

计算机程序设计基础（1）

15 位运算

清华大学电子工程系

杨昉

E-mail: fangyang@tsinghua.edu.cn



15、位运算

□ 二进制位运算

□ 位段

二进制位运算



● 位运算指对二进制位进行的运算

□ 每个二进制位中只能存放0或1

□ 通常，将一个数据用二进制数表示后，最右边的二进制位称为最低位（第0位），最左边的二进制位为最高位

位运算符	意义
&	按位与
	按位或
^	按位异或
~	按位取反
<<	左移
>>	右移

- 在这六种位运算符中，其中的按位取反是单目运算符，只有一个运算对象，其他均为双目运算符，有两个运算对象
- 位运算的运算对象只能是整型（包括 short, int, long, long long, unsigned 各种）或 char型 数据，但不能是实型数据

二进制位运算



- 例15-1：已知采用补码方式-13的二进制表示为**11110011**，21的二进制表示为**00010101**，这两个数进行六种位运算结果

□按位与： 11110011 亦即 $-13 \& 21 = 17$

(&) 00010101

00010001

□按位或： 11110011 亦即 $-13 | 21 = -9$

(|) 00010101

11110111

二进制位运算



□**按位异或**：异或运算规则为两位相同则返回0，两位不同则返回1

即有： $0 \wedge 0 = 0$, $1 \wedge 1 = 0$, $0 \wedge 1 = 1$, $1 \wedge 0 = 1$

$$\begin{array}{r} 11110011 \\ (^) 00010101 \\ \hline 11100110 \end{array} \quad \text{亦即 } -13 \wedge 21 = -26$$

□**按位取反**： $\sim 11110011 = 00001100$ 亦即 $\sim(-13) = 12$

$\sim 00010101 = 11101010$ 亦即 $\sim 21 = -22$

二进制位运算



□**左移运算符**：将运算对象中的每个二进制位向左移动若干位，从左边移出去的高位部分被**丢弃**，右边空出的低位部分**补0**

- $00010101 \ll 2 = 01010100$ ，将21左移两位，即 $21 \ll 2 = 84$

□**右移运算符**：将运算对象中的每个二进制位向右移动若干位，从右边移出去的低位部分被**丢弃**。但左边空出的高位部分视**具体情况而定**

- 若右移对象为无符号整型数，则右移后左边空出的高位部分补0
- 若右移对象为一般整型数或字符型数据，当该数据的最高位为0，则右移后左边空出的高位部分补0
- 当该数据的最高位为1，补0还是补1与编译器有关，补1的称为**算术右移**，补0的称为**逻辑右移**

二进制位运算



- 位运算和之前介绍的所有运算符优先级顺序如下

- 由高到低：**!** --> **~** --> **算术运算符** --> **<<、>>** --> **关系运算符**
--> **&、^、|** --> **&&** --> **||** --> **赋值运算符**

- C语言直接对比特位进行操作存在困难，但可以通过位操作实现以下的功能

- 取出指定位：取出short型变量x的低8位： **$x \& 0x00FF$**

- 将指定位置零：将short型变量x的低8位清零： **$x \& 0xFF00$**

- 思考：指定位置置1？



●位段的定义

- 程序设计中常常要用到一些标志信息，往往只需几个比特位即可，但如果用整型数来表示则会浪费存储空间
- 位段操作可以解决这个问题，位段结构类型的一般形式为

struct 位段结构类型名
{成员表};

- 例15-2：右边定义了一个类型名为packed_d的位段

```
struct packed_d{  
    unsigned short f1:2;  
    unsigned short f2:1;  
    unsigned short f3:3;  
    unsigned short f4:2;  
    unsigned short f5:5;  
};
```




□定义了一个位段结构类型，名为packed_d，共包含5个成员（又称为**位段**），每个成员均为**无符号short类型**

- f1占2个二进制位；

- f2占1个二进制位；

- f3占3个二进制位；

- f4占2个二进制位；

- f5占5个二进制位

□至于这些位段在存储单元中的具体存放位置，是由**编译系统**来分配的，一般用户不必考虑

□在计算机系统中，各位段在存储单元中一般是**从右到左**（即从低位到高位）顺序分配的

位段结构



□ 上述位段结构类型需要占2个字节(即16个二进制位)，其存储结构

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
空				f5				f4		f3			f2	f1	

□ 定义了位段结构类型后，就可以定义位段结构类型的变量

- 可以将位段结构类型与该类型的变量分开定义
- 也可以在定义位段结构类型的同时定义该类型的变量
- 还可以直接定义无名的位段结构类型的变量

□ 位段结构成员的引用方式与引用一般结构体成员的方式相同

□ 对位段进行赋值时，要考虑到该位段所占用的二进制位数

- 如果所赋的数值超过了位段的表示范围，则自动取其低位数字

位段注意事项



- 位段成员的类型必须是unsigned型
- 在位段结构类型中，可以定义无名位段，这种无名位段具有位段之间的分隔（或占位）作用
 - 若无名位段的宽度值为0，则下一个位段从一个新的字节开始存放
- 每个位段（成员）所占的二进制位数一般不能超过一个机器字长（即编译器中一个字的长度，一般为32位）
- 位段不能说明为数组，也不能用指针指向位段成员
 - 否则编译会出现错误信息 **error C2104: 位域上的“&”被忽略**

位段注意事项



- 不能用 **sizeof** 求段位成员的大小
- 在位段结构类型定义中，可以包含 **非位段** 成员
 - 非位段成员可以定义在结构体中 **任何位置**，其引用方式与普通结构体成员的引用方式 **完全相同**
 - 非位段成员总是从下一个本类型所占长度 **整数倍** 的位置开始存放，当前数据类型剩余空间 **不再使用**
- 位段结构体类型变量中的位段成员可以在一般的表达式中被引用，并被 **自动转换** 为相应的 **整数**

```
struct packed_x{  
    int n;  
    unsigned int f1:2;  
    unsigned int f2:1;  
};
```


计算机程序设计基础（1）

16 期末复习

清华大学电子工程系

杨昉

E-mail: fangyang@tsinghua.edu.cn

考试安排



●考试时间、地点

□2024年01月13日(周六)晚19:00~20:30 六教6A203

(只有90分钟)

●考查内容

□教材上半部分所有内容，第13章位运算除外

●考试题型

□10道程序完成填空题，每题2个填空，每空1分；10道读程序写结果题目，每题2分；闭卷考试，答题时间90分钟

□平时成绩30分+期中机考10分+期末机考20分+期末笔试40分

学习方法——总结有助于提高



- 只学习不总结，就像无头苍蝇无法提高
- 在学习编程的过程中，要**不断复习**和**总结**之前的知识，并将其归纳和升华
- 总结有助于把握学习脉络，为未来学习提供指导

● **回顾历史、总结经验，从中得出规律性认识，进而映照现实、指导实践，远观未来、把握大势，是我们党的一个好传统**

C语言主要内容



- 本学期从以下几个方面出发来介绍C语言有关知识

知识点大类	主要内容
基础知识	数据类型、表达式、宏定义、编译预处理
数据读写	输入与输出、文件操作
程序设计	顺序结构、分支结构、循环结构、函数设计
数据结构	数组、指针、结构体与联合体

不仅C语言的学习是这样，在其他语言的学习过程中，也是从这几个方面出发，**由浅入深，逐步掌握**

程序设计=算法+数据结构+方法+工具



●程序设计的过程

□问题分析

- 分析问题性质：从未知问题转化为已知问题，例如逻辑判断问题
- 分析输入输出：I/O控制；文件读写；输入输出设备选择
- 决定采用的模型和方法

□结构的设计

- 控制结构，决定程序的运行过程：顺序、选择、循环结构，模块设计
- 数据结构，决定数据在计算机中存放：数组，指针，结构体与联合体

程序设计=算法+数据结构+方法+工具



□ 算法设计：考虑算法代价和可靠性，在有限代价下获得可靠结果

□ 流程描述

- 各类流程图：传统流程图、结构化流程图
- 自然语言、算法语言
- 编程实现：选择合适的编程语言，掌握基本数据类型和语法

□ 程序的编写、调试和运行

- 编辑程序 -> 编译、链接生成可执行文件 -> 执行可执行文件，运行程序
- 善用断点进行debug

程序设计=算法+数据结构+方法+工具



●浮点数排序问题

□问题分析

- 输入输出数据: 浮点数
- 方法: 选择排序

□结构特性设计

- 控制结构:
 - 循环结构实现数组的遍历操作
 - 选择结构控制交换操作的进行
- 数据结构: 浮点数数组

□算法设计: 时间复杂度 $O(n^2)$

□编程实现: 见右侧代码

```
1 void selesort(double p[], int n) {  
2     int i, j, k;  
3     double d;  
4     for (i = 0; i <= n - 2; i++) {  
5         k = i;  
6         for (j = i + 1; j <= n - 1; j++)  
7             if (p[j] < p[k]) k = j;  
8             if (k != i) { //交换元素位置  
9                 d = p[i]; p[i] = p[k]; p[k] = d;  
10            }  
11        }  
12    }
```

数据类型：进制转换



●将十进制转换为二进制：除2取余法

- 将十进制数除以2，得到一个商数和一个余数
- 再将商数除以2，又得到一个商数和一个余数
- 继续这个过程，直到商数等于零为止
- 得到的余数逆序对应二进制数的各位数字

2		97	
2		48	余数为1
2		24	余数为0
2		12	余数为0
2		6	余数为0
2		3	余数为0
2		1	余数为1
		0	余数为1, 商为0

●将二进制转换为十进制：位权求和法

- 将二进制整数的第n位乘以位权 2^n
- 将每一位的值进行求和，即得到十进制数

$$(97)_{10} = (1100001)_2$$

数据类型：原、反、补、偏移码



- 原码：二进制定点数表示

- 原码的符号位在最高位，0表示正，1表示负，数值部分按一般的二进制形式表示

- 反码：正数的反码和原码相同

- 负数的反码是对该数的原码除符号位外各位取反

- 补码：正数的补码和原码相同

- 负数的补码是在该数的反码的最后(即最右边)一位上加1

- 偏移码：其补码的符号位取反即是偏移码

基础知识：数据类型变量



●基本数据类型变量定义

变量类型	表示	存储空间	数值范围
基本整型变量	<u>int</u>	4B	$-2^{31} \sim 2^{31}-1$
长整型变量	<u>long [int]</u>	4B	$-2^{31} \sim 2^{31}-1$
超长整型变量	<u>long long [int]</u>	8B	$-2^{63} \sim 2^{63}-1$
短整型变量	<u>short [int]</u>	2B	$-2^{15} \sim 2^{15}-1$
无符号整型	<u>unsigned [int]</u>	2B	$0 \sim 2^{16}-1$
单精度实型变量	<u>float</u>	4B	$-10^{38} \sim 10^{38}$
双精度实型变量	<u>double</u>	8B	$-10^{308} \sim 10^{308}$
字符型变量	<u>char</u>	1B	$-2^7 \sim 2^7-1$

