C语言知识要点

iceman@zju.edu.cn

用Turbo C编程的步骤

(1) 编辑：进入DOS方式，转到TC的安装目录，运行Turbo C，编辑源程序。
①进入DOS方式：在Windows2000或Windows XP下进入DOS方式只要"开始->运行"再输入command。
②转到TC的安装目录：如果TC安装在D:\TC下面，则在进入DOS方式后，再输入以下命令：

```
d:
 cd \tc
```

③运行Turbo C：只要再输入以下命令：

```
tc
```

以上3步可以简化为：”我的电脑”-> D:盘 -> TC文件夹 -> 双击TC.EXE
④编辑源程序：直接输入源程序，输入完毕，按F2(File->Save)保存。
(2) 编译(compile)
选择菜单Compile->Compile to OBJ，编译产生目标文件。如果有错则转第(1)步。
(3) 连接(link)
选择菜单Compile->Link EXE file，连接产生EXE文件。如果有错则转第(1)步。
(4) 运行(run)
选择菜单Run->Run，运行EXE文件。按Alt+F5(Run->User screen)观看程序输出结果。

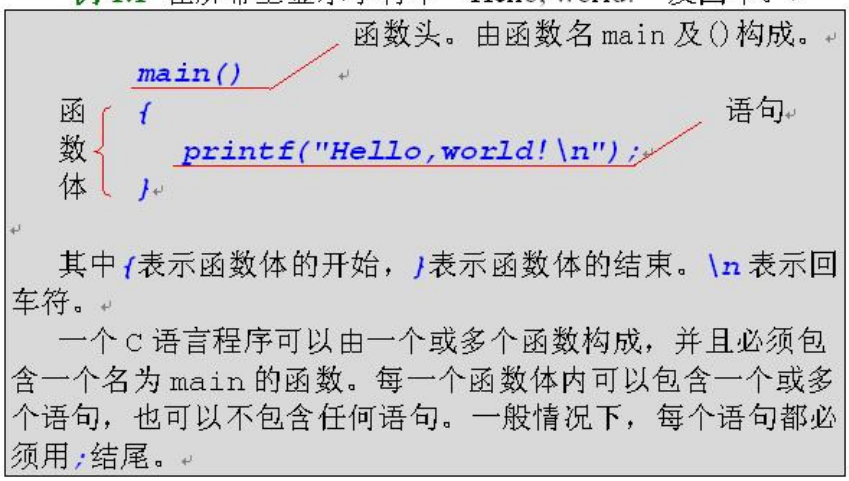
程序与程序设计

program(程序): A sequence(系列) of instructions(指令) that can be executed(执行) by a computer. The term can refer to the original source code(源代码) or to the executable (machine language) version(可执行程序). Also called software.

programming(程序设计): The art and science of creating computer programs. Programming begins with knowledge of one or more programming languages, such as Basic, C, Pascal, or assembly language. Knowledge of a language alone does not make a good program. Much more can be involved(涉及), such as expertise(专门知识) in the theory of algorithms(算法), user interface(用户界面) design, and characteristics(特性) of hardware devices. Computers are rigorously(严格) logical machines, and programming requires a similarly(类似) logical approach(方法) to designing, writing (coding) (编写代码), testing, and debugging(查错) a program. Low-level languages, such as assembly language, also require familiarity(熟悉) with the capabilities of a microprocessor and the basic instructions built into it. In the modular(模块的) approach advocated by many programmers, a project(工程) is broken into(分成) smaller, more manageable(可管理的) modules-stand-alone functional units that can be designed, written, tested, and debugged separately before being incorporated(合并) into the larger program.

例1.1 C程序的基本结构

例 1.1 在屏幕上显示字符串“Hello, world!”及回车。



其中{表示函数体的开始，}表示函数体的结束。\n 表示回车符。

一个 C 语言程序可以由一个或多个函数构成，并且必须包含一个名为 main 的函数。每一个函数体内可以包含一个或多个语句，也可以不包含任何语句。一般情况下，每个语句都必须用;结尾。

例1.3 输入变量x与y的值，输出其中较大者的值。(分支结构)

```
#include <stdio.h>
void main()
{
   int x, y, max;  /* 定义3个整型变量 */
   scanf("%d %d", &x, &y); /* 输入x与y */
   if (x>=y)  /* 若x大于等于y */
      max=x;
   else        /* 否则，即当x小于y时 */
      max=y;
   printf("max=%d\n", max); /* 输出max */
}
```

例1.4 编程求出1+2+3+…+100的和(循环结构)

例 1.4 编程求出 1+2+3+...+100 的和(循环结构)

```
main()
{
   int i, sum;     /* 定义 2 个整型变量 */
   i = 1;
   sum = 0;
   while(i<=100) /* 若 i 小于等于 100 则循环 */
   {
      sum = sum + i;
      i = i + 1;
   }
   printf("sum=%d\n", sum); /* 输出 sum */
}
```

例1.5输入变量a与b的值，输出其中较大者的值；

再输入变量m与n的值，也输出其中较大者的值。

```
int max(int x, int y) /* 先定义一个函数max */
{                          /* x与y是函数的参数 */
   int z;                  /* 函数内部定义一个变量z*/
```

```
    if (x>=y)
       z = x;
    else
       z = y;
    return z;              /* 返回函数值 */
}

main()
{
    int a, b, max_ab;
    int m, n, max_mn;

    scanf("%d %d", &a, &b);
    max_ab=max(a,b);
    printf("max_ab=%d\n", max_ab);
    scanf("%d %d", &m, &n);
    max_mn=max(m,n);
    printf("max_mn=%d\n", max_mn);
}
```

运算符：[P. 331] [P. 85]

(1) 算术运算符： + - * / %
x = 5 % 3 x=2

(2) 关系运算符： == != > >= < <=

(3) 逻辑运算符： && || !
并且 或者 非

要表达代数中 a>b>c这种条件，应该写成：
if(a>b && b>c)
不能写成if(a>b>c)

复合语句

例如：输入 float 类型变量 a、b 的值，计算 c=a/b。若 b=0 则输出“divided by zero”，若 b≠0 则计算 c 的值并输出。

```
main()
{
   float a, b, c;
   scanf("%f %f", &a, &b);
   if(b != 0)   这里不能有分号
   {
      c = a/b;
      printf("c=%f\n", c);
   }  这里也不能有分号，因为{}隐含表示一个语句的结束
   else
      printf("Divided by zero");
}
```

注意：上述程序中 if 后面所跟的由一对{}括起来的语句称为复合语句。复合语句可以看作是一个整体，即可以看作是一个语句，并且后面不用再加分号。

函数原型

原型(prototype)：对函数的参数个数、参数类型、以及函数的返回值类型加以说明。

如果把max()写到main()后面，则要求把max()的函数头抄一遍并加分号(即函数原型)放到main前面：

```
int max(int x, int y);   函数max()的原型后面有分号
main()
{
   int maxab;
   maxab = max(10,20);
}
int max(int x, int y)      函数头后面无分号
{
   int z;
   if (x>=y)
      z = x;
   else
      z = y;
   return z;
 }
```

库函数原型

自定义函数与库函数都有函数原型。
查库函数的原型：把光标移到函数名下，按Ctrl+F1
所有.h文件都在TC\Include文件夹中。
在math.h中有以下这句话：

```
double  sqrt  (double x);
```

~~double sqrt (double x);~~ 如果删除它...

```
#include <math.h>        ←则必须包含math.h
#include <stdio.h>
main()
{
   int y;
   y = sqrt(4);
   printf("y=%d\n", y);
}
```

错误分析：

```
y=2x; 错误 --> y=2*x 正确

y=5/9; 结果y=0，因为这是整除。

scanf("%d\n", &x);  输入格式中不要有\n

float x;
scanf("%f", &x); 不能用%d格式

double y;          lf:long float=double
scanf("%lf", &y);  double类型输入时不能用%f

printf输出double变量时可以用%lf，也可以用%f。

scanf("%d %d", &x, &y); 输入100,200 错误
```

应该输入100　　200回车　或 100回车200回车

scanf("%d,%d", &x, &y); 输入100 200 错误
应该输入100,200回车

scanf("x=%d,y=%d", &x, &y); 则应该输入
x=100,y=200

小数格式

%.2f 表示小数点后保留 2 位

%6.1f 表示整数+小数+小数点一共=6 位
　　小数点后保留 1 位，即小数部分是 1 位
123.4　%2.1f 输出时，因为格式位数不足，故仍旧输出 123.4
123.4　%6.1f 输出时，输出空格 123.4
①float 可以达到约 7 位有效数字，double 可以达到约 17 位有效数字；
②float 类型输入及输出都用%f，double 类型输出时可以用%lf 也可以用%f，但是 double 类型输入时必须使用%lf。

