C语言

C语言基础

概念

1. **程序**：一系列指令序列，即人机对话的语言
2. **语言分类**
    - **低级语言**：机器语言、汇编语言
    - **高级语言**：C,C++等
3. 程序设计的三大基本结构
    1. **顺序结构**
    2. **选择结构**
    3. **循环结构**
4. 源程序：高级语言编写的程序
5. 目标程序：二进制代码表示的程序
6. 开发一个C程序的基本过程：
    - **编辑(.c 源程序)->编译(.obj 目标程序)->连接(.exe 可执行程序)->运行**
7. C代码编译成可执行程序经过4步
    - 预处理->编译>汇编->链接
8. C语言是一种**结构化的程序设计语言**,简明易懂，功能强大，适合于各种硬件平台。C语言**即具有高级语言特点，又具有低级语言的功能。**
9. C程序可以有零个以上的输入但必须得有1个以上的输出
10. 基本结构

```c
#include <stdio.h>
int main(){
    printf("Hello World\n");
    return 0;
}
```

11. **函数是程序的基本单位。一个程序有且只有一个main()函数，位置任意**程序执行**从main开始，到main结束**，不能被其他函数调用
12. 函数的基本单位为语句
13. **分号为语句的分隔符**
14. 大括号标识一个语句组，成对使用
15. 若**主函数为 void 可省略 主函数还可以是int**

```c
#include <stdio.h>
int main(){
    int a,b,c;
    a=4;
    b=10;
    c=a+b;
    printf("%d\n",c);
    return 0;
}
```

标识符

1. 标识符：在C语言中，有许多符号的命名如变量名、函数名、数组名等，都必须遵守一定**的规则**，按此规则命名的符号**称为标识符**
2. 标迟符的命名规则：
    - **字母、数字、下划线_构成的有限序列**
    - **以字母或者下划线开头**
    - 标识符不能包含除以外的任何特殊字符，如：%、#、逗号、空格等。
    - 标识符**必须以字母或（下划线）开头。**
    - 标识符不能包含空白字符（换行符、空格和制表符称为空白字符）
    - C语言中的某些词（例如int和float等）称为**保留字**，具有特殊意义，不**能用作标识符名。**
    - C语言**区分大小写**，因此标识符price.与标识符PRICE是两个不同的标识符。
3. 分类
    1. **关键字**(32个)：C语言规定了一批标识符，他们在C语言中代表着固定的含义，**不能另做它用。**

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

    2. **预定义标识符**：C语言**语法允许**用户把这类标识符**另做它用**，但是**这些标识符将失去系统规定**的功能。

        比如：**printf define main scanf**

    3. **用户标识符**：由用户根据需要定义的标识符称为用户标识符

注释

- ```
  //行注释
  /*注释内容不是程序代码，是给其他人员增强理解能力加上的文字说明当程序执行时注释内容视为空 */
  //块注释
  ```

常量

1. 常量是在程序中保持不变的量
2. 常量用于定义具有如下特点的数据
    - 在程序运行中**保持不变**
    - 在程序内部频繁使用
    - 用define关键字定义**符号常量**
    - 分类：**整型常量、实型常量、字符常量、字符串常量、符号常量**
      举例：1 2.5 'a' “abc”
  ```

整型常量

1. 十进制表示：用一串连续的数字(0~9)表示十进制数。
   例如：345 3684 0 -23456 **只有十进制负数前面加-。**
2. 八进制表示：以数字**0开头**的一个连续数字序列，序列中只能有**0-7**这八个数字。
   例如：045 067451等。 而：019 423 -078都是非法的八进制数。（**无负数**）
3. 十六进制表示：以**0X或0x开头**的连续数字和字母序列，序列中只能有0-9、A-F和a-f这些数字和字母，字母a、b、c、d、e、f分别对应十进制数字10、11、12、13、14、15,大小写均可。**(0x里面x小写就用小写a-f 大写X就是A-F)**（**无负数**）

实型常量

1. 小数形式
   - 由数字和小数点组成，**必须有小数点。**(整数部分，小数部分为0可省略小数点不能省略)
   - 例如：3.14 0.15 .56 78. 0.0
2. 指数形式
   - 以**幂**的形式表示，以字母e或E后跟一个以10为底的幂数。
   - 形式：数字 字母+(e/E) 指数(正负)
   - 字母e或E的前后及数字之间不得有空格（实型变量也遵守这个规则)
   - 技巧记忆：**e前e后必有数，e后必须是整数(负数0整数)!**
   - 例：2.3e5 500e-2 .5e3 4.5e0，而e4 .5e3.6 .e5 e都不合法。

字符常量

- 定义：在程序中用**一对单引号把一个字符**括起来，作为字符常量
- 例'A','a','t','!','*','\n'等。Ascll A值65 a值97
- 字符常量只能**用单引号**括起来，**不能用双引号**； 如：“A”不是字符常量
- 字符常量**只能包含一个字符**；如：'abc',错误。
- **区分大小写**； 如：'A'和'a
- C语言约定采用 \ 开头的字符序列 如 \101 八进制 x开头十六进制 \xhh
- 注意：**\x十六进制就两位**因为3位超过ASCLL了 **\转义字符加8进制16进制需要注意越界**

转义字符	意义	ASCII码值（十进制）
\a	响铃(BEL)	007
\b	退格(BS)，将当前位置移到前一列	008
\f	换页(FF)，将当前位置移到下页开头	012
\n	换行(LF)，将当前位置移到下一行开头	010
\r	回车(CR)，将当前位置移到本行开头	013
\t	水平制表(HT)（跳到下一个TAB位置）	009
\v	垂直制表(VT)	011
\\	代表一个反斜线字符"\'	092
\'	代表一个单引号（撇号）字符	039
\"	代表一个双引号字符	034
\?	代表一个问号	063
\0	空字符(NUL)	000
\ddd	1到3位八进制数所代表的任意字符	三位八进制
\xhh	十六进制所代表的任意字符	十六进制

注意：

1. 区分，斜杠："/" 与 反斜杠："\",此处不可互换

2. \xhh 十六进制转义不限制字符个数 '\x000000000000F' == '\xF' [3]

- **字符**是**按**其代码 **ASCLL码(整数值)形式存储**，所有字符数据都作为整数来处理。因此，字符量可以**参与整数运算。**

字符串常量

1. 定义：在程序中用一**对双撇号把若干个字符**括起来 例：“123”，“boy”,“a”等。
2. 字符串常量只能用**双撇号**，不能用单引号括起来；如：‘A’不是字符串常量
3. 字符常量和字符串常量区别
    - 分界符不同**字符串常量双撇号**，**字符常量单撇号**
    - **字符串常量内容多个字符**，**字符常量一个字符**
    - **占用空间不同**字符串常量存储结尾**\0** (占用空间字节个数是**字符数加1**因为\0也占用一个字节)字**符常量占用1个字节**
    - 字符参与运算，**字符串不能参与运算**

符号常量

1. 用**编译预处理宏定义**一个符号名的方法来代表一个常量
2. 格式：**#define 宏名 宏值**

```c
#include <stdio.h>
#define PI 3.14159
main(){
    float r;
    double s;
    r=5.0;
    s=PI*r*r;
    printf("s=%f\n",s);
}
```

常变量

1. 定义a为一个整型变量，指定其值为3,而且在变量存在期间其值不能改变
2. 常变量与常量的异同是
    - **常变量**具有变量的基本属性:有类型，占存储单元，只是**不允许改变其值。**
    - 常变量是有名字的不变量，而常量是没有名字的不变量。
    - **有名字**就便于在程序中被引用。

```c
const int a=3
```

变量

1. 定义：在**程序运行过程中，值可以改变**的量
2. 注意：
    - 每个**变量**有一个**名**字作为标识，属于用户标识符。
    - **变量**必须**先定义后使用**（定义后还得赋值才能用）
    - 变量代表了内存中的若干个存储单元，**变量名**实际上是以一个名字**代表**的一个**存储空间**（存储地址）
    - **变量名**和**变量值**是两个不同的概念
        - 定义变量时指定该变量的名字和类型
        - 从变量中取值，实际上是通过变量名找到相应的内存地址，从该存储单元中读取数据。
3. 格式：**数据类型 变量名**； 如：int i;
4. 变量初始化
    - **边定义变初始化： 数据类型 变量名=值**；（注意区分变量名和变量值）
    - **先定义后初始化：数据类型变量名**； 变量名=值;
5. 变量类型
    - **int(4个字节) char(1个字节) float(单精度4字节) double(双精度4字节)**（长度和机器有关）
6.

类型/编译器	16位编译器	32位编译器	64位编译器
void	0	0	0
char	1	1	1
char *	2	4	8
short int	2	2	2
int	2	4	4
float	4	4	4
double	8	8	8
long	4	4	8
long double	8	12	16
long long	8	8	8

运算符

基本运算符

1. 基本的算术运算符

表 3-4 最常用的算术运算符

运算符	含义	举例	结果	优先级
++	自增运算符	++a	a 的值加 1	14
--	自减运算符	--a	a 的值减 1	
*	乘法运算符	a*b	a 和 b 的乘积	13
/	除法运算符	a/b	a 除以 b 的商	
%	求余运算符	a%b	a 除以 b 的余数	
+	加法运算符	a+b	a 和 b 的和	12
-	减法运算符	a-b	a 和 b 的差	

   - 另外，对于+和-俩个运算符，还具有取原值和取负值的含义 如：a=+a;a=-a;
2. 算术运算符
   - / % 双目/二元 从左向右 + - 双目 从左向右 若以上5个运算符混合 *** / % 高于+ -**，但是**小括号()改变优先级**
   - **%求余要求两个操作数都为整数** 如果**两个数是负数看分子 分子为负结果为负数 例子：5%3 =2 2%5 =2 -5%7=-5 7%-5=2**

3. 类型转换(自动类型转换
    - char　char->int
    - int char->int
    - int　int->int
    - char/int　float/double--->double
    - float　float/double--->double
    - double double--->double
    - 注意：商 取整 向0 取整（下取整）
    - **如果参与运算的操作数包括float或double，运算结果都是double**
    - **如果参与运算的操作数只包括int或char型，运算结果是int型**
4. ++ -- 自增 自减 单目 需要一个操作数**（整型）**
    - **前缀 ++i --i 先变后用新值 先自增1在用新值 i=i+1在用新i 减同理**
    - **后缀 i++ --i 先用原值后变 先用i i在自增1 先用 i 原值 i=i+1**
    - **注意：前缀和后缀只有在表达式才有区别若作为单独语句功能相同没有区别**
5. 强制类型转换：（类型名）(表达式)
    - **强制类型转换只会改变当前表达式的类型**（一个变量定义后在生命周期中类型不会改变）
    - 如：3.5%2错误　(int）3.5%2正确
6. **运算数类型不一致，**系统自动进行类型转换（**由低向高**转换)
   例：**3/2的结果就是 1　3/2.0的结果就是1.5**

赋值运算符

1. 值运算符和赋值表达式：变量名=表达式
2. **优先级倒数第二，结合方向：左<——右**
3. **是一种赋予的关系而不是等价的关系**
4. = **左侧只能是变量**，不能是表达式
5. 如果**赋值运算符两侧的类型不一致**，但都是**基本类型**时，在赋值时要进行类型转换。类型转换是由**系统自动进行的**，转换的规则是
    - 将**浮点型数据**（包括单、双精度）赋给**整型变量**时，先对浮点数取整，即**舍弃**小数部分，然后赋予整型变量。**int i=3.65; printf("%d",i);//输出3**
    - 将**整型数据赋给单、双精度变量时 数值不变** 但**以浮点数形式存储到变量**中
    - 将一个**double型**数据赋给**float**变量时，先将双精度数转换为单精度，即**只取6~7位有效数字**，存储到float型变量的**4个字节**中。应**注意**双精度数值的大小不能超出float型变量的**数值范围**；将一个**float**型数据赋给**double**型变量时，**数值不变**，在内存中以**8个字节存储，有效位数扩展到15位。**
    - **字符型数据**赋给**整型变量时**，将字符的**ASCII码赋给整型变量。**
    - 将一**个占字节多的整型数据**赋给一**个占字节少的整型**变量或**字符变量**时，只将其**低字节原封不动地送到被赋值的变量**（即发生"**截断**")。

复合赋值表达式

1. 复合赋值运算符
   +=、-=、*=、/=、%=、<=、>=等（两个运算符之间不能有空格)
2. **借助复合的赋值运算符**将形如
    - 变量名=变量名+表达式"的表达式
    - 简化成："**变量名+=表达式**"的形式
    - 说明：凡是有赋值运算符参加的运算都是**从右往左算**例
        - a+=3 等价于a=a+3
        - x * =y+8 等价于x=x*(y+8)

    - x%=3等价于x=x%3
3. 嵌套赋值
    - 结合方向：自右向左
    - 每一个赋值符号出现时，计算一下右边表达式的值
    - 如果给变量赋值，记录变量的变化

> 已知 `int a,b=5,c=4;`计算表达式a+=a-=a=b+c的值。　　得0

逗号运算符和逗号表达式

1. 定义：用逗号运算符将表达式连接起来的式子
2. 一般形式：表达式1，表达式2，表达式3，…，表达式n
3. 求解过程
    - **从左到右**一个一个求解，**最后一个表达式的值**就是整个、逗号表达式的值。
    - 结合方向左--右
    - 优先级最低
    - 例如
        - a=3　a=3,a+3结果：表达式的值为6
        - b=a+3,a-3 结果：表达式的值为0
        - a=(2,3,4)a的值是多少？ a=4
        - a=2,3,4 a的值是多少？　a=2 后面不保留

C语言中的逻辑值

1. 逻辑值只有两个，分别用“真”和“假”表示。
2. **任何基本类型的值都可作为逻辑值**使用。
    - C语言没有专门的逻辑值，**所有非0**的值被都被当作“真使用，而0值被当作“假”使用。

关系运算符

1. 系运算实际上是“比较运算”
2. C语言的关系运算符共6种

操作符	含义
==	等于
!=	不等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于

3. **优先级**：后四种优先级高于前两种 **结合方法**：从左向右
4. 关系运算符的操作数可以是**变量、常量或表达式。**
5. **计算结果=逻辑值（真或假）** 在C语言中，**“0”表示“假”，“非0”表示“真”**

逻辑运算符

1.

运算符	含义	举例	说明
!	(逻辑非)	!-1 结果0	非取反真变假 假变真
&&	(逻辑与)	a&&b	逻辑与同真为真其他为假
||	(逻辑或)	a||b	逻辑或同假为假其他未真

2. 双目 || && 单目 !
3. ! → && →|| (**!为三者中最高**)
4. 短路：**&&与遇见0就短路 后面不算**结果0 **||或遇见1就 不算结果1**

