进制转换

原码，补码，反码

1. 计算机中，正数原码，补码，反码的表示是一样的。只有负数不一样，可以粗略地认为是处理负数问题。
2. 比如-1，的表示形式是由原码(10000001)，转换为反码(11111110)，转换为补码(11111111)存储的
3. 反码：除符号位以外，其他位都取反
4. 补码：转换为反码后对反码加 1 得到补码
5. 二进制的正负：二进制数第一位为符号位，正时为 0，为负时为 1

进制的标准化呈现方式

- 二进制：0b 或 0B(不区分大小写)开头，**二进制不满 8 位时，要用 0 补位。二进制只要首位为 1，那就是负数**。比如：0b101
- 八进制：以 0 开头。比如：-101，01777
- 十进制：和平时一样，不需要任何前缀
- 十六进制：0x 或 0X（不区分大小写）开头。比如 0x1e2c

进制转换总结

- 十进制转其他进制，都是除以要转换的进制数然后整数部分取余倒序组合，小数部分乘以要转换的进制数，保留整数部分，拿前面剩下的小数部分不断取整数部分，正序组合
- 其他进制转十进制，都是对每个数拆开，然后每个数乘以其他进制在当前位置的次方相加。
- 二进制其他进制，转十进制如上，转八进制/转十六进制都差不多，分别拆为 3,4 位数的二进制位，然后分别求其二进制位，从左到右组合
- 其他进制转二进制，十进制转二进制如上，八进制/十六进制进制都是每个位数分割，然后分别求 3,4 位的二进制位，不够再补码，然后从左到右组合
- 口诀：**二进制转其他的，以二进制为准，其他进制转十进制的，以其他进制为准**

十进制和二进制

// 十进制 0-9 二进制0-1 八进制0-7 十六进制0-9和A-F 即0123456789ABCDEF

/*
十进制转二进制(倒序组合)
如：4 4/2=2余0 取出0(看下面) 2/2=1余0 取出0 1/2无法取余 取出1
  然后倒序排列得100

二进制转十进制，83为例（二进制首位为1，则为负数）
1.83的二进制为1010011，以8个二进制位为基准，进行补码后为01010011
计算 0x2^7+1x2^6+0x2^5+1x2^4+0x2^3+0x2^2+1x2^1+1x2^0 = 83
2.-83的二进制，“先取反，再加1”，83的二进制为01010011，取反后为10101100，再加1位10101101

(负数)以上面-83转换后的二进制为例子。
1.10101101转二进制，“先取反，再加1”，取反后为01010010，加1后为01010011，然后-(0x2^7+1x2^6+0x2^5+1x2^

*/

十进制和十六进制

/*

十进制转十六进制
原理：十进制小数转换成十六进制小数采用“整数除16取整，倒序输出，小数乘16取整，正序输出”法。
如 27

整数部分 27/16=1 余11 取出B 1/16无法取余 取出1 则整数部分得1b
所以27.21的十六进制为0x1B

十六进制转十进制(0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)
比如：1B，个位B代表11，十位数1代表1，就等于1x16^1+11x16^0 = 16+11 = 27

*/

十进制和八进制

/*
十进制转八进制
原理：十进制整数转成八进制的小数采用 “整数除8取整，倒序输出，小数乘8取整，正序输出”法。
比如：求200.44的八进制（省略3位小数）
整数部分200/8 = 25 余0，25/8 = 3 余 1，,3无法除以8，倒序组合整数部分就是 310
小数部分0.44*8 = 3.25 取出3，0.25*8 = 4.16 取出4，0.16*8 = 1.28，取出1，正序组合小数部分就是341
所以200.44的八进制就是310.341

八进制转十进制
比如：求310的八进制
310 = 3*8^2+1*8^1+0*8^0 = 192+8 = 200
*/

二进制和十六进制

/*

二进制转十六进制(每4个拆分,2^4=16，正序求值组合)
如：01010011(也就是十进制的83)转十六进制，
按每4个进行拆分，也就是0101和0011，然后0101 = 1x2^0+1x2^2 = 5，0011 = 1x2^0+1x2^1 = 3
组合后就是53，所以01010011的十六进制就是53

十六进制转二进制
如上面的53，把5和3分别拆开，分别求每个数的4位bit二进制数，5的二进制为101，补满4位后为0101
3的二进制为11补满4位后为0011，两个组合起来就是01010011
*/

二进制和八进制

```
/*
二进制转八进制(每3个拆分,因为2^3=8,不足按3个进行补码，正序求值组合),
比如01010011(也就是十进制的83)转八进制,
按每3个进行还分，也就是001,010,011，然后001 = 1x2^0 = 1，010 = 1x2^1 = 2，011 = 1x2^1+1x2^0 =3
所以01010011的八进制就是123


八进制转二进制(每个数拆开，按3位取二进制，不足补码)
如上面的123，把每个数拆开，也就是1,2，3,分别求每个数的3位bit二进制数,
1的二进制为1，补码后为001,2的二进制为10，补码后为010,3的二进制为11，补码后为011,
所以8进制的123转换成二进制后为001010011
*/
```

01.程序设计与 C 语言

```c
#include <stdio.h>
int max(int x, int y) // 函数首部
{
  int z; // 声明部分
  /*执行部分*/
  if (x > y)
  {
    z = x;
  }
  else
  {
    z = y;
  }
  return z;
}
```

基本概念

基本概念

```
char c1 = '\'';       //后面接一个特殊字符
char c2 = '\110';       //后面接一个三位的八进制数
char c3 = '\x0d';       //后面接一个两位的十六进制数，最大值127
```

重要概念

1. **C 语言主要是由函数构成，函数是 C 语言的基本单位**
2. C 语言的函数由
    i. 函数首部
    ii. 函数体。包括**声明部分**和**执行部分**

```
/*
一个函数由两部分组成
1.函数首部
 int     max   (int     x, int  y)
类型名  函数名    参数类型 参数名
2.函数体，分为声明部分和执行部分
*/
```

3. 一个 C 语言程序总是从 main 函数开始执行的。不管 main 函数在头或者尾
4. 每个语句和数据声明的最后必须有一个分号(😜
5. C 语言本身没有输入输出语句。输入输出由库函数 scanf 和 printf 等函数来完成
6. 可以用//对语句进行注释
7. 程序的概念：一组计算机能识别和执行的指令，一个 C 语言源程序文件中包括 3 个部分：预处理指令，全局声明和函数定义。
8. 上机运行一个 C 程序必须经过 4 个步骤：**编辑，编译，连接和执行**。
9. 源程序：用高级语言编写的程序，计算机只能识别由 0 和 1 组成的二进制指令(机器指令)，所以需要将源程序通过**编译程序**编译成二进制形式的目标程序，再将该目标程序与系统的函数库以及其他目标程序连接起来，形成**可执行的目标程序**
10. 高级语言的编译过程：源程序(f.c)->通过编译程序编译成->二进制形式的目标程序(f.obj)->将目标程序与系统函数库以及其他目标程序连接->可执行的目标程序(f.exe)
11. 机器语言：一个型号机器语言的指令的集合。也叫低级语言。只能识别 0 和 1
12. 高级语言：无法直接识别和执行二进制指令。
13. 程序：一组计算机能识别和执行的指令。

输入和输出打印

1. 输出：printf("a=%d")

```
/*
参考链接：http://www.manongjc.com/detail/62-mtyzthbdovmstil.html
print函数中自增/自减运算符：在printf中，运算规则变为从右向左，输出规则为从左向右，容易产生一些问题

比如：
int i=1;
printf("%d====%d====%d",i++,++i,++i);

打印出的是3====4====4
步骤：

先计算，从右到左
1. ++i; i = 2
2. ++i; i = 3
3. i++; i = 3，再运算i= i++ =4

再输出，从左到右
i++ // 运算之前，i=3，所以输出3，
++i // 此时i=4,输出当前值 i=4
++i // 此时i=4,输出当前值 i=4
其实在运算过程中，遇到i++这种需要先赋值后运算的情况，编译器会将运算前的值存储在寄存器中，以便在运算完成之

*/
```

2. 输入: scanf("%d", &a)，**scanf 是以空白符(空格、制表符、换行等等)为结束标志的,当遇到空白符是就会结束一次输入**

02.程序的存储和运算

位，字节和地址

1. 位(bit)：二进制位，是存储信息的最小单位，值为 0 或 1
2. 字节(byte): 一个存储器包含很多二进制位，用位来管理很不方便，所以将 8 个二进制位组成一组成为字节
3. 地址：计算机存储器包含很多存储单元，操作系统把所有存储单元以字节为单位编号，这就是存储地址