单步跟踪演示

请下载step.exe后观看演示过程。

for循环

```
#include <stdio.h>
main()
{
   int i, sum=0;
   for(i=1; i<=100; i++)
      sum += i; /* 循环体 */
   printf("sum=%d\n", sum);
}
```

①在刚开始 for 循环时做(只做一次)
②每次循环前做
③每次循环后做

for 循环过程：① → ② → 循环体 → ③

上面的 for 相当于以下的 while 循环：

```
    i = 1; /* 在循环前做（只做一次）*/
    while(i<=100) /* 每次循环前做 */
    {
       sum += i; /* 循环体 */
       i++; /* 每次循环后做 */
    }
```

错误分析

scanf("%d", x); 正确写法：&x ，可以通过跟踪发现。

函数内部变量不可与函数头部变量(形参)相同：

```
int max(int a, int b)
{
   int a, b; 错误：重复定义
   int c;
```

```
    c=a+b;
    return c;
}
```

变量名不可与函数名相同

```
int max(int a, int b)
{
    ...
}
main()
{
    int x=100, y=200, max;
    max = max(x,y); 变量名不可以与函数名相同
}
```

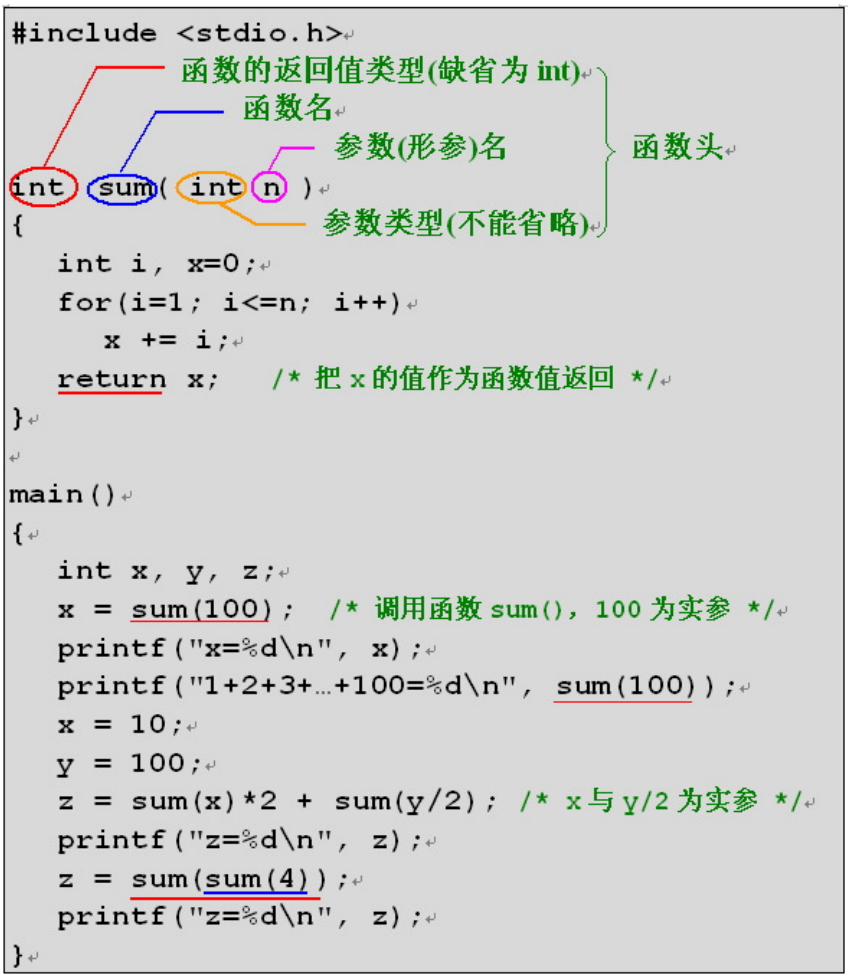
不同函数中(包括函数头及函数体)允许出现同名变量，
并且同名变量互不影响：

```
int max(int a, int b)
{
    int z;
    if(a>b) z=a; else z=b;
    a=0; b=0; 这里故意把a、b改成0，但是并不会破坏main()中a、b的值。
   return z;
}
main()
{
    int a=100, b=200, z;
    z = max(a,b);
    printf("a=%d, b=%d, z=%d", a, b, z);
}
```

函数的形参值已经通过实参代入，
因此不要对它重新赋值或输入

```
int max(int a, int b)
{   int z; 一般来说，函数内部不要输入输出
   scanf("%d %d", &a, &b); 错误
   if(a>b) z=a; else z=b;
    return z;
}
main()
{
    int a, b, z;
    scanf("%d %d", &a, &b);
    z = max(a,b);
    printf("z=%d\n", z);
}
```

函数定义与函数调用




```
#include <stdio.h>
int sum( int n )
{
   int i, x=0;
   for(i=1; i<=n; i++)
      x += i;
   return x;    /* 把 x 的值作为函数值返回 */
}

main()
{
   int x, y, z;
   x = sum(100);  /* 调用函数 sum(), 100 为实参 */
   printf("x=%d\n", x);
   printf("1+2+3+…+100=%d\n", sum(100));
   x = 10;
   y = 100;
   z = sum(x)*2 + sum(y/2); /* x与y/2为实参 */
   printf("z=%d\n", z);
   z = sum(sum(4));
   printf("z=%d\n", z);
}
```

函数定义

①若某个函数的返回值类型为void，则表示该函数无返回值，
所以该函数不可以在return时带一个值返回，
只能使用 return; 或者遇到函数体结束符 } 自动返回。

②若某个函数的返回值类型省略，则表示该函数的返回值类型为int。

③若某个函数的参数为void，则表示该函数无任何参数；
若函数的参数为空，则表示该函数的参数个数任意，参数类型任意。

函数调用

如果函数定义出现在函数调用之前，则可以省略函数原型说明。
如果函数定义出现在函数调用之后，则必须在函数调用之前说明函数原型。

if语句

① else总是与离它最近的，位于同一代码块内的，
尚未与其它else结合的if配对。
② if(*条件*) 后面只能跟一个语句，若要在*条件*成立的情况下
执行两个或两个以上的语句时，必须把这些语句用{}括起来，并且{}后不可再加分号。

③ 如果不在 if(*条件*) 语句；后面立即跟一个else，则
此if语句到此结束，即此if语句是一个单独的、不与else配对的语句。

字符类型

C 语言规定，字符等价于该字符的 ASCII 码，例如：
'A' == 65　　'0' == 48　　'1' == 49
char c='A'; 与 char c=65; 等效
字符(character)要用单引号引起来，不能用双引号。
用双引号引起来的叫字符串(string)。
'\n'==10　'\t'==9 都是一个字符
char c='A';
c++;　此时 c='B'
用 getchar()可以输入 1 个字符，还可以用 scanf()
输入：
char c;
scanf("%c", &c); 相当于 c=getchar();
用 putchar()可以输出 1 个字符，也可以用 printf()
输出：
char c='A';
printf("%c", c); 相当于 putchar(c);
以上会输出 A 这个符号，若要输出'A'的 ASCII 码，则：
printf("%d", c); 此时输出 65

switch语句

```
#include <stdio.h>
void main()
{
   int x, y;
   char operator;
   float z;
   int ok=1;
   printf("请输入\n");
   scanf("%d%c%d", &x, &operator, &y);
   switch(operator)
   {
   case '+':
        z = x + y;
        break;
   case '-':
        z = x - y;
        break;
   case '*':
        z = x * y;
        break;
   case '/':
        if(y!=0)
           z = (float)x / y;
        else
        {
           printf("被零除!\n");
           ok=0;
        }
        break;
   default:
```

```
        printf("错误的运算符！\n");
        ok=0;
    } /* switch语句结束 */
    if(ok==1)
       printf("%d%c%d=%f\n", x , operator, y, z);
}
```

while循环
输入一行字符(回车结束)，把大写字母转化成小写，其它字符照样输出。

```
#include <stdio.h>
void main()
{
    char c;
    c = getchar();
    while(c!='\n')
    {
       if (c>='A' && c<='Z')
          c += 32;
       putchar(c);
       c = getchar();
    }
}
```

do...while循环
输入一行字符(回车结束)，把大写字母转化成小写，其它字符照样输出。

```
#include <stdio.h>
void main()
{
    char c;
    do
    {
       c = getchar();
       if (c>='A' && c<='Z')
          c += 32;
       putchar(c);
    } while(c!='\n');
}
```