```
&&断路规则：
int a=0,b=0;
计算下列表达式执行后a和b的值以及表达式的值
1）a++&&b++ a=1,b=0,result=0
2）a++&&++b a=1,b=0,result=0
3）++a&&b++ a=1,b=1,result=0
4）++a&&++b a=1,b=l,result=1
int a=1,b=1;
计算下列表达式执行后a和b的值以及表达式的值
1）a--&&b-- a=0,b=0,result=1
2）a--&&--b a=0,b=0,result=0
3）--a&&b-- a=0,b=1,result=0
4）--a&&--b a=0,b=1,result=0
||断路规则
int a=0,b=0;
计算下列表达式计算后a,b的值以及表达式的值
1）a++||b++  a=1,b=1 result =0
2）a++||++b   a=1,b=1 result =1
3) ++a||b++  a=1,b=0 result =1
4) ++a||++b  a=1,b=0,result =1
int a=1,b=1:
计算下列表达式计算后a,b的值以及表达式的值
1）a--||b-- a=0,b=1 result =1
2)a--||--b  a=0,b=1,result =1
3)--a||b-- a=0,b=0,result =1
4)--a||b-- a=0,b=0,result =0
```

5. 运算符的优先次序：！—>算数运算符—>关系运算符—>&&—>||—>赋值运算符
6. 逻辑运算符与逻辑表达式
   - **表示逻辑运算结果时：1“真”，0“假”**
   - **判断一个量是否为“真”时：0“假”，非0“真”** 注意：**将一个非零的数值认作为“真”**

条件运算符

1. 条件表达式的一般形式为

```
表达式1?表达式2:表达式3
```

2. 条件运算符的执行顺序
   - 求解表达式1
   - 真：求解表达式2，值为整个条件表达式的值

- 假：求解表达式3，值是整个条件表达式的值
- **注：唯一的三目运算符 记忆： 真前假后**

3. **算术>关系>逻辑>条件>赋值>逗号**
4. 结合性：“自右至左”
5. 以下为合法的使用方法

```
a>b?(max=a):(max=b);
a>b?printf("%d",a):printf"%d",b)
```

长度测试运算符

1. c语言提供了测试数据长度运算符sizeof
2. 其功能是给出exp所占用的内存字节数。

```
sizeof(exp)
//其中，exp可以是类型关键字、变量或表达式。
//例如
sizeof(double),sizeof(x)
```

流程控制结构

顺序结构

语句

1. 语句：C语言中描述计算过程的最基本单位。 由分号；结束。
2. 顺序结构：按语句在程序中出现的顺序逐条执行，没有分支、没有转移。

复合语句和空语句

1. 复合语句
   - 定义：用一对**花括号**把若干语句括起来构成一个语句组。
   - 注意：
     - 花括号内语句的数目不限
     - 在**花括号外面不能加分号**
2. 空语句

```
main(){
 ;
}
```

数据输出

1. 基本概念
   - 输出：把数据从计算机**内部**送到计算机**外部设备**上的操作称为"输出”
2. 注意：C语言本身不提供输入和输出语句，但是有输入和输出函数。
3. 在使用输入输出函数时，要在程序文件的开头用预编译指令
   - 按指定路径查找文件
   - 源程序文件所在目录
   - C编译系统指定的include目录

```
#include "c:\cpp\linclude\lmyfile.h"
#include <stdio.h> //根目录
#include "stdio.h" //用户目录
```

printf()函数

1. 作用：在终端设备上按指定格式输出
2. 语句：**printf (格式控制，输出项表)**;
3.

格式字符	说　明
d,i	以带符号的十进制形式输出整数（正数不输出符号）
o	**以八进制无符号形式输出整数（不输出前导符 0 ）**
x,X	**以十六进制无符号形式输出整数（不输出前导符0x），用x则输出十六进制数的a～f时以小写形式输出，用X时，则以大写字母输出**
u	以无符号十进制形式输出整数
c	以字符形式输出，只输出一个字符 'a'
s	输出字符串"a" "ab"
f	**以小数形式输出单、双精度数，隐含输出6位小数 %f double 15-16位，%f 6位 float 6-7位 1.0f/3=0.333333**
e,E	以指数形式输出实数，用e时指数以"e"表示(如1.2e+02)，用E时指数以"E"表示(如1.2E+02)=1.2*10的2次幂
g,G	选用%f或%e格式中输出宽度较短的一种格式，**不输出无意义的0**。用G时，若以指数形式输出，则指数以大写表示 64.000000 %g 64

附加字符	说　明
l	长整型整数，可加在格式符 d 、 o 、 x 、 u 前面）
m (代表一个正整数)	数据最小宽度
n (代表一个正整数)	对实数，表示输出 n 位小数；对字符串，表示截取的字符个数
-	输出的数字或字符在域内向左靠

4. 常用格式字符
   - d格式符：用来**输出一个有符号的十进制整数** 输出int型数据
   - 指定列宽：**%md** 右对齐
     - **实际宽度>设定宽度m时，按实际宽度输出；**
     - **实际宽度<设定宽度m时，数据右对齐，左边用空格补位。**
   - **%-md** 左对齐
     - **实际宽度>设定宽度m时，按实际宽度输出；**
     - **实际宽度<设定宽度m时，数据左对齐，右边用空格补位。**
   - c格式符：用来输出一个字符

     ```
     char ch='a';
     printf("%c",ch); //输出字符：a
     printf("%5c",ch); //输出字符：空4个a
     ```

   - s格式符：用来输出一个字符串

     ```
     printf("%s","CHNA") //输出字符串：CHINA
     ```

     - %m.ns:指定输出字符串数据的宽度为m,
       - 若n<m,**截取字符串中左端n个字符，输出靠右**，左端空格补齐。
       - 若n>m,输出字符串中截取的前n位字符字符。
   - f格式符：用来输出实数，以小数形式输出
     - %m.nf对于float、double数据可以用"m.n"形式
     - **m指定数据总宽度**，**n指定小数部分位数**，即精度
     - **%-m.nf输出数据左对齐；**

- **%+m.nf输出数据带正负号；**
- **%0m.nf不够位数用0补位；**
- ```
  //用%输出实数，只能得到6位小数。
  double a=1.0;
  printf("%f\n",a/3);
  0.333333
  ```
- ```
  //指定数据宽度和小数位数：用%m.nf(右对齐)  m列宽 n小数位数
  printf("%20.15f\n",1.0/3);
  //空4位 0.333333333333333 15位小数
  printf("%.0f\n",10000/3.0):
  //3333
  float n=9.478689;
  printf("%f",n);
  //输出结果：9.47869
  //默认情况下精确到六位小数 上面是7位四舍五入
  //m宽度，表示所有的数字和小数点所占的位数，不够20位右对齐。如果m比较小比如1输出的数位数大于m就没有作用
  //n精度（精确到小数点后多少位）
  ```

- **e(E)**格式符。指定以**指数形式输出**实数
  - %e,VC++给出小数位数为6
  - 小数点前必须有而且只有1位非零数字

    ```
    printf("%e",123.456;
    //输出：1.234560 e+002  002表示10的2次方
    ```

  - %m.ne 设置列宽和小数位数 m列宽 n小数位数

    ```
    printf("%13.2e",123.456)
    //输出：     1.32 e+001 前面4个空格
    ```

5. 注意
   - 遇到%号字符，按后面输出列表变量的值代替
   - 遇到 \ 转义符体现功能
   - 其他一般字符原样输出
   - 字符型用%d输出是字符码的值
   - ```
     k=12;
     printf("k=%%d\n",k);
     //输出 k=%d
     //%遇到%是输出%的意思
     ```

6. 注意事项
   - 格式控制中应包含与输出项**一一对应**的输出格式说明，类型必须匹配；
   - 若格式说明的**个数少于输出项个数**，则多余的输出项不予输出

     ```
     如：int x=12,y=28;printf('%d',x,y);
     ```
     ```

- 若格式**说明的个数多于输出项个数**，输出乱码。

```
int x=12,y=28;
printf("%d%d%d",x,y);
```

- **同一变量，不同形式**，出现在同一条输出函数调用中。

```
//如：
int k=21;
printf("%d,%d",k,++k);
//输出22,22
```