不用类型数字的存储方式

1. 整数的存储方式

一个十进制整数会转换成二进制数存放在存储单元之中，一个字节有八个二进制位，首位用来标识正负，剩余 7 个位数用来表示数字即(2^7)-1，即十进制的 127。但是这样无法满足超过 127 的其他整数的要求，所以现在使用都用 4 个字节来表示一个整数，最大整数约为 21 亿**(负数补码看下面)**

2. 实数的存储方式

对于实数，如(123.456),一律采用指数形式存储，会被转化为标准化指数形式 0.123456*10^3，即(小数点前面是 0，小数点后的第一位数字不是 0)，在计算机中一般以 4 个字节存储一个实数，其中以 3 个字节存储数值部分(包括数符)，以 1 个字节存放指数部分(包括指数符号)

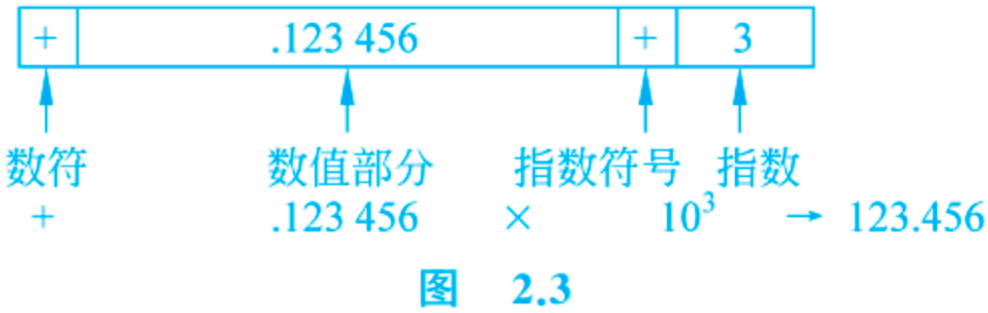


图 2.3

3. 字符的存储方式

字符包括(A,a)，专用字符(@,;)等。计算机并不是将字符本身放到存储单元中(因为存储单元只能存放二进制信息)，而是将字符的**代码（ASCII 码）**存储到响应的存储单元中，比如 A 在 ASCII 码中代表 65，65 的二进制是 1000001，补位后为 01000001

整型常量的表现形式

1. 十进制整数。如 123,-456，4。
2. 八进制整数，在程序中凡是以 0 开头的数都被认作八进制数

```
// 八进制数如：0123,0345
// 但是 078这是是错误的，因为八进制的数最高只到7
```

3. 十六进制整数，逢 16 进 1，在程序中以 0x 开头的数都被认作十六进制数，

```
// 0x123 如何表达十进制呢?
0x123 = 1x16^2+2x16^1+3x16^0=291 // 123代表十进制数291。
// 0x2a
0x2a = 2x16^1+10x16^0 = 32+10 = 42 // 2a代表十进制42
-0x2a = -(2x16^1+10x16^0 = 32+10 ) = -42 // -2a代表十进制-42
// 0xfee
0xfee =  15x16^2+14x16^1+14x16^0= 4078 // fee代表十进制4078
```

常量和变量

1. **常量**：在程序运行过程中其值不能改变的量
2. **变量**：在程序运行过程中其值可以改变的量

变量

1. 变量名：代表一个变量地址，通过地址可以找到存储单元，可从改存储单元读取其值。
2. 变量名的取名规则：**变量名的第一个字符必须是字母或下划线，其后必须是字母，数字或下划线**
3. **要弄清楚变量名,变量地址,存储单元、变量的值的相互关系,提到变量名,就马上想到它代表一个变量地址,通过这个地址可以找到相应的存储单元,可向该存储单元存放一个值,或者从该存储单元读取其值。**

变量的种类

1. 整型变量 int: 一般占据 4 个字节即 2^-31 到 2^31 -1 , 如果需要改变变量的字节数，可以定义**长整形 long int**或**短整形 short int**,此外(变量的值常常是正的，为了利用变量的数值范围，可以使整型比那辆的存储单元的第一位不用来表示数值符号，这样可以将存储的范围扩大一杯，即在前面加**unsigned**)
2. 实型常量 float(浮点数: 分为十进制小数形式(0.123),-50,10 不算，这种为整型,指数形式(123x10^3,在计算机中用 123e3 或 123E3 代表，但字符 e 之前必须需有数字，且 e 后面的指数必须为整数))。**在计算机中一般以 4 字节存储一个实数。这 4 字节可分成两个部分:一般以 3 字节存放数值部分(包括数符),以 1 字节存放指数部分(包括指数的符号)**

```
// 实型(浮点数)常量的表现形式
// 1. 十进制小数
0.123,123.23
// 指数形式 以3.1415926为例
0.314159e1
```

实型变量分 3 类，单精度(float 占 4 字节(10^-38 到 10^38)，有效位数 7 位),双精度(double 占 8 字节 10^-308 到 10^308，有效位数为 15 位)，长双精度(long double 站 8/16，可以用 sizeof 方法测试)，可以使用 sizeof 方法查看所占字节数

```
// 实型数据的舍入误差
float a = 3.14159268723;
printf("%f", a); // 3.141593 由于只能提供7位有效数字(第七位2还不是精确的)，所以3.14159后面的小数并不是
```

```
// 为了提高数据精度，可以将变量定义为双精度类型。
```

3. 字符型变量 char, 只允许存储一个字符(非字符串)，如果 char a ="a"也是错的，因为字符串"a"实际上有 2 个字符 "a\0"
    - 转义字符

```
// 字符数据和整型数据在一定条件下可以通用
// 下面两个表达的意思是一样的
char a  = 97
char a = 'a'
```

```
// 字符串和字符的差异
char a = 'a';
char a1[] = "a";
printf("%d\n", sizeof(a)); // 1
printf("%d\n", sizeof(a1)); // 2 实际上字符串后面还有个\0
```

符号常量

```
// #define不是C语句，它是一个预编译指令 符号常量不占存储单元
```

```
// 错误用法
// 不能对符号常量指定类型 如 double PI;
#define PI 3.1415;
// 不能对符号常量进行赋值
PI = 3.14;
```

算术运算符和算术表达式

基本算术运算符

1. 普通运算符+, -, *(乘), /(除), %(取余)
2. 自增自减运算法++i，--i，i++,i--
3. 算术表达式(算术运算符将算术对象相连接，如 i*j/c-1.5*sin(x)+'m'这种)。
   在进行运算时，不同类型的数据会转成同一类型
   总结：**字节少的数据转换成字节多的类型**
   i. char 和 short 型转换为 int 型
   ii. float 型一律转换为 double 型
   iii. 整形(int,short,long)数据和 double 型数据进行运算，先将整形转换为 double 型
4. 强制类型转换，使用 (类型名) 表达式 可以强制转换类型。会产生一个中间变量，原来的变量类型不会发生变化。例：(double)a
5. 按位与运算符(&)：参加运算的两个数，换算为二进制(0、1)后，进行与运算。只有当相应位上的数都是 1 时，该位才取 1，否则该为为 0。
   - 例如： 3&5 即 0000 0011 & 0000 0101 = 0000 0001 因此，3&5 的值得 1。
6. 左右移运算符：左移<<为乘，右移>>为除，移数比如 8>>1 = 8/2, 8>>2 = 8/4，即所余 n 为 2^n

第二章提高部分

整型变量类型的取值范围

0. char: 1 字节
1. int：2 字节[-2^15 - (2^15-1)]/4 字节[-2^31 - (2^31-1])
2. unsigned int：2 字节[0 - (2^16-1)]/4 字节[0 - (2^32-1])
3. short：2 字节 [-2^15 - (2^15-1)]
4. unsigned short：2 字节 [0 - (2^16-1)]
5. long：4 字节[-2^31 - (2^31-1])
6. unsigned long：4 字节[0 - (2^32-1])
7. long long：8 字节()[-2^63 - (2^63-1])
8. unsigned long long：[0 - (2^64-1])
9. float：4 字节
10. double：8 字节

C 语言中的数据类型

1. 基本数据类型：整型(字符，布尔，整数)，浮点型(小数)
2. 枚举类型 enum
3. 空类型 void
4. 派生类型(数组，指针，结构体，共用体，函数)

