1、 导言（简述C语言基本操作）

第一个C语言程序：

```c
# include <stdio.h> // 包含标准库的信息

int main()
{
    printf("hello, world\n");
    return 0;
}
```

说明：

- 一个C语言程序，都是由**函数**和**变量**组成的。函数包含一些语句，以指定所要执行的计算操作；变量则用于存储计算过程中的值。
- 函数的命名没有限制，但**每个程序都从main函数开始执行**，意味着每个程序都包含main函数。
- main函数通常**调用其他函数**，被调用的函数可以是自己编写的，也可以来自函数库。如`# include <stdio.h>`
- **函数中的语句用花括号{ }括起来**
- **各条语句均以分号（; ）结束**

C语言注释：

```c
// 此处为单行注释
/* 此处为多行注释
    多行注释 */
```

常用C语言数据类型： int（整型）、char（字符，一个字节）、float（浮点数）

```c
celsius = 5 * (fahr-32) / 9 ;
```

在C语言以及其他语言中，整数除法将执行舍位，结果中任何的小数部分都会被舍弃

printf的不同用法

```c
printf("hellp, world\n");

int a = 3; float b = 5.732;

printf("输入数字为%d\n",a);  //按照十进制整型打印
printf("输入数字为%3d\n",a); //按照十进制整型打印，至少6个字符宽度

printf("输出数字为%f\n",b);  //按照浮点数打印
printf("输出数字为%3f\n",b); //按照浮点数打印，至少6个字符宽度

```

```c
printf("输出数字为%.2f\n",b);     //按照浮点数打印，小数点保留两位
printf("输出数字为%6.2f\n",b);    //按照浮点数打印，至少6个字符宽度，小数点保留两位

printf("%s",longest);    //输出字符数组的一整句话
```

符号常量

```c
# define 名字 替换文本
# define MaxSize 10
```

字符输入/输出

- 标准库中提供了一次读/写一个字符的函数，其中常用的是getchar和putchar两个函数。
- getchar会读取回车，空格键，如果不想读取回车，可以再加一个getchar来"吃掉"回车

```c
c = getchar();
putchar(c)
```

结束EOF：

- EOF全称是End Of File（C语言标准函数库中表示文件结束符），通常在文本的最后表示资料结束。C语言中数据都是以字符的ASCII代码值来存放的。ASCII代码值的范围是0~127，不可能出现-1，因此可以用EOF作为文件结束标志，我们可以把EOF作为'-1'理解。
- **EOF定义在头文件<stdio.h>中，是一个整型数int。**
- 可以用ctrl + z 输入**EOF**

```c
/*
读取一个字符
    while (该字符不是文件结束指示符)
        输出刚读取的字符
        读取下一个字符
*/

# include <stdio.h>
void main(){
    int c;

    c = getchar()   ;
    while (c != EOF){
        putchar(c);
        c = getchar();
    }
}
```

上述部分代码可以优化为：

```
while ((c=getchar()) != EOF){
    putchar(c);
    c = getchar();
}
```

统计输入的字符数量：

- 自增： a++ / ++a
- 自减：a-- / --a
- for循环语句的循环体是空的，但又必须有一个循环体，因此用单独的分号代替，称为空语句，建议单独放一行

```
# include <stdio.h>
// 统计输入字符的数量

void main(){
    int nc;
    for (nc=0; getchar() != EOF; nc++)
        ;   // 空语句
    printf("%3d",nc)
}
```

实际运行结果总是不一样，这里考虑加上一个getchar()”吃掉“回车，结果运行成功。

```
# include <stdio.h>
// 统计输入字符的数量

void main(){
    int nc;
    for (nc=0; getchar() != EOF; nc++)
        getchar();  // ”吃掉“回车！！！
    printf("%3d",nc)
}
```