注：B代表字节(Byte)



●例16-1：英文大小写字母的转换

□每一个英文小写字母比它相应的大写字母的ASCII码大32

```
1 #include <stdio.h>
2 void main() {
3     char x, y, c1, c2;
4     x = 'A' ;
5     y = 'B' ;
6     c1 = x + 32;
7     c2 = y + 32;
8     printf("x=%c, y=%c\n", x, y);
9     printf("c1=%c, c2=%c\n", c1, c2);
10 }
```

可以写成：

```
c1= x - 'A' + 'a' ;
c2= y - 'A' + 'a' ;
```

```
x=A, y=B
c1=a, c2=b
请按任意键继续...
```


数据类型：sizeof 运算符



● 给出一个变量或某种数据类型的量在计算机内存中所占字节数

□ 1. 求表达式计算结果所占内存的字节数

一般形式： sizeof(表达式) 或 sizeof 表达式

□ 2. 求某种数据类型的量所占内存的字节数

一般形式： sizeof(类型名)

```
1 #include <stdio.h>
2 void main() {
3     printf("sizeof(char)=%d\n", sizeof(char));
4     printf("sizeof(int)=%d\n", sizeof(int));
5     printf("sizeof(double)=%d\n", sizeof(double));
6     printf("sizeof(3)=%d\n", sizeof(3));
7     printf("sizeof(3.46)=%d\n", sizeof 3.46);
8 }
```

```
sizeof(char)=1
sizeof(int)=4
sizeof(double)=8
sizeof(3)=4
sizeof(3.46)=8
请按任意键继续...
```

基础知识\表达式与运算符



类型	运算符	说明
赋值运算符	=、+=、-=、*=、/=、%=	从右到左 ，自动进行类型转换，还可强制转换
算术运算符	*, /、+、-、%	从左到右,(*,/,%)优先级 高 , 能先执行就先执行 ；%只适用 整型 ； 从低到高 且 逐步 转换，但 不变数据
关系运算符	>、>=、<、<=、==、!=	(<,<=,>,>=)优先级 高 ，注意 == 用法
逻辑运算符	&&、 、!	从左到右， 能先执行就先执行 ， 不是所有都执行 ； ! > && > ，可赋给 整型 或 字符型 ， 非假(0)即真(1)
单目运算符	++、--、(+、-)	在前在后 有别 ， 不能有空格 ；常量与表达式 不可用
逗号运算符	,	分隔符或 逗号表达式 (最后一个子表达式值)
sizeof	sizeof 或 sizeof()	某种变量或数据类型在计算机 所占字节数
位运算符	&、 、^、~、>>、<<	

单目运算符 > ! > 算术运算符 > 关系运算符 > && > || > 赋值运算 > 逗号运算符 25

运算符：类型转换



● 自动类型转换

□ 表达式中数据类型不一致时,系统将进行 **自动类型转换**

□ 转换原则是 **从低到高**

□ 类型转换 **不改变数据类型**

□ 类型转换是 **逐步进行的**

- 仅考虑运算符两边数据类型 **是否匹配**

(低) int → unsigned int → long → double (高)

↑ 必定转换

char或short

↑ 必定转换

float (C语言统一用双精度运算)

```
1 #include<stdio.h>
2 void main() {
3     printf("%lf\n", 10/4 + 2.5);
4 }
```

● 强制类型转换

□ 强制类型转换的形式为: **(类型名) (表达式)**

□ 自动类型转换无法应对复杂情况, 尽量采用强制类型转换

4. 500000
请按任意键继续.....

运算符：数据类型强制转换



● 辨析：注意浮点数除法

□ 以下几种计算方式将得到浮点数结果

1	sum += 1.0/n;
2	sum += 1/(double)n;
3	sum += (double)1/n;

将其用作循环条件时，防止因为结果错误**掉入死循环**！

□ 以下几种计算方式将得到整型数结果（向下取整）：因为1和n都是整型变量，当 $n > 1$ 时， $1/n$ 的值总为0

1	sum += 1/n;
2	sum += (double)(1/n);

表达式：运算顺序



- 请辨析逻辑表达式 $!(5 > 3) \ \&\& \ 1 \ || \ 2 < 4 - !0$ 的运算顺序

$!(5 > 3)$ $\&\& \ 1 \ || \ 2 < 4 - !0$

$!1$ $\&\& \ 1 \ || \ 2 < 4 - !0$

$0 \ || \ 2 < 4 -$ $!0$

$0 \ || \ 2 <$ $4 - 1$

$0 \ || \$ $2 < 3$

$0 \ || \ 1$

1

表达式：逻辑表达式



- 例16-2：有甲乙丙三人，每人说一句话如下

- 甲说：乙在说谎；乙说：丙在说谎；丙说：甲乙都在说谎；试写出能确定谁在说谎的条件（即逻辑表达式）

- 分别用整型变量abc表示甲乙丙三人，值为1则代表该人说真话，为0则代表在说谎

- 若甲说的是真话，则有 $a==1 \&\& b==0$ ，等价于 $a \&\& !b$ ，若甲在说谎，则有 $!a \&\& b$ ，因此根据甲的话，则有 $a \&\& !b \ || \ !a \&\& b$

- 同理，根据乙的话，则有 $b \&\& !c \ || \ !b \&\& c$

- 若丙是真话，则有 $c==1 \&\& a+b==0$ ，等价于 $c \&\& !(a+b)$ ，若丙在说谎，则有 $!c \&\& a+b$ ，因此根据丙的话有 $c \&\& !(a+b) \ || \ !c \&\& a+b$

- 上面三个条件需同时成立，因此是“逻辑与”的关系，最后得到表达式如下

$$(a \&\& !b \ || \ !a \&\& b) \&\& (b \&\& !c \ || \ !b \&\& c) \ || (c \&\& !(a+b) \ || \ !c \&\& a+b)$$

基础知识\宏定义



类型	定义方法	说明
符号常量定义	<code>#define</code> 符号常量名 字符串	带有符号常量名的地方用字符串进行替换
带参数宏定义	<code>#define</code> 宏名(参数表) 字符串	不仅对字符串进行替换，还进行参数替换。需将参数和字符串都括起来
带#的宏定义 (字符串化)	<code>#define</code> 宏名(参数表) #<标识符>	把标识符中参数替换为参数对应字符串
带#的宏定义 (字符串连接)	<code>#define</code> 宏名(参数表) <标识符> ## <宏变量> >	把标识符中参数的字符串与宏变量字符串连接起来

符号常量名一般用大写字母；只进行简单替换，不做语法检查；双引号字符串不替换；多参数间加逗号；#define不加分号且独占行；可用续行符跨行定义；需注意作用域范围

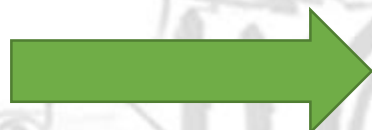
宏定义：宏定义的使用



● 例16-3：宏定义使用时注意括号的使用

```
1 #include <stdio.h>
2 #define F(x) ((x)*(x)*(x)+(x)*(x))
3 void main() {
4     double f, x, y;
5     printf("input x, y: ");
6     scanf("%lf, %lf", &x, &y);
7     /*输入的两个数据之间用逗号分隔*/
8     f=F(x)*F(y+1);
9     printf("f=%f\n", f);
10 }
```

经预处理后



```
1 #include <stdio.h>
2 void main() {
3     double f, x, y;
4     printf("input x, y: ");
5     scanf("%lf, %lf", &x, &y);
6     /*输入的两个数据之间用逗号分隔*/
7     f= ((x)*(x)*(x)+(x)*(x)) *
8        ((y+1)*(y+1)*(y+1)+(y+1)*(y+1));
9     printf("f=%f\n", f);
10 }
```

基础知识\编译预处理



类型	定义	说明
文件包含	<code>#include</code> “文件名” (所属的文件目录中寻找所包含的文件, 再按标准方式检索其他目录) <code>#include</code> <文件名> (标准方式检索其他目录)	将指定文件中的全部内容插入到该命令所在的位置后一起被编译; 一次只能包含一个, 可嵌套使用, 不可递归包含
条件编译	<code>#ifdef</code> , <code>#else</code> , <code>#endif</code> / <code>#ifdef</code> , <code>#endif</code> <code>#ifndef</code> , <code>#else</code> , <code>#endif</code> / <code>#ifndef</code> , <code>#endif</code> <code>#if</code> , <code>#else</code> , <code>#endif</code> / <code>#if</code> , <code>#endif</code> <code>#if</code> , <code>#elif</code> , <code>#else</code> , <code>#endif</code> / <code>#undef</code>	只在满足一定条件时才进行编译; 或者当满足条件时对一部分语句进行编译, 可减少目标程序长度, 减少运行时间。 <code>#ifdef</code> 只关注定义本身, 与具体值无关
<code>#pragma</code>	<code>#pragma token-string</code>	<code>once</code> 让编译器把指定文件只包含一次; <code>warning</code> 让编译器不再显示这类警告
<code>#line</code>	<code>#line</code> 数字[“文件名”]	让编译器编译显示错误信息时, 改变当前所显示的行号和文件名



● 例16-4：文件包含命令尖括号与引号的使用

□ 源文件中调用A.h

□ A.h中调用B.h

调用stdio.h使用尖括号
调用A.h与B.h使用双引号

```
1 //A.h
2 #include "B.h"
3 #define A 100
```

```
1 //B.h
2 #define B 200
```

```
1 #include <stdio.h>
2 #include "A.h"
3 void main() {
4     int x = A, y = B;
5     printf("%d\n", x);
6     printf("%d\n", y);
7 }
```

```
100
200
请按任意键继续...
```




● 例16-5：条件编译命令

- 对键盘输入的字符，将大写字母转换成小写字母，其他字符不变

```
input ch:A
a
请按任意键继续...
```

```
1 #define LOW 1
2 #include <stdio.h>
3 void main() {
4     char ch;
5     printf("input ch:");
6     scanf("%c", &ch);
7     #ifdef LOW
8         if (ch >= 'A' && ch <= 'Z' )
9             ch = ch - 'A' + 'a'; /*大写字母转换成小写字母*/
10    #else
11        if (ch >= 'a' && ch <= 'z' )
12            ch = ch - 'a' + 'A'; /*小写字母转换成大写字母*/
13    #endif
14    printf("%c\n", ch);
15 }
```