  - 原因：printf函数其参数**从右往左进行处理**，先计算++k。显示值时，从左往右。其实k是先执行的但是进入栈了++k后进栈 但是栈后进先出

putchar函数

1. 用putchar函数既可以输出**可显示字符**，也可以输出**控制字符**和**转义字符**
2. putchar(c)中的**c**可以是**字符常量、整型常量、字符变量或整型变量**（其值在字符的ASCII-代码范围内)。

```
putchar('A');//输出大写字母A
putchar(x);//输出字符变量x的值
putchar('\101');//也是输出字符A
putchar('65');//也是输出字符A
putchar('\n');//换行
//对控制字符则执行控制功能，不在屏幕上显示。
```

数据输入

1. 输入：从计算机**外部设备**将数据送入**计算机内部**的操作称为"输入"。

scanf函数

1. 作用：
2. 在终端设备上输入数据
3. 语句：scanf(格式控制，地址表列)；
   - **格式控制**含义同printf函数
   - **地址表列**可以是变量的地址，或字符串的首地址
4. 注意：格式控制必须与你对应的变量的类型相等，否则会出现意想不到的数据。
   - 记忆：**第一部分格式控制的形式在终端输入数据**　一模一样！
5. scanf函数中的格式声明

格式控制符	说明
%c	读取一个单一的字符
%hd、%d、%ld	读取一个十进制整数，并分别赋值给 short、int、long 类型
%ho、%o、%lo	读取一个八进制整数（可带前缀也可不带），并分别赋值给 short、int、long 类型
%hx、%x、%lx	读取一个十六进制整数（可带前缀也可不带），并分别赋值给 short、int、long 类型
%hu、%u、%lu	读取一个无符号整数，并分别赋值给 unsigned short、unsigned int、unsigned long 类型
%f、%lf	读取一个十进制形式的小数，并分别赋值给 float、double 类型
%e、%le	读取一个指数形式的小数，并分别赋值给 float、double 类型
%s	读取一个字符串（以空白符为结束）
%g、%lg	既可以读取一个十进制形式的小数，也可以读取一个指数形式的小数，并分别赋值给 float、double 类型
- %+格式字符，中间可以插入附加的字符，**附加字符必须原样输入。**

```
scanf("a=%f,b=%f,c=%f",&a,&b,&c);
//a=是附加字符输入的时候必须输入a=...
```

- scanf函数中**没有精度控制。**
- scanf中要求给出**变量地址**，如给出变量名则会出错。
- scanf中
  1. 对整型输入可以控制格式，一般是%d,若出现
     - %md,“截取”，截取m位数据给对应的变量
     - %*nd,“跳跃”，从对应位置开始跳跃n列，该控制字符不取值，下一个控制字符再进行取值给对应的变量
  2. 对浮点型，不能进行%m.nf的格式设置
- 在输入多个数值数据时，若格式控制串中没有非格式字符作输入数据之间的间隔则**可用空格，TAB或回车作间隔。**C编译系统在遇到**空格、TAB、回车或非法数据**（如对“%d"输入“12A”时，“A”即为非法数据)**时即认为该数据结束。**
- **在用“%c”格式声明输入字符时，空格字符和“转义字符”中的字符都作为有效字符输入。**
- 如输入的数据与格式指示字符不一致时，虽然编译能够通过，但结果将不正确。

6. 使用scanf函数时应注意的问题

- ```
  scanf("%f%f%f",a,b,c);//错
  scanf("%f%f%f",&a,&b,&c);//对
  scanf("a=%f,b=%f,c=%f",&a,&b,&c);
  //132错
  //a=1,b=3,c=2对
  //a=1b=3c=2错
  scanf("%c%c%c",&c1,&c2,&c3);
  ```
  ```

```
//abc对
//a b c 错
对于scanf("%d%c%f",&a,&b,&c);
//若输入1234a123o.262
 a=1234 b=a c=123 剩下不要但不是不存在
//若输入1234 a 123o.262
 a=1234 b=空格 c=空的没有录入
```

getchar函数

1. 函数**没有参数。**
2. 函数的值就是从输入设备得到的字符，
3. 只能接收**一个**字符。
4. 如果想输入多个字符就要用**多个函数。**
5. 不仅可以从输入设备获得一个**可显示的字符**，而且可以获得**控制字符。**
6. 用getchar函数得到的字符可以赋给一个字符变量或整型变量，也可以作为表达式的一部分。如，putchar(**getchar()**);将接收到的字符输出。

选择结构

1. 选择结构：根据条件进行判断真假，执行不同的操作。

if语句

1. 简单if语句的一般形式为

```
if(表达式){
    <语句块>
}
else{
    <语句块>
}
```

   - if语句中的"表达式"**可以是关系表达式、逻辑表达式、甚至是数值表达式。**其中最直观、最容易理解的是关系表达式。
   - **如果if后面的语句没有括号就表示一条语句是if的语句块**
   - **else子句为可选的**，即可以有，也可以没有。
   - 如果条件**为真**，语句**执行一个语句或一组语句**；
   - 如果条件**为假**，则**执行if语句后面(else)的语句**（如果有)。
   - **闰年判定**：年份能被4整除**但**(&&)不能100整除，**或者**(||)年份能被400整除

```
//编写程序，输入一个年份，判断是否为闰年
#include <stdio.h>
int main(){
    int y;
    scanf("%d",&y);
    if((y%4==0&&y%100!=0)||y%400==0)
       printf("闰年")
    else printf("平年")
}
```

2. 嵌套if
   - 匹配规则

- **有else必有if**
- 匹配：**else与它前面的、紧挨着的、未被匹配的相匹配。**
- ```
  #include "stdio.h"
  main(){
  int a=1,b=2,c=3,d=4;
  if(a=b)
   {b++;c+=2;}
  else
   b+=2;c+=3;
  d=a+b+c;
  printf("%d,%d,%d,%d",a,b,c,d);
  }
  //答案  a=2 b=3 c=8 d=13
  ```

switch语句

1. switch语句用来实现多分支选择结构
2. switch语句的作用**是根据表达式的值，使流程跳转到不同的语句。**
3. switch语句的一般形式

```
switch(表达式){  //整型/字符型/枚举类型的表达式
 case 常量表达式1:语句1;
 case 常量表达式2:语句2;
      ...
 case 常量表达式n:语句n; //不能相同
 default :语句n+1;  //位置任意
}
```

4. 在使用switch结构时应注意以下几点
   - 括号内的“表达式”，**其值的类型应为整数类型（包括字符型）。**
   - 在case后的**各常量表达式的值不能相同**，否则会出现错误；
   - case和常量表达式之间要**有空格**，如：case 10
   - 在case后，可以**省略语句**；也可以有多个语句，多个语句不用{}括起来；
   - 每个case语句后**都应该有一个break语句**；
   - 各case和default子句的先后顺序可以变动；
   - **多个case标号可以共用一组执行语句。**
   - default子句可以省 位置任意 **如果没有匹配的就执行default语句然后退出**
   - **break用于结束跳出本switch**
   - 注：**没有break的时候，只要有一个case匹配，剩下的语句都执行，直至结束switch**
5. 比较多重if和switch结构
   - 多重if结构用来实现两路、三路分支比较方便，而switch:结构实现三路以上分支比较方便。
   - 在使用switch结构时，应注意分支条件要求是简单数据类型(int、char)表达式，而且case语句后面**必须是常量表达式。**
   - 有些问题只能使用多重if结构来实现，例如要判断一个值是否处在某个**区间**的情况。
6. 例子
   - ```
     //1.输入一个整数，判断是正数、负数还是零
     #include "stdio.h"
     main(){
     int a;
     scanf("%d",&a);
         if(a>0)printf("正数");
     ```

```
     else if(a==0) printf("0");
    else printf("负数");
}
//【例5-3】给出一百分制成绩，要求输出成绩等级'A'、B'、'C'、D'、E'。90分以上为'A',80~89分为'B',70~79分为C',60~69分为'D',60分以下为E'。
#include <stdio.h
main(){
    float score;
    char grade;
    scanf("%f",&score);
    switch((int)(score/10)){
    case 10:
    case 9:grade='A';break;
    case 8:grade='B';break;
    case 7:grade-'C';break;
    case 6:grade='D';break;
    default:grade='E';
    }
printf("成绩是%5.1f,相应的等级是%cn",score,grade);
}
```

循环结构

- 需要多次重复执行一个或多个任务的问题一般用循环解决

while循环

1. while循环的一般语法

```
while(表达式)
{语句}
```

2. 工作原理
    - 计算表达式的值，当值为真（非0）时，执行循环体语句，一旦条件为假，就停止执行循环体。如果条件在开始时就为假，那么不执行循环体语句直接退出循环
    - 若循环体为一条语句不用花括号
3. 规则
    - 循环中使用的变量 必须初始化
    - 循环体中的语句必须实现修改循环条件的值，避免死循环。
    - 例题

```
int a=0,b=1;
while(a++) b++;
printf("%d,%d",a,b);//1,1

int a=2,b=1;
while(a--)b++;//a=-1,b=3
printf("%d,%d",a,b);
//-1,3
//求1到100数的和
#include <stdio.h>
int main(){
    int i=1,sum=0;
    while(i<=100){
```

```
        sum+=i;
        i++;
    }
    printf("sum=%d",sum);
    return 0;
}
```

do while循环

1. do while循环的一般语法

```
do
{
语句;
...
}while(表达式);
```

2. 工作原理
   - 它**先执行循环体中的语句，然后再判断条件**是否为真如果**为真则继续循环；如果为假，则终止循环。**
   - 例子

```
//求1到100数的和
#include <stdio.h>
int main(){
    int i=1,sum=0;
    do{
        sum+=i;
        i++;
    }while(i<=100);
    printf("sum=%d",sum);
    return 0;
}
```

3. 区别
   - while循环**先判断后执行。**
   - do-while循环**先执行后判断**，循环将**至少**执行**一次。**

for循环

1. for语句的一般形式

```
for(表达式1;表达式2;表达式3)
{循环体} //若为1条语句不用花括号
```

2. 规则
   - **表达式1设置初始条件，只执行一次。可以为零个、一个或多个变量设置初值执行**
   - **表达式2循环条件表达式，用来判定是否继续循环。**在每次执行循环体前先执行此表达式，决定是否继续执行循环 **为真执行循环体为假退出循环**
   - **表达式3作为循环的调整器**，例如使循环变量变化，它是在执行完循环体后才进行的。
3. for循环的表达式
   - **for语句中的各个表达式都可以省略 注意：分号分隔符(；)不能省略**
   - 省略表达式1:省去循环变量赋初值，应在for语句之前有给循环变量赋初值语句。

- **省略表达式2**:即不判断循环条件，**表达式2恒真**，**应在循环体内设法结束循环（一般用if+break),**否则将成为死循环。
- 省略表达式3：省去修改循环变量值的操作，需要在循环体内具有更改循环变量的值的语句。
- 省略三个表达式：都省略，for前须有变量初始化语句，循环体内须有控制循环结束的语句，须有改变变量值的语句。