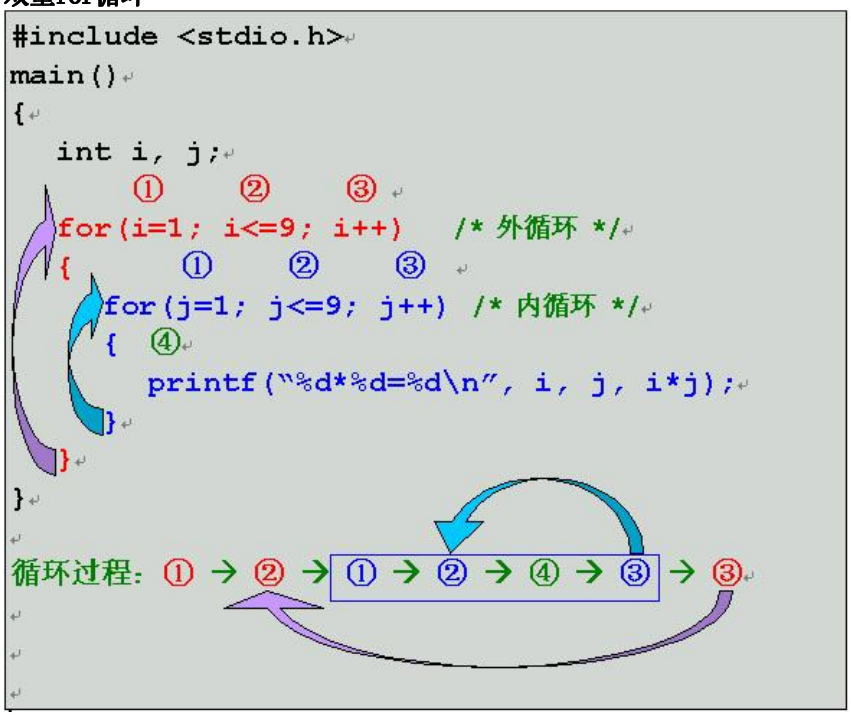
do...while循环

由计算机随机生成一个1到100之间的整数，用户输入一个数进行猜测，
若猜不中则继续输入另一个数直到猜中为止。计算机对用户输入分别
作出"太大"、"太小"及"猜中"的判断。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void main()
{
    int computer_number, my_number;
    srand((unsigned)time(NULL));     /* 随机数初始化 */
    /* TC中可以简写为：randomize(); */
    computer_number=rand()%100+1;    /* 产生一个[1,100]的随机数 */
    /* TC中可以简化为: computer_number=random(100)+1; */
    do
    {
        printf("Please input a number\n");
        scanf("%d", &my_number);
        if(my_number > computer_number)
```

```
            printf("Too big!\n");
        else if(my_number < computer_number)
            printf("Too small!\n");
        else
            printf("You have got it!\n");
    } while(computer_number != my_number);
}
```

双重for循环



[P.79 例4-9] 输出所有小于500的素数(解法1)

例 2. 输出所有小于 500 的素数。[P.79 例 4-9]

```
#include <stdio.h>
main()
{
   int i, x, count=0;
   for(x=2; x<500; x++)
   {
      for(i=2; i<=x-1; i++)
      {
         if(x%i==0) break;
         /* 若 x 能被 i 整除，则跳出内循环 */
      }
      if(i==x) /* 若 i==x，则肯定不是中间跳出 */
      {
         printf("%d\n", i);
         count++;
      }
   }
   printf("count=%d\n", count);
}
```

[P.79 例4-9] 输出所有小于500的素数(解法2)

```
#include <stdio.h>
main()
{
   int i, x, count=0;
   int prime;
   for(x=2; x < 500; x++)
   {
      prime = 1;          /* 先假定x是素数 */
      for(i=2; i <= x-1; i++)
      {
         if(x%i==0)
         {
            prime = 0;   /* 若x能被i整除，则x不是素数。*/
            break;       /* 跳出内循环 */
         }
      }
      if(prime != 0)     /* 若x为素数 */
      {
         printf("%d\n", x);
         count++;
      }
   }
   printf("count=%d\n", count);
}
```

break

在1到1000之间找出第一个能被3、7、13整除的数。

```
#include <stdio.h>
void main()
{
   int i;
   for(i=1; i<=1000; i++)
   {
      if(i%3==0 && i%7==0 && i%13==0)
         break;
   }
   if(i>1000)
      printf("Not found!\n");
   else
      printf("%d\n", i);
}
```

continue

找出[100,200]范围内所有各位数字之和为奇数的奇数。

```
#include <stdio.h>
main()
{
   int i, x, sum;
   for(i=100; i<=200; i++)
   {
      if(i%2==0)
         continue;
      sum=0;
      x=i;
      while(x!=0)
      {
```

```
            sum = sum + x%10;
            x = x / 10;
        }
        if(sum%2==0)
            continue;
        printf("%d ", i);
    }
}
```

用break跳出两层循环

```
int x, a, b;
int f;
for(x=6; x<=100; x=x+2)
{
    f = 0;
    for(a=2; a<=x-1; a++)
    {
        for(b=2; b<=x-1; b++)
        {
            if(prime(a)!=0 && prime(b)!=0 && a+b==x)
            {
                printf("%d = %d + %d\n", x, a, b);
                f = 1;
                break;
            }
        }
        if(f==1)
            break;
    }
}
```

数据的存储

① bit：位
② byte：字节
1 字节 = 8 位
1个字节最多可以表示256(2的8次方)个数：

二进制	十进制	十六进制
0000 0000	0	0x00
0000 0001	1	0x01
0000 0010	2	0x02
0111 1111	127	0x7F
1111 1111	255	0xFF

③ word：字
1字 = 2 字节 = 16 位
一个字最多可以表示65536(216)个数：

二进制	十六进制	十进制
0000 0000 0000 0000	0x0000	0
0000 0000 0000 0001	0x0001	1
0000 0000 0000 1111	0x000F	15
0000 0001 0000 0000	0x0100	256
1111 1111 1111 1111	0xFFFF	65535

④ double word：双字
1 双字 = 2 字 = 4字节 = 32 位
一个双字最多表示0x00000000到0xFFFFFFFF之间
共2的32次方（4,294,967,296）个数。

非符号数

byte、word、double word在C语言中分别对应为以下三种数据类型：