数据读写\输入输出格式字符



格式字符	说明
c	输入(出)一个 <u>字符</u>
d	输入(出) <u>带符号的十进制整型数</u>
i	输入(出) <u>整型数</u> ，整型数可以是带先导0的八进制数，也可以是带先导0x(或0X)的十六进制数，输入时候会转换成十进制
o	输入(出) <u>八进制格式整型数</u> ，可以带先导0，也可以不带
x	输入(出) <u>十六进制格式整型数</u> ，可以带先导0x或0X，也可以不带
u	输入(出) <u>无符号十进制形式整型数</u>
f (lf)	输入(出) <u>以小数形式浮点数</u> （单精度用f，双精度数用lf）
e (le)	输入(出) <u>以指数形式浮点数</u> （单精度用e，双精度数用le）
g	输入(出)自动决定输出格式为e和f中较短的一种，不打印无效的零
p	输出变量的内存地址
s	输入(出)一个 <u>字符串</u> ，直到遇到"\0"

数据读写\输入输出格式字符



格式字符	说明
l	<u>输入(出)</u> 长度修饰: 整型 %ld, %lu, %lo, %lx; 浮点型 %lf, %le
h	<u>输入(出)</u> 长度修饰: 整型 %hd
m	整型 <u>输入(出)</u> 位宽: 超过自动突破; 不足右对齐, 左补空格, 达到位宽 字符串 <u>输出</u> 宽度: 超过自动突破; 不足右对齐, 左补空格, 达到位宽
m.n	浮点型 <u>输入(出)</u> 精度位宽和精度, 可以只有.n, 也可以m.0, 还可以.0 m超过自动突破; 不足右对齐, 左补空格; n超过四舍五入; 不足右边补0
.n	浮点型 <u>输出</u> 精度, 见上; 字符串 <u>输出</u> 宽度, 超过截断, 不足右对齐, 左补空格
0	前导0符号: 整型 <u>输出</u> 右对齐, 左边补0; 可以写作%0m或%.m表示
+	正号符号: <u>输出</u> 正数总带+号
-	左对齐符号: <u>输出</u> 左对齐
*	<u>输出</u> 变位宽输出; <u>输入</u> 跳过某个数
%	<u>输入(出)</u> 格式说明引导符

数据读写\格式输入函数



知识点	格式	说明
格式输出语句	<code>printf ("格式控制", 输出列表);</code>	格式说明符与输出列表中的量 <u>一一对应</u> ， <u>类型一致</u> ， <u>个数相同</u>
格式输入语句	<code>scanf ("格式控制", 内存地址表);</code>	取地址运算符 <u>&</u> ，输入项与格式说明符必须 <u>一一对应</u> ；输入数据时，两个数据之间用“ <u>空格</u> ”、“ <u>Tab</u> ”、“ <u>回车</u> ”来分隔。
字符输出函数	<code>putchar(c)</code>	在屏幕当前光标位置处输出项目c所表示的一个字符。c可以是 <u>字符型常量</u> 、 <u>字符型变量</u> 、 <u>整型变量</u> 或 <u>整型表达式</u> 、 <u>转义字符</u>
字符输入函数	<code>getchar(c)</code>	接收从键盘输入的一个字符，由键盘输入的字符在屏幕上显示，并且以回车结束
	<code>_getch()</code>	使用 <u>_getch()</u> ，由键盘输入的字符不在屏幕上显示；
	<code>_getche()</code>	使用 <u>_getche()</u> ，由键盘输入的字符在屏幕上显示

输入与输出：格式输出函数



● 例16-6：各种情况下的输出示例

□ `int k=1234; double f =123.456;`

```
1 printf("%12.0f\n", f);
2 printf("%.f\n", f);
3 printf("%e\n", f);
4 printf("%13e\n", f);
5 printf("%13.8e\n", f);
6 printf("%3.8e\n", f);
7 printf("%.8e\n", f);
8 printf("%13.2e\n", f);
9 printf("%13.0e\n", f);
10 printf("%.0e\n", f);
11 printf("%g\n", f);
12 printf("%5g\n", f);
13 printf("%10g\n", f);
14 printf("%g\n", 123.456789);
```

```
123
123
1.234560e+02
1.234560e+02
1.23456000e+02
1.23456000e+02
1.23456000e+02
1.23e+02
1e+02
1e+02
123.456
123.456
123.456
123.457
```

```
1 printf("%06d\n", k);
2 printf("%.6d\n", k);
3 printf("%012.6f\n", f);
4 printf("%013.2e\n", f);
5 printf("%s\n", "abcdefg");
6 printf("%10s\n", "abcdefg");
7 printf("%5s\n", "abcdefg");
8 printf("%.5s\n", "abcdefg");
9 printf("%-6d\n", k);
10 printf("%-12.2f\n", f);
11 printf("%-13.2e\n", f);
12 printf("%+-6d%+-12.2f\n", k, -f);
13 printf("%4.1f%%\n", 12.5);
```

```
001234
001234
00123.456000
000001.23e+02
abcdefg
  abcdefg
abcdefg
abcde
1234
123.46
1.23e+02
+1234 -123.46
12.5%
```


输入与输出：格式输入函数



●例16-7：格式输入函数

```
1 #include <stdio.h>
2 void main() {
3     int a, b;
4     float c, d;
5     scanf("%d%d%f%f", &a, &b, &c, &d);
6 }
```

12 78 12.5 7.6
请按任意键继续...

□格式说明符之间有**其他字符**，要从键盘输入**同样字符**

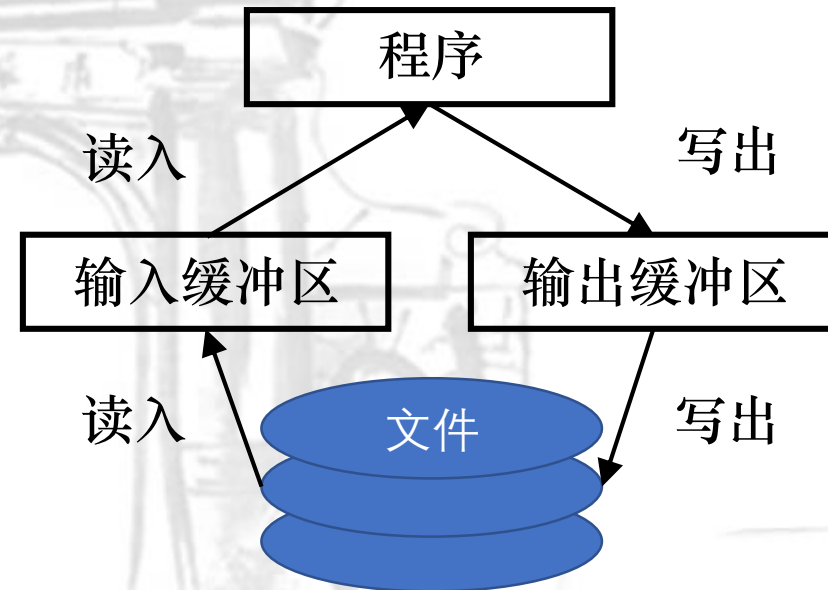
```
1 #include <stdio.h>
2 void main() {
3     int a, b;
4     float c, d;
5     scanf("a=%d, b=%d, c=%f, d=%f", &a, &b, &c, &d);
6 }
```

a=12, b=78, c=12.5, d=7.6
请按任意键继续...

- 定义：对文件的操作都是通过库函数来实现的
 - 文件系统可以分为高级文件系统和低级文件系统，这是按照是否有缓冲文件系统来划分的

- 高级文件系统示意图

- 指系统自动地为正在被使用的文件在内存中开辟一个缓冲区
- 输入输出都引入缓冲区
- 读入和写出都经过缓冲区



文件操作：文本与二进制文件



● 文本文件

□ 文本文件又称为ASCII文件。在这种文件中，每个字节存放一个字符的ASCII码值。每个整数或实数是由每位数字的ASCII字符组成

- 举例：.c、.cpp等源程序文件是文本文件

● 二进制文件

□ 二进制文件中的数据与该数据在内存中的二进制形式是一致的，是把整数的补码、浮点数的IEEE 754表示直接写入文件中，其中一个字节并不代表一个字符。可执行文件都是二进制文件

- 举例：.jpg、.avi、.docx等都是二进制文件

文件操作：文件操作函数



函数名	函数原型	详细说明
fopen	FILE *fp=fopen("文件名","文件打开方式");	选择以何种方式打开 文本文件 或者 二进制文件
fclose	fclose(fp);	当一个文件使用完后将其 关闭并释放该缓冲区
fgetc	char fgetc(FILE *fp);	从指定的文件读取字符
fputc	int fputc(char ch, FILE *fp);	将字符写入到指定的文件流中
fgets	char *fgets(char * string, int n, FILE *fp);	读入一行字符（不超过n-1个字符）存放到由string的存储空间中，结束后将 自动 在字符串最后 加'\0'
fputs	int fputs(char *string, FILE *fp);	将指定的字符串写到文件fp中。若写成功，则 返回非负值 ；若写不成功，则 返回EOF
feof	int feof(fp);	专门用来判断文件 是否结束

文件操作：文件操作函数



函数名	函数原型	详细说明
fread	<code>int fread(void *buffer, int size, int count, FILE *fp);</code>	从指定的文件中以 <u>二进制</u> 格式 <u>读入</u> 一组数据到指定的内存区
fwrite	<code>int fwrite(void *buffer, int size, int count, FILE *fp);</code>	将一组数据以 <u>二进制</u> 格式 <u>写到</u> 指定的文件中
fscanf	<code>int fscanf(文件指针, 格式控制, 地址参数表);</code>	从指定的 <u>文本</u> 文件中格式化地 <u>读入</u> 数据
fprintf	<code>int fprintf(文件指针, 格式控制, 输出参数表);</code>	格式化地 <u>写入</u> 数据到指定的 <u>文本</u> 文件中
rewind	<code>void rewind(FILE *fp);</code>	将文件的读写指针移动到文件的 <u>开头</u>
fseek	<code>int fseek(FILE *stream, long offset, int origin);</code>	将文件的读写指针移动到 <u>指定的位置</u>
ftell	<code>long ftell(FILE *fp);</code>	返回文件的 <u>当前</u> 读写位置（出错返回-1）
fflush	<code>int fflush(FILE *fp);</code>	<u>清空</u> 文件的输入输出 <u>缓冲区</u> ，使文件缓冲区中的内容立刻输出到实际的文件中