4. 循环的嵌套
   - 循环的嵌套：一个循环体内又包含另一个完整的循环结构
   - 3种循环(while循环、do..while循环和for循环)可以互相嵌套
5. 例子

```
//编程找出100以内的素数
#include <stdio.h>
int main(){
    int i,k,flag;
    for(i=2;i<=100;i++){
        flag=0;
        for(k=2;k<i;k++){
            //除了自己和1能除尽其他的数除尽了就不是素数
            //素数又叫质数（prime number），有无限个。质数定义为在大于1的自然数中，除了1和它本身以外不再有其他因数。
            if(i%k==0)
             flag=1;
        }
        if(flag==0){
            printf("%d ",i);
        }
    }
}
//求1到100数的和
#include <stdio.h>
int main(){
    int i,sum=0;
    for(i=1;i<=100;i++){
        sum+=i;
    }
    printf("sum=%d",sum);
    return 0;
}
//从键盘输入两个正整数，求其最大公约数和最小公倍数。
//两个数的乘积=两个数的最大公约数*两个数的最小公倍数
#include <stdio.h>
int main(){
    int m,n,k=1,i;
    scanf("%d%d",&m,&n);
    for(i=2;i<=m;i++){
        if(m%i==0&&n%i==0) k=i;
    }
    printf("%d和%d最大公约数%d,最小公倍数%d",m,n,k,m*n/k);
}
//编写程序，求1-1/2+1/3-1/4+....-1/100；#include <stdio.h>
 int main() {
    float sum=0;
    int i,sign=1;
    for(i=1; i<=100; i++){
        sum+=sign*1.0/i;
        sign=-sign;//通过加符号的方式
```

```
    }
    printf("sum=%f",sum);
}
```

break语句和continue语句

1. break语句提前终止循环
    - break语句可以用来从循环体内跳出循环体，即提前结束循环，接着执行循环下面的语句。**跳出所在的循环**
    - break语句用于do-while、while、for循环中时，**可使程序终止循环而不执行循环后面的语句**
    - **break语句通常在循环中与if语句一起使用。若条件值为真，将跳出循环**
    - 如果已执行break语句，就**不会执行循环体中位于break语句后的语句**
    - **注意：break语句只能用于循环语句和switch语句之中，而不能单独使用。**
    - 例子

```
//求200到300之间所有的素数
#include <stdio.h>
int main(){
    int i,j;
    for(i=200;i<300;i++){
        for(j=2;j<i;j++){
            if(i%j==0){
                break;
            }
        }
        if(j==i)
        printf("%d ",i);
    }
}
```

2. continue语句提前结束本次循环
    - 用于**提前结束本次循环**，可以用continue语句
    - continue语句一般用在循环里
    - **while和do...while会跳转到循环条件； for会跳转到“表达式3”。**
    - continue语句的作用是**跳过循环体中剩余的语句而执行下一次循环**
    - 对于while和do-while循环，continue语句执行之后的动作是条件判断：**对于for循环，随后的动作是变量更新。**
3. break语句和continue语句的区别
    - break跳出整个循环 （大跳）
    - continue跳出本次循环（小跳） （还继续执行循环）

函数

函数介绍

1. **一个C程序由一个或多个程序模块组成**，每一个程序模块作为**一个源程序文件**
2. **一个源程序文件由一个或多个函数以及其他有关内容（如指令、数据声明与定义等)组成**
3. **c程序的执行是从main函数开始的**
4. **函数是C程序的基本单位**

5. **所有函数都是平行的**即**函数不能嵌套定义。函数间可以互相调用，但不能调用main函数main函数是被操作系统调用的。**
6. 函数分类（从用户使用的角度）
    - **系统函数**（库函数）：由系统提供的，用户不必自己定义，可直接使用
    - **用户自己定义的函数**：解决用户专门需要而定义的函数。
7. 函数分类（函数的形式看）
    - **无参函数。**在调用无参函数时，主调函数不向被调用函数传递数据。
    - **有参函数。**主调函数在调用被调用函数时，通过参数向被调用函数传递数据。

函数的定义和返回值

1. 定义函数
    - 在程序中用到的所有函数**“先定义，后使用”**
    - 定义函数应该包括的功能**指定函数名字**、**函数返回值类型**、函数实现的功能以及参数的个数与类型
2. 一般形式

```
函数类型 函数名（参数表）//无参函数参数表无内容
{
    函数体 //实现函数功能的语句序列，注意即使1条语句也要花括号括起来
}
```

3. 函数的类型
    - 函数的返回值类型，可以是int char double...
    - **函数只有动作没有返回值**此时函数的类型为 **void**
    - **函数定义时的函数类型**和**return语句返回类型不一致以定义类型为准**
    - **一般函数省略类型默认为int类型** （主函数省略类型默认为void）
4. 函数名
    - 见名晓意，符合命名规则的标识符
    - 标识符后面**紧跟着小括号(参数表)**，**确定是函数名**而不是一般变量
5. 参数表（）
    - **定义时，参数表后无其他符号**
    - （）无参的参数表
    - （参数类型1 参数1，参数类型2 参数2，...）有参的参数表
    - 注意：**即便所有参数是相同类型，也要各自体现参数类型**

```
int sum(int a,int b,int c)
```

    - **函数名后边的参数表为形式参数表**（所有参数的类型和名必须完整体现）
    - **而调用函数时，函数名后面的参数为实际参数表**（只需要体现出实际参数，不需要体现类型）

```
int fun(int a,char b,double c){} //形式参数表
fun(10,m,1.25*d)//实际参数表
```

6. 函数体{ }
    - 定义时函数体后无其他符号
    - 函数功能实现的部分，由0条以上的执行语句构成
7. return语句

- 在函数体中，**由return语句反回值**，一个函数**可能有多条return语句**，**执行到哪个return哪个return起作用**
- 若**return语句返回值类型和函数类型不一致，以定义时函数类型为准**

函数的声明

1. 函数声明：**对即将用的函数先声明**，交代有此函数，可以用，**函数完整定义一般在主函数之后**
2. 一般形式

```
//函数类型 函数名(参数表); //函数头部;
int fun(int a,int b);
//等价
int fun(int,int); //更合适
```

- 注意：函数声明，函数类型 ，函数名，参数表中参数类型和个数必须与定义函数时的类型，**函数名。参数表（参数的类型和个数）一 一对应**
- ```
  以下属于正确的函数原型声明的是
  1.void f(int x,y);
  2.int f(int x;int y);
  3.int f(x,y);
  4.int f(int x,int y)
  5.void f();
  6.int f(int);
  答案 5,6
  ```

函数调用

1. 函数调用语句
   - 直接调用

     ```
     函数名();//这时不要求函数带回值，只要求函数完成一定的操作
     ```

   - 函数表达式

     ```
     c=max(a,b);//这时要求函数带回一个确定的值以参加表达式的运算
     ```

   - 函数参数

     ```
     m=max(a,max(b,c));//函数调用作为另一个函数调用时的实参
     ```

2. **赋值兼容**是指实参与形参类型不同时，能**按不同类型数值的赋值规则进行转换。**
3. 函数调用的过程
   - 在定义函数中指定的形参
   - **在未出现函数调用时**，它们**并不占内存**中的存储单元。
   - **在发生函数调用时，函数的形参才被临时分配内存单元。**
   - 将**实参**的**值传递给对应形参。**
   - **在执行函数期间**，由于形参已经有值，就可以利用形参进行有关的运算。
   - **通过return语句将函数值带回到主调函数。**应当注意**返回值的类型**与**函数类型一致。**如果函数不需要返回值，则不需要return语句。这时函数的类型应定义为void类型。
     ```

- **调用结束，形参单元被释放。**
  注意：实参单元仍保留并维持原值，没有改变。如果在执行一个被调用函数时形参的值发生改变，**不会改变**主调函数的实参的值。因为实参与形参是两个不同的存储单元。

4. **实参向形参的数据传递是"值传递"，单向传递，只能由实参传给形参而不能由形参传给实参。**实参和形参在内存中占有不同的存储单元，实参无法得到形参的值
5. 例题

```
对于函数
void f(int a,float b){.....}
以下是正确调用的是
1.f(32,28);
2.int i;i=f(21,23.2);
3.f('a',43.1);
4.f('a','b');
5.f(34.2);
6.f(34,45,1.2);
7.f();
答案1 3 4
```

参数传递

1. 调用函数时，由**实际**参数向**形式**参数传递，**单向**传递
   - 此时：形式参数会分配空间，存储实际参数传递过来的值
2. 函数调用三步
   - 参数传递 实——>形 **单向**传递
   - 执行函数体
   - 返回到调用函数的位置
3. 参数个数(调用函数时)
   - 函数调用时实参个数，看最外层括号所包含的逗号的个数，实参个数=逗号个数+1

函数的递归调用

1. 递归调用定义：在调用一个函数过程中又出现**直接或间接地调用该函数自身**
2. 递归调用的基本形式
   - 直接递归调用：函数体的执行部分出现本函数自身的调用
   - 间接递归调用：其函数体内并没有出现自身的调用，而是出现在被调用函数的函数体执行部分当中
3. 满足条件
   - 要解决的问题转化为一个**新问题，方法与原问题相同**，只是**处理对象有规律地递增或递减**
   - 必须有**递归结束**的出口，**一般借助if条件实**现。
   - ```
     //求n!
     long fac(int n){
         long f=1;
         int i;
         for(i=1;i<=n;i++){
             f=f*i;
         }
         return f;
     }
     //用递归调用方法计算n!。 fac(n)=n*fac(n-1)
     long fac(int n){
     ```
     ```

```
    if(n==1) return 1;
    else return n*fac(n-1);
}
```

全局变量和局部变量

1. 从变量起作用的范围分为
    - **局部变量**：在**函数内或复合语句内定义**的变量
    - **全局变量**：在**函数之外定义**的变量
2. 局部变量
    - 程序执行过程中，动态地分配存储空间
    - 调用函数时，系统会自动给局部变量分配存储空间，调用结束时就自动释放空间。
    - 函数中定义的变量、函数形参，复合语句中定义的变量等。
    - **在函数内定义的变量是局部变量。**
3. 全局变量
    - 程序的编译单位是源程序文件，一个源文件可以包含一个或若干个函数。
    - **在函数之外定义的变量称为外部变量**，外部变量是**全局变量**（也称**全程变量**）
    - **全局变量**可以为本文件中其他函数所共用。它的有**效范围为从定义变量的位置开始到本源文件结束。**
4. 全局变量与局部变量同名时的使用
    - 在**局部变量的作用范围内，使用同名的局部变量而全局变量暂时被屏蔽**，即**同名全局变量在该函数中不起作用。**
    - **全局变量不是看哪用而是看哪定义**
5. 变量的存储方法
    - 从变量值**存在的时间（即生存期）**来观察，有的变量在程序运行的**整个过程都是存在**的，而有的变量则是**在调用其所在的函数时才临时分配存储单元**，而在函数**调用结束后该存储单元就马上释放了，变量不存在**了。
    - 也就是说，变量的存储有两种不同的方式：**静态存储方式**和**动态存储方式。**
    - **静态存储方式：是指在程序运行期间由系统分配固定的存储空间的方式。**
    - **动态存储方式：则是在程序运行期间根据需要进行动态的分配存储空间的方式**
6. 静态局部变量(static局部变量)
    - 用关键字**static**进行声明
    - 希望函数中的某些局部变量在函数调用结束后不消失而继续**保留原值**，即其占用的存储单元不释放，在下一次再调用该函数时，该变量已有值（上一次函数调用结束时的值），指定该局部变量为“静态局部变量”
    - 特殊：**第一次遇见分配空间，存储值，后边再次出现，会保留原有操作过的结果**

存储类别

1. 在C语言中，每一个变量和函数都有两个属性：**数据类型**和**数据的存储类别**
2. **存储类别**指的是数据在内存中存储的方式（如静态存储和动态存储）
3. 根据变量的存储类别，可以知道变量的作用域和生存期。
4. 定义变量的一般形式如下：**存储类别标识符 数据类型标识符 变量名**；
5. C的存储类别包括4种
    - 自动的(auto)：**未说明存储类型的变量都是auto变量。**
    - 静态的(static)：分配在静态存储区，若无赋初值，编译时系统自动赋值为0
    - 寄存器的(register)：
        - 只有局部自动类变量和形式参数可以作为寄存器变量，其它不可以

        - 寄存器的数目有限，只能说明少量的寄存器变量
        - 该类变量没有地址，不能用求地址运算
    - 外部的(extern)：在函数的外部定义的全局变量

编译预处理

1. 编译**预处理是**指对**源程序编译之前调用预处理程序**，对源程序中的预处理命令（以#号开始的命令）进行识别与处理，**产生一个新的源程序**，然后再由编译程序对预处理后的源程序进行常规的编译，得到可执行的目标代码。
2. **编译预处理语句以#号开头**以它占用一个单独的书写行，**语句不用分号作为结束符。**

宏定义

1. 宏定义是用一个标识符来代表一个字符串，其中标识符为宏名
2. 编译预处理时会**把宏名替换成它所代表的的字符序列**
3. 宏定义有两种形式
    - 不带参数的宏定义
    - 带参数的宏定义
4. 形式：**#define 宏名 宏体**

不带参数的宏定义

1. 宏名通常用大写字母，宏定义一般写在程序开头
2. 宏名是用户定义的标识符，不得与程序中的其它名字相同
3. **宏名不是变量，不占有内存**
4. 在宏定义时，如果一行写不下时，需要在上一行结尾**加上“\”来表示续行。**
5.)宏定义时如果在行末有分号，则替换时连分号一起进行替换。宏定义时注意替换后的实际情况（适当加括号）；
6. **宏定义可以嵌套，替换时是逐层替换**