整型常量类型的取值范围

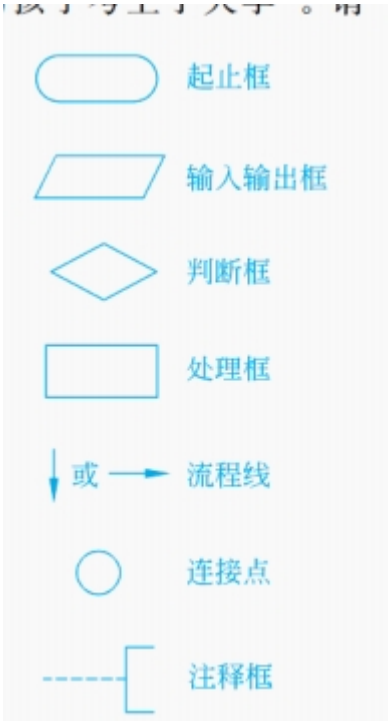
1. 取值范围按照值的大小自动分配
2. 在一个常量后加 l 或 L，则认为是 long int 常量，如 432L，
3. 常量后面加字母 u 或 U，则认为是 unsigned int

03.顺序程序设计

第三章 顺序程序设计

程序，数据结构与算法

1. 程序：一组计算机能识别和执行的指令(**第一章有定义**)
2. 程序的组成：
    - 对数据的描述：即**数据结构**
    - 对操作的描述：即**算法**
3. 算法的作用：解决做什么和怎么做的问题
4. 算法的类别：**数值运算算法**(比如：求圆的面积)和**非数值运算算法**(比如：求某个表的排序)
5. 表示算法的方式：自然语言，流程图，N-S 流程图，伪代码



ASCII 码特殊位置

1. 0-9：48-57
2. A-Z：65-90
3. a-z：97-122

程序的三种结构

1. 顺序结构
2. 选择结构，又称判断结构或分支结构，比如 if(a>b)
3. 循环结构

由以上三种基本结构构成的程序称为**结构化程序**

C 语言语句

1. 控制语句
    - do...while();
    - for()....
    - while()....
    - if()...else...
    - switch
    - continue 结束本次循环
    - break 中止执行 switch 或循环

- return
- goto 转向

2. 函数调用：函数调用语句由一个函数调用加分号构成
3. 表达式语句：表达式语句由一个表达式加一个分号构成
4. 空语句：；
5. 复合语句：用{}把一些语句括起来称为复合语句

赋值语句和赋值表达式

3. 区别：赋值表达式的末尾没有分号，赋值语句的末尾必须有分号在一个表达式中可以包含一个或多个赋值表达式，但决不能包含赋值语句。

```
// if里面可以包含赋值表达式，但不能包含赋值语句
if((a=b)>0) t=a; // 合法
if((a=b;)>0) t= a; // 不合法
```

不同数据类型之间赋值

1. 浮点数赋值给整型变量，先对实数取整(舍去小数部分，然后赋予整型)

```
int i;
float b = 3.1523;
i = b; // 3
```

2. 整形数据赋给浮点数单精度或双精度，数值不变，以浮点数形式存储到变量中

```
int i = 23;
float b = i; // 23.000000
double b = i; // 23.00000000000000
```

3. double 赋值给 float，截取前面 7 位有效数字，存放到 float 变量的存储单元(4 字节)中。将 float 赋值给 double 时，数值不变，有效位数扩展到 16 位，在内存中以 8 字节存储

```
double d = 123.456789e100;
float f = d; // 1.#INF00
```

4. 字符型数据赋予整形变量时，将字符的 ASCII 码赋值给整形变量

5. 将占字节多的整型赋值给占字节少的整型或字符变量，只会将低字节原封不动低送到该变量(即发生截断)

```
int i = 289;
char c = 'a';
c = i; // 打印出 !(感叹号) 289的二进制为0000000100100001，将其截断后为00100001转换为二进制后为33，33
```

6. 总结，**字节大的赋值给小的，做截断，字节小的赋值给大的，做增加**

数据输入输出的概念

1. **输入输出是以计算机为主体而言的**，
2. **C 语言本身不提供输入输出语句**，输入和输出操作是由 C 函数库中的函数来实现的
3. 在使用系统库函数时，应当在程序中使用预编译指令"**#include**"， <stdio.h>是将有关的文件内容包括到用户源文件中，这种文件被称为**头文件**

```
// 头文件的两种形式
#include <stdio.h> // 标准形式(直接找到C编译系统所在子目录，一般都是标准模式)
#include "stdio.h" // 其他形式(先在用户当前目录中找，找不到再按标准形式找)
```

4. 字符数据的输入输出：putchar 输出字符, getchar 输入字符
5. 简单格式的输入输出：printf 格式输出函数，格式输入函数
    - printf：%d,%i 输出十进制整型，%c 输出字符，%s 输出字符串，并，%f 输出实数，**编译器对 float 类型默认输出 6 位小数** %lf 输出长实数，即双精度数据，%e 输出指数形式的实数
    - scanf(格式控制，地址表列)。解析，如 scanf("%d%d%d", &a, &b, &c);，其中"&"是地址运算符，&a 指变量 a 在内存中的地址。**注：输入时可以以空格，Enter 或 Tab 键分隔输入的数据，但是不能用逗号**
    - scanf 如果读取的是数字，那么在读入数字时会忽略空格(字符) 和 换行符(enter 或'\n')
    - scanf 如果 读入的是 字符 就会把空格，换行，当成一个字符
6. printf 修饰符：使用(%m.nf)，m 必须大于 n
7. 专业名词：
    - 格式控制：scanf 函数和 printf 函数中双撇号中的部分
    - 格式声明：由%和格式字符组成。如%d,%c,%7.2f
    - 格式字符：用来指定各种输出格式。如 d,c,f,e,g 等
    - 附加格式字符(也称修饰符)，对格式字符的作用作补充说明，如%3d,%7.2f
    - %%输入一个%，%lf 输入 double 类型，%f 输入 float 类型,%o 输入 8 进制，是字符 o 不是数字 0
8. 固定宽度的输出

- %mc：即 c 字符宽度为 m 列
- %ms：即 s 字符串占 m 列

```
printf("%e", 1235.2321); // 1.235232e+003
```

提高部分

无符号和有符号之间的赋值

1. 有符号整数赋值给长度相同的无符号整形变量时，按字节原样赋值(连原有的符号位也作为数值一起传送)

```
unsigned short a;
short int b;
b = -1;
a =b; //65535

// 解答
-1转换成二进制进制，-1补码后为0000000000000001(short占2字节)，首位为0，负数情况下首位为0，则取反为1111
由于a是无符号变量，第一位不代表数值的符号，16位都用来表示数值，所以他的值为65535
```

2. 将无符号整数赋值给长度相同的有符号整型变量

```
unsigned short int a;
short int b;
a = 65535;
b=a; //-1
```

输出数据时的格式控制

04.选择结构程序设计

C 语言的逻辑判断基础

1. 逻辑量：就是参加逻辑运算的变量、常量

2. 逻辑值：逻辑运算的值

   C 语言在表示一个逻辑值时(如关系表达，逻辑表达式的值)时，以 1 代表真，以 0 代表假，在判别一个逻辑量的值时，以 非 0 作为真，0 作为假
3. C 语言中主要用 if 语句实现选择结构，用 switch 语句实现多分支选择结构

运算符分类

1. 关系运算符(<,<=,>, >=优先级相同(高)，==,!=(低))
2. 算术运算符+,-,*,//
3. 赋值运算符 +=,-=, /=, *=, =
4. 逻辑运算符，!(非)，&&和|| => 只返回 0 或 1
5. 条件运算符 ?: => a > b ? a: b
6. 其中按计算优先级：!(非) > 算术运算符 > 关系运算符 > &&和|| > 条件运算符 > 赋值运算符
   - **记法：非自增减指针地址算术求余加减关系等于按位与与条件赋值逗号**
7. 其中按运算顺序
   - 从右到左：**非条件赋值**
   - 从左到右：排除上面的其他类型运算符

关系运算符和关系表达式

1. 关系运算符：用来比较大小的符号，看上面**运算符分类**
2. 关系表达式：诸如 x>0，a+b>c 等的表达式，包含了>和<等这样的比较符号的式子，这些式子的值不是数值，而是一个逻辑值("真"或"假")

学习 switch,if, if-else-if 语句

1. switch 语句
   - switch 后面的表达式，可以是数值型或字符型类型数据, 如果前面的每一个 case 都加了 break，default 阶段可以不用加 break;
   - switch 语句为什么要加 break，因为 case 只是起语句标号作用，看课后题第四章例子，如果第一个通过了，后面的 case 判断也会连续输出
2. if-else:
   - if 中每个内嵌语句必须以分号结束。
   - else 子句不能作为语句单独使用，它必须是 if 语句的一部分，与 if 配对
   - else 子句总是与它上面最近的未配对的 if 配对
3. 选择结构的嵌套：选择结构中又包含一个或多个选择结构，这称为选择结构的嵌套。