文件操作：文本与二进制文件写入



● 例16-8：文本文件和二进制文件写入

```
1  #include <stdio.h>    //包含头文件"stdio.h"
2  void main() {
3      int a=12345, b=567890; float f=97.6875f; double f2=97.6875;
4      FILE *fp;
5      fp=fopen("data1.txt", "w");           //文本文件
6      fprintf(fp, "%d %d %f %f\n", a, b, f, f2);
7      fclose(fp);                          //关闭文件
8      fp=fopen("data2.txt", "wb");          //二进制文件
9      fwrite(&a, sizeof(int), 1, fp);       //块写入
10     fwrite(&b, sizeof(int), 1, fp);
11     fwrite(&f, sizeof(float), 1, fp);
12     fwrite(&f2, sizeof(double), 1, fp);
13     fclose(fp);                          //关闭文件
14 }
```

数据读写\文件打开格式



● 文件打开方式

标志	含义	详细说明
r	只读	为读打开一个文件。若指定的文件不存在，则 <u>返回空指针值NULL</u>
w	只写	为写打开一个新文件。若指定的文件已存在，则其中 <u>原有内容被删去</u> ；否则 <u>创建一个新文件</u>
a	追加写	向 <u>文件尾增加数据</u> 。若指定的文件不存在，则 <u>创建一个新文件</u>
r+	读写	为读写打开一个文件。若指定的文件不存在，则 <u>返回空指针值NULL</u>
w+	读写	为读写打开一个新文件。若指定的文件已存在，则其中 <u>原有内容被删去</u> ；否则 <u>创建一个新文件</u>
a+	读与追加写	为读写向文件尾增加数据打开一个文件。若指定的文件不存在，则 <u>创建一个新文件</u>

文件操作：文件的读写



● 例16-9：读入一行文本

- 第1行不足19个字符，连回车符也读入到字符串中并输出到了屏幕
- 第2行超过19个字符，因此fgets只读入前19个字符，fgets下一次读入了第2行剩余的字符连同回车符

```
1 #include <stdio.h>
2 #include <string.h>
3 void main() {
4     char a[20]; int n; FILE *fp;
5     fp=fopen("d:\\1.txt", "r");
6     for (n=0;n<3;n++) {
7         fgets(a, 20, fp);
8         printf("%s%d\n", a, strlen(a));
9     }
10    fclose(fp);
11 }
```

1.txt的内容为:

```
abcdefghijklmnpqr
abcdefghijklmnpqrstuvwxy
```

```
abcdefghijklmnpqr
19
abcdefghijklmnpqrs 19
tuvwxyz
8
请按任意键继续.....
```

程序设计\顺序与分支结构



类型	定义	说明
顺序结构	语句1; 语句2; 语句3;	顺序执行相关语句
分支结构 (if语句)	<pre>if(条件1) 语句1; else if(条件n) 语句n; else 语句n+1;</pre>	if语句: if (条件) 语句; if+复合语句,约束范围 if-else语句: if (条件) 语句1; else 语句2; if-else可以 嵌套 , 条件设置 和 顺序 需注意
分支结构 (条件运算符)	表达式1 ? 表达式2: 表达式3	方便、紧凑的分支语句,结合方向 从右到左 优先级 比赋值运算符高 ,比 关系、算术运算符低
分支结构 (switch结构)	<pre>switch(表达式) { case 常量表达式1: 语句1; ... case 常量表达式n: 语句n; default: 语句n+1; }</pre>	case 判断条件 , 语句n 执行命令 , default 备选 常量表达式必须是 整形 或 字符型 ,且 互不相同 , 但 顺序任意 ; default 可以没有 ; 内部顺序执行, 但break可从语句中 跳转出来 ,且需结合使用
复合语句: {...}实现, }后 不加分号 , 变量 生命周期 , 变量的 掩蔽		

顺序结构：生命周期



● 例16-10：变量的掩蔽现象

- 在复合语句的嵌套结构中，有时在内层与外层定义了同名的变量
- 这种情况按照局部优先的原则，内层复合语句中的变量掩蔽外层
- 复合语句的同名变量，直到内层复合语句结束，外层复合语句的同名变量才可以访问到
- 内层复合语句中对内层定义的变量的执行结果也不带回到外层

```
1  #include <stdio.h>
2  void main() {
3      int x, y;
4      x=10;
5      y=100;
6      { int x;
7          x = 20;
8          printf("y=%d\n", y);
9          printf("x=%d\n", x);
10     }
11     printf("x=%d\n", x);
12 }
```

```
y = 100
x = 20
x = 10
请按任意键继续.....
```


分支结构：if-else 语句



● 例16-11：条件设置顺序影响执行效率：对比两种实现

实现方法A

```
1 #include <stdio.h>
2 void main() {
3     float x, y;
4     printf("Input x:");
5     scanf("%f", &x);
6     if (x>=50.0) y = 0.0;
7     else if (x>=40.0) y = 50-x;
8     else if (x>=20.0) y = 30-0.5*x;
9     else if (x>=0.0) y = 20.0;
10    else if (x>=-10.0) y = 2*x+20;
11    else y = 0.0;
12    printf("x = %f, y = %f\n", x, y);
13 }
```

实现方法B

```
1 #include <stdio.h>
2 void main() {
3     float x, y;
4     printf("Input x:");
5     scanf("%f", &x);
6     if (x>=20.0) {
7         if (x>=50.0) y = 0.0;
8         else if (x>=40.0) y = 50-x;
9         else y = 30-0.5*x;
10    } else {
11        if (x>=0.0) y = 20.0;
12        else if (x>=-10.0) y = 2*x+20;
13        else y = 0.0;
14    }
15    printf("x = %f, y = %f\n", x, y);
16 }
```

分支结构：条件运算符



● 例16-12：条件运算符精简代码

□ 从键盘输入一个x，计算并输出下列分段函数值：

$$y = \begin{cases} x^2 - 1, & x < 0 \\ x^2 + 1, & x \geq 0 \end{cases}$$

if语句需要两句



条件语句仅需一句

```
1 #include <stdio.h>
2 void main() {
3     float x, y;
4     printf("Input x:");
5     scanf("%f", &x);
6     if(x<0) y = x*x-1;
7     else y = x*x+1;
8     printf("y = %f\n", y);
9 }
```

```
1 #include <stdio.h>
2 void main() {
3     float x, y;
4     printf("Input x:");
5     scanf("%f", &x);
6     y=(x<0)?(x*x-1):(x*x+1);
7     printf("y = %f\n", y);
8 }
```

分支结构：switch 结构



● 例16-13：switch结构中break作用辨析

□ 注意"break;"语句**非常重要**，如果将其去掉，则：

```
1  #include <stdio.h>
2  void main() {
3      float grade;
4      printf("Input grade:");
5      scanf("%f", &grade);
6      switch (int(grade/10)) {
7          case 10:
8          case 9: printf("A!\n");
9          case 8: printf("B!\n");
10         case 7: printf("C!\n");
11         case 6: printf("D!\n");
12         default: printf("E!\n");
13     }
14 }
```

```
Input grade: 100
A!
B!
C!
D!
E!
请按任意键继续.....
```

程序设计\循环结构



类型	定义	说明
当型循环 (while)	<code>while</code> (表达式) 循环体语句;	先判断后执行;条件满足执行循环;不满足退出循环 可能一次都不执行
直到循环 (do-while)	<code>do</code> 循环体语句 <code>while</code> (表达式);	先执行后判断;条件满足退出循环;不满足继续执行 至少执行一次
循环结构 (for)	<code>for</code> (表达式1;表达式2;表达式3) 循环体语句;	精简地实现当型循环;三个表达式分别给 循环变量 赋初值、判断循环条件、循环变量增值 ; 表达式1、3 可省略 ,但分号 不可省 ;表达式2 必须有
<code>break</code>	<code>break</code> ;	退出当前循环结构(或switch结构)
<code>continue</code>	<code>continue</code> ;	结束本次循环执行,但不退出循环结构

循环可嵌套,内循环**完全嵌套**于外循环中,内外循环**不交叉**,内外循环控制变量**不重名**
`while`和`do-while`语句循环**次数有限**和循环**次数未知**;需要有**改变循环条件语句**
`for`语句事先知道循环的**起始点**和**终止点**及其**循环次数**的问题

循环结构：当型与直到型循环



- 例16-14：下列C程序的功能是计算并输出 $n!$ （阶乘）值，其中 n 从键盘输入

```
1 #include <stdio.h>
2 void main() {
3     int n, k;
4     double s;
5     printf("input n:");
6     scanf("%d", &n);
7     k=1; s=1.0;
8     while(k<n) {
9         k++;
10        s = s*k;
11    }
12    printf("n! = %lf\n", s);
13 }
```

当型

```
1 #include <stdio.h>
2 void main() {
3     int n, k;
4     double s;
5     printf("input n:");
6     scanf("%d", &n);
7     k=0; s=1.0;
8     do {
9         k++;
10        s = s*k;
11    } while(k<n);
12    printf("sum = %lf\n", s);
13 }
```

直到型

循环结构：for语句



- 例16-15：下列C程序的功能是计算并输出n!（阶乘）值，其中n从键盘输入

```
1  #include <stdio.h>
2  void main() {
3      int n, k;
4      int s;
5      printf("Input n:");
6      scanf("%d", &n);
7      s=1;
8      for(k=1; k<=n; k++) {
9          s = s*k;
10     }
11     printf("n! = %d\n", s);
12 }
```

```
Input n: 4
4! = 24
请按任意键继续.....
```

循环结构：循环的中断



● 例16-16：输出100 ~ 200之间所有能被7和9整除的自然数

```
1 #include <stdio.h>           //包含头文件
2 void main() {
3     int n;
4     for(n=100; n<=200; n++) { //遍历100~200的三位数
5         if ((n%7!=0) || (n%9!=0))
6             continue;         //找不到就进入下次循环
7         printf("%d\n", n);
8     }
9 }
```

```
if ((n%7==0)&&(n%9==0))
    printf("%d\n", n);
```

126
189
请按任意键继续……

程序设计\函数的定义与使用



类型	定义	说明
函数定义	类型标识符 函数名 (形参名称与类型列表) {说明部分 语句部分}	函数没有从属关系，不可嵌套定义，互相独立，可互相调用
函数向前引用说明	函数类型 函数名(形参1类型 形参名1, 形参2类型 形参名2, ...);	也称函数原型。函数类型、形参类型和个数需与原函数一致,形参名可省略
函数调用	函数名(形参名称与类型列表)	函数调用可以在表达式中也可以单独一个语句。函数可以嵌套调用
形参与实参结合	数值结合：实参地址和形参地址互相独立，直接将实参值传送给形参并放在形参地址	形参的改变不影响调用程序的实参值，只能单向传递，通过返回值传递改变
局部变量与全局变量	局部：函数内部定义，函数范围内有效 全局：函数外定义，定义位置到文件结束	作用域不同。不同函数局部变量可重名。全局变量所有函数都可访问，可以被局部变量掩蔽