```
unsigned char           非符号的字符类型
unsigned short int      非符号的短整型
```

unsigned long int　　非符号的长整型

正数的表示：最高位=0

8位正数

最小值	最大值	进制
00000000	01111111	二进制
0x00	0x7F	十六进制
0	127	十进制

16位正数

最小值	最大值	进制
0000000000000000	0111111111111111	二进制
0x0000	0x7FFF	十六进制
0	32767	十进制

32位正数

最小值	最大值	进制
0x00000000	0x7FFFFFFF	十六进制
0	2147483647	十进制

负数的表示：二进制补码(two's complement)

8位负数

最小值	最大值	进制
10000000	11111111	二进制
0x80	0xFF	十六进制
-128	-1	十进制

16位负数

最小值	最大值	进制
1000000000000000	1111111111111111	二进制
0x8000	0xFFFF	十六进制
-32768	-1	十进制

32位负数

最小值	最大值	进制
0x80000000	0xFFFFFFFF	十六进制
-2147483648	-1	十进制

符号数的表示

8位符号数(在C语言中对应的数据类型为char)

最小值	最大值	进制
10000000	01111111	二进制
0x80	0x7F	十六进制
-128	+127	十进制

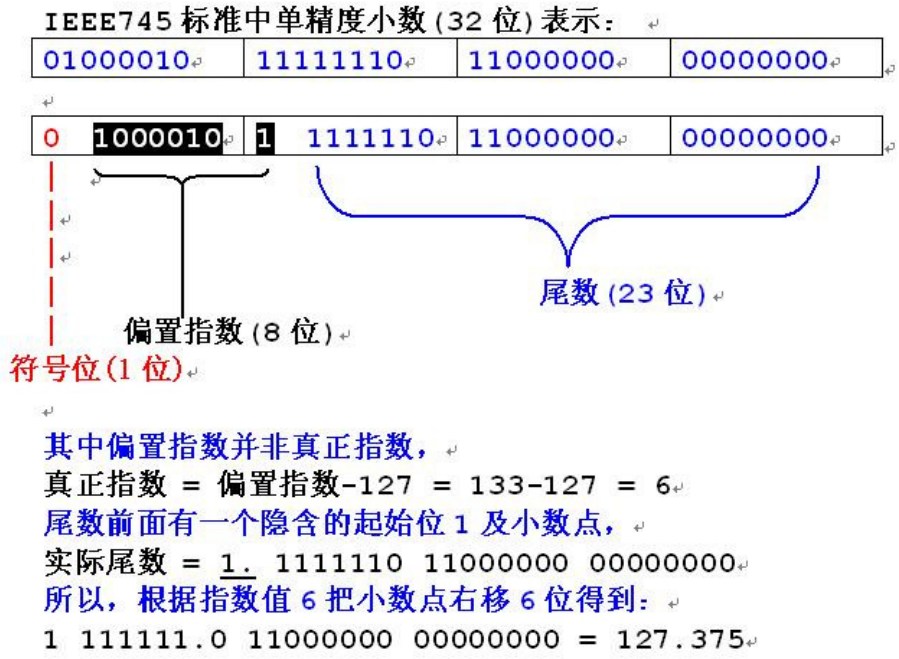
16位符号数(在C语言中对应的数据类型为short int)

最小值	最大值	进制
1000000000000000	0111111111111111	二进制
0x8000	0x7FFF	十六进制
-32768	+32767	十进制

32位符号数(在C语言中对应的数据类型为long int)

最小值	最大值	进制
0x80000000	0x7FFFFFFF	十六进制
-2147483648	+2147483647	十进制

小数的表示



其中偏置指数并非真正指数，
真正指数 = 偏置指数-127 = 133-127 = 6
尾数前面有一个隐含的起始位 1 及小数点，
实际尾数 = 1. 1111110 11000000 00000000
所以，根据指数值 6 把小数点右移 6 位得到：
1 111111.0 11000000 00000000 = 127.375

在C语言中，字符与它的ASCII码是等价的。字符在本质上是一个8位数。
例如：

```
char c;
 c = 'A';  等价于  c = 65;
 c = '0';  等价于  c = 48;
 c = 'A'+1; 等价于 c = 65+1;
 c = 'Z'-1; 等价于 c = 90-1;
```

数字0~9与字符'0'~'9'的相互转化
例如：

```
3 + '0' -->  '3'  因为 3+'0'=3+48=51='3'
 9 + '0' --> '9'  因为 9+'0'=9+48=57='9'
 '8' - '0' --> 8  因为 '8'-'0'=56-48=8
 '1' - '0' --> 1  因为 '1'-'0'=49-48=1
```

大小写字母的相互转化
大写字母 + 32 = 小写字母
小写字母 - 32 = 大写字母

基本数据类型
C的基本数据类型：

```
char     字符型
int       整型
float    单精度浮点型
double   双精度浮点型
void      空类型
```

类型修饰符：

```
signed      符号的
unsigned    非符号的
short        短的
long         长的
```

类型修饰符与基本数据类型的组合：

```
signed  char   ==  char
 unsigned char
 signed short int == short int == short
```

```
    unsigned short int == unsigned short
    signed long int == long int == long
    unsigned long int == unsigned long
    short float == float
    long  float ==  double
```

注意：在Turbo C中，int指short int；在VC++中，int指long int。
long int的输入/输出格式：%ld
int的输入/输出格式：%d
short int的输入/输出格式：%hd
输入输出格式：

```
char    %c   %d
unsigned char   %c   %u
int    %d
unsigned int  %u
long int   %ld
unsigned long int   %lu
float  %f
double   %lf
```

整型常量/常数(constant)的表示方法
前缀0表示8进制，前缀0x表示16进制，后缀U表示非符号整型(unsigned int)，
后缀L表示长整型(long int),后缀UL表示非符号长整型(unsigned long int)。

实型常量的表示方法

```
float   f;
double  d;
f = 3.14;
f = 314E-2;    /* 3.14 */
f = .314;      /* 0.314 */
d = 314.;      /* 314.0 */
printf("f=%f, d=%lf\n", f, d);
printf("%f, %lf\n", 3.14F, 3.14159);
```

加有F后缀的常量是float型，不加F后缀则被默认为是double型。
双精度浮点型数输出时的格式控制符可以用%lf也可以用%f，
但输入时必须用%lf。

字符型常量的表示方法
直接加单引号：'A'   'a'   '3'   '*'
特殊字符：'\n' '\t' '\b' '\\' '\''
          '\r' '\a'
16进制：'\x0A' '\x7' '\xFF'
8进制： '\12'   '\7'   '\0' '\377'

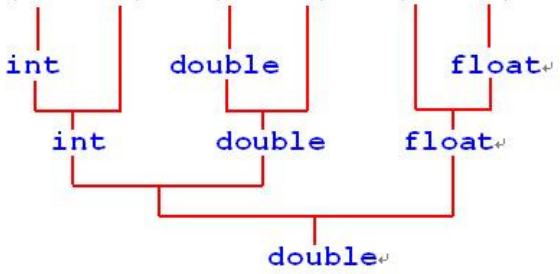
数据类型的自动转化
① 先进行char与float类型的提升(promotion)：

```
   char -> int
    unsigned char -> unsigned int
    short int -> int
    float -> double(仅适用于C89标准,不适用于C99)
```

② 小类型服从大类型：

```
   int -> unsigned int -> long int ->
    unsigned long int -> double
```

```
char ch;
int i;
float f;
double d, r;
r = (ch / i) + (f * d) - (f + i);
```



数据类型的强制转化

```
double d;
long int n;
int x = 200;
d = (double)x * x; /* d=40000.0 */
n = 200 * (long int)x; /* n=40000 */
d = (float)1/2; /* d=0.5 */
```

符号扩充与零扩充

当把位数较短的符号数转化位数较长的数要进行符号扩充

```
char c = -1;
int x;
x = c;
printf("x=%d\n", x);  /* -1 */
```

当把位数较短的非符号数转化位数较长的数要进行零扩充

```
unsigned char c = -1;
int x;
x = c;
printf("x=%d\n", x); /* 255 */
```

C语言中的真与假

(1) 若条件表达式的值≠0，则该表达式的值为真(True)；
若条件表达式的值=0，则该表达式的值为假(False)。

```
int x=50, y;
if (x)       /*  相当于if (x!=0) */
   y = 1;
else
   y = 0;
```

(2) 若关系运算和逻辑运算表达式的值为真，则该表达式的值等于1；
若表达式的值为假，则该表达式的值等于0。

```
int  x=3, y=5, z;
int a=1, b=2, c=0;
z = x == y;     /* z = 0 */
z = (x < y)+3   /* z = 4 */
z = a==b==c;    /* z = 1 */
z = a < c < b; /* z = 1 */
```

位运算符

&　|　^　~　>>　<<

```
char x=5, y=6, z;
z = x&y; /* 与运算:
              x =  0000 0101
          &)  y =  0000 0110
              ---------------
              0000 0100 结果 z = 4 */

z = x|y; /* 或运算:
              x =  0000 0101
          |)  y =  0000 0110
              ---------------
              0000 0111 结果 z = 7 */

z = x^y; /* 异或运算:
              x =  0000 0101
          ^)  y =  0000 0110
              ---------------
              0000 0011 结果 z = 3 */

z = ~x;  /* 非运算(求反):
              x =  0000 0101
           ~  x =  1111 1010 结果 z = -6 */

z = y<<1; /* 左移:
              y   = 0000 0110
           y<<1   = 0000 1100 结果 z = 12 */

z = y>>1; /* 右移:
              y   = 0000 0110
           y>>1   = 0000 0011 结果 z = 3 */
```

逻辑运算

&&　||　!