统计输入的行数：

- 区分 “等于“ 和 “赋值”，== 和 =
- 单引号''中的字符表示一个整型值，该值等于此字符再ASCII字符集中的值

```c
# include <stdio.h>
 // 统计输入的行数

void main(){
    int c,nl;

    nl = 0;
    while ((c=getchar()) != EOF)
        if (c == '\n')
            ++nl;
    printf("%d\n",nl);
}
```

统计输入的单词数量：

- 单词的定义： 其中不包含空格、制表符或换行符的字符序列
- nl，行数量；nw。单词数量；nc，字符数量

```c
# include<stdio.h>
// 统计输入的单词数量

# define IN 1   // 在单词内
# define OUT 0  // 在单词外

void main(){
    int state, nl, nw, nc, c;   //此步骤极其关键，务必在使用变量前声明类型

    state = OUT;
    c =  nl = nw = nc = 0;

    while ((c=getchar()) != EOF){
        nc++;
        if (c == '\n')
            nl++;
        if (c==' ' || c=='\t' || c=='\n')
            state = OUT;
        else if (state == OUT){     //此处一定要注意是==，不是赋值语句=
            state = IN;
            nw++;
        }
    }
    printf("输入的行数量%d\n输入的单词数量是%d\n输入的字符数量是%d\n",nl,nw,nc);
}
```

统计输入的各个数字0-9的，空白字符（空格符、制表符、换行符）和其他所有字符的数量

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

```
void main(){
    int nwhite,nother;  //nwhite(空白字符数量)    nother(其他字符数量)
    int ndigit[10]; //分别代表数字0-9的数量
    int i, c;

    nwhite = nother = 0;
    for (i=0; i<10; i++)
        ndigit[i] = 0;

    while ((c=getchar()) != EOF){
        if (c>='0' && c<='9')
            ndigit[c-'0']++;    //字符类型的可以进行加减操作，根据ASIIC码值
        else if (c==' ' || c=='\t' || c=='\n')
            nwhite++;
        else
            nother++;
    }

    printf("数字0-9的数量依次是：");      //输出数字数量
    for (i=0; i<10; i++)
        printf("%d\t",ndigit[i]);
    printf("\n");

    printf("空白字符的数量是：%d\n其他字符的数量是：%d",nwhite,nother);
}
```

函数：

- 函数为计算的封装提供了一种简便的方法，此后使用函数不需要考虑它是如何实现的
- 如printf、getchar、putchar都是函数库中提供的函数
- 下面是幂函数power(m,n)，标准库中有pow(m,n)
- **自定义函数需要在main函数前**
- **一般来说，返回0表示正常终止，返回值非0表示出现异常情况或出错结束条件**

```
# include <stdio.h>

int power(int m, int n){
    int i;
    int ans;
    ans = 1;

    if (m==0)   // 0的阶乘永远是1
        ;
    else{
        for (i=0; i<n; i++)
        ans = ans*m;
    }
```

```
    return ans;      // 返回值给函数
}

void main(){
    int a;
    a = 0;

    a = power(3,2);
    printf("3的平方阶乘结果是：%d",a);

    return 0;
}
```

字符数组：

- 需求：读取一组文本行，并把最长的文本行打印出来

```
// 详见课本27面，此处偷懒不编程了ssss
```

2、类型、运算符与表示符

- 变量和常量是程序处理的两种基本数据对象。声明语句说明变量的名字和类型，也可以指定变量的初始值。
- 运算符指定要进行的操作。表达式则把变量与常量组合起来生成新的值。
- **对象的类型决定该对象可取值的集合以及可以进行的操作。**

变量名 命名规则：

- 名字是由字母和数字组成的序列，但第一个字符必须为字母；
- 下划线“_”被看作字母，通常用于命名较长的变量名，提高可读性；
- 由于库例程的通常以下划线开头，依次变量名不要以下划线开头
- 大小写字母是由区别的
- 对于内部名而言，至少前31个字符是有效的；对于外部名，仅前6个字符有效，并且不区分大小写；
- 注意区分关键字，且使用关键字的时候必须小写