程序设计\函数的定义与使用



类型	定义	说明
动态变量与静态变量	<p>静态变量: 程序运行期间由系统分配固定的存储空间, 整个程序运行期间都不释放</p> <p>动态变量: 程序运行期间根据需要动态分配存储空间。调用时分配, 调用后释放</p> <p>外部变量: 变量的向前引用说明; 其他文件可使用; 静态外部变量控制作用域</p>	存储类别决定了该数据的 存储区域、作用域、生存期 。形参 不能 为静态变量。静态变量 默认初值0 , 作用域为 本文件 。默认自动类型, 用 static 说明静态变量, 用 extern 说明的外部变量
内部函数与外部函数	<p>static 类型标识符函数名(形参表)</p> <p>[extern] 类型标识符函数名(形参表)</p>	内部函数只能被 本文件中其他函数 调用; 外部函数能被 其他文件中函数 调用
递归	<p>直接递归: 直接调用函数本身</p> <p>间接递归: 通过调用别的函数调用自身</p>	注意递归与操作的 时序问题 。找出递归的 初始值 和 递归表达式

函数模块：形参与返回值



● 例16-17：计算1到5之间各自然数的阶乘值

函数的向前引用说明

被调用函数p的返回值是int型，此函数有一个int型形参

```
1 #include<stdio.h>
2 void main() { /*主函数*/
3     int n, m;
4     int p(int);
5     //说明要调用的函数p() 返回值是int型, 有一个int型形参
6     for (m = 1; m <= 5; m++)
7         printf("%d!=%d\n", m, p(m));
8 }
9 //计算阶乘值的函数, 返回值为int型
10 int p(int n) {
11     int k, s;
12     s = 1;
13     for (k = 1; k <= n; k++) s = s * k;
14     return(s);
15 }
```

```
1!=1
2!=2
3!=6
4!=24
5!=120
请按任意键继续...
```

- 主函数main()：通过循环调用函数p()计算并输出m!的值
- p()：它的功能是计算阶乘值，例如，p(m)是计算m!

函数模块：局部变量与全局变量



●例16-18：采用全局变量进行迭代法求根

```
1 #include <stdio.h>
2 double x; //定义全局变量
3 int subroot(double eps) {
4     int m;
5     double x0, f(double);
6     m = 0;
7     do {
8         m = m + 1;
9         x0 = x;
10        x = f(x0);
11    } while ((m <= 100) && (fabs(x - x0) >= eps));
12    if (m > 100) printf("FAIL!\n");
13    printf("x=%f\n", x);
14    return(m);
15 }
```

```
16 void main() {
17     int m, subroot(double);
18     double eps;
19     x = 1.0;
20     eps = 0.000001;
21     m = subroot(eps);
22     printf("m=%d\n", m);
23     printf("x=%f\n", x);
24 }
25 double f(double x) {
26     return 1.0 + atan(x);
27 }
```

```
x=2.132268
m=10
x=2.132268
请按任意键继续...
```

函数模块：静态变量



● 例16-19：静态变量与动态变量区别

❑ 函数调用结束后其内存不会消失而保留原值，在下一次调用时仍为 上次调用结束时的值

❑ 静态变量时若不赋初值，则编译时 自动赋初值0

去掉static

```
1 int ksum(int n) {  
2     static int x = 0;  
3     x = x + n;  
4     return(x);  
5 }  
6 #include <stdio.h>  
7 void main() {  
8     int k, ksum(int);  
9     for (k = 1; k <= 5; k++)  
10         printf("sum(%d)=%d\n", k, ksum(k));  
11 }
```

sum(1)=1
sum(2)=3
sum(3)=6
sum(4)=10
sum(5)=15
请按任意键继续...

```
1 int ksum(int n) {  
2     int x = 0;  
3     x = x + n;  
4     return(x);  
5 }  
6 #include <stdio.h>  
7 void main() {  
8     int k, ksum(int);  
9     for (k = 1; k <= 5; k++)  
10         printf("sum(%d)=%d\n", k, ksum(k));  
11 }
```

sum(1)=1
sum(2)=2
sum(3)=3
sum(4)=4
sum(5)=5
请按任意键继续...

函数模块：外部变量



● 例16-20：变量值交换

□ 用 **extern** 使全局变量定义点之前的函数中也能使用该全局变量

```
1 #include <stdio.h>
2 void main() {
3     extern int x, y; /*x与y定义为外部变量*/
4     void swap();
5     scanf("x=%d, y=%d", &x, &y);
6     swap();
7     printf("x=%d, y=%d\n", x, y);
8 }
9 int x, y;
10 void swap() {
11     int t;
12     t = x; x = y; y = t;
13     return;
14 }
```

x=1, y=2
x=2, y=1
请按任意键继续...

若在主函数中没有用extern说明变量x和y，则x和y变成主函数的局部变量，**掩蔽**全局变量x和y。swap不能将主函数中的x和y的值交换

x=1, y=2
x=1, y=2
请按任意键继续...

函数模块：内部函数与外部函数



● 例16-21：调用外部函数

```
1  /* file.c */
2  #include <stdio.h>
3  void main() {
4      int x = 10, z, y, f(int), g(int);
5      y = f(x); z = g(x);
6      printf("x=%d\n y=%d\n", x, y);
7      printf("z=%d\n", z);
8  }
```

```
1  /* file1.c */
2  static int f(int x) { /*f是内部函数*/
3      return x * x;
4  }
5  /* g调用本文件中的内部函数f */
6  int g(int x) {
7      return f(x);
8  }
```

```
1  /* file2.c */
2  int f(int x) {
3      return x * x * x;
4  }
```

同一个项目中虽然有同名函数f，但因为其中一个是static类型的，是**内部函数**，因此编译时不会出现重名错误

```
x=10
y=1000
z=100
请按任意键继续...
```



● 例16-22：上楼梯问题

□ 用递归方法编写函数 $f(n)$ ：一共有 n 个台阶，某人上楼一步可以跨1个台阶，也可以跨2个台阶，问有多少种走法

● 分析：

□ 当 $n=1$ 时，共1种走法； $f(1)=1$ ；

□ 当 $n=2$ 时，一步走一个或两个台阶，共2种走法； $f(2)=2$ ；

□ 假设已经知道 $n-1$ 时的走法，那么当 n 时，可归结为两种情况

- 一步1个台阶，剩 $n-1$ 个台阶
- 一步2个台阶，剩 $n-2$ 个台阶

递归式： $f(n)=f(n-1)+f(n-2)$

函数模块：递归调用



● 例16-22：上楼梯问题

□ 用递归方法编写函数 $f(n)$ ：一共有 n 个台阶，某人上楼一步可以跨1个台阶，也可以跨2个台阶，问有多少种走法

```
1 #include <stdio.h>
2 int f(int n) {
3     if (n == 1)
4         return 1;
5     else if (n == 2)
6         return 2;
7     else
8         return f(n - 1) + f(n - 2);
9 }
10 void main() {
11     int n;
12     scanf("%d", &n);
13     printf("T=%d\n", f(n));
14 }
```

递归式： $f(n)=f(n-1)+f(n-2)$

3
T=3
请按任意键继续...

10
T=89
请按任意键继续...

30
T=1346269
请按任意键继续...

数据结构\数组的定义与初始化



类型	定义	说明
一维数组	<p>一维数组: 类型说明符 数组名[常量表达式]</p> <pre>int z[] = { 0, 0, 0, 0, 0 }; int a[10] = {1, 2, 3, 4, 5}; static int y[5];</pre>	<p>下标范围是0到N-1, 只能逐个引用元素; 常量表达式必须为整型, 但不能是变量; 部分元素赋值后, 后面自动赋初值0; 外部数组和静态数组赋初值0或给定值</p>
二维数组	<p>二维数组: 类型说明符 数组名[常量表达式1] [常量表达式2];</p> <pre>int c[][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8}; int d[][4] = {{1, 2}, {5}, {9, 10, 11}};</pre>	<p>二维数组存储顺序是以行为主的; 分行赋初值, 部分赋值后续自动为0; 全部或分行赋值可省略第一维长度说明</p>
字符数组	<pre>char 数组名[常量表达式]; char 数组名[常量表达式1][常量表达式2];</pre>	<p>字符数组中的一个元素只能存放一个字 符, 初始化与数组类似</p>
字符串	<pre>char a[] = "how do you do?"; 错误: char b[15]; b = "China"; char b[15]; b[15] = "China";</pre>	<p>一对双撇号括起来, 隐含包括一个结束符 '\0', 字符串可以对字符数组进行初始化</p>

数组：一维数组初始化



● 例16-23：不同类型的数组初始化

- 数组可以只给前若干元素赋初值，后面的元素将自动赋初值0
- 对程序进行编译连接时，外部数组和静态数组就自动赋初值（0或者给定的初值）

```
1 #include <stdio.h>
2 int k, x[5];
3     /* 外部数组变量会自动初始化为0 */
4 void main() {
5     static int y[5];
6     /*用static说明的局部静态数组会自动初始化为0 */
7     int z[] = { 0, 0, 0, 0, 0 };
8     /* 定义不写长度, 由初值个数确定数组长度*/
9     for (k = 0; k<5; k++)
10         printf("%5d%5d%5d\n", x[k], y[k], z[k]);
11 }
```

```
0  0  0
0  0  0
0  0  0
0  0  0
0  0  0
请按任意键继续...
```

数组：二维数组初始化



●例16-24：二维数组赋初值省略第一维长度

```
1 #include <stdio.h>
2 void print_num(int a[3][4]) {
3     for (int i = 0; i < 3; i++) {
4         for (int j = 0; j < 4; j++)
5             printf("%d ", a[i][j]);
6         printf("\n");
7     }
8 }
9 void main() {
10     int c[][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
11     //等价于int a[4][4], 最后3个元素值为0
12     int d[][4]={ {1, 2}, {5}, {9, 10, 11} }; //等价于int a[3][4], 缺省者为0
13     print_num(c);
14     printf("\n");
15     print_num(d);
16 }
```

```
1 2 3 4
5 6 7 8
9 10 11 12
13 0 0 0
```

```
1 2 0 0
5 0 0 0
9 10 11 0
```

请按任意键继续...