提高部分

条件表达式和条件运算符

1. 条件运算符(?😀
2. 条件表达式：max = a > b? a: b;

05.循环结构程序设计

循环结构

1. **循环结构就是用来处理需要重复处理的问题的。所以，循环结构又称为重复结构。**
2. 构成有效循环的条件：
    - 需要重复执行的操作，这称为**循环体**
    - **循环结束的条件**，即在什么情况下停止重复的操作
3. **循环的嵌套：一个循环体内由包含另一个完整的循环结构**
4. **内嵌的循环中还可以嵌套循环，这就是多层循环**
5. 循环的三种方式：while,do...while,for 循环，

while 语句

1. 特点：先判断表达式，后执行循环体。

do...while 语句

1. 特点：先执行循环体，然后判断循环条件是否成立。

for 语句

1. 结构：**for(表达式 1：循环变量赋初值;表达式 2：循环条件;表达式 3：循环变量增值) { 4 语句 }**, 其中：
    - 表达式 1 可以省略，但表达式 1 后面的分号不能省略。
    - 表达式 2 可以省略，如果表达式 2 省略，会出现无限循环
    - 表达式 3 也可以省略，但是要另外设法保证循环能正常结束；
    - 表达式 1,2,3 可以一起省略，省略后相当于 while(1)
    - 总结:**表达式 123 都可以省略，但是分号要保存**，其执行顺序为 1243
2. continue 情况下会执行表达式 3，break 情况下会直接中断掉

switch

1. 当 switch 命中后，命中字段没有 break;那会自动流入 default

特殊语句

1. break 语句：提前退出循环，**只能运用于循环语句和 switch 语句中**
2. continue 语句：结束本次循环

06.利用数组处理批量数据

数组介绍

1. 数组：**数组是有序数据的结合**，用一个统一的数组名和下标来唯一地确定数组中的元素

一维和多维数组的定义

1. 数组的命名规则和变量相同
2. 定义数组时需要指定数组中元素的个数；比如：int a[10];
3. **常量表达式不能包含变量，即数组大小不依赖于程序运行过程中变量的值**

```
int n;
scanf("%d", n);
int a[n]; // 不允许
```

4. **数组的下标可以是整型常量，也可以是整型表达式**

```
int a[2+1],int a[2*3]
```

5. **如果整形数组的长度超过默认初始化的值，那其他的非默认项则都为 0，如果是字符数组则为默认为'\0'**

```
// 一维数组的定义规范
// => 类型符 数组名[常量表达式]

// 整形
// 一维数组
// 定长 10代表数组长度
int a[10] = {1,2,3,4,6} // 长度为10，但是只有5个，则后5个的初值自动为0
// 自适应长度(根据内部定义的长度)
int a[] = {1,2,3,4,5,6} // 长度为6

// 二维数组的定义规范
// => 类型名 数组名[常量表达式][常量表达式]

// 二维数组的定义和初始化
float a[3][4], b[5][10] // 不能写成float a[3,4]
int a[3][4] = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
int a[3][4] = int a[][4] = {1,2,3,4,5,6,7,8,9,10,11,12};  // 会根据每4个为一组
int a[3][4] = {{1},{5},{9}}; // 对每一行的第一列赋值，其余数为0
// 获取二维数组值 a[1][2]

// 字符型
char a[10] = "shuhong"
```

字符数组和字符串的定义

1. C 语言中，将字符串作为字符数组来处理，C 语言设置了'\0'标志，C 语言在处理字符串常量时会自动加一个'\0'作为结束符，在内存中也会将这个'\0'标识一起存储在内存中。所以假如 char c[10] = "Happy"在内存中的长度是 6，但是 strlen 方法打印是 5，strlen 不含空结束字符
2. 如果字符数组的长度超过默认初始化的值，那未**赋值的部分元素值默认为空字符(\0)**
3. **如果初始化时的字符个数超过数组长度，则按语法错误处理**
4. printf，scanf 输入输出字符串时的注意项：
    - 输出字符不包括结束符'\0'
    - 如果数组长度大于字符串实际长度，也只输出到遇'\0'结束
    - 如果一个字符数组包含一个以上'\0'则遇到第一个'\0'输出就结束
    - 如果利用 scanf 函数输入多个字符串，则在输入时以空格分隔，**最后面不用加换行符\n**

```
char c[10] = { 'H', 'a', 'p', 'p', 'y' }; // strlen =5
char c[10] = {"Happy"} // strlen =5
char c[10] = "Happy" // strlen =5 与 char c[10] = { 'H', 'a', 'p', 'p', 'y', '\0', '\0', '\0', '
printf("%s", c);
```

数组方法

1. gets(char[])：从终端输入一个字符串到字符数组，输入成功返回字符串第一个字符的地址，输入失败则返回 NULL
2. puts(char[])：将一个字符串输出到终端
3. strcat(char1[]，char2[])：连接两个字符数组中的字符串，把字符串 2 接到字符串 1 的后面，定义时 char1[]长度必须足够大，连接时会将 char1[]后面的'\0'取消。**返回 str1**
4. strcpy(char1[]，str2)：将字符串 2 复制到字符数组 1 中去(直接替代)，**返回 str1**
5. strncpy(char1[]，str2, n)：将字符串 2 中的 n 个字符复制到字符数组 1 中去
6. strcmp(str1, str2)：从左到右按 ASCII 码大小，比较字符串 1 和字符串 2，**如果字符串 1 等于字符串 2，则返回值值为 0；如果字符串 1>字符串 2，则函数值为一个正整数，反之函数值为一个负整数**

```
/*
字符串比较从第一个字符开始比较，将每个字符转换成对应ASCII码
比如girl和goel，g和g比较ASCII码一直，转移到i和o比较，o比i大，所以girl > goel
*/
```

6. strlen(char[])：测试字符串长度，直到空结束字符，但不包括空结束字符。
7. strlwr(str)：将字符串中大写字母换成小写字母
8. strupr(str)：将字符串中小写字母换成小写字母

如何清空字符串和整形数组

```
// 清空字符串
char str[100] = "******";
str[0] = '\0';

// 清空整形数组 循环然后每个置为0
```

提高部分

字符数组的操作

1. 字符数组的赋值：
   - 不能用赋值语句将一个字符串常量或字符数组直接赋值给一个字符数组，只能用 strcpy 来赋值。如下不合法

```
// dsds
str1 = "china";
str = str2;
```

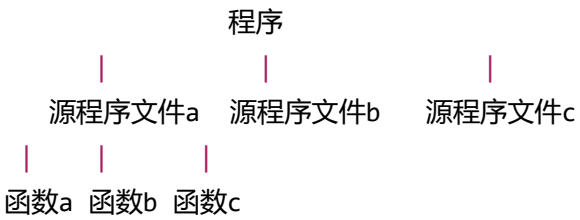
2. 字符数组所占大小

```
char c[] = "china"; // 定义为字符串形式的，长度会在其实际内容基础上加1，可用sizeof查看
char d[] = {'c', 'h', 'i', 'n', 'a'}; // 定义为字符数组的，长度为其实际内容长度

char str1[] = "ismyline";
char str2[8] = "ismyline";
int z= 4;
printf("%d\n", sizeof(str1)); // 9 字符数组不指定长度时，会自动加上\0，长度会从8变成9
printf("%d\n", sizeof(str2)); // 8 指定长度，输出8
printf("%d\n", sizeof(z)); // 4 int类型占4个字符
printf("%s\n", str2); // ismylineismyline 限定了个数后，因为会自动添加\0，突破边界，会造成怪异打印

// strlen为例，除了未指定长度，且字符分开初始化的，都是按实际长度
char str1[] = {'C', 'z'};  // 3
char str2[] = {"Cz"};      // 2
char str3[] = "Cz";        // 2
char str4[3] = {'C', 'z'}; // 2

// sizeof为例
// 如果字符分开初始化的，都是按实际长度或所定义的长度
// 如果字符没分开，则会在基础上加1
char str1[] = {'C', 'z'};  // 2
char str2[] = {"Cz"};      // 3
char str3[] = "Cz";        // 3
char str4[3] = {'C', 'z'}; // 3
```