```
#define PI 3.14
#define ADDPI(PI+1)
#define TWOADDPI (2*ADDPI)
```

7. 宏名不能替换字符串中与宏名相同的字符串，也不能替换标识符中的字符串
8. 标识符被宏定义后，在取消这次宏定义之前，不允许重新对它进行宏定义 取消宏定义的命令为：#undef 宏名
9.
```
//替换
#include "stdio.h"
#define s 2;
main(){
int a=2,b=3;
a*=s+b;
printf("%d",a);
} //结果为4 注意有结束符号替换的时候一起带过去
```

带参数的宏定义

1. 一般格式为： #define宏名（形参表列） 宏体
2. 其中，define是宏定义命令符，宏名是一个标识符，形参表列是用逗号分隔开的标识符序列，每个标识符称为形式参数，宏体是包含形参的字符序列。
3. 说明
   - 带参数的宏定义中，“宏名”与“（形参表列）”之间不加空格。否则，就成为不带参数的宏定义了
   - 如果实参是表达式，宏替换时实参表达式替换形参，不求解实参表达式的值。为了能够正确地进行宏替换，一般将宏体和各形参都用圆括号括起来。
   - ```
     #define MX(x,y)x*y
     #define MY(x,y)(x)*(y)
     #define NX(x,y)(x)>(y)?(x):(y)
     #define NY(x,y)((x)>(y)?(x):(y))
     void main(){
     int val,a,b;
     scanf("%d%d",&a,&b);
     val=MX(a+4,b+3); //宏替换后为val=a+4*b+3
     printf("MX=%d\n",val);
     val=MY(a+4,b+3);//宏替换后为va=(a+4)*(b+3)
     printf("MY=%d\n",val);
     val=NX(a+4,b+3)*5;//宏替换后为val=(a+4)>(b+3)?(a+4):(b+3)*5
     printf("NX=%d n",val);
     val=NY(a+4,b+3)*5;//宏替换后为val=(a+4)>(b+3)?(a+4):(b+3)*5
     printf("NY=%d n",val);
     }
     ```

文件包含

1. “文件包含”指一个C源文件通过#include命令将另一个C源文件的全部内容包含进来
2. 一般格式为

   ```
   #include<文件名>
   或者
   #include "文件名"
   例如：
   #include <stdio.h>
   #include "file1.c"
   ```

3. 文件包含控制行可出现在源文件的任何地方，通常都放在文件开头
4. 一个include只能指定一个包含文件。如果需要把多个文件包含到源文件之中，必须使用多个#include命令行
5. 当被包含文件修改时，凡包含此文件的所有源程序都必须重新进行编译

条件编译

1. 一般情况下，源程序中所有的语句都参加编译，但有时也**希望根据一定的条件去编译源文件的不同部分**，这就是条件编译。**条件编译使得同一源程序在不同的编译条件下得到不同的目标代码。**
2. ```
   1.格式一
   #ifdef表达式
       程序段1
   #else
   ```

```
    程序段2
#endif
或者：
ifdef表达式
 程序段1
 作用：求解表达式的值，如果为非零（真），编译程序段1，否则编译程序段2。可以根据需要省略#else部分。
2.格式二
#ifdef标识符
    程序段1
#else
    程序段2
#endif
作用：当标识符已经被#define命令定义过，则编译程序段1，否则编译程序段2，可以根据需要省略#c1sc分支。
3.格式三
#ifndef标识符
程序段1
#else
程序段2
#endif
作用：当标识符未被define命令定义过，则编译程序段1，否则编译程序段2可以根据需要省略else分支。
```

数组

一维数组

定义一维数组

1. 定义：数组是**相同类型**的**数据元素**构成的**有限集合**
2. 数组：**先定义**数组，**后使用数组里的元素**
3. 一般形式

```
类型 数组名[长度];
```

- 类型： char double float 结构体类型
- 数组名：符合命名规则的标识符
- 长度：**整型的常量表达式不能包含变量** 表明数组中数据的个数

```
int a[10];
```

数组初始化

1. ```
   //在定义数组的同时，给各数组元素赋值
   int a[10]={0,1,2,3,4,5,6,7,8,9};

   int a[10]={0,1,2,3,4};
   //部分赋值，相当于
   int a[10]={0,1,2,3,4,0,0,0,0,0};

   int a[10]={0,0,0,0,0,0,0,0,0,0};
   //相当于
   int a[10]={0};
   ```
   ```

```
int a[5]={1,2,3,4,5};
//可写为
int a[]={1,2,3,4,5};//省略长度，后必须有初始化列表

//使用
a[0] a[1] a[2]

//其他的初始化情况：
int arr[10]={10,9,8,7,6,5,4,3,2,1,0};//错误！越界了
int arr[10]={9,8,7,5};//正确，后面的6个元素初始化为0
int arr[]={9,8,7};//正确：元素个数为3
int arr[]={}; //错误，到底是几个元素？
```

2. 注意
    - 数组下标从0开始
    - 数组元素在内存中按顺序连续存放
    - 数组名代表数组的首地址，是地址常量，即score与&score[0]的值相同
    - 编译系统对数组元素的下标值不作任何检查，当下标值超出有效变化范围时，也不会给出错误信息，但此时已经出现越界错误。因此需要编程时注意。

引用数组

1. 使用数组一般形式

```
数组名[下标表达式]
数组名[i]
```

    - 下标表达式（整型常或表达式）
    - 0<= i <=长度-1
    - 注意：只能**逐个引用**数组元素，**不能一次整体调用**整个数组全部元素的值

赋值

1. 只能逐个对数组元素进行操作（字符数组例外）
2. 一般维数组的处理用**一重循环**来实现，用**循环变量**的值对应**数组元素的下标**

```
 int i,a[10];
    for(i=0;i<10;i++){ //i就是数组小标值 小于数组下标值
        scanf("%d",&a[i]);//输入
        printf("%d",&a[i]);//输出
    }
```

    - **scanf("%s",数组名) 可以输入一个字符串**
3. 例子

```
//编写程序，定义一个整型数组a[10],输入各个元素的值，求各个元素的平均值，(结果小数点后保留2位)
#include <stdio.h>
int main(){
    int a[10],i,sum=0;
    float avg;
    for(i=0;i<10;i++){
        scanf("%d",&a[i]);
        sum+=a[i];
```

```
    }
    avg=sum/10.0;
    printf("%.2f",avg);
    return 0;
}
//编写程序，定义一个整型数组a[10],输入各个元素的值，求其中的最大元素及其下标。
#include <stdio.h>
int main(){
    int a[10],i,z=0;
    for(i=0;i<10;i++){
        scanf("%d",&a[i]);
         if(a[i]>a[z]){
            z=i;
        }
    }
    printf("xb=%d,max=%d",z,a[z]);
}
```

二维数组

定义数组

1. 形式

```
数据类型 数组名[][];  [内为整型常量表达式]
```

   - **可以理解为int a[行] [列]**

```
例：int a[3][4];  //12个整型的元素
    double b[5][6];
    char c[5][20]
    []代表维度在二维数组中,第一[]个为行数第二个[]为列数
    存储空间按行优先顺序存储
    int a[3][4];  a[0][0]  a[0][1] a[0][2] a[0][3]
                  a[1][0]  a[1][1] a[1][2] a[1][3]
                  a[2][0]  a[2][1] a[2][2] a[2][3]
```

2. 数组元素：数组名[行下标] [列下标] 0<=行下标 列下标<=长度-1

数组初始化

1. 按行赋初值

```
int a[2][3]={{1,2,3},{4,5,6}}; 123 456
int a[2][3]={{1},{4,5,6}}; 100 456
```

2. 按数组元素存放顺序赋初值

```
int a[2][3]={1,2,3,4,5,6}; 123 456
int a[2][3]={1,2,3}; 123 000
```

3. **省略行数**（根据初值个数和列声明自动确定行数 **不可以省略列**）

```
int b[][3]={1,2,3,4,5,6,7,8,9,10};4行123 456 789 1000
int c[][3]={{1,2},{3}}; 120 300
```

使用数组

1. 一般二维数组的处理用**二重循环**来实现，用循环变量的值控制数组元素的下标

```
  int a[3][4],i,j;
    for(i=0;j<3;i++){
        for(j=0;j<4;j++){
            scanf("%d",&a[i][j]);//输入
            printf("%d",&a[i][j]);//输出
        }
    }
```

2. 例题

```
//3编写程序，定义一个二维数组a[10][10],输入每个元素的值求其中最大元素及其下标
#include <stdio.h>
int main(){
    int a[10][10],i,j,k=0,l=0;
    for(i=0;i<10;i++){
        for(j=0;j<10;j++){
            scanf("%d",&a[i][j]);
            if(a[k][l]<a[i][j]){
                k=i;
                l=j;
            }
        }
    }
    printf("zdys=%d,xb[%d] xb[%d]",a[k][l],k,l);
    return 0;
}
//4.编写程序，定义一个二维数组a[10][10],输入每个元素的值求对角线上元素的和 对角线一个是两个坐标相等 例a[10][10] 就是a[0][0] a[..][..]   a[9][9]，一个是两个坐标相加等于坐标数-1 例a[10][10] 就是a[0][9] a[9][0]
#include <stdio.h>
int main(){
    int a[10][10],i,j,sum;
    for(i=0;i<10;i++){
        for(j=0;j<10;j++){
            scanf("%d",&a[i][j]);
            if(i==j||i+j==9){
                sum+=a[i][j];
            }
        }
    }
    printf("%d",sum);
}
```

字符数组

1. 字符数组的定义
   - C语言中没有独立的字符串类型，字符串的存放与处理是在字符型数组中进行的。字符数组同其他数组一样，可以是一维的，也可以是多维的。
   - 用来存放字符数据的数组是字符数组。在字符数组中的一组元素内存放一个字符。

     ```
     char c[10];
     c[0]='i';c[1]='c';c[2]='m';c[3]='b';
     ```

   - 由于字符型数据是以整数形式(ASCI代码)存放的，因此也**可以用整型数组来存放字符数据。**
2. 字符数组的初始化

   ```
   char c[10]={'d','d','e','w','e'};
   ```

   - 如果在定义字符数组时不进行初始化，则数组中各元素的值是不可预料的
   - 如果花括号中提供的初值个数（即字符个数）大于数组长度，则出现语法错误
   - **如果初值个数小于数组长度**，则只将这些字符赋给数组中前面那些元素，**其余的元素自动定为空字符**（即**'\0'**）。
   - 如果提供的初值个数与预定的数组长度相同，**在定义时可以省略数组长度**，系统会自动根据初值个数确定数组长度。
3. 字符数组的引用
   - ```
     数组名[i]   0<= i <=长度-1
     ```