数据结构\数组与函数



类型	定义	说明
字符数组 输入输出	<pre>scanf("%c%c", &a[1], &a[2]); printf("%c\n", a[2]); scanf("%s%s", b, c); printf("%s\n", b);</pre>	%c 输入/输出时, 为 数组元素地址/元素 ; %s 输入输出时, 输入输出均为 数组名 ; 输入自动加 结束符'\0' ; 输出遇 '\0' 则结束
字符串处 理函数	<pre>strcat(s1, s2); 字符串2连接到1的后面; strcpy(s1, s2); 将字符串2拷贝到1中; strncpy(s1, s2, n); 将字符串2前n个字符 复制到字符数组1中; strcmp(字符串1, 字 符串2); 按照字典序比较两个字符串大小;</pre>	<pre>strlwr(字符串) 将字符串中大写字母转成小写; strupr(字符串) 将字符串中小写字母转成大写; sprintf(字符数组名, "输出格式", 变量列表); 将结果输出到字符数组中; sscanf(字符数组名, "输入格式", 变量列表); 从字符数组中读入数据;</pre>
形参与实 参数组	<pre>函数类型 函数名(数组类型 形参数组名); void exchange(int a[], int b[], int n)</pre>	形参与实参数组采用 地址结合 , 实现 双向传递 ; 一维形参数组一般 不指定大小
二维数组 作函数参 数	<pre>正确: int a[2][4]; int a[][4]; 错误: int a[2][]; int a[][]; matmul((int*)a, (int*)b, 2, 4, 3);</pre>	可省略 第一维大小 , 不可省略 第二维 ; 实参是二维数组, 形参是一维数组, 结合还是 地址结合 , 但需 强制类型转换

数组：字符数组与字符串



● 例16-25：字符数组的输入与输出

```
1 #include <stdio.h>
2 void main() {
3     char a[5];
4     scanf("%c%c", &a[1], &a[2]);
5     a[0] = 'a'; a[3] = 'd'; a[4] = '\0';
6     printf("%s\n", a);
7     char b[20], c[20];
8     scanf("%s%s", b, c);
9     printf("%s\n", b);
10    printf("%s\n", c);
11 }
```

```
bc
abcd
helloworld
debug
helloworld
debug
请按任意键继续...
```

数组：字符串函数



● strlen(字符串1)

- 求字符串长度(不包括结束符'\0')
- “字符串”可以是字符数组名,也可以是字符串常量

● 例16-26: strlen()与sizeof()

```
1 #include <stdio.h>
2 #include <string.h>
3 void main() {
4     char s[10] = "abcde";
5     printf("%d\n", strlen(s));
6     printf("%d, %d\n", sizeof(s), strlen(s));
7 }
```

```
5
10, 5
请按任意键继续...
```

数组：数组作为函数参数



● 例16-27：形参数组与实参数组的结合

□ 形参数组与实参数组间采用地址结合，实现数据双向传递


```
1 #include <stdio.h>
2 void exchange1(int a[], int b[])
3     { for (int k = 0; k < 10; k++) a[k] = b[k]; }
4 void exchange2(int a, int b)
5     { a = b; }
6 void main() {
7     int a[10] = {0, 0, 0}; int a1 = 0, b1 = 233;
8     int b[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
9     exchange1(a, b);
10    exchange2(a1, b1);
11    for (int k = 0; k < 10; k++)
12        printf("%d ", a[k]);
13    printf("\na1 = %d , b1 = %d\n", a1, b1);
14 }
```

1 2 3 4 5 6 7 8 9 10
a1 = 0, b1 = 233
请按任意键继续...

函数中进行改变后，
数组a的值改变，
但变量a1的值不变

数据结构\指针



知识点	使用示例	说明
指针的定义	<code>int *p;</code>	用以存放其他变量所占存储空间的首地址的变量叫做 指针 ;指针 定义 时前面有“ * ”
直接访问	<code>int a; a = 5;</code>	直接通过 变量名 进行操作
间接访问	<code>int a,*p=&a; *p=5;</code>	通过指针对变量进行修改, 指针在 使用 时变量名前加“ * ”表示 间接存取
悬浮指针	<code>int a,*p; *p=a;</code> 	指针变量中具有 确定地址值 后才能被引用
指针作为形参	<code>void f (int*p){...}</code>	通过 地址结合 ,实现形参与实参的 双向传递
指向指针的指针	<code>int **q;</code>	指向指针型数据的指针
多级间接访问	<code>int x, *p, **q; p = &x;q = &p;**p = 3;</code>	通过指针实现间接访问,称为 一级间接访问 ,通过指向指针的指针实现 二级间接访问 。依此类推,C语言允许多级间接访问

指针：指针变量作为函数参数

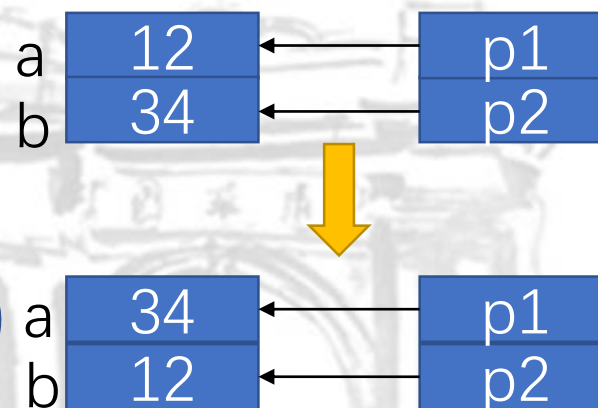


●例16-28：指针作为函数形参可以实现数据的双向传递

```
1 #include <stdio.h>
2 void swap(int *p1, int *p2) {
3     int t;
4     t=*p1; *p1=*p2; *p2=t;
5     return;
6 }
7 void main() {
8     int a, b;
9     scanf("%d,%d", &a, &b);
10    printf("a=%d, b=%d\n", a, b);
11    swap(&a, &b);
12    printf("a=%d, b=%d\n", a, b);
13 }
```

参数传递是
传值变量的
地址值

方便对比测试，输入固定为12, 34



12,34
a=12,b=34
a=34,b=12
请按任意键继续……

将p1和p2所指单元的内容交换，因此a和b的值被交换了

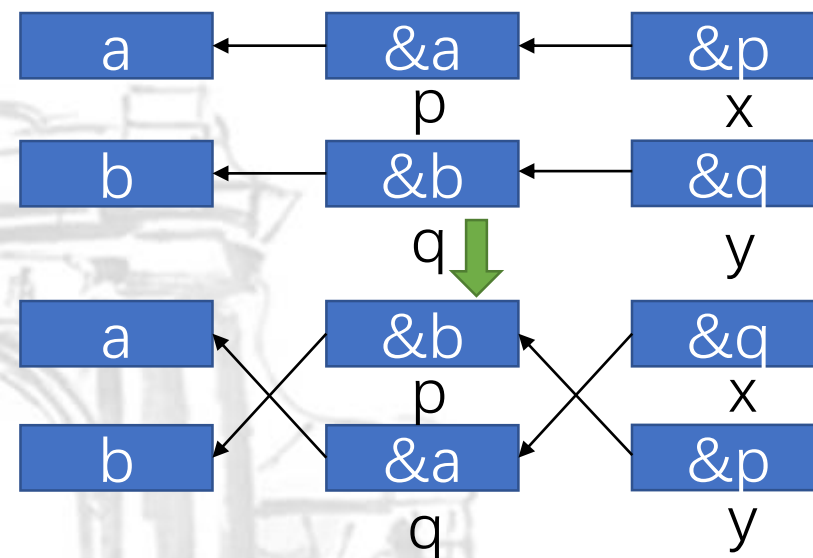
指针：指向指针的指针



●例16-29：阅读以下程序

```
1 #include <stdio.h>
2 void main() {
3     int a=12, b=34, *p, *q, **x, **y;
4     int *m, **n;
5     p = &a, x = &p;
6     q = &b, y = &q;
7     n = y, y = x, x = n;
8     m = p, p = q, q = m;
9     printf("%d %d %d %d %d %d\n", **x, **y, *p, *q, a, b);
10 }
```

12 34 34 12 12 34
请按任意键继续……



面对这种问题，可以像右边一样通过框图的方式来理解，方块旁标注变量名称，方块内部标注变量数值，箭头表示指针变量指向的变量

数据结构\指针与数组



知识点	使用示例	说明
指针数组	<code>int *a[10];</code>	每个元素均为 指针类型数据的数组
一维数组名	<code>int a[5];</code> 有 <code>a=&a[0]</code>	一维数组名实际上是对应类型的 常量指针
指向一维数组元素的指针	<code>int a[5],*p=&a[1]</code> , 则 <code>p[1]</code> 、 <code>*(p+1)</code> 、 <code>*(a+2)</code> 、 <code>a[2]</code> 等价	指向数组元素的指针加一表示 指向下一个元素 ; 数组元素可用下标法或指针法表示
二维数组	<code>inta[2][2]={ {1,2},{3,4}};</code>	静态二维数组以行为主 线性存储 于内存
二维数组名	<code>int a[5][5];*(a+i)+j=&a[i][j];</code> <code>a[i][j] = *(*a+i)+j</code>	二维数组名也是指针, 但需分清 行指针 和 指向数组元素 的指针
一维数组作为函数参数	<code>void f(int *a){...};</code> <code>void f(int a[]){...};</code>	掌握形参和实参分别用 数组名 和 指针变量 时的各种等价写法
数组名作实参, 指针作形参	<code>void f(int *b){...};</code> <code>f((int *)a);</code>	使用时需将二维数组名 强制类型转换 为 指针变量 或 行指针变量
指针数组作为实参	<code>void f(int **b){...};</code> <code>b[i]=&a[i][0]</code>	将 指针数组元素 分别指向对应行首地址 75

指针：数组指针作为函数参数



●例16-30：主函数定义了一个 5×4 的矩阵，asd()函数对其赋值

□二维数组在内存中以行为主的方式一维顺序排列存储

□可以将形参设置为一维数组的方式，同时对下标做相应转换

```
1 #include <stdio.h>
2 void asd(int *b, int m, int n) {
3     int k=1, i, j;
4     for (i=0; i<m; i=i+1)
5         for (j=0; j<n; j=j+1) {
6             b[i*n+j]=k;
7             k=k+1;
8         }
9 }
```

```
10 void main() { //行指针强制类型转换
11     int i, j, a[5][4];
12     asd( (int *)a, 5, 4);
13     for (i=0; i<5; i++) {
14         for (j=0; j<4; j++)
15             printf("%5d", a[i][j]);
16         printf("\n");
17     }
18 }
```

```
1  2  3  4
5  6  7  8
9 10 11 12
13 14 15 16
17 18 19 20
请按任意键继续.....
```

指针：二维数组行指针作为函数参数



●例16-31：二维数组行指针对于矩阵赋值

□同样的，也可以将形参设置成二维数组行指针的形式

□这样的缺点是需要知道矩阵的列数，缺乏通用性

```
1 #include <stdio.h>
2 void asd(int (*b)[4], int m, int n) {
3     int k=1, i, j;
4     for (i=0; i<m; i=i+1)
5         for (j=0; j<n; j=j+1) {
6             b[i][j]=k;
7             k=k+1;
8         }
9 }
```

```
10 void main() {
11     int i, j, a[5][4];
12     asd( a, 5, 4);
13     for (i=0; i<5; i++) {
14         for (j=0; j<4; j++)
15             printf("%5d", a[i][j]);
16         printf("\n");
17     }
18 }
```

```
1  2  3  4
5  6  7  8
9 10 11 12
13 14 15 16
17 18 19 20
请按任意键继续.....
```