07.用函数实现模块化程序设计

函数的特性

1. 从用户使用角度看，函数分为**库函数**和**用户自定义函数**
2. 从函数的形式看，函数分**有参函数**和**无参函数**
3. 程序结构图



4. 函数不能嵌套定义，函数可以互相调用，但是 main 函数不能互相调用。

函数定义

```
int Column; // 定义全局变量
static int Column // 静态全局变量，只允许当前文件引用，不允许外部文件引用;
int max(int x, int y) {
  if(x > y) return x;
  else return y;
}

int main()
{
  int max(int x, int y); // 函数声明形式1
  int max(int s, int b) // 函数声明形式2 改变参数名称
  int max(int, int) // 函数声明形式2 默认不适用参数
  printf("%d\n", max(3, 4));
}

// 函数定义的一般形式
/*
类型名 函数名()
{
   函数体
}
*/
// 调用函数
/*
 函数名()
 函数名(实参表列)
*/
```

函数调用过程(以上面函数定位为例)

1. 在定义函数中指定的形参，在未出现函数调用时，他们并不占用内存中的存储单元，在发生函数调用时，函数 max 中的形参被分配内存单元
2. 将实参对应的值传递给形参
3. 在执行 max 函数期间，由于形参已经有值，就可以进行有关的运算
4. 通过 return 语句将函数值带回到主调函数
5. 调用结束，形参单元被释放
6. **调用函数的方式**：
    - 函数语句：pinrt_star()
    - 函数表达式：c = 2 * max(a,b);

- 函数参数:printf("%d", max(a,b));

函数调用(普通调用)

重点：**函数形参是按值传递的(传整个数组除外，数组是按地址传递)，就算函数内改变原来参数的值，原来参数的值也不受影响（包括传数组某一项）**，

1. 函数定义：**指对函数功能的确立**，它是一个完整的，独立的函数单位。
2. 函数声明的作用：把函数名，函数参数的个数和参数类型等信息通知编译系统，编译系统能据此识别函数并检查函数调用是否合法，即**对函数调用的合法性进行检查。C 语言对形参数组的大小不做检查，因此形参数组可以不指定大小**
3. 函数声明的说明: 如果被调用函数的定义出现在 main 函数之前，那可以不加函数声明
4. 函数原型：就是**函数声明，函数声明也叫函数原型**

```
/**
函数调用过程范例:
void main() {
   int max(int, int); // 函数声明简写，不写参数名，只写参数类型
   int max(int u, int v) // 函数生命简写，参数名不用x,y，直接使用u,v
}
int max(int x, int y) { // 形参x,y不占内存中的存储单元
   if(x>y) {
      return x;
   } else {
      return y;
   }
}
1. 在定义函数中指定的形参，在未出现函数调用时，他们并不占内存中的存储单元，直到函数发生调用时，函数max中
2. 将实参对应的值传递给形参
3. 在执行max期间，由于形参已经有值，就可以进行有关的运算
4. 通过return语句将函数值带回到主调函数
5. 调用结束后，形参单元被释放
 */
```

函数调用(传整个数组情况)

1. 用变量名或数组元素名作函数参数时，传递的是变量的值，用数组名作函数参数时，传递的是数组首元素的地址。
2. 由于数组名代表数组首元素的地址，因此只是将数组的首元素的地址传输给对应形参
3. 函数调用时虚实结合的方式：**1. 值传递 2. 地址传递**

函数的嵌套调用和递归调用

嵌套调用：在调用一个函数过程中调用另外一个函数，称为函数的嵌套调用。
递归调用：在调用函数过程中直接或间接地调用该函数本身，称为函数的递归调用。

多维数组作为函数实参

1. 在被调用函数中对形参数组进行声明时，可以指定每一维的大小，也可以省略第一维的大小说明，但不能只指定第一行

```
// 合法
float fn(float arr[5][5])
float fn(float arr[][5])
// 不合法
float fn(float arr[][])
float fn(float arr[4][])
```

变量的作用域和生存期

1. 变量的作用域：指**变量的有效的范围**，分为局部变量和全局变量
2. 变量的生存期：指**变量存在的时间**
3. 内部变量/局部变量：**在函数或复合语句中定义的变量，只在本函数或复合语句内范围有效。**使用时才分配存储单元，不占用空间。也叫**动态存储方式**,函数内的变量，函数调用完释放回收空间
   - 主函数内的变量也是局部变量
   - 不同函数中可以使用相同名字的变量，他们互不干扰
   - 形参也是局部变量
   - 在函数内部可以在复合语句中定义变量，这些变量只在复合语句中有效
   - 复合语句：**将两条或以上的语句用{}括起来的语句**
4. 外部变量/全局变量：在函数外定义的变量，提前分配存储单元，占用空间。**全局变量的有效范围从定义变量的位置开始到本源程序文件结束，在此范围内可以为本程序文件中的所有函数所共用**，也叫**静态存储方式**，在函数外定义的变量，程序运行过程中都存在。
5. **C 语言程序组成**：一个程序可以包含**一个或若干个源程序文件，而一个源程序文件可以包含一个或若干个函数。**全局变量普通情况下只在一个文件内生效
6. 变量的可见性：一个变量在其作用域内可以被引用，则称此变量在作用域内"可见"
7. 变量的生存期：如果一个变量值在某一时刻是存在的，则认为这一时刻属于该变量的“生存期”,或称该变量在此时刻“存在”

```
// 全局变量定义
#include <stdio.h>
int Row, Column;

void main() {

}
```

变量的存储类别

1. 每一个变量和函数都有两个属性：**数据类型**和**数据存储类别**，定义变量时除了需要定义数据类型外，还可以指定定义其存储类别

变量的存储类别分类

1. auto: 声明自动变量，比如函数形参和在函数中定义的变量，在函数调用结束时就自动释放这些存储空间，所以这类变量被称为**自动变量**，auto 关键字可以省略。它属于动态存储类别

```
auto int b,c = 3;
```

2. static：声明静态变量，函数中的静态局部变量在调用结束后不消失而保留原值。该变量在整个程序运行期间都不释放空间(缩略图 239)，静态局部变量不赋初值时，会自动赋初值 0(对数值型变量)或空字符(对字符变量)，如果希望外部变量只限于被本文件引用，只能用于本文件的外部变量，称为**静态外部变量**
   - static 对局部变量：使变量由动态存储改变为静态存储方式。
   - static 对全局变量：使全局变量局部化，只能被本文件引用

```
static int f = 1; // 只初始化一次，在程序运行期间都不释放空间，不能被外部其他文件所引用
```

3. register：寄存器变量。一些频繁使用的变量(比如循环 10000 次)，每次循环都要引用局部变量，这种频繁操作的变量每次存储都会花费很多时间，为提高效率直接放 cpu 的寄存器中，增加存取速度

```
register int f;
```

4. extern：**声明外部变量的作用范围**，
   - 作用 1：**在一个文件内扩展外部变量的作用域**， 因为全局变量的作用域只在于从定义点到文件结束(全局变量不一定写在最头部)，这样会导致上面的函数引用不到全局变量(缩略图 241)
   - 作用 2：**将外部变量的作用域扩展到其他文件**

```c
// global.c
int globalVar = 10;

// main.c
#include <stdio.h>

extern int globalVar;  // extern关键字声明全局变量，引用global.c内的变量

int main() {
  extern int juan; // 拓展全局变量的作用域
  printf("Global variable: %d\n", globalVar);
  printf("%d\n", juan); // 2
  return 0;
}

int juan = 2; // 当前变量有效范围从该位置开始到本源程序文件
```

内部函数和外部函数

1. 内部函数：又称**静态函数**，只能被本文件中的其他函数所调用，不同文件中有同名的内部函数互不干扰，比如 static int fun(int a, int b);
2. 外部函数：定义时在函数处加 extern 的函数，可以被其他文件调用，**extern 可省略**，比如 extern int fun(int a, int b);

```c
// functions.c
#include <stdio.h>

void printMessage()
{
    printf("Hello, extern!\n");
}

// main.c

int main() {
  // 引用外部函数
  extern void printMessage();
}
```