C语言的基本数据类型：

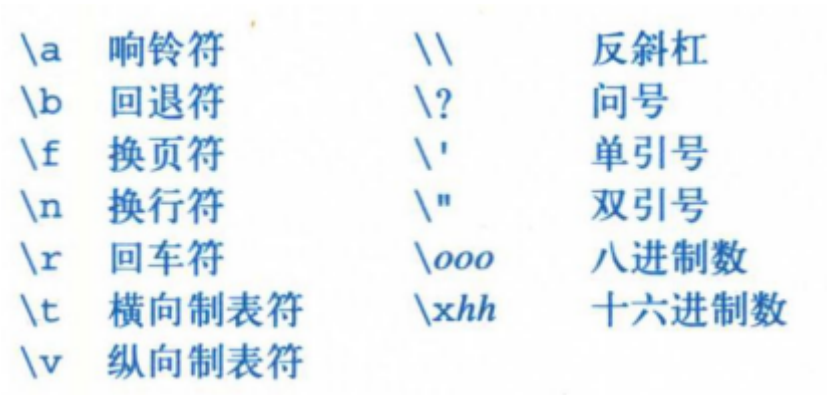
- char 字符型 占一个字节长度
- int 整型
- float 单精度浮点型
- double 双精度浮点型

此外，可以在基本数据类型前加上一些限定符：
例如： 1、short int 和 long int （short通常16位，long通常32位）
2、signed int 和 unsigned int（unsigned取值是0-255，signed取值是-128-127）

常量：

- 后缀：long类型的常量以字母l或L结尾，如123456789L，若一个证书太大无法用int类型表示，也会当作long类型处理。无符号常量以字母u或U结尾，后缀ul或UL表示unsigned long。
- 前缀：整型数除了十进制白哦是外，还可以用八进制和十六进制表示。前缀ox或OX表示十六进制，前缀o或O表示八进制。
- 常量表达式：`# define MaxSize 10` 可以在程序中任何位置出现

转义字符：



字符串常量：
字符串常量就是字符数组，字符串的内部表示使用一个空字符'\0'作为字符串的结尾，因此存储字符串的物理存储单元数比括在双引号中的字符数多一个。
标准库`<string.h>`中`strlen(s)`可以返回字符串参数s的长度，但是不包括结尾的空字符。

```
// strlen函数：返回s的长度
int strlen(char [s]){
    int i;
    i = 0;

    while (s[i] != '\0')
        i++;
    return i;
}
```

一个字符和字符串之间的区别：x' 与 “x” 是不同的，*前者是一个整数，其值是字母x在ASICC码值对应的数值，后者是一个字符（即字母x）以及一个结束符 '\0' 的字符数组。*

枚举常量：

- 枚举是一个常量整型值的列表

- 枚举为建立常量值与名字之间的关联提供了更加便利的方式。相对于#define语句而言它可使常量值自动生成
- 详细信息可以看此链接枚举常量及应用，讲述的极其详细。

变量的声明：

- 所有变量都必须先声明后使用
- 一个声明指定一种变量类型，后面所带的变量表可以包含一个或多个该类型的变量
- 可以在声明的同时对变量进行初始化

```
int lower, upper, step;
char c,line[1000];
int love=520; hate=250;
```

const限定符限定：

- 该限定符指定变量的值不能被修改，对于数组而言，const限定符指定数组所有元素的值都不能被修改

```
const doubel e = 3..14159265;
const char msg[] = "warning";
```

算数运算符、关系运算符、逻辑运算符：

- +、-、*、/、%
- >、>=、<、<=、==、!=
- &&、||
- 根据优先级的不同，添加圆括号

自增运算符（++）与自减运算符（--）：

- 它们既可用作前缀运算符（如，++n），也可用作后缀运算符（如，n++）
- 不同点，表达式++n先将n的值递增1，然后在使用n的值；而n++则是先使用变量n的值，然后再将n的值递增1
- 在不需要使用任何具体值且仅需要递增变量的情况下（循环中i，j变量），前缀后缀方式的效果相同

