",i+1);
        scanf ("%d",&pst[i].score);
        getchar ();
        pst[i].visual=1;
        printf ("\n");
    }
}

```

```

}
void sort (STUDENT *psr,int m)
{
    int n = 0;
    int q = 0;
    STUDENT t;
    for (;n<m-1;n+=1)
    {
        if (psr[n].visual==1)
        {
            for (;q<m-1-n;++q)
            {
                if (psr[q].score > psr[q+1].score)
                {
                    t = psr[q];
                    psr[q] = psr[q+1];
                    psr[q+1] = t;
                }
            }
        }
    }
}

void output (STUDENT *psq ,int k)
{
    int i = k-1;
    int j = 1;
    for (;i>=0;--i)
    {
        if (psq[i].visual==1)
        {
            printf ("%s总分%d, 排%d名\n",psq[i].name,psq[i].score,j);
            j+=1;
        }
    }
}

void help (int* i)
{
    printf ("你还有什么需要帮助的吗? \n");
    printf ("输入1进行修改, 输入2进行删除, 输入3退出\n");
    scanf ("%d",&i);
    getchar ();
}

void revise (PSTUDENT pst,int i)
{

```

```

int t = 0;
printf ("此人哪里有问题? \n");
printf ("输入1改名字, 输入2改性别, 输入3改年龄, 输入4改成绩: ");
scanf ("%d",&t);
getchar ();
switch (t)
{
    case 1:
    {
        printf ("请输入要改的名字: ");
        scanf ("%s",pst[i].name);
        getchar ();
        break;
    }
    case 2:
    {
        printf ("请输入要改的性别: ");
        scanf ("%s",pst[i].sex);
        getchar ();
        break;
    }
    case 3:
    {
        printf ("请输入要改的年龄: ");
        scanf ("%d",&pst[i].age);
        getchar ();
        break;
    }
    case 4:
    {
        printf ("请输入要改的成绩: ");
        scanf ("%d",&pst[i].score);
        getchar ();
        break;
    }
    default: printf ("error,请重新输入");
}
printf ("已完成更改! \n");
}

void detail (int* p)
{
    printf ("是否查看某个人的详细信息? (是输入1, 不需要并退出输入2)");
    scanf ("%d",&*p);
    getchar ();
}

void see (PSTUDENT pst)

```

```

{
    int i = 0;
    printf ("输入你想看排第几的考生的详细信息： ");
    scanf ("%d",&i);
    getchar ();
    i = i-1;
    printf ("该考生姓名为： %s\n",pst[i].name);
    printf ("该考生性别为： %s\n",pst[i].sex);
    printf ("该考生年龄为： %d\n",pst[i].age);
    printf ("该考生成绩为： %d\n",pst[i].score);
}
void judge (PSTUDENT pstu,int a)
{
    int e = 0;
    int i = 0;
    int k = 0;
    for (i=0;i<10000;+ +i)
        {
            w :
                printf ("第几名出了问题呢： ");
                scanf ("%d",&e);
                getchar ();
                revise (pstu,a-e);
                printf ("处理好了吗（输入1好了， 输入2没有）： ");
                scanf ("%d",&k);
                getchar();
                if (k == 1)
                {
                    break;
                }else if (k == 2)
                {
                    goto w;
                }
                else
                {
                    printf ("error,请重新输入！ ");
                    getchar ();
                }
                getchar ();
            }
        }
}
void delet (PSTUDENT pst,int a)
{
    int i = 0;
    printf ("请输入想删除的考生的排名： ");

```

```

scanf ("%d",&i);
getchar ();
pst[a-i].visual=0;
printf ("已完成，按回车显示新版数据\n");
}

```

二、整蛊软件（谨慎打开，但也不会怎样）

```

#include<stdio.h>
#include<windows.h>
int main(void)
{
    char c[100];
    char ch[100];
    char ch2[100];
    char ch1[100] = { "我是伞兵" };
    system("title surprise,mother fucker");
    MessageBoxA(0, "你的电脑被我绑架了，输入 '我是伞兵' 即可解除", "system warning SOS", 0);
    scanf("%s", &ch);
    getchar();
    if (*ch == *ch1)
    {
        MessageBoxA(0, "你的电脑控制权已经交还给你", "哈哈，你个伞兵", 0);
    }
    else
    {
        MessageBoxA(0, "我生气了，这是你最后的机会，输入我是伞兵，不然让你感受我的愤怒", "system warning SOS", 0);
        scanf("%s", &ch2);
        getchar();
        if (*ch2 == *ch1)
        {
            MessageBoxA(0, "你的电脑控制权已经交还给你", "哈哈，你个伞兵", 0);
        }
        else
        {
            MessageBoxA(0, "毁灭吧，我累了，放弃抵抗吧，伞兵", "system warning SOS", 0);
            system("shutdown -s -t 6");
            printf("warning,十秒自动关机已启动，感受痛苦吧！");
            while (1)
            {
                for (int i = 0; i < 16; i++)
                {
                    sprintf(c, "color %x%x", i, 16 - i);
                    system(c);
                    Sleep(50);
                }
            }
        }
    }
}

```



```

    }
}

getchar();
return 0;
}

```

三、懒人QQ自动登录

```

#include<stdio.h>
#include<stdlib.h>
#include<windows.h>
void ope(void);
void input1(void);
void click(void);
int main(void)
{
    ope();
    input1();
    click();
    return 0;
}
void ope(void)
{
    system("这里放QQ打开的地址");
    return;
}
void input1(void)
{
    Sleep(4600);
    keybd_event('密码，一次一个字符', 0, 0, 0);
    keybd_event('重复', 0, 2, 0);
    Sleep(50);
    keybd_event('密码，一次一个字符', 0, 0, 0);
    keybd_event('重复', 0, 2, 0);
    Sleep(50);
    //密码位数多可复制粘贴，dddd
    return;
}
void click(void)
{
    Sleep(60);
    keybd_event(0x0D, 0, 0, 0);
    keybd_event(0x0D, 0, 2, 0);
}

```