指针

1. 如果在程序中定义了一个变量，在对程序进行编译时，系统就会给这个变量分配内存单元。编译系统根据程序中定义的变量类型，分配一定长度的空间。内存区的每一个字节有一个编号，这就是“**地址**”。
2. 由于通过地址能找到所需的变量单元，可以说，**地址指向该变量单元**，**将地址形象化地称为“指针”**。
3. 指针：**指针是一种用来存放某个变量或函数的地址值的变量**
4. 存储单元的**地址**和存储单元的**内容**是**两个不同的概念。**
5. 指针：**指针是一种用来存放某个变量或函数的地址值的变量**

变量指针

定义指针

1. 定义类型(先定义后使用)

   ```
   数据类型 * 指针变量名;
   ```

   - 数据类型：指针可以指向的数据元素的类型
   - 指针变量名：用户自定义名称
   - 指针变量前面的 * 表示该变量为指针型变量。**指针变量**名则不包含 *

     ```
     例:int *p;//p为指针
     ```
     ```

2. **指针变量中只能存放地址**（指针），不要将一个整数赋给一个指针变量
3. **一个指针，在赋值指向变量之前，是无法进行取内容操作的**

使用指针

1. 用变量的地址给指针变量赋值（使用取地址运算符&）

```
例:int *p;//p为指针
   int n;
   int *p=&n;
```

   - 注：**只能用同类型变量的地址进行赋值！**
   - **指针赋值只能是同类型变量的地址**
2. 赋空值

```
int *p;
p=0;
p=NULL;
```

   - 注意：**赋空值不指向任何变量**
3. 指向指针的指针

```
int **p,*s,k; //定义的p是一个指向指针的指针变量
k=20;
S=&k;
p=&s;
//*s==k  **p==k
```

4. 对指针变量的操作
   - 直接访问：通过变量本身对变量进行存取的方式。
   - 间接访问：通过指针变量实现对变量的访问方式。
   -
```
//间接访问的过程是：
//由指针变量得到变量的地址，根据该地址找到变量的存储区，再对该存储区的内容进行存取，从而实现了对变量的间接访问
int a,*p=&a;
a=10 /*直接访问，结果a的值为10*/
*p=*p+10;/*间接访问，结果a的值为20*/
```

5. & *运算符
   - &取地址运算符
   - **& 和 *互逆**

```
&(*p)→p
*(&a)→a
```

   - 区分 p &p *p

```
double d=3;
double *p=&d;
用指针可以间接的操作它所指向的数据元素即*p==d
prinf("%d,%d,%d,%d",d,p,&p,*p)
d,d的地址,指针的地址,d
*p=10;间接改变 d=10
//p为地址变量
//*p表示的是地址所指向的值
//&p表示取p地址变量的地址
```

6. 指针的运算
   - ++/-- * 运算级别相同 **同时出现，从右向左进行计算**
   - ```
     int a=[10],*p;
     p=a;
     a[0]=1;
     a[1]=2;
     *p++; //的含义:用的是*p,然后指针P进行自增1 然后指针p指向a[1]
     (*p)++; //的含义:指针p指向的的变量a的值进行自增1 a[0]的值自增1变成2
     ```

一维数组指针

定义数组指针

1. 数组名：**指针常量**(值是数组的起始地址,也为数组第一个元素的地址)**不可改变**
2. 指针：**变量**，让其指向数组后，可以借助指针变量操作数组元素

```
//可以用一个指针变量指向一个数组元素
int a[10]={1,3,5,7,9,11,13,15,17,19};
int *p;
p=&a[0]; //等价于p=a;  int*p=a  int *p=&a[0];
//*p==a[0];  *p==*a==a[0];
```

3. 注意：**数组名a不代表整个数组**，只**代表数组首元素的地址。**
4. p=a;的作用是把a数组的首元素的地址赋给指针变量p,而不是"把数组a各元素的值赋给p"
5. 注意 **a[10] 数组a不可以做左值更改 a是一个常量**

使用数组指针

1. 指针数组运算
```

运算	含义
p++	p指向**后一个**数组元素
p--	p指向**前一个**数组元素
p+i	指向**当前元素后第i个数组元素**
p-i	指向**当前元素前第i个数组元素**
p-q	两个指针之间相差的元素个数
关系运算	p<q：p指的元素在前，p>q：p在后 p==q:指向同一元素
a+i	数组a中下标为i的元素的地址

- 两个指针相减，如p1-p2(只有p1和p2都指向同一数组中的元素时才有意义)（两个指针的运算只能相减）**结果是两个地址之差除以数组元素的长度** 注意：**两个地址不能相加**
- 两个相同类型的指针可以进行关系运算，其结果反映的是两个指针所指向的地址之间的**前后位置关系。**
- 两个指针pa和pb,它们所指向的数据类型相同，对于**关系式"pa<pb",当结果为真时，表示pa在pb之前**，否则表示在相等或之后。另外，**指针也可以与NULL进**行相等的判断，以说明指针**是否为空指针。**
- ```
  1.p++;//使p指向下一元素a[1]
  2.*p;//得到下一个元素a[1]的值
  3.*p++://由于++和·同优先级，结合方向自右而左，因此它等价于*(p++)。先引用p的值，实现*p的运算，然后再使p自增1*/
  4.*(p++)//先取*p值，然后使p加1
  5.*(++p)://先使p加1，再取*p
  6.++(*p)//表示p所指向的元素值加1，如果p=a,则相当于++[a],若a[0]的值为3，则a[0]的值为4。注意：是元素a[0]的值加1，而不是指针p的值加1
  7.*(p--)//相当于a[i--],先对p进行*运算，再使p自减
  8.*(++p)//相当于a+i,先使p自加，再进行*运算
  9.*(--p)//相当于a[--i]，先使p自减，再进行*运算
  ```

2. 数组元素引用
   - 下标法：如a[i]形式
   - 指针法：如 *(a+i) 或 *(p+i)
   - ```
     p=a; //条件  0<i<长度
     p[i]==a[i];
     *(a+i)==*(p+i);
     //数组指针可以减法不可以加法
     //减法比如p=a  a[p-a]==a[0]
     //a数组是常量不能改变但可以进行表达式 例 a+5 没有赋值可以
     //注：*p++是先取p指向的变量，在将p向后移动 如果当前p指向a[3],则p[2]并不代表a[2],而是a[3+2],即a[5]。
     ```

二维数组指针

1. 定义简单指针
   - 即根据二维数组存储结构定义的指针，它指向二维数组中的一个元素。
2. 二维数组的存储
   - 元素排列的顺序是按行存放

```
int a[3][3]
int*p1=&a[0][0];
```

3. 定义行指针

```
int(*p)[4]=a;
//表示P是一个指针变量，他指向二维数组的一行，每行都是含有4个元素的一维数组，且指针p被初始化为指向二维数组a的首行，
//p+i表示指向二维数组第i行的指针。
//*(p+i)表示指向第i行第一个元素的指针，与p[i]等价
//(p+i)+j表示指向第i行第j个元素的指针，与p[i]+j等价。
//*(*(p+i)+j)表示第行j列元素的值，与p[i][j]们等价。
```

字符串指针

定义字符串

1. 一维数组存放字符串
   - 在C语言中没有专门的字符串变量，通常用一个字符数组来存放一个字符串
   - 在C语言中，是将**字符串**作为**字符数组来处理的。**
   - 字符数组和字符串的区别是：字符串的末尾有一个空字符'\0'
2. 定义

```
1:char str[]={'s','t','i','n','l','\0'};
2:char str1[]={'s','t','i','n','\0'};
3:char str2[10]={"string"};  //占数组长度是字符串长度加1
4:char str3[10]="string";
5:char str4[]="string";
```

   - C语言中字符串常量被隐含处理成以 \0 结尾的无名字符型一维数组
   - 注意：字符串数组不能超过数组长度 数组名是常量不可以赋值
   - 不能把一个字符串赋值给一个字符变量

```
char*sp,s[10];
s="Hello!"不合法
//原因：s和字符串"Hello!"都是地址常量，不可以互相赋值
sp="Hello!"//合法，表示sp指向字符串“Hello!"首地址
sp="hell" //不合法指针直接赋值字符串此时指针是字符串常量
char str[10]="Hello!",str2[10]
str1=str2//不合法 str1和str2都是字符数组首地址常量
```

3. **两个连续的单引号表示空字符**，空字符它**不占内存**，故其**不能称之为字符常量**，常量是要有地址的，**空字符串是两个连续的双引号**，但**因其占一个字节**，故其是**字符串常量。**

字符数组的输入和输出

1. 字符串的输出
   - 逐个字符输入输出。用格式符%c输入或输出一个字符

```
#include <stdio.h>
int main(){
char c[15]={'I','a','m','a','s','t','u','d','e','n','t','.'};
int i;
for(i=0;i<15;i++)
printf("%c",c[i]);
printf("\n");
return 0;
}
```

   - 将整个字符串一次输入或输出。用%s格式符，对字符串(string)输入输出

```
#include <stdio.h>
int main(){
char c="China";
printf("%s",c);
return 0;
}
```

     - 输出的字符中**不包括结束符'\0'**
     - 用“%s”格式符输出字符串时，**printf函数中的输出项是字符数组名**，而**不是数组元素名**
     - 如果数组长度大于字符串的实际长度，也只输出到**遇\0'结束**
     - 如果一个字符数组中包含一个以上\0'，**则遇第一个\0'时输出就结束**
2. 字符数的输入

```
char str1[5],str2[5],str3[5];
scanf("%s%s%s",str1,str2,str3);
```

   - 如果利用一个scanf函数输入多个字符串，则应在**输入时以空格分隔**
   - scanf函数中的输入项如果是字符数组名，**不要再加地址符&**，因为在C语言中**数组名代表该数组第一个元素的地址**（或者说数组的起始地址）

字符串处理函数

1. 输出字符串的函数 **puts(字符数组)**
   - 作用：将一个字符串（以\0结束的字符序列)输出到终端。
   - 用puts函数输出的字符串中**可以包含转义字符。**
   - 在**用puts输出时将字符串结束标志\0** 转换成**\n**,即**输出完字符串后换行。(自动换行**)
2. 输入字符串的函数 **gets(字符数组)**
   - 用：从终端输入一个字符串到字符数组，并且得到一个函数值该函数值是字符数组的起始地址
   - 注意 用puts和gets函数**只能输出或输入一个字符串。**
3. 字符串连接函数。strcat(字符数组1，字符数组2)
   - 作用：把两个字符数组中的字符串连接起来，**把字符串2接到字符串1的后面**，**结果放在字符数组1中**，函数调用后得到一个函数值——字符数组1的地址。