```
int x=50, y;
if (x*10-1)        /*  相当于if (x*10-1 != 0)，条件为真 */
    y = 1;
else
    y = 0;

int a=1, b=2, c=0, z;
z = a==b==c;  /* z = 1 */
z = a<c<b;    /* z = 1 */
z = a+b>c;    /* z = 1 */
z = a+(b>c);  /* z = 2 */
z = a || b;   /* z = 1 */
z = a && b;   /* z = 1 */
z = !(a>b);   /* z = 1 */
z = !a;       /* z = 0 */
```

自增自减运算符

设exp是一个表达式，则表达式exp++的值等于表达式exp的原值，
表达式exp--的值也等于表达式exp的原值；
表达式++exp的值等于exp加1以后的值，
表达式--exp的值等于exp减1以后的值。

冒泡法排序

```
#include <stdio.h>
#include <stdlib.h>
void main()
```

```
{
   int a[5], i, j, t;
   randomize();
   printf("排序前随机生成的5个数如下：\n");
   for(i=0; i<5; i++)
   {
      a[i]=10+random(99-10+1);
      printf("%d ", a[i]);
   }
   for(i=4; i>=1; i--)/*每次循环找出一个最大值*/
   {       /* 先确实a[4],再确定a[3],a[2],a[1] */
      for(j=0; j<i; j++)/* a[j]为a[i]前的值 */
      {
         if(a[j] > a[j+1])/* 若前大后小则交换 */
         {
            t=a[j]; a[j]=a[j+1]; a[j+1]=t;
         }
      }
   }
   printf("\n排序后结果如下：\n");
   for(i=0; i < 5; i++)
      printf("%d ", a[i]);
   putchar('\n');
}
```

冒泡(沉底)排序过程如下所示：

i=4,j=0	33	55	44	11	22
	33	55	44	11	22
i=4,j=1	33	55	44	11	22
	33	44	55	11	22
i=4,j=2	33	44	55	11	22
	33	44	11	55	22
i=4,j=3	33	44	11	55	22
	33	44	11	22	55
i=3,j=0	33	44	11	22	
	33	44	11	22	
i=3,j=1	33	44	11	22	
	33	11	44	22	
i=3,j=2	33	11	44	22	
	33	11	22	44	
i=2,j=0	33	11	22		
	11	33	22		
i=2,j=1	11	33	22		
	11	22	33		
i=1,j=0	11	22			
	11	22			

字符数组及整型数组的输出

```
char x[5]="ABCD";
printf("%s", x); 会输出ABCD
                 把数组x当作字符串
int a[3]={10,20,30};
printf("%s", a); 错误!
%s要求后面是char类型的数组名或者字符串常量
for(i=0; i<3; i++)    正确
   printf("%d ", a[i]);
```

输入字符串保存到字符数组的3种方法

输入字符串可以用getchar()循环实现(写法1)

```
char b[100];
char a;
int i=0;
a = getchar();  假定输入ABC回车
while(a != '\n')
{
   b[i] = a;         b[0]='A'   b[1]='B'   b[2]='C'
   i++;
   a = getchar();    'B'  'C'   '\n'
}
b[i] = '\0';         b[3]='\0'
printf("b=%s\n", b);
```

输入字符串可以用getchar()循环实现(写法2)

```
int i=0;
char c, s[100];
while( ( c=getchar() ) != '\n')
{  s[i]=c;
   i++;
}
s[i]='\0';
```

输入字符串也可以用gets()实现(写法3)

```
char s[100];
gets(s);  假定输入ABC回车，则数组s中的内容如下：
s: 'A'  'B'  'C'  '\0'
```

选择法排序

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
   int a[5], i, j, k, t;
   randomize();
   printf("排序前随机生成的5个数如下：\n");
   for(i=0; i<5; i++)
   {
      a[i]=10+random(99-10+1);
      printf("%d ", a[i]);
   }
   for(i=0; i <=3; i++)/*每次循环找出一个最小值*/
   {       /* 先确实a[0],再确定a[1],a[2],a[3] */
       k=i; /* 假定a[i]是最小值 */
      for(j=i+1; j<=4; j++)/*a[j]在a[i]后*/
      {
         if(a[j] < a[k])/*若有更小值则设为最小*/
            k=j;
      }
      /* 交换a[i]与最小值a[k] */
      t=a[i]; a[i]=a[k]; a[k]=t;
   }
   printf("\n排序后结果如下：\n");
   for(i=0; i<5; i++)
      printf("%d ", a[i]);
   putchar('\n');
}
```

把十进制字符串转化成一个整数

```
#include <stdio.h>
void main()
{
   char s[10]="1234";
```

```
    int i=0, x=0;
    while(s[i]!='\0')
    {
       x = x*10 + s[i] - '0';
         i++;
    }
    printf("x = %d\n", x);
}
```

把十六进制字符串转化成一个整数

```
#include <stdio.h>
void main()
{
    char s[10]="1A3F";
    int i=0, x=0;
    while(s[i]!='\0')
    {
       if(s[i]>='0' && s[i]<='9')
          x = x*16 + s[i] - '0';
         else
            x = x*16 + s[i] - 'A' + 10;
         i++;
    }
    printf("x = %d\n", x);
}
```

把一个整数转化成十进制字符串

```
#include <stdio.h>
void main()
{
    int x = 1234, y;
    int i, n;
    char s[10];
    y = x;
    n = 0;
    do
    {
       y /= 10;
         n++;
    } while(y!=0);
    y = x;
    s[n]='\0';
    for(i=n-1; i>=0; i--)
    {
       s[i] = y % 10 + '0';
         y /= 10;
    }
    puts(s);
}
```

把一个整数转化成十六进制字符串

```
#include <stdio.h>
void main()
{
    int x = 1234;
    char s[10];
    int i, d;
    for(i=3; i>=0; i--)
```

```
    {
        d = x % 16;
        x /= 16;
        if(d < 10)
           s[i] = d + '0';
        else
           s[i] = d - 10 + 'A';
    }
    s[4]='\0';
    puts(s);
}
```

程序跟踪技巧：

1. F7/F8 跟踪一步。F7可以进入函数内部。
2. Ctrl+F7观察变量

以十六进制形式观察变量a的值，可以输入a,x; 十进制形式则输入a,d;
观察一维数组，直接输入数组名；
观察二维数组a的各行，可以输入a[0]、a[1]…
观察形参数组a(假定元素个数为10)，可以输入*a,10d

3. F4 运行到光标处(Go to cursor)
4. Ctrl+F8 设断点(breakpoint)
5. Ctrl+F2 从头开始跟踪(program reset)

用移位法把一个短整型数转化成十六进制输出

```
#include <stdio.h>
main()
{
  unsigned short int x;
  int i;
  char a[10];
  char t[]="0123456789ABCDEF";
  scanf("%u", &x);
  for(i=0; i<4; i++)
  {
        a[i]= t[ ( x<<i*4 ) >> 12 ];
  }
  a[i]='\0';
  printf("%s\n", a);
}
```

字符串逆序

```
#include <stdio.h>
#include <string.h>
void main()
{
    char s[100], t;
    int i, len;
    puts("Please input a string:");
    gets(s);
    printf("s before change:%s\n", s);
    len = strlen(s); /* 求字符串长度，不包括'\0' */
    for(i=0; i < len/2; i++) /* 两头对称交换 */
    {
       t=s[i];
       s[i]=s[len-1-i];
       s[len-1-i]=t;
    }
    puts("s after change:"); /* 用puts()输出 */
    puts(s);
    puts("s after change:"); /* 用循环输出 */
    i=0;
    while(s[i]) /* 相当于while(s[i]!='\0') */
    {
```

```
        putchar(s[i]);
        i++;
    }
    putchar('\n');
}
```

strcpy()、strcat()、strcmp()

```
#include <stdio.h>
#include <string.h>
void main()
{
    char pass[20], temp[20];
    puts("Please input your ID:");
    gets(temp);
    strcpy(pass, "M"); /* 字符串复制 */
    /* 以上这句相当于：
       pass[0]='M'; pass[1]='\0';
    */
    strcat(pass, temp); /* 字符串连接 */
    if(strcmp(pass, "M1234")==0)/*比较*/
       puts("Welcome!");
    else
       puts("Access denied!");
}
```

统计单词个数

```
#include <stdio.h>
void main()
{
    char c0, c, s[100];
    int i, count=0;
    gets(s);
    c0=' '; /* 假定前面是空格 */
    for(i=0; i < strlen(s); i++)
    {
       c=s[i];
       /* 若当前位置不是空格且前面是空格，则
          必定是一个单词的开始 */
       if(c!=' ' && c0==' ')
          count++;
       c0=c;
    }
    printf("words=%d\n", count);
}
```