做题中的易错点

1. 函数中不管有没有 return 语句，都会返回一个值，只不过没定义的时候这个返回值不确定，

08.善于使用指针

指针及相关概念

1. C 语言定义变量时对该变量分配内存单元，每个内存单元都会有一个编号，这就内存单元的地址，**地址指向该变量单元，将地址形象化地称为"指针"**，C 语言的地址都是带类型的地址，只有纯地址是无法对内存单元进行访问的
2. **直接访问**：如 int a=3; printf("%d", a); 这种形式
3. **间接访问**：如 int a=3, *p = &a; printf("%d", a); 这种形式
4. 指针：一个变量的地址
5. **如果一个指针变量中存放了一个整型变量的地址，则称这个变量是指向整型变量的指针变量**
6. 指针变量：有一个变量专门用来存储另一变量的地址，即为**指针变量**，指针变量就是地址变量，指针变量的值是地址
    - 指针变量名前面的*表示该变量的类型为指针变量。
    - 定义指针变量必须指定基类型，因为不同类型的所占字节不同
    - 赋给指针变量的是变量地址而不能是任意类型的数据，而且只能是与指针变量的基类型相同类型的变量的地址。
    - 指针变量中只能存放地址，不能*p1=100;系统无法分辨它是否是地址
    - 将指针变量赋予一个地址，包含了两个信息：1. 纯地址 2. 地址的基类型

```
// 定义指针变量
int *p => 基类型 *指针变量名;
int a =100;
// *p1 代表p1所指向的变量，也就是变量a
int *p1 = &a; // 可以初始化也可以不初始化，
printf("%d", *p1);
/*
上面 int *p1表示定义p1为整型指针变量，变量名为p1
p1 =  &a; 表示把a的地址赋予p1，这个时候p1就指向了a;
后面*p1代表p1所指向的对象，也就是a

即*p => 变量a，其中*号表示该变量是指针变量，*p1指变量p所只指向的变量(即实际值，也就是存储单元)
p => 变量a的地址
*/
```

指针变量赋值过程中发生了什么(以下面代码为例子)

```
int a;
int *p=&a;
```

1. 定义一个指向整型类数据的指针变量 p
2. **把 p 整型变量地址&a 赋值给 p，使得 p 指向 a，其中&a 包含 2 个信息：纯地址和 a 的类型。赋值时会先检查&a 和 p 的类型是否相同，不相同不能赋值**

指针变量作为函数参数

1. 指针变量作为实参，是按值传递的, 数组除外
2. **不能通过调用函数来改变实参指针变量的值，但可以改变实参指针变量所指变量的值，数组除外**

```
// 由上面给出的解释

// 1. 不能在所调用的函数里面直接更换指针变量的地址，如 int * a; a= b;
// 2. 能在函数里面直接使用 int a; a = *p, *p = 123; 即直接改变了指针变量所指变量的值
```

3. 假如 int a[3] = {1,2,3}, *p =a; 则也可以用 p[i]打印出当前值

通过指针引用数组

```
int a[10];
int *p;
p = &a[0]; // 或者p=a，两者等价
```

1. 如果指针变量 p1 和 p2 都指向同一数组，然后 p2-p1，结果是两个地址之差除以数组元素的长度，得出的结果代表了**元素之间相差的个数，可以得出两个元素间的相对距离**
2. 如果指针变量 p1 和 p2 都指向同一数组，然后 p2+p1，则无意义
3. 指针引用数组元素的方法
   - 下标法：a[i]
   - 指针法：*(a+i)
4. 如果指针 p1 已指向数组中的某一个元素，则 p+1 指向同数组中的下一个元素,**程序执行 p+1 时并不是将 p 的值(地址)简单地加 1,而是加一个数组元素所占用的字节数。**
5. *(a+i),*(p+i),a[i]三者是等价的,**在编译时，就是按数组首元素地址加上相对位移量得到所找元素的地址**

```
int a[5] = {1,2,3,4,5}
int *p;
p = &a[2]; // 或者p=a，两者等价
printf("%d", *(p+2)); // 打印5，p初始指向3
```

以变量名和数组名作为函数参数的比较

实参类型	变量名	数组名
要求形参的类型	变量名	数组名或指针变量
传递的信息	变量的值	实参数组首元素的地址
通过函数调用能否改变实参的值	不能	能改变实参数组元素的值

通过指针引用字符串

1. 使用字符指针变量和字符数组的区别：
   - 字符指针变量中存放的是地址，而不是将字符串放到字符指针变量中
   - 对字符数组的赋值，有些可以赋值，有些则不行，即**数组可以在定义时整体赋初值，但不能再赋值语句中整体赋值，指针变量除外**

```
// 不合法
char str[14];
str = "I love china";  // 不合法
str[] = "I love china";  // 不合法
// 合法
char *a;
a =  "I love china";
```

   - **数字符数组中个元素的值可以改变，但字符指针变量指向的字符串常量那个中的内容不可以被替代**数

```
char a[] = "House";
char *b = " House"; // 字符型指针指向一个常量，不能修改
a[2] = r; // 合法
b[2] = r; // 非法：字符串常量不能直接改变
```

   - **对字符指针变量，如果未初始化情况下，不能随便赋予内存单元，因为没有地址的内存单元可能会存在一个不可预料的值**

```
// 合法
char str[10];
scanf("%s", str);
// 不合法
char *a;
scanf("%s", a);
// 合法
char *a,str[10];
a= str;
scanf("%s", a);
```

- **字符数组是常量，不能加减。但字符指针变量可以加减**

易错地方

1. 输入两个值，按从大到小排列

```c
// 运行下面代码 输入5,9 打印出来的也是5,9
// C语言中实参变量和形参变量的数据传递是单项的值传递，用指针变量也要遵守这一情况
int main()
{
  void swap(int *p1, int *p2);
  int *z1, *z2, a, b;
  scanf("%d,%d", &a, &b);
  z1 = &a;
  z2 = &b;
  if (a < b)
  {
    swap(z1, z2);
  }
  printf("max=%d,min=%d\n", a, b);
  return 0;
}


// 正确
void swap(int *p1, int *p2)
{
  int temp;
  temp = *p1;
  *p1 = *p2;
  *p2 = temp;
}

// 错误
void swap(int *p1, int *p2)
{
  int *p;
  p = p1;
  p1 = p2;
  p2 = p;
}

// 出错
void swap(int *p1, int *p2)
{
  int *temp;
  *temp = *p1;
  p1 = *p2;
  p2 = *temp;
```

```
}
```
这里未给指针变量temp赋值，因此temp中并无确定的值(即地址不可预见)，所以temp指向的存储单元也是不可预见的，

提高部分

1. 多维数组的指针：每多一层地址，表示多一层数组，数组之间不像一维数组那样连续

```
int a[3][4] = {{1,3,5,7}, {9,11,13,15}, {17,19,21,23}};
/**
 上面a是数组名，有3行，同时也代表着二维数组首元素的地址，首元素是由4个整型元素所组成的一维数组，因此a代表
*/
a+1; // 代表a[1][0]的地址
a+2; // 代表a[2][0]的地址
int (*p)[4]; // 定义一个指向有4个元素的一维数组，如果有p=a+1; 则p指向a[1]的地址；,如果想到拿a[1][2]的
```

2. **指向函数的指针： 数据类型 (*指针变量名)(函数参数表列);**

```
int (*p)(int, int)
int getMax(int a, int b) {
  return a-b;
}
p = max; // 指针指向函数
(*p)(a,b); // 调用
```

3. **返回指针值的函数：类型名 *函数名(参数表列);**

```
// 返回指针值的函数
int *a(int x)
  {
    int *pt;
    pt = &x;
    return pt;
  }
  int *p;
  p = a(4);
  printf("%d", *p);
```

4. 指针数组：即数组里面的每一项都是指针/地址

```
// int *p[4];
int a = 1; int *pa = &a;
int b = 2; int *pb = &b;
int c = 3; int *pc = &c;
int d = 4; int *pd = &d;
int e = 5; int *pe = &e;

int* arr1[5] = { &a,&b,&c,&d,&e };
int* arr2[5] = { pa,pb,pc,pd,pe };
printf("%d", *arr2[2]); // 打印3
```

5. 多重指针：即**指向指针的指针**

```
int a = 100;
int *p1 = &a;
int **p2 = &p1;
int ***p3 = &p2;
printf("%d, %d, %d, %d\n", a, *p1, **p2, ***p3); // 打印 100,100, 100,100
```

6. 总结：**定义指针型变量的时候，只要是*号包含在括号内的，如(*p)，那么就属于指向 xx 的指针变量**

定义	含义
int i	定义整型变量 i
int *p	p 为指向整型数据的指针变量
int a[n]	定义整型数组，它有 n 个元素
int *p[n]	定义指针数组，它由 n 个指向整型数据的指针元素组成
int (*p)[n]	p 为指向含 n 个元素的一维数组的指针变量
int f()	f 为返回整型的函数
int *p()	p 为返回一个指针的函数，该指针指向整型数据
int (*p)()	p 为指向函数的指针，该函数返回一个整型值
int **p	p 是一个函数指针，它指向一个指向整型数据的变量
void *p	p 是一个指针变量，基类型为 void(空类型), 不指向具体的数据

指针运算小结

1. 指针变量的空值 NULL：指针变量可以有空值，即该指针变量不指向任何变量，NULL 是一个符号常量，在 stdio.h 头文件里面已经对 NULL 进行了定义,**任何指针变量或地址都可以与 NULL 做相等或不相等的比较**

```
if(p == NULL)...
```