```
char str1[30]={"People's Republic of "}
char str2={"China");
printf("%s",strcat(str1,str2));
//输出：People's Republic of China
```

4. 字符串复制函数 strcpy(字符数组1，字符串2)
   - 作用：将字符串2**复制**到字符数组1中去。
   - **“字符数组1”必须写成数组名形式**，“字符串2”可以是字符数组名，也可以是一个字符串常量。
   - **不能用赋值语句将一个字符串常量或字符数组直接给一个字符数组。字符数组名是一个地址常量**，它不能改变值，正如数值型数组名不能被赋值一样。
   - 可以用strncpy函数将字符串2中前面n个字符复制到字符数组1中去。

```
strncpy(str1,str2,2);
//将str2中最前面2个字符复制到str1中，取代str1中原有的最前面2个字符。但复制的字符个数n还应多于str1中原有的字符（不包括10）。
```

5. 字符串比较函数 strcmp(字符串1，字符串2)
   - 作用：比较字符串1和字符串2。
   - 字符串比较的规则是：将两个字符串自左至右逐个字符相比（**按ASCII码值大小比较**），直到出现不同的字符或遇到\0为止。
     - 如**全部字符相同，则认为两个字符串相等**
     - 若**出现不相同的字符**，则**以第1对不相同的字符的比较结果为准。**比较的**结果由函数值带回**
   - 返回值
     - 如果字符串1与字符串2**相同，则函数值为0。**
     - 如果**字符串1>字符串2，则函数值为一个正整数。**
     - 如果**字符串1<字符串2，则函数值为一个负整数。**

6. 转换为小写的函数strlwr
   - 其一般调用格式为： strlwr(str);
   - strlwr的作用是将字符串中大写字母转换成小写字母。

7. 转换为大写的函数strupr
   - 其一般调用格式为： strupr(str);
   - strupr函数的作用是将字符串中小写字母转换成大写字母。

8. 测字符串长度的函数 strlen(字符数组)
   - 作用：测试字符串长度的函数。函数的值为**字符串中的实际长度 不包括\0'在内**

```
#include <stdio.h>
#include <string.h>
int main(){
char str[10]="China";
printf("%od,%d n",strlen(str),strlen("China"));
}
//输出5,5
```

9. 字符数组应用举例

```
//写程序，输入一个字符串，输出该字符串的长度
#include <stdio.h>
int main(){
```

```
    char a[100];
    int i;
    gets(a);
    while(a[i]!='\0'){
        i++;
    }
    printf("%d",i);
    return 0;
}//字符串编程的标准模板！！！（套路）
//编写程序，输入一个字符串，分别输出其中数字字符、字母字符、以及其它字符的个数
#include <stdio.h>
int main(){
    char a[100];
    int i=0,j=0,k=0,n=0;
    gest(a);
    while(a[i]!='\0'){
        if(a[i]>='0'&&a[i]<='9'){
           j++;
        }else if((a[i]>='A'&&a[i]<='Z')||(a[i]>='a'&&a[i]<='z')) k++;
        else n++;
        i++;
    }
    printf("%d,%d,%d",j,k,n);
//编写程序，输入一个字符串，拷贝到另一个字符串中，不能使用strcpy
#include <stdio.h>
int main(){
    char a[100],b[100];
    int i=0;
    gets(a);
    while(b[i]=a[i]){
        i++;
    }
    puts(b);
    return 0;
}
//编写程序，输入一个字符串，逆序输出该字符串。比如：输入"hello"输出："olleh
#include <stdio.h>
int main(){
    char a[100];
    int i=0;
    gets(a);
    while(a[i]!='\0'){
        i++;
    }
    while(i>=0){
        printf("%c",a[i--]);
    }
    return 0;
}
```

字符串和指针

1. 有名字符串和无名字符串
    - 在程序设计中，可以**采用字符型数组**存放和处理字符串，也可以使用**字符指针**来创建一个字符串

```
char str[]="hello! ";  //用字符型数组存放字符串
char*str="hello! ";    //用字符型指针指向字符串 常量不能修改
char *str1;
str1 ="hello!";
```

- 前者可以称为**有名字符串**，**数组名**就是该**字符串的名**字。后者称为**无名字符串**它是一个**字符串常量**，**保存在永久存储区中。str指针**只存放了该字符串常量的**首地址。**
- 用**字符型数组存放字符串时**，**可以对字符串进行修改**，而用**字符指针指向**的字符串是**不能修改**的。

2. 字符指针变量和字符数组的比较

- 存储单元的内容。编译时**为字符数组分配若干存储单元**，以存放各元素的值，而对**字符指针变量，只分配一个存储单元**(Visual C+为指针变量分配4个字节)

```
char *a;
scanf("%s",a);//错误的

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

- **指针变量的值**是可以改变的，而**字符数组名代表一个固定的值，不能改变**
- **字符数组中各元素的值**是**可以改变**的（可以对它们再赋值），**但字符指针变量指向的字符串常量中的内容是不可以被取代的**（不能对它们再赋值）

```
char a[]="House";
a[2]='r'; //正确

char *b="House";
b[2]='r'; //错误
```

3. 字符指针数组

- 字符指针数组：一个数组中的各个元素都是字符指针

```
char *数组名[整型常量表达式];
char *a[5];//定义一个长度为5的字符指针数组，5个数据元素都是字符型指针
int *数组名[整型常量表达式];
int *a[5];//定义一个长度为5的整型指针数组，5个数据元素都是整型型指针
```

4. 指向字符串的指针

- **字符串指针指向是该指针输出字符串的首字符**

```
char *ps,a[]="We change live";
int n=10;
ps=a;
ps=ps+n;
printf("%s\n",ps); //输出结果live
printf("%s\n",a);  //输出结果 We change live
```

5. 例题

```
// 大小写转换
```

```c
#include <stdio.h>
int main(){
    char a[80];
    char *p=a;
    printf("请输入一行文本按回车结束\n");
    gets(a);
    while(*p!='\0'){ //==*p
        if(*p>='A' && *p<='Z'){
            *p+=32; //大写转小写
            p++;
        }else{
         printf("请输入一行文本按回车结束\n");
         gets(a);
        }
    }
    printf("%s",a);
    return 0;
}
#include <stdio.h>
int main(){
    char a[80];
    char *p=a;
    printf("请输入一行文本按回车结束\n");
    gets(a);
    while(*p){ //==*p
        if(*p>='A' && *p<='Z'){
            *p+=32; //大写转小写
            p++;
        }else{
         printf("请输入一行文本按回车结束\n");
         gets(a);
        }
    }
    printf("%s",a);
    return 0;
}
```

指针与函数

1. 指针做参数传递变量
    - **指针作参数，可以将实参的地址传递给形参，从而实现由形参修改实参的效果。**
    - 函数的传递方式
        - 传值调用：不能改变实参
        - 传地址调用：可以改变实参
2. 例题

```c
#include<stdio.h>
void swap(int *,int *);
void main(){
    int a=3,b=5;
    printf("交换前：a=%d,b=%d\n",a,b);
    swap(&a,&b); //指针作为参数
    printf("交换后：a=%d,b=%d\n",a,b);
}
void swap(int *x,int *y) { //定义交换函数，形参为指针
```

```
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
//输出 a=3,b=5 a=5,b=3

#include<stdio.h>
void swap(int *,int *);
void main(){
    int a=3,b=5;
    printf("交换前：a=%d,b=%d\n",a,b);
    swap(&a,b); //指针作为参数
    printf("交换后：a=%d,b=%d\n",a,b);
}
void swap(int *x,int y) { //定义交换函数，形参为指针
    int temp;
    temp=*x;
    *x=y;
    y=temp;
}
//输出 a=3,b=5 a=5,b=5
```

- **如果函数内没有进行取值计算 *x 等就交换地址值 不改变主函数的值**
- 用指针作参数，需要注意
  - 函数参数声明为指针
  - 在函数体内使用指针的间接引用
  - 调用时，用地址作实参
- **函数返回值可以是指针**：函数的调用出现在赋值语句中，对一个指针进行赋值操作，把函数调用的结果赋值给一个指针变量

3. 数组名作为函数参数

- 用数组名作为函数参数时，在**主调函数**和**被调函数中**需要**分别定义数组**
- 要求**形参与实参是相同类型的数组**
- 当**数组名**作**为函数参数时**，**传递的是该数组在内存中的起始地址**，而**不**是**传递数组元素的值。**
- **形参数组和实参数组实际上占用同一个内存区域**，**系统不为形参数组另外分配存储单元**，因此在**定义函数时形参数组**可以在数组名后面跟一个**空的方括号，不指明长度。**

```
#include <stdio.h>
#include <stdlib.h>
void f(int a[],int n) {
    int i;
    for(i=0; i<n; i++)
        a[i]=a[i]+1;
    }
int main() {
    int a[5]= {1,2,3,4,5};
    int i=0;
    f(a,5);
    for(i=0; i<5; i++)
        printf("%d ",a[i]);
    return 0;
    }
//输出2 3 4 5 6
```

结构体

typedef

1. 格式

```
typedef 已有类型名 新类型名
```

```
//例如
typedef int INTEGER; //INTEGER m,n; 等价于int m,n;
typedef char* CHARP; //CHARP P; 等价于 char *p;
```

2. **typedef为声明新类型名的关键字，已有类型名可以是C提供的标准类型**还可以是**用户自己定义的类型名。**
3. 新类型名是用户定义的标识符，是用户为已有类型指定的新名字
4. **typedef将已存在的类型用新名代替**
5. 用typedef 声明新类型优点
    - 可以使熟悉某种语言的人在使用℃语言时沿用原来的习惯用法
    - 提高程序可读性
    - 可以简化书写，命名一个简短明确的类型名代替复杂类型表示方法

枚举

1. 枚举是一种用户定义的数据类型，它用关键字 enum 以如下语法来声明

```
enum 枚举类型名字 {名字0,…,名字n};
```

2. 枚举类型名字通常并不真的使用，要用的是在大括号里的名字，因为它们就是就是常量符号，它们的类型是int，值则依次从0到n

```
enum colors{ red, yellow,green };
```

    - 就创建了三个常量，red的值是0，yellow是1，而green是2
3. 枚举量
    - 声明枚举量的时候可以指定值（默认是0）

```
enum COLOR { RED=1,YELLOW,GREEN = 5};
```

```
#include <stdio.h>
enum COLOR {RED=1,YELLOW,GREEN=5,NumCOLORS};
int main() {
    printf("code for YELLOW is %d\n",YELLOW);
    printf("code for GREEN is %d\n",GREEN);
    return 0;
}
//输出 YELLOW 为2 GREEN为5
```

    - 枚举只是int

```c
#include <stdio.h>
enum COLOR {RED=1,YELLOW,GREEN=5,NumCOLORS};
int main() {
    enum COLOR color =0;
    printf("code for GREEN is %d\n",GREEN);
    printf("and color is %d\n",color);
    return 0;
}
//GREEN 5 color=0
```

- 即使给枚举类型的变量赋不存在的整数值也没有任何warning或error
- 虽然枚举类型可以当作类型使用，但是实际上很(bu)少(hao)用
- 如果有意义上排比的名字，用枚举比const int方便
- 枚举比宏（macro）好，因为枚举有int类型

结构体

结构体基本形式

1. 形式

```c
//类型说明形式：
struct 结构体名    // struct 结构体名=结构体类型名
{
 类型名1 结构成员名表1;
 类型名2 结构成员名表2;
 ...
 类型名n 结构成员名表n;
};
类似于图纸，不分配存储空间
```