用数组作为函数参数的2个例子，注意实参、形参的用法

例1. 用myputs()输出一个字符串

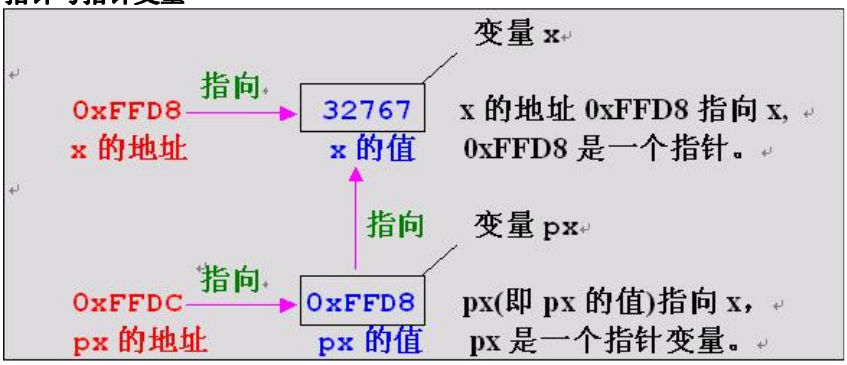
```
void myputs(char s[])
{
    int i=0;
    while(s[i]!='\0')
    {
       putchar(s[i]);
         i++;
    }
    putchar('\n');
}
main()
{
    char a[100];
    gets(a);
```

```
   myputs(a);
}
```

例2．用reverse()把一个字符串逆序

```
void reverse(char s[]);
main()
{
   char a[]="ABCD";
   reverse(a);
   puts(a);
}
void reverse(char s[])
{
   int n=strlen(s), i;
   char t;
   for(i=0; i<n/2; i++)
   {t=s[i]; s[i]=s[n-1-i]; s[n-1-i]=t;}
}
```

指针与指针变量



解读指针定义

如何解读指针的定义？先看右边，再看左边。

例如：

```
int  ( * p[3] ) [4] ;
```

①p 是数组

②数组的每个元素都是指针

③每个指针都指向一个数组

④这个数组的每一个元素都是 int

所以，p 是数组指针数组，即由数组指针构成的数组。

```
int  (*p)(int, int); /* 定义一个指针 p，它指
   向一个函数，该函数有两个 int 类型的参数，并且
   返回一个 int 类型的值。因此，p 是一个函数指针。
   p is a pointer to a function returning int.
   */
```

指针运算

```
main()
{
   int x[4]={1,2,3,4}, y1, y2, *p, *q;
   char s[]="ABC", *pc;
   double d[3]={1.2, 3.4, 5.6}, *pd;
   /******** 指向int的指针 *********/
   p = &x[0];
   printf("p=%p, *p=%d\n", p, *p);
   p++;
   printf("p=%p, *p=%d\n", p, *p);
   q = &x[3];
```

```
   printf("q=%p, *q=%d\n", q, *q);
   q -= 3;
   printf("q=%p, *q=%d\n", q, *q);
   y1 = p - q;
   y2 = (int)p - (int)q;
   printf("y1=%d, y2=%d\n", y1, y2);
   if(p > q) puts("指针p大于q");
   /********* 指向char的指针 **********/
   pc = &s[0];
   printf("pc=%p, *pc=%c\n", pc, *pc);
   pc++;
   printf("pc=%p, *pc=%c\n", pc, *pc);
   y1 = pc - &s[0];
   y2 = (int)pc - (int)&s[0];
   printf("y1=%d, y2=%d\n", y1, y2);
   /********* 指向double的指针 ********/
   pd = &d[0];
   printf("pd=%p, *pd=%lf\n", pd, *pd);
   pd++;
   printf("pd=%p, *pd=%lf\n", pd, *pd);
   y1 = pd - &d[0];
   y2 = (int)pd - (int)&d[0];
   printf("y1=%d, y2=%d\n", y1, y2);
}
```

野指针与空指针

野指针(wild pointer)

```
int a=123;
int *p;
*p = a;   此处，p是野指针。
```

当p本身没有赋值时，却对*p进行引用，此时p就是野指针。

正确用法：要对*p进行引用，必须先对p赋值。例如：

```
int a=123, b, *p=&b;
*p = a;   正确。相当于b=a
```

空指针(NULL pointer)

```
int *p;
p=NULL; 或 p=0;
```

此时，p为空指针。

NULL定义在头文件stdio.h中，它的值为0：

```
#define NULL 0
```

空指针就是不指向任何对象的指针。若p=0，则*p没有意义。

因为C语言规定，0地址不存放任何变量。

空指针的作用是为了和有效的指针进行比较。例如：

```
int * f(int a[], int x)
{  int i;
   for(i=0; i<3; i++)
   {
      if(a[i]==x)
         break;
   }
   if(i==3)
      return NULL;
   else
      return &a[i];
}
main()
{ int a[3]={10,20,30}, *p;
   p = f(a, 20);
   if(p!=NULL)
      printf("%d is found!", *p);
}
```

指针作为函数的参数(被调用者改变调用者的变量)

```
#include <stdio.h>
void swap(int *p1, int *p2);
main()
{
   int x=123, y=456;
   printf("x=%d, y=%d\n", x, y);
   swap(&x, &y);
   printf("x=%d, y=%d\n", x, y);
}
void swap(int *p1, int *p2)
{
   int t;
   t = *p1;
   *p1 = *p2;
   *p2 = t;
}
```

利用指针作参数，可以返回多个值。

```
#include <stdio.h>
void f(int, int, int *, int *);
main()
{
   int x, y;
   f(1, 2, &x, &y);
   printf("x=%d, y=%d\n", x, y);
}
void f(int x, int y, int *p1, int *p2)
{
   *p1 = x+y;
   *p2 = x-y;
}
```

数组名 ≡ 数组首元素的地址

```
#include <stdio.h>
main()
{
   int a[3]={1,2,3};
   int sum=0, i;
   for(i=0; i < 3; i++)
      sum += *(a+i);  /* 相当于sum += a[i]; */
   printf("sum=%d\n", sum);
}
```

指针[i] ≡ 数组元素(指针所指的第i个元素)

```
#include <stdio.h>
main()
{
   int a[3]={1,2,3}, sum=0, i;
   int *p;
   p = a; /* p=&a[0]; */
   for(i=0; i < 3; i++)
      sum += p[i];  /*  sum+=*(p+i); */
   printf("sum=%d\n", sum);
   for(p=a; p < a+3; p++)
      sum += *p;
   printf("sum=%d\n", sum);
   for(p=a; a < p+3; a++)  /* 错误! */
      sum += *a;
}
```

数组作参数≡指针作参数

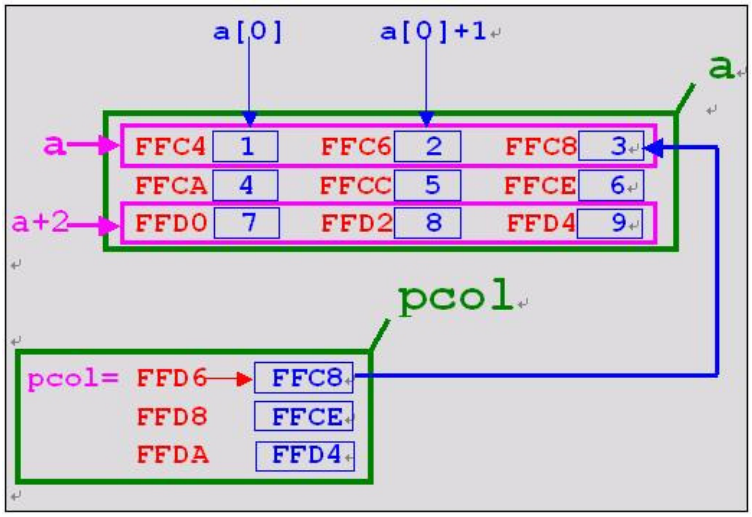
要点：当用数组名如a作函数的实参时，传递的不是整个数组，
而是该数组首元素的地址即&a[0]，即传递的是一个指针。
当用数组如int a[]作函数的形参时，这个形参并非真的是一个数组，
而只是一个指针变量即int *a。

```
#include <stdio.h>
void reverse(int a[], int n);
main()
{
   int x[4]={1,2,3,4}, i;
   reverse(x,4);     /* 用数组名作实参 */
                     /* 也可以写成 &x[0] */
   for(i=0; i < 4; i++)
      printf("%d ", x[i]);
}
                     /* 用数组作形参 */
void reverse(int a[], int n)
{                    /* 等价于 int *a */
   int i, t;
   for(i=0; i < n/2; i++)
   {
      t=a[i]; a[i]=a[n-1-i]; a[n-1-i]=t;
      /* 也可以写成:
     t=*(a+i);
      *(a+i)=*(a+n-1-i);
      *(a+n-1-i)=t;
      */
   }
}
```