指针：指针数组作为函数参数



●例16-32：指针数组对矩阵赋值

□指针数组作为实参，每个元素指向二维数组对应行的首地址

```
1 #include <stdio.h>
2 void asd(int **b, int m, int n) {
3 //或void asd(int *b[], int m, int n)
4     int k=1, i, j;
5     for (i=0; i<m; i++)
6         for (j=0; j<n; j++) {
7             b[i][j]=k;
8             k++;
9         }
10 }
```

```
11 void main() {
12     int i, j, a[5][4], *b[5];
13     for (i=0; i<5; i++)
14         b[i] = &a[i][0];
15     /* 指针数组指向二维数组
16        每一行的第一个元素 */
17     asd( b, 5, 4);
18     for (i=0; i<5; i++) {
19         for (j=0; j<4; j++)
20             printf("%5d", a[i][j]);
21         printf("\n");
22     }
23 }
```

```
1  2  3  4
5  6  7  8
9 10 11 12
13 14 15 16
17 18 19 20
请按任意键继续.....
```


数据结构\动态内存



函数名	用法	说明
malloc()	(类型*) malloc(申请的字节数)	使用时需对申请后的内存进行 强制类型转换 ，赋值失败返回 NULL
calloc()	(类型 *) calloc(元素个数n, 类型占据字节数size)	在内存的动态存储区中分配n个长度为size的连续空间， 初始化为0 ，赋值失败返回 NULL 。能够申请比malloc() 更大的动态内存空间
realloc()	指针名 = (数据类型*) realloc(指针名, 新的内存长度)	在原先申请内存块的基础上，再重新申请一块更长的（或更短）的内存块，以实现内存块的 动态增长 ，同时 保留原数据
free()	free(指针名)	释放动态分配的内存，防止内存泄漏

指针：动态内存分配



●例16-33：用指向指针的指针实现行数与列数都不定的二维数组

```
1 #include<stdlib.h>
2 #include<stdio.h>
3 void main() {
4     double **p;
5     int m, n, i, j, k;
6     scanf("%d %d", &m, &n);
7     //为p开辟存放m个指向行的指针的内存空间
8     p = (double**)malloc(sizeof(double*)*m);
9     //为指向每一行的指针开辟存放n个变量的空间
10    for(k=0; k<m; k++)
11        p[k] = (double*)malloc(sizeof(double)*n);
12    //为了简明去掉了检测申请内存失败的代码
13    for(i=0; i<m; i++)
14        for(j=0; j<n; j++)
15            p[i][j]=n*i+j;
16    for(i=0; i<m; i++) {
17        for(j=0; j<n; j++)
18            printf("%3.0f", p[i][j]);
19        printf("\n");
20    } //释放内存的顺序与申请时相反
21    for(k=m-1; k>=0; k--)
22        free(p[k]);
23    free(p);
24 }
```

向函数中传递这种方式生成的动态二维数组的方式与传递用指针数组生成的二维数组的方式类似

```
3 4
0 1 2 3
4 5 6 7
8 9 10 11
请按任意键继续.....
```

数据结构\字符串与指针



知识点	示例	说明
字符串的表示：字符指针与数组	<pre>char s[20]= "Hello world" ; char *s = "Hello world" ;</pre>	二者初始化方法相仿，都能够用来作为 函数参数
字符指针自身可以修改	<pre>char *s = "Hello world" ; s= s+4;</pre>	修改之后的s值为 "o world"
字符指针所指向的字符串不能修改	<pre>char *s = "Hello world" ; s[3]= 'f' ;</pre> ✗	字符指针指向的字符串是 常量字符串
字符数组名无法修改	<pre>char s[20]= "Hello world" ; s = s+4;</pre> ✗	数组名是 常量指针 ，无法进行修改
字符数组中的值可以进行修改	<pre>char s[20] = "Hello world" ; s[1]= 'a' ;</pre>	修改后为 "Hallo world"
不能通过输入函数让字符指针变量指向一个字符串	各种错误示例见对应章节课件	根本原因：对于通过键盘输入的字符串，系统 不分配存储空间
字符串作为函数参数	<pre>void f(char *s){...} void f(char s[]){...}</pre>	字符数组和字符指针分别用作函数形参/实参

指针：紧凑型字符串数组



●例16-34：利用紧凑型字符串数组实现根据年月日判断星期几

```
1 #include <stdio.h>
2 void main() {
3     char *Weeks[ ]={"Sunday", "Monday", "Tuesday",
4     "Wendesday", "Thursday", "Friday", "Saturday"};
5     int year, month, day, weekday, i;
6     int months[12]={0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30};
7     scanf("%d%d%d", &year, &month, &day);
8     if (year%4==0 && year%100!=0 || year%400==0)
9         months[2]++;
10    for (i=2; i<12; i++)
11        months[i] += months[i-1];
12    year--;
13    weekday=year+year/4-year/100+year/400+months[month-1]+day;
14    weekday %= 7;
15    printf("%s\n", Weeks[weekday]);
16 }
```

2021 12 03

Friday

请按任意键继续……

普通年份除以7余1，闰
年除以7余2。且依据现
行历法倒推，公元元年
1月1日是礼拜一

数据结构\函数指针



知识点	示例	说明
指向函数的指针定义	类型标识符 (*指针变量名)();	最后的() 不能省略
函数指针的使用	(*函数指针变量名)(实参表) 或 函数指针变量名(实参表)	对函数指针变量运算 没有意义 的
函数指针数组	int (*p[5]) ();	使函数指针数组每个元素分别指向不同的函数, 减少代码冗余
函数指针变量作为函数参数	int root(double (*f())){...}	函数指针指向不同函数入口地址时, 在函数中可 调用不同的函数
返回指针值的函数	int* f(...){...}	函数可以返回 任何类型 的指针值, 指针函数通过 函数名 返回指针值
main函数的形参	main(int argc, char *argv[]) {...}	argv是一个 字符型指针数组 , 指向不同的字符串; argc的值是argv指向的 字符串个数+1

指针：用函数指针调用函数



- 例16-35：从键盘输入一个大于1的正整数n,当n为偶数时,计算 $1+1/2+1/4+\dots+1/n$ 当n为奇数时计算 $1+1/3+1/5+\dots+1/n$

n为偶数时计算值的函数even()：

```
1 double even(int n) {  
2     int k; double sum;  
3     sum=1.0;  
4     for (k=2; k<=n; k=k+2)  
5         sum=sum+1.0/k;  
6     return sum;  
7 }
```

n为奇数时计算值的函数odd()：

```
1 double odd(int n) {  
2     int k; double sum;  
3     sum=1.0;  
4     for (k=3; k<=n; k=k+2)  
5         sum=sum+1.0/k;  
6     return sum;  
7 }
```

指针：用函数指针调用函数



主函数常规方式调用函数

```
1 #include <stdio.h>
2 void main() {
3     int n, k;
4     double even(int), odd(int);
5     printf("input n;");
6     scanf("%d", &n);
7     if (n>1) {
8         if (n%2==0) /*n为偶数*/
9             printf("even=%9.6f\n", even(n));
10        else /*n为奇数*/
11            printf("odd=%9.6f\n", odd(n));
12    }
13    else printf("ERR!\n");
14 }
```

主函数通过函数指针调用函数

```
1 #include <stdio.h>
2 void main() {
3     int n, k;
4     double even(int), odd(int), (*p)(int);
5     printf("input n;");
6     scanf("%d", &n);
7     if (n>1) {
8         if (n%2==0) p=even;
9         /*n为偶数, 指针变量p指向函数even()*/
10        else p=odd;
11        /*n为奇数, 指针变量p指向函数odd()*/
12        printf("sum=%9.6f\n", (*p)(n));
13    }
14    else printf("ERR!\n");
15 }
```

二者的最终效果是等价的！

数据结构\结构体与联合体



类型	定义	说明
结构体	<code>struct 结构体类型名 {成员表};</code> <code>struct 结构体类型名 变量表;</code> <code>struct 结构体类型名 {成员表} 变量表;</code> <code>struct {成员表} 变量表;</code>	不同类型成员的 复合类型 , 结构体类型名定义的是 类型 , struct 不可省略; 结构体变量名.成员名 ; 结构体可嵌套; 可定义结构体数组,与函数结合参考数组
结构体指针	<code>p=&st1: st1.num, (*p).num,</code> <code>p->num, p[0].num</code>	指向结构体类型 变量或数组(或数组元素) 的起始地址, " -> "被称为 指向运算符
联合体	<code>union 联合体名 {成员表};</code> 引用: 联合体变量名.成员名	各种不同数据 共用同一段存储空间 , 按 最大成员 分存储空间, 成员首地址相同
枚举类型	<code>enum 枚举类型名 {枚举元素列表};</code> <code>enum 枚举类型名 {枚举元素列表} 变量表;</code> <code>enum {枚举元素列表} 变量表;</code>	变量的值 只限于列举出来的值的范围 内; 可以给枚举量 赋值 ,还可 整型值强制转换

自定义类型: **typedef 原类型名 新类型名;** 增加了一个**类型别名**,而没有定义新的类型

链表: 每个存储结点包含**数据域**和**指针域**,可进行链表**查找、插入、删除、逆转**

结构体：结构体变量的定义



● 例16-36：定义"日期"结构体类型，并引用内部成员

```
1 #include <stdio.h>    //包含头文件"stdio.h"
2 struct date {
3     int year, month, day;
4 };
5 void main() {          //定义main函数，这是程序的主体
6     struct date a={2017, 12, 8};    //定义date型变量a
7     printf("%02d/%02d/%d\n", a.month, a.day, a.year);
8 }
```

12/08/2017
请按任意键继续……

结构体：结构体指针作为函数参数



● 例16-37：结构体指针作为函数参数

```
1 #include <stdio.h>    //包含头文件"stdio.h"
2 #include <string.h>    //包含头文件"string.h"
3 struct student {
4     int num; char name[10]; char gender; int age; float score;
5 };
6 void change(struct student *t) {
7     printf(" t=%4d%6s%2c%3d%6.2f\n", t->num, t->name, t->gender, t->age, t->score);
8     t->score=95.0;
9     printf(" t=%4d%6s%2c%3d%6.2f\n", t->num, t->name, t->gender, t->age, t->score);
10 }
11 void main() {
12     struct student st={101, "Zhang", 'M', 19, 89.0}; //定义student型变量st
13     printf("st=%4d%6s%2c%3d%6.2f\n", st.num, st.name, st.gender, st.age, st.score);
14     change(&st);
15     printf("st=%4d%6s%2c%3d%6.2f\n", st.num, st.name, st.gender, st.age, st.score);
16 }
```

```
st= 101 Zhang M 19 89.00
t= 101 Zhang M 19 89.00
t= 101 Zhang M 19 95.00
st= 101 Zhang M 19 95.00
请按任意键继续.....
```