2. 用户自定义复杂的数据类型由多个数据成员构成 **结构体**
   - 一个结构体类型可以由若干个成员的成分构成
   - ```c
     //例如
     struct Student
     {int num;
     char name[20];
     char sex;
     int age;
     float score;
     char addr[30];
     };
     ```
   - **每个成员都要有类型和名**
3. 说明
   - 结构体类型并非只有一种，而是可以设计出许多种结构体类型
   - 成员可以是另一个以定义的结构体类型（结构体的嵌套）
     ```

```
struct date
{
int year;
int month;
int day;
};
struct student{
int num;
char name[20];
char sex;
struct date birthday;
float score;
};
```

4. 定义结构体类型变量

- **先声明结构体类型，再用结构体类型定义该类型变量**（常用）

```
struct Stu1
{
 char name[50];
 int age;
};
struct Stu1 s1={"Mike",18},s2;
//struct Stu1表示结构体类型 对应的内容进行相应的初始化
```

- 边定义类型的边声明结构体变量

```
struct Stu2{
    char name[50];
    int age;
}s3={"Lily",20},s4;
```

- 不说明类型名，而直接定义结构体 类型变量

```
//其一般形式为
struct //无名
{成员表列}//变量名表列;
//创建一个无名的结构体类型。
//例如
struct
{
char name[50];
int age;
}s5={Alice”,21};
//s5为无名的结构体类型的变量
```

- 使用typedef说明一个结构体类型名，再用 新类型名定义变量

```
typedef struct //（有无皆可）
{
  char name[50];
  int age;
}STU;  //STU说明为本结构体的新类型的名
STU s6={"mike",23};//不可再写关键字struct
```

```
//可以用名
typedef struct A //（有无皆可）
{
  char name[50];
  int age;
}STU;  //STU说明为本结构体的新类型的名
STU s6={"mike",23};//不可再写关键字struct
struct A s7; //等价于 STU s7c
```

5. 总结

- **结构体类型不分配存储空间**不能对类型进行操作，类型可以定义变量
- **结构体变量为实体，则分配数据对应所有空间总和**
- **结构体类型中的成员**可以与**程序中的变量同名**，但二者不代表同一对象。例如，程序中可以定义一个变量age,它与struct student中的age是不同的变量，**互不影响。**
- **初始化时，将所有初值按照各成员的顺序排列。**
- 对结构体变量的引用具体到每一个数据成员用 . 成员指向运算符

  . **前为一般的结构体变量** —> **前为结构体变量的指针** . —> **级别最高**

```
#include <stdio.h>
struct student{
    int num;
    char name[20];
    float score;
};
int main(){
    int i;
    float sum=0;
    struct student stu;
        printf("请输入第%d个学生信息 学号 姓名 成绩\n",i+1);
        scanf("%d%s%f",&stu.num,stu.name,&stu.score);
        printf("%d%s%f\n",stu.num,stu.name,astu.score);
}
```

结构体数组

1. 定义结构体数组一般形式

```
1.struct 结构体名 {成员表列}数组名 [数组长度];
2.先声明一个结构体类型，然后再用此类型定义结构体数组：
结构体类型 数组名[数组长度];
如：struct Person leader[3];
//定义了一个结构数组leader,共有3个元素，leader[0] ——leader[2],每个数组元素都具有
struct leader的结构形式
```

- 每个数组元素都具有结构体的结构形式
- 注：结构体数组每一个元素都是结构体类型的变量
- 对结构体数组初始化的形式是在定义数组的后面加上：={初值表列};

```
struct Person leader[3]={{"Li",0},{"Zhang",0},{"Wang",0}}
```

2. 结构体数组使用

- 数组名[下标].成员名
  leader[0].成员名

- ```
  #include <stdio.h>
  struct student{
      int num;
      char name[20];
      float score;
  };
  int main(){
      int i;
      float sum=0;
      struct student stu[3];
      for(i=0;i<3;i++){
          printf("请输入第%d个学生信息 学号 姓名 成绩\n",i+1);
          scanf("%d%s%f",&stu[i].num,stu[i].name,&stu[i].score);
          sum+=stu[i].score;
      }
          printf("总成绩为%d",sum);
  }
  #include <stdio.h>
   struct STU{
      int num;
      char name[20];
      float com;
      float eng;
      float tota;
   }a[3]={{20211,"张海燕",93,79},{20212,"刘明",78,90},{20213,"李娜",85,83}};
   int main(){
      int i;
      for(i=0;i<3;i++){
          a[i].tota=a[i].com+a[i].eng;
          printf("学号%d 姓名%6s 总成
  绩%3.f\n",a[i].num,a[i].name,a[i].tota);
       }
   }
  ```

结构和指针

1. 一个指针当用来指向一个结构时，称之为**结构指针变量**
2. 一般形式为

```
//C语言允许像定义其他类型的指针一样定义结构体类型的指针。
struct 结构名*结构指针变量名;
struct student*ps;//ps可以指向struct student类型的变量
struct student stu;//定义struct student类型的变量stu
ps=&stu;
//指针ps指向变量stu我们可以通过指针ps访问它所指向的变量su(具体的说是访问su的成员)，例如：
(*ps).num引用学生变量的学号
(*ps).name引用学生变量的姓名
```

3. 通过结构指针可以访问该结构变量的成员，一般形式为
```

```
(*结构指针变量).成员名 或者 结构指针变量->成员名
struct stu a;
struct stu *pstu;
pstu=&a;
(*pstu).num或者 pstu->num ,  a.num
//(*pstu)等价a
```

- 例子

```
#include<stdio.h>
struct student {
    int num;
    char name[20];
    float score;
};
int main() {
    struct student stu= {10001,"wang",98};
        struct student *ps;
            ps=&stu;
        printf("学号：%d\n姓名：%sln成绩：%fn",(*ps).num,ps->name,ps-
>score);
        return 0;
    }
```

4. 指向结构体数组的指针

- 我们可以定义结构体指针来指向结构体变量，也可以定义结构体指针指向结构体数组中的元素

```
struct student[10];//定义结构体数组st
struct student*ps;//定义结构体指针变量ps
ps=st;
```

- 给指针变量ps赋值为数组首地址，使ps指向数组首元素st[0]

```
#include<stdio.h>
struct student {
    int num;
    char name[20];
    float score;
};
int main(){
struct student st[5]= {{10001,"wang",98},{10002,"i",85},
{10003,"zhang",56},{10004,"zhao",473},{10005,"qian",73}};
struct student *ps;
ps=st;
   printf("不及格学生名单为：\n");
for(int i=0; i<5; i++,ps++)
    if(ps->score<60) {
        printf("学号%d 姓名%s 成绩%f",ps->num,ps->name,ps->score);
    }
}
```

5. 指向结构体的指针作为函数参数

- C语言中允许用结构体变量和指向结构体变量的指针做函数参数

- 用结构体变量做实参时，采用的参数传递方式是“值传递”，即将实参（结构体变量）各成员的值依次传递给形参（结构体变量）的各个成员，形参相当于实参的副本，这种传递方式在空间和时间上开销较大
- 此外，采用这种参数传递方法，当被调用函数的形参被改变时，其值是不能返回给主调函数的实参的。所以一般较少采用这种方法，更多的是用结构体指针作函数参数，参数传递时将实参（结构体变量）的地址传递给形参（结构体指针变量）

```
#include<stdio.h>
struct student {
    int num;
    char name[20];
    float score;
};
void display(struct student *p) {
    printf("学号：%d姓名：%s成绩：%f \n",p->num,p->name,p->score);
}
int main() {
    struct student stu= {10001,"wang",98};
    struct student *ps;
    ps=&stu;
       display(ps);
}
```

联合 union

1. 存储
   - **所有的成员共享一个空间**
   - 同一时间只有一个成员是有效的
   - **union的大小是其最大的成员所占的空间**
2. 初始化
   - 对第一个成员做初始化

文件

文件的概念

1. 概念：指存储在外部介质上数据的集合
2. 分类（从用户或者操作系统使用的角度（逻辑上）把文件分为）
   - 文本文件
   - 二进制文件
3. 介绍
   - 文本文件（又称ASCII文件）
     - 基于字符编码的文件，每一个字节放一个字符的ASCII代码。如果要求在外存上**以ASCII代码形式存储**，则需要在**存储前进行转换**
   - 二进制文件
     - 基于值编码的文件，数据在内存中是以**二进制形式**存储的，如果不加转换地输出到外存，就是二进制文件
   - **字符**一律以**ASCII**形式存储
   - **数值型**数据既可以用**ASCII**形式存储，也可以用**二进制**形式存储
4. 存取方式

- **顺序存取**
  - 每当打开这类文件，进行读或写操作时，总是从文件的开头开始，从头到尾顺序地读或写
- **直接存取**
  - 可以通过调用C语言的库函数去指定开始读或写的字节号，然后直接对此位置上的数据进行读，或把数据写到此位置上

5. 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（如文件的名字、文件状态、文件当前位置等。)这些信息保存在一个结构体变量中。该结构体类型由系统声明（不需要用户定义），取名为：**FILE**
   - 在程序中可以直接用**FILE类型**名定义变量。
   - 文件指针（包含在stdio.h头文件中）
   - 格式：**FILE * 指针变量名**；（FILE必须大写）

```
FILE *fp
```

文件操作常用函数

1. 文件操作的4个基本步骤
   - 步骤1：利用FILE**定义文件类型指针。**
   - 步骤2：**打开文件**
   - 步骤3：对文件进行读写或定位等相关**操作。**
   - 步骤4：**关闭文件**

```
#include <stdio.h>
#include <stdlib.h>
void main() {
    char ch;
    FILE *fp; //定义文件指针变量；步骤1
    fp=fopen("letter.txt","w");//新建并打开一个文件；步骤
    if (fp==NULL) { //判断文件打开成功与否
        printf("\n Opening file error");
        exit(0);
    }
    for (ch='A'; ch <='Z'; ch++)
        fpute(ch,fp); //将变量ch中的字符写入文件；步骤3
    fpute('\n',fp); //最后写入一个换行符
    fclose(fp); //关闭文件；步骤4
}
```

2. fopen函数的使用格式

```
FLE*fopen(char*filename,char*mode); //函数说明形式
fopen(文件名,使用文件方式); //函数调用形式
```

```
//说明：若要打开的文件在当前目录下，可以只使用文件名若不在当前目录下，则应该给出文件的路径。
fp=fopen("E:\\test\file8_1.txt","r");
```

   - fopen函数调用后**返回一个指向FILE的指针**，如果**不能打开**文件，函数**返回空指针NULL**

- 文件路径：**当前路径直接用文件名 其他路径需说明：注意用 **
使用方式	含义
r	以**只读**方式打开一个文本文件
r+	以**读/写**方式打开一个文本文件
w	以**只写**方式建立一个**新的文本文件**
w+	以**读/写**方式建立一个**新的文本文件**
a	向文本文件末尾追加数据（**打开或新建**）
a+	以读/写方式打开一个文本文件
rb	以只读方式打开一个二进制文件
rb+	以读/写方式打开一个二进制文件
wb	以只写方式建立一个新的二进制文件
wb+	以读/写方式建立一个新的二进制文件
ab	向二进制文件末尾追加数据
ab+	以读/写方式打开一个二进制文件

3. 关闭文件
   - 文件的关闭(fclose函数)
   - fclose（文件指针);
   - 返回值：fclose函数也带回一个值，当顺利地执行了关闭操作，则返回值为0，否则返回非0。(**0表示成功 非0表示有错误**)
   - **文件使用完之后应关闭文件防止文件丢失**