指针与二维数组

```
#include <stdio.h>
main()
{
   int a[3][3]={{1,2,3},{4,5,6},{7,8,9}};
   int (*prow)[3], *pcol[3];
   int *p, **pp;
   pcol[0]=&a[0][2];
   pcol[1]=&a[1][2];
   pcol[2]=&a[2][2];
   printf("%d,%d,%d\n",
            *pcol[0], *pcol[1], *pcol[2]);
   /*****************************************/
   prow=&a[0];
   printf("%d\n", (*prow)[1] );
   printf("%d\n", *((*prow)+2) );
   printf("%d\n", *(*(prow+1)+2) );
   printf("%d\n", a[2][1]);
   printf("%d\n", *(a[2]+1) );
   printf("%d\n", *(*(a+1)+2) );
   /*****************************************/
   prow=a+1; /* prow=&a[1]; */
   printf("%d\n", *(*(prow+1)+2) );
   printf("%d\n", *(prow[1]+1) );
   printf("%d\n", *prow[1] );
   printf("%d\n", prow[1][2] );
   printf("%d\n", **(a+1) );
   printf("%d\n", *(*a+1) );
   printf("%d\n", **a+1 );
   printf("%d\n", (*a)[1] );
   /*****************************************/
   p = a[0];      /*  p = &a[0][0]; */
   pp = pcol;     /* pp = &pcol[0]; */
   p++; pp++;
```

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



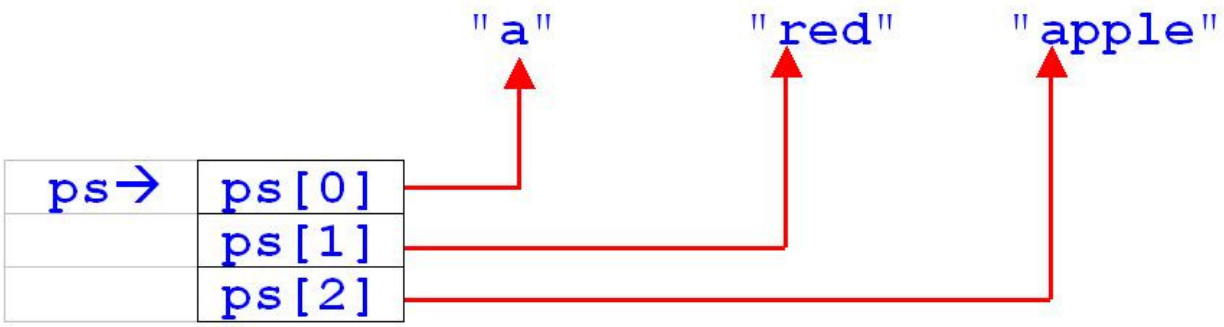
二维数组作参数≡一维数组的指针作参数

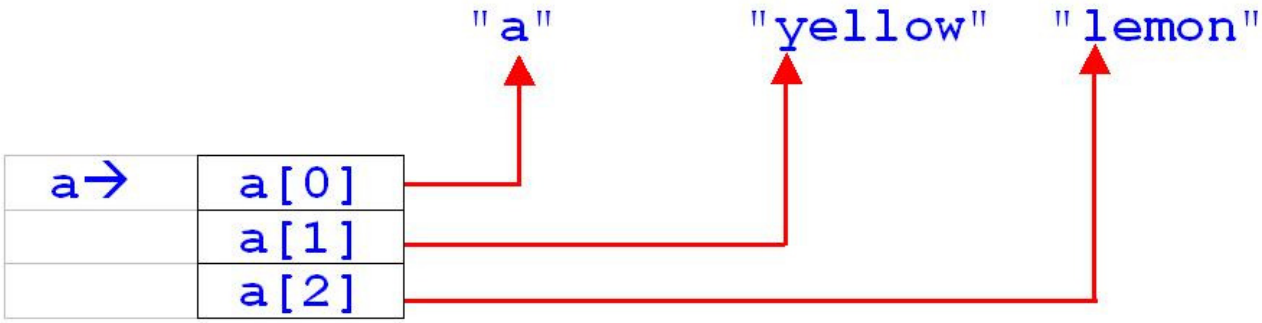
要点：当用二维数组名如a作函数的实参时，传递的不是整个数组，而是该数组首行的地址即&a[0]，即传递的是一个一维数组指针。当用数组如int a[][3]作函数的形参时，这个形参并非真的是一个数组，而只是一个指针变量即int (*a)[3]。

```
#include <stdio.h>
int f(int a[][3], int m, int n);
main()
{
    int x[2][3]={{1,2,3},{4,5,6}};
    int sum;    /* 二维数组名作实参，也可写成 &x[0] */
    sum = f( x, 2, 3 );
    printf("sum=%d\n", sum);
}
    /* 二维数组作形参，等价于 int (*a)[3] */
int f( int a[][3], int m, int n )
{
    int sum=0, i, j;
    for(i=0; i < m; i++)
        for(j=0; j < n; j++)
            sum += a[i][j];
          /* 等价于 sum += *(*(a+i)+j); */
    return sum;
}
```

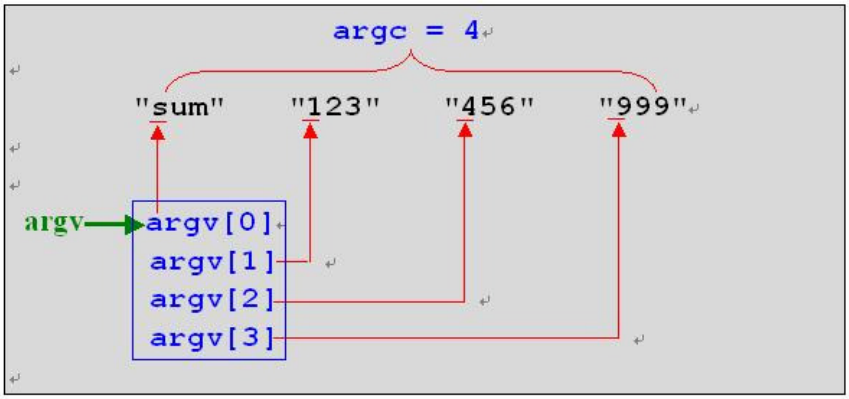
字符指针数组与二维字符数组

```
#include <stdio.h>
void main()
{
    char *ps[3]={"a", "red", "apple"};
    char a[3][7]={"a", "yellow", "lemon"};
    int i;
    for(i=0; i < 3; i++)
        puts(ps[i]);
    for(i=0; i < 3; i++)
        puts(a[i]);
    putchar(ps[2][0]); puts(ps[2]+1);
    putchar(*a[1]); puts(*(a+1)+1);
}
```



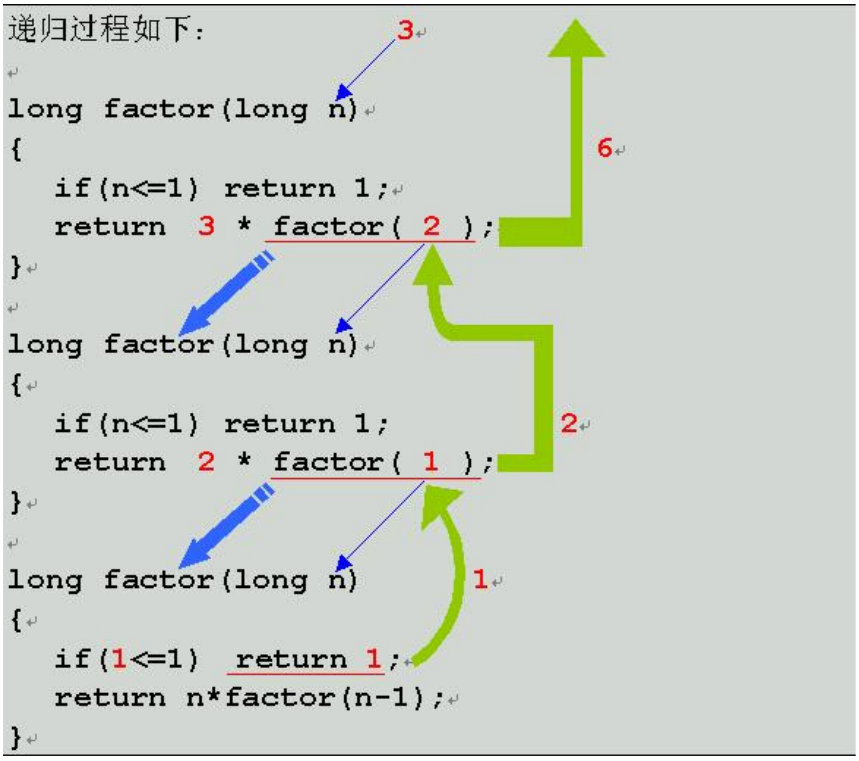


main()的参数




```
#include <stdio.h>
#include <stdlib.h>
   /* 字符指针数组作形参，等价于 char **argv */
void main(int argc, char *argv[] )
{
   int i, sum=0;
   printf("一共有%d个参数\n", argc);
   for(i=0; i<argc; i++)
      printf("参数%d=%s\n", i, argv[i]);
   for(i=1; i<argc; i++)
      sum += atoi(argv[i]);
   printf("sum=%d\n", sum);
}
```