2. 指针运算小结
   - 例如：p++,p--,p+i,p-i 等均是指针变量加(减)一个整数。将该指针变量的值(地址)h 和它指向的变量所占用的内存单元字节数相(减)
   - 两个指针变量可以相减，相减之差是两个指针之间的元素个数
   - 两个指针变量可以比较，指向前面元素的指针变量小于后面元素的指针变量

{2,3,4,5}, **q = p，则*p[0] = 1, **(q+2) = 4, A

09.使用结构体类型处理组合数据

结构体 struct

1. 定义：由用户自己建立由不同类型数据组成的组合型的数据结构，被称为**结构体**。在其他高级语言中把这种形式的数据结构称为“**记录**”
2. 特点：**结构体变量所占内存长度是各成员占的内存长度之和**，每个成员分别占有自己的内存单元
3. **结构体规则：**
   - 结构体变量中的成员的引用方式为：结构体变量名.成员名
   - 嵌套的结构体，则用类似于 student1.birthday.month
   - 对结构体的变量的成员可以像普通变量一样进行各种运算
   - 同类型的结构体变量可以相互赋值，比如下面例子可以直接 student1 = student2;
   - 可以引用结构体变量成员或结构体变量的地址,

```
scanf("%d", &student2.num);
scanf("%o", &student2);
```

4. **结构体指针：指向结构体数据的指针,一个结构体变量的起始地址就是这个结构体变量的指针。**
5. 定义结构体的方式：**只能对结构体变量赋值，存取或运算，而不能对一个类型赋值，存取或运算。在编译时对类型是不分配空间的，只对变量分配空间**

```
// 定义
struct 结构体名 {  // struct student合起来即为结构体类型名
  成员表列
} 变量名表列
// 例子1：先声明结构体类型，再定义该类型的变量
struct student {
  int num;
  char name[20];
  char sex;
  char addr[20];
}
struct student student1, student2 = { 10101, "shuhongxie", 'M', "guangzhou" };;
// struct student => 结构体类型名  student1,student2 => 结构体变量名
// 例子2：在声明类型的同时定义变量
struct student {
  int num;
  char name[20];
  char sex;
  char addr[20];
} student1, student2 = { 10101, "shuhongxie", 'M', "guangzhou" }; // 直接声明和定义
// 例子3：不指定类型名而直接定义结构体类型变量
struct  {
  int num;
  char name[20];
  char sex;
  char addr[20];
} student1, student2 = { 10101, "shuhongxie", 'M', "guangzhou" }; // 直接声明和定义
```

结构体数组

```
// 定义1
struct 结构体名 {
  成员表列
} 数组名[数组长度];
// 定义2
结构体类型 数组名[数组长度];
struct person leader[3]; // 未初始化
struct person leader[3] = { "Li", 0, "Zhang", 0, "Func", 0 }; // 已初始化

// 示例
struct person {
  char name[20];
  int count;
} leader[3] = {"Li",0,"Zhang",0,"Fun",0};
```

结构体指针

1. 定义：**结构体指针就是指向结构体数据的指针，一个结构体变量的起始地址就是这个结构体变量的指针**
2. 指向运算符：(*p).num 可以变成 p->num;

```
struct student *pt;

struct student {
  int num;
  float score;
  struct student *next;
} std;
pt = &std;
/*
调用函数体的值，可以写成以下三种形式
1. std.num
2. (*p).num
3. p->num
*/
```

用结构体变量和结构体变量的指针做函数参数

1. 结构体变量或结构体成员作为实参进行函数调用时，采取的是“值传递”，等于将结构体变量或结构体成员变量的值传给形参。但如果结构体成员是数组的话除外
2. 结构体指针变量作为实参进行函数调用时，传输的是结构体的地址

结构体和动态链表

1. 使用 malloc 方法**动态分配内存**

共用体类型

1. 特点：**结构体变量所占内存长度是各成员占的内存长度之和，共用体变量所占内存长度等于最长的成员的长度**
2. 由于共同体变量中的各个成员共用同一块存储空间，因此，在任一时刻，只能存放一个成员的值。
3. 共用体变量中起作用的成员值是最后一次被赋值的成员值。即再次赋值会覆盖之前的值。
4. 共用体变量的地址和它成员的地址都是同一地址
5. **可以对结构体进行赋值，但不能对共用体变量赋值也不能企图引用共用体变量来得到成员的值。**

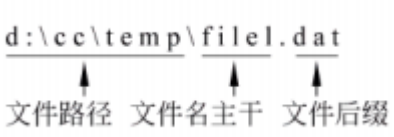
```
union data {
  short int i;
  char ch;
  float f;
} a, b,c;
// 共用体不能用变量进行初始化 比如  a = { 2, 'Z', '3.1' }
// 只能这样
a.i = 20;
a.ch = 'X';
a.f = 3.14142;
// 不能引用共用体变量，只能引用共用体变量中的成员 比如a.i
```

枚举类型(不重要)

```
// 定义
enum weekday { sun, mon, tue, wed, thu, fri, sat }; // sun,mon,tue 被称为枚举元素或枚举常量，他们是
// 使用
// workday,week_end 被称为枚举变量，他们的值只能是sun到sat之一
enum weekday workday,week_end; // 枚举变量
workday = mon;
week_end = sun
// 也可以直接定义枚举变量
enum weekday { sun, mon, tue, wed, thu,fri,sat } workday,week_end;
```

10.利用文件保存数据

C 文件的有关概念

0. 文件类型：**数据文件**，**程序文件**
1. 程序文件：包括源文件(.c)，目标文件(.obj)，可执行文件(.exe)，这种文件是用来存放程序的，以便实现程序的功能。
2. 数据文件：**供程序运行时读写的数据。**
3. 文件：**存储在外部介质上数据的集合**，如：磁盘，U 盘等。
4. 输入输出流：**表示了信息从源到目的端的流动。**输入操作时，数据从文件流向计算机内存，输出操作时，数据从计算机流向文件(如打印机，磁盘)
5. 流式文件：以字节为单位的文件，由一连串字节组成，中间没有分隔符，对文件的存取是以字节为单位的文件。
6. 数据文件的分类：**ASCII 文件**，也称作文本文件和**二进制文件**
7. 文件标识：**文件路径，文件主干，文件后缀**
    - 

8. 文件缓冲系统：系统自动在内存区为程序中每一个正在使用的文件开辟一个**文件缓冲区。**
9. 文件缓冲区：系统在读写程序时在内存中开辟的数据源与数据目标中间的一个用于保存完整数据内容的缓冲区域。
10. 文件类型指针：**在打开文件时，会在内存建立一个文件信息区，存放文件的有关特征和当前状态。这个信息的数据组织成结构体类型，系统将其命名了 FILE 类型。文件类型指针就是指向文件结构体类型变量的指针，即指向 FILE 文件类型的指针**

11. 通过文件指针访问文件的好处：可以随机访问文件，有效表示数据结构，动态分配内存，方便使用字符串，有效使用数组。
12. 文件信息区：用来存放文件的有关信息(如文件名，文件状态，文件当前位置等)，系统将其命名为 FILE 类型
13. 文件的打开和关闭：所谓的“打开”是指为文件建立相应的信息区和文件缓冲区。“关闭”是指撤销文件信息区和文件缓冲区，使文件指针变量不再指向该文件。

文件名

文件名包括三部分的文件标识

例如：d:\cc\temp\file1.dat

1. 文件路径 => d:\cc\temp\
2. 文件名主干 => file1
3. 文件后缀 => .dat （一般不超过 3 个字母）

数据文件分类

1. **ASCII 文件**：又称 text 文件，每一个字节存放一个字符的 ASCII 代码
2. **二进制文件**：把内存中的数据按其在内存中的存储形式原样输出到磁盘上存放

```
/**
 * 比如 10000用ASCII形式存储到键盘上，则需要占5个字节(一个字符占1一个字节)，
 * 而用二进制形式输出，则在磁盘上只占2个字节
 */
```

文件缓冲区

1. 定义：C 语言采用"**缓冲文件系统**"处理文件，是指系统自动在内存区为程序中每一个正在使用的文件开辟一个**文件缓冲区**，从内存中向磁盘输出数据必须先送到内存中的缓冲区，装满缓冲区后才送到磁盘，如果从磁盘向内存中读入数据，则一次从磁盘文件将一批数据输入到内存缓冲区(充满缓冲区)，然后再从缓冲区逐个地将数据送到程序数据区(给程序变量)，缓冲区大小由各个具体的 C 编译系统确定。
2. 为什么要用缓冲区？ 缓冲区是为了让低速的输入输出设备和高速的用户程序能够协调工作，并降低输入输出设备的读写次数。
    - 例如，我们都知道硬盘的速度要远低于 CPU，它们之间有好几个数量级的差距，当向硬盘写入数据时，程序需要等待，不能做任何事情，就好像卡顿了一样，用户体验非常差。计算机上