结构体：链表定义



● 链表一般结构

- 由多个存储节点构成，每个存储节点包含数据域和指针域，分别存放数据元素和下一结点元素的地址
- 可以用如下示意图表示



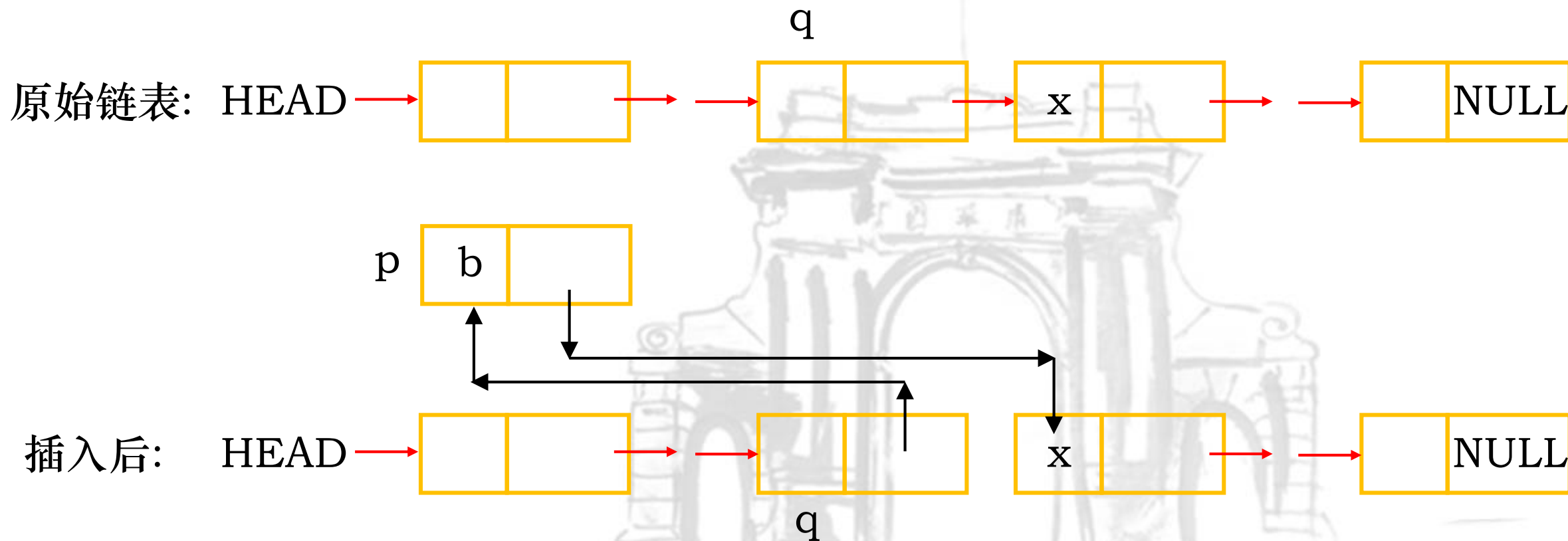
□ 逻辑结构



结构体：链表插入元素



● 链表插入元素示意图



结构体：链表插入运算



● 例16-38：链表插入元素

```
1 struct node {
2     ET data; struct node *next; };
3 void inslst(struct node ** head, ET x, ET b) {
4     struct node *p,*q;
5     p=(struct node *)malloc(sizeof(struct node)); //申请一个新结点
6     if(p==NULL) {
7         printf("can't get memory!\n"); exit(1); }
8     p->data=b; //置结点的数据域
9     if(*head==NULL) { //链表为空
10         *head=p; p->next=NULL; return; }
11     if ((*head)->data==x) { //在第一个结点前插入
12         p->next=*head; *head=p; return; }
13     q=lookst(*head, x); //寻找包含x的前一个结点q
14     p->next=q->next; q->next=p; //结点p插入到q之后
15     return;
16 }
```

结构体：链表删除元素

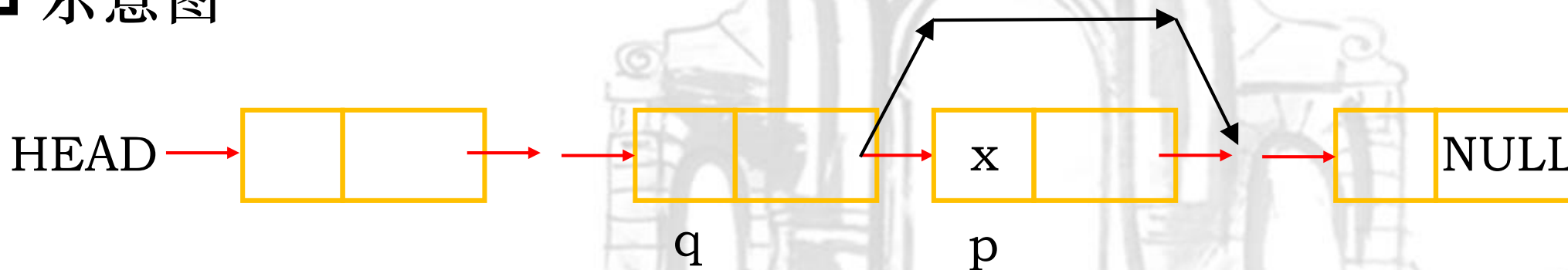


● 链表删除元素举例

□ 将结点q后的结点p从链表中删除,即让结点q的指针指向包含元素x的结点p的后一个结点,即令 $q \rightarrow next = p \rightarrow next$;

□ 将包含元素x的结点p释放

□ 示意图



结构体：链表逆转



- 链表逆转

- 依次遍历并逆转

- 逆转链表示意图



联合体：联合体的定义



● 例16-39：联合体的使用

```
1 #include <stdio.h>    //包含头文件"stdio.h"
2 union EXAMPLE {
3     struct {
4         int x, y;
5     } in;
6     int a, b;
7 } e;
8 void main() {
9     e.a=1;
10    e.b=2;
11    e.in.x=e.a*e.b;
12    e.in.y=e.a+e.b;
13    printf("%d, %d\n", e.in.x, e.in.y);
14 }
```

4, 8

请按任意键继续……

- ❑ 对于联合体e，实际上是a、b、in共享内存
- ❑ 进一步，因为in中有x和y，实际上是a、b、in.x共享内存
- ❑ in.y的内存单元在in.x之后
- ❑ 在执行e.a=1; e.b=2;后，实际上a和b的值都是2
- ❑ 执行e.in.x=e.a*e.b; 之后，a和b和in.x的值都变成4

枚举类型：枚举类型的定义



- 例16-40：根据键盘输入的一周中的星期几（整数值），输出其英文名称

```
input n:2
Tuesday
请按任意键继续.....
```

```
1 #include <stdio.h>    //包含头文件"stdio.h"
2 void main() {
3     enum week{sun ,mon, tue, wed, thu, fri, sat} weekday;
4     int n; printf("input n: "); scanf("%d",&n);
5     if ((n>=0)&&(n<=6)) {
6         weekday = (enum week)n;
7         switch(weekday) {
8             case sun: printf("Sunday\n");    break;
9             case mon: printf("Monday\n");    break;
10            case tue: printf("Tuesday\n");    break;
11            case wed: printf("Wednesday\n"); break;
12            case thu: printf("Thursday\n");   break;
13            case fri: printf("Friday\n");     break;
14            case sat: printf("Saturday\n");   break;
15        }
16    }
17    else printf("ERR\n");
18 }
```

●下面给出几道考试样题和输出，请同学们自行复习

```
1 #include <stdio.h>
2 void main() {
3     int a[3][3] = { 1, 3, 5, 7, 9, 11, 13, 15, 19 };
4     int i, j, sum1 = 0, sum2 = 0;
5     for (i = 0; i < 3; i++)
6         for (j = 0; j < 3; j++)
7             if (i == j) sum1 += a[i][j];
8     for (i = 0; i < 3; i++)
9         for (j = 2; j >= 0; j--)
10            if (i + j == 2) sum2 += a[i][j];
11     printf("%d %d\n", sum1, sum2);
12     return;
13 }
```

考察：二维数组，循环与分支结构

sum1计算二维数组对角线和
sum2计算反对角线和

输出：29 27

实战演练



```
1 #include <stdio.h>
2 #include <string.h>
3 void main() {
4     char st[21] = "hahaa\0\t\' \\'";
5     printf("%d,%d,%s\n", strlen(st), sizeof(st), st);
6     return;
7 }
```

考察：字符串及字符串函数，strlen结果不包含结束符' \0'，sizeof计算整个数组大小

输出: 5,21,hahaa

```
1 #include <stdio.h>
2 void main() {
3     char a[4][5] = { "DCBA", "EFGH", "IJKL", "MNOP" };
4     int i;
5     for (i = 0; i < 4; i++)
6         printf("%c", (*(a + i) + i));
7     return;
8 }
```

考察：二维字符数组，数组元素的指针法表示
 $*(a + i) + i$ 等价于输出 $a[i][i]$

输出: DFKP



```
1 #include <stdio.h>
2 int f(int a) {
3     static int c = 2;
4     int b = 5;
5     b = b + 1; c = c + 1;
6     return a + b + c;
7 }
8 void main() {
9     int i;
10    static int a = 2;
11    for (i = 0; i < 3; i++) printf("%d ", f(a++));
12    return;
13 }
```

考察：静态变量
函数第一次调用 $a=2, b=6, c=3$
函数第二次调用 $a=3, b=6, c=4$
函数第三次调用 $a=4, b=6, c=5$
因为 c 是**静态变量**，在函数调用结束后依然**保持**
可以在编译器中加入**断点调试**，查看各变量的值，理解程序的运行过程

输出: 11 13 15

实战演练



控制台输入: how do you do+回车

```
1 #include <stdio.h>
2 int main() {
3     char str1[] = "how do you do";
4     char str2[20], str3[20], *p1, *p2, *p3;
5     p1 = str1; p2 = str2; p3 = str3;
6     scanf("%s", p2);
7     gets(p3);
8     printf("%s ", p2);
9     printf("%s", p1);
10    printf("%s\n", p3);
11    return 0;
12 }
```

str1被赋值” how do you do” ,
scanf()读到**空格停止**，因此str2被赋值 “how” ,
此时**输入缓冲区**内剩余
“ do you do” , 因此
str3被赋值 “ do you do”
(注意最前面有个空格字符)

输出: how how do you do do you do

THANKS