递归



用辗转相除法求最大公约数

```
int f(int m, int n)
{
   int x;
   x = m % n;
   if(x==0)
      return n;
   else
      return f(n, x);
}
main()
{
   int a=6, b=10, c;
   c = f(a,b);
   printf("c=%d\n", c);
}
```

动态变量

```
int  sum(int n)
{
   int x=0;
   while(n) {
      x += n;
      n--;
   }
   return x;
}

main()
{
   int n, x;
   n=10;
   x=100;
```

```
    {
        int y;
        y=sum(n);
        y=y+sum(x);
        printf("y=%d\n", y);
    }
    printf("n=%d, x=%d\n", n, x);
}
```

局部静态变量

```
int  next_number(void)
{
    static int x=0;  /* 局部静态变量 */
    x++;
    return x;
}

int  next_number_plus(void)
{
    int x=0;
    x++;
    return x+next_number();
}

main()
{
    int x;
    x = next_number_plus();
    printf("1st call:%d\n", x);
    x = next_number_plus();
    printf("2nd call:%d\n", x);
    x = next_number_plus();
    printf("3rd call:%d\n", x);
}
```

结构类型定义与结构变量定义

```
#include <stdio.h>
struct student {    /* 结构类型定义 */
    char name[10];
    int score;
};   /* 结构类型名是 struct student */

main()
{
    struct student x, y, a[3];
    int i;
    x.score=50;  /* 结构变量与成员之间用.隔开 */
    y.score=60;
    a[0].score=70;
    a[1].score=a[0].score+10;
    a[2].score=a[1].score+10;
    strcpy(x.name, "Tom");/*不能直接对 name 赋值*/
```

不定义结构类型，直接定义结构变量

```
#include <stdio.h>
main()
{
   struct {    /* struct 后面的 TAG 名可以省略*/
      char *name;
      int score;
   } x, y;     /* x, y 是定义的两个结构变量 */
   x.name="Tom"; /*x.name 可以赋值，因为它是指针*/
   x.score=50;
   y.name="Jerry"; y.score=60;
   printf("%s,%d\n", x.name, x.score);
   printf("%s,%d\n", y.name, y.score);
}
```

同时定义结构类型与结构变量

```
#include <stdio.h>
struct student { /*既定义结构类型又定义结构变量*/
   char name[10];
   int score;
} x={"Tom",50}, y={"Jerry",60};
main()
{
   struct student z={"Donald",80};
   if (z.score>x.score && z.score>y.score)
     printf("Mr. %s is the greatest!", z.name);
}
```

用typedef定义结构类型

```
#include <stdio.h>
typedef struct {
   char name[10];
   int score;
} STUDENT;  /* STUDENT 是一个结构类型*/
main()
{
   STUDENT x={"Tom",50}, y;
   scanf("%s %d", y.name, &y.score);
   if(y.score > x.score)
      printf("%s is greater than %s!", 
              y.name, x.name);
   else
      printf("%s is greater than %s!",
              x.name, y.name);
}
```

结构指针与函数

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

```
struct st {
   char name[10];
   int score;
};
void input(struct st *p); /* 结构指针作参数 */
struct st * find(struct st a[], int n, char *name);
/* 此函数返回一个结构指针 */
main()
{
   struct st a[3], *p;
   char name[10];
   int i;
   for(i=0; i < 3; i++)
      input(&a[i]); /* 实参为结构指针 */
   scanf(" %s", name);
   p = find(a, 3, name);
   if(p==NULL)
      puts("Not found!");
   else
      printf("name=%s, score=%d\n",
      p->name, p->score);
}


void input(struct st *p)
{
   scanf(" %s %d", p->name, &p->score);
}
struct st * find(struct st a[], int n, char *name)
{
   int i;
   for(i=0; i < n; i++) {
      if( strcmp(a[i].name, name)==0 ) break;
   }
   if(i>=n)
      return NULL;
   else
      return &a[i]; /* 或return a+i;*/
}
```

与文件操作有关的函数

`FILE * fopen(char *filename, char *mode);`
以filename为文件名，以mode方式打开文件，返回一个文件结构指针。

`int fclose(FILE *fp);`
关闭文件，fp为打开文件时返回的文件结构指针。

`int fgetc(FILE *fp);`
从文件中读取一个字符，返回值为读取的字符。

`int fputc(int c, FILE *fp);`
把字符c写入文件。

`int feof(FILE *fp);`
返回文件是否结束(End of File)，若结束返回非零值，否则返回0。

`int fscanf(FILE *fp, char *format, ...);`
以format格式从文件中读取变量的值。

`int fprintf(FILE *fp, char *format, ...);`
以format格式把变量的值写入文件。

文本文件的打开方式

"r"　只读方式(read only)。以该方式打开文件时要求文件已存在。
"w"　只写方式(write only)。以该方式打开文件时若文件不存在则新建一个，若已存在则删除原文件再新建一个。
"a"　追加方式(append)，只写。

以该方式打开文件时若文件不存在则新建一个，若已存在则打开该文件，原内容不删除。
"r+" 读写方式。以该方式打开文件时要求文件已存在。
"w+" 读写方式。以该方式打开文件时若文件不存在则新建一个，若已存在则删除原文件再新建一个。

复制一个文本文件并输出

```
#include <stdio.h>
void main()
{
   FILE *fp1, *fp2;
   char c;
   fp1=fopen("file1.c", "r");
   fp2=fopen("file2.c", "w");
   if(fp1==NULL || fp2==NULL) {
      puts("打开文件失败！");
      exit(0);
   }
   while(!feof(fp1)) {
      c = fgetc(fp1);
      fputc(c, fp2);
      putchar(c);
   }
   fclose(fp1);
   fclose(fp2);
}
```

输入学生成绩并排序

键盘输入5个学生的姓名、学号、成绩，并以文本格式写入文件student1.dat中，关闭文件。
再从该文件中重新读取这5个学生的姓名、学号、成绩，
然后按成绩从高到低的顺序把这些数据写入student2.dat中。

```
#include <stdio.h>
struct st {
   char name[10];
   int num;
   int score;
};
main()
{
   struct st a[5];
   FILE *fp;
   int i, j, n, max;
   char tname[10];
   int tnum, tscore;
   for(i=0; i < 5; i++) /* 输入5个姓名,学号,成绩 */
      scanf("%s %d %d", a[i].name,
      &a[i].num, &a[i].score);
   fp = fopen("student1.dat", "w");
   for(i=0; i < 5; i++) /* 写入5个姓名,学号,成绩 */
      fprintf(fp, "%s %d %d\n", a[i].name,
      a[i].num, a[i].score);
   fclose(fp);
   fp = fopen("student1.dat", "r");
   n = 0; /* 先假定文件中存放0个学生的记录 */
   while(!feof(fp)) {/* 若文件未结束，则读取记录 */
      fscanf(fp, "%s %d %d ", a[n].name,
      &a[n].num, &a[n].score);
      n++; /* 记录个数加1 */
   }

   for(i=0; i < n-1; i++) { /* 用选择法排序 */
      max = i;
      for(j=i+1; j <= n-1; j++)
         if(a[j].score > a[max].score) max=j;
```

```
            strcpy(tname, a[i].name); /* 交换姓名 */
            strcpy(a[i].name, a[max].name);
            strcpy(a[max].name, tname);
            tnum = a[i].num; /* 交换学号 */
            a[i].num = a[max].num;
            a[max].num = tnum;
            tscore = a[i].score; /* 交换成绩 */
            a[i].score = a[max].score;
            a[max].score = tscore;
        }
        fp = fopen("student2.dat", "w");
        for(i=0; i < n; i++) /* 写入排序后的记录 */
            fprintf(fp, "%s %d %d\n", a[i].name,
            a[i].num, a[i].score);
        fclose(fp);
}
```