绝大多数应用程序都需要和硬件打交道，例如读写硬盘、向显示器输出、从键盘输入等，如果每个程序都等待硬件，那么整台计算机也将变得卡顿。

- 但是有了缓冲区，就可以将数据先放入缓冲区中（内存的读写速度也远高于硬盘），然后程序可以继续往下执行，等所有的数据都准备好了，再将缓冲区中的所有数据一次性地写入硬盘，这样程序就减少了等待的次数，变得流畅起来。

文件指针类型

1. 定义：缓冲文件系统中，关键的是文件类型指针，简称**文件指针**，每个被使用的文件都在内存中开辟一个相应的**文件信息区**，用来存放文件的有关信息(如文件名，文件状态，文件当前位置等)。这些信息是保存在一个结构体变量中的，该结构体变量由系统声明存放在头文件"stdio.h"中，叫FILE，在程序中可以直接使用 FILE 类型定义名词。

```
// 定义file类型变量
FILE F;
// 定义指向文件型数据的指针变量:
FILE *fp;
```

文件的打开和关闭

1. fopen(文件名，使用文件方式:r/w/a/rb/wb/ab/r+/w+/a+/rb+/wb+/ab+) 失败返回 NULL，成功返回文件首地址。**使用 fopen 时，若以"读"的模式打开，文件必须存在；若以"写"的模式(只要含有 w)打开时，文件可以不存在，会自动创建**
2. fclose(文件指针) 成功返回 0，失败返回 EOF(-1)，**关闭就是撤销文件信息区和文件缓冲区，使文件指针变量不再指向该文件，使文件指针变量与文件脱钩，此后不能通过该指针对原来与其相联系的文件进行读写，，如前所述,在向文件写数据时,是先将数据输出到缓冲区,待缓冲区充满后才正式输出给文件。如果当数据未充满缓冲区而程序结束运行,就会将缓冲区中的数据丢失。用 fclose 函数关闭文件,可以避免这个问题,它先把缓冲区中的数据输出到磁盘文件,然后才释放文件指针变量。**

```
FILE *fp;
fp = fopen("a1", "r");
fclose(fp);
```

文件的顺序读写

向文件读取字符

1. fgetc(fp)： 从 fp 指向的文件读入一个字符，成功返回所读字符，失败返回 EOF(-1)

2. fputc(ch, fp)：把字符 ch 写到文件指针变量 fp 所指向的文件中，成功返回输出字符，失败返回 EOF(-1)，**写入时会清空文件原有的文本**
3. feof(in)：检测文件尾标志 EOF 是否已被读过。被读过表示文件有效字符已全部读完，返回 非 0，否则返回 0
4. 文件读取为什么是按顺序的：访问磁盘文件是逐个字节进行的，为了知道当前访问到第几自己，系统用**文件读写位置标记**来表示**当前所访问的位置**，每访问完一个字节后，当前位置标记就指向下一字节。为了知道对文件的访问是否完成，在一个文件的有效字符的后面一字节中，系统自动设置了一个**文件尾标志**，用 EOF 表示,在 stdio 头文件 EOF 被定义为-1，当读完全部有效字符后，读写位置标记就指向 EOF，再执行一次读操作就读入 EOF,这个时候表示有效字符已经读完。

向文件读取字符串

1. fgets(str,n ,fp)：从 fp 指向的文件中读入一个长度为 n-1 的字符串，存放到字符数组 str 中。成功返回 str 数组首元素, 失败返回 NULL,**在读完一个字符前遇到\n 或 EOF,读入即结束，但将所遇到的换行符\n 也作为一个字符读入**
2. fputs(str, fp)：把字符 str 写到文件指针变量 fp 所指向文件中，输出成功返回 0，失败返回非 0 值

文件的格式化读写

1. fprintf(文件指针，格式字符串，输出表列)；对文件按格式进行输出输出到指定文件
2. fscanf(文件指针，格式字符串，输入表列)

```
// 例如 将整形变量i和实型变量f按照%d和%6.2f的格式输出到fp指向的文件中
fprint(fp, "%d,%6.2f", i ,f);
```

用二进制方式读写文件(只能用于二进制文件)

buffer: 一个地址，对于 fread 来说，是读入数据的存放地址，对 fwrite 来说是输出数据的地址
size: 要读写的字节数
count: 要进行读写多少个 size 字节的数据项
fp：文件型指针

1. 读：fread(buffer,size,count, fp);
2. 写：fwrite(buffer,size,count,fp);
3. fread 和 fwrite 如果成功，函数返回值为 count 的值

文件的随机读写

使用 rewind 函数使文件位置标记指向文件头

1. 重置文件位置标记：rewind(文件类型指针)

2. 移动文件位置标记，改变文件位置标记的位置：fseek(文件类型指针，位移量，起始点)，**用于二进制文件**
3. 测定文件位置标记的当前位置(从文件头开始)：ftell(fp)，如果返回-1L 那表示出错，如果函数调用时出错那返回 error

```
//     起始点          名字            用数字代表
//    文件开始         SEEK_SET          0
//文件位置标记当前位置  SEEK_CUR          1
//     文件末尾        SEEK_END          2
fseek(fp, 100L, 0); // 将文件位置标记移到离文件头100字节处
fseek(fp, 50L, 1); // 将文件位置标记移到离当前位置后面50字节处
fseek(fp, -10L, 2); // 将文件位置标记从文件末尾处向后退10字节
```

系统自定义的文件类型指针

3 个**标准文件**指针，**为了方便用户，系统在程序开始运行时，自动打开下面 3 个标准文件**

1. stdin(标准输入文件指针)，指向在内存中与**键盘**相对应的文件信息区，因此用他进行输入就蕴含了**键盘输入**
2. stdout(标准输出文件指针)，指向在内存中与**显示器屏幕**相对应的文件显示区
3. stderr(标准出错文件指针)，用来输出出错的信息，指向在内存中和**显示器屏幕**相应的文件显示器，因为，在程序运行出错时的信息就输出到显示器屏幕。
4. stdprn(标准打印文件)，一般指打印机。

fread 和 fwrite 用于二进制文件的输入输出

文件读写的出错检测

1. ferror 函数：可以用 ferror 函数检查输入输出函数的错误，调用方式为 ferror(fp)；返回 0 时表示为出错，返回 EOF 值表示出错，执行 fopen 时 ferror 的函数初始值默认为 0
2. clearerr：使文件错误标志和文件阶数标志置为 0

各函数返回值分类

- fclose: **关闭 fp 所指的文件，释放文件缓冲区**，完成返回 0，出错返回 EOF(-1)
- feof：**检查文件是否结束**，没读完返回 0，遇到文件结束符返回非 0
- fgetc：**从 fp 所指定的文件中取得下一个字符**，成功返回所得字符，出错返回 EOF(-1)
- fgets：**从 fp 指向的文件读取一个长度为(n — 1)的字符串,存入起始地址为 buf 的空间**，成功返回地址 buf，失败或出错返回 NULL

- fputc: **将字符 ch 输出到 fp 指向的文件中**，成功返回该字符，失败返回 非 0
- putc: **将字符 ch 输出到 fp 指向的文件中**，成功返回该字符，失败返回 EOF
- fputs：**将 str 指向的字符串输出到标准输出设备**，成功返回 0，失败返回 非 0
- getchar：**从标准输入设备读取下一个字符**，成功返回所读字符，文件结束或出错返回 -1
- putchar：**把字符 ch 输出到标准输出设备**，输出的字符 ch。若出错,返回 EOF
- fopen: **以 mode 指定的方式打开名为 filename 的文件**，成功返回文件指针，失败返回 NULL(0)
- fseek：文件位置指针移动，成功返回当前位置，否则返回-1
- ferror：返回值为 0(假),表示未出错，返回 EOF(-1)表示出错
- clearerr：使文件错误标志和文件结束标志为 0,, 无返回值
- 返回 NULL 的 fopen,fgets, 成功返回字符的 getc/fgetc/fputc/getchar/putchar
- 成功返回 0 的 fclose,fputs
- 返回 EOF 的 fgetc,putc,putchar
- fgetc,fgets,fputc,fputs,fread,fwrite 的 fp 参数都在最后，其他的 fp 参数都在最前