位运算符：
C语言提供了6个位操作运算符，这些运算符只能作用于整型操作数，即只能只作用于有符号或无符号的char、short、int、long类型

& 按位与（AND）
| 按位或 （OR）
^ 按位异或（XOR）
<< 左移
>> 右移
~ 按位求反（一元运算符）

赋值运算符与表达式：

```
i = i + 1;
i += 1; //赋值运算符
i += 1; i *= 5;
```

条件表达式：

```
if (a > b)
    z = a;
else
    z = b;
```

上述代码可用条件表示式（三元运算符）简化

```
// expr1 ? expr2 : expr3;
z = (a >b) ? a : b;     //a大于b吗？是》a，不是》b
```

3、控制流（判断、循环、分支）

语句与程序块：

- C语言中，分号是语句结束符
- 用一对花括号"{"与"}"把一组声明和语句括在一起构成一个复合语句（也叫程序块 ）

if-else 语句：

- **在有if语句嵌套的情况下使用花括号**
- **各个表达式将被依次求值，一旦某个表达式结果为真，则执行与之相关的语句，并且种植整个语句序列的执行**
- 各条语句可以是单条语句，也可以是花括号括起的复合语句
- else语句 和 else if语句 可省略

```
// C语言else-if结构
if (表达式)
    语句;
else if (表达式)
    语句;
else if (表达式)
    语句;
else
    语句
```

switch语句：

```
switch(表达式){
    case 常量表达式：语句序列;
    case 常量表达式：语句序列;
    default: 语句序列;
}
```

使用switch语句统计输入的数字、空白字符以及其他字符出现的次数：

- 当多个switch函数并列时，没有语句序列，则执行最后一个语句学列
- 后面需要break和defaul

```
# include <stdio.h>

void main(){
    int c, i, nwhite, nother, ndigit[10];

   nwhite = nother = 0;
   for (i=0; i<10; i++)
       ndigit[i] = 0;
    while ((c=getchar()) != EOF)
    {
        switch (c)
        {
            case '0':   case '1':   case '2':   case '3':   case '4':
            case '5':   case '6':   case '7':   case '8':   case '9':
                ndigit[c-'0']++;
                break;
            case ' ':   case '\n':  case '\t':
                nwhite++;
                break;
            default:
                nother++;
                break;
        }
    }
    printf("digits = ");
    for (i=0; i<10; i++)
```

```
        printf(" %d ",ndigit[i]);
    printf("\n");
    printf("white space = %d\n",nwhite);
    printf("other space = %d\n",nother);

    return 0;
}
```

while循环与for循环：

- 在没有初始化或重新初始化操作时，使用while循环语句更加自然
- 若语句需要执行简单的初始化和变量递增，使用for循环更加合适

下面是无限循环的案例，可以借助其他手段（如break或return语句）终止执行

```
for (;;){
    ……
}
```

break语句与continue语句：

- break语句：直接跳出当前循环
- continue语句：结束当前循环，开始下一次循环

4、函数与程序结构

- C语言在设计中考虑了函数的**高效性**和**易用性**两个元素。故C语言程序一般都由许多小的函数组成，而不是少量较大的函数组成。
- 如果函数定义中省略了返回值类型，则默认为int类型
- **函数之间可以通过参数、函数返回值以及外部变量进行通信**
- 函数在源文件中出现的次序可以是任意的，但**需要在main函数之前被声明。**
- 被调用函数通过return语句向调用者返回值，return语句后面可以跟任何表达式，不一定需要有返回值
- 如果函数带有参数，则要声明它们；如果没有参数，使用void进行声明（代码的健全性）
- 注意函数返回值类型，double等

变量： 自动变量只能在函数内部使用，*从其所在的函数被调用时变量开始存在，在函数退出时也将消失*；而外部变量是永久存在的，他们的值在依次函数调用到下一次函数调用之间保持不变。如果不同函数需要共享某些数据，而这两个函数互不调用对方，最方便的就是定义外部变量（全局变量），而不是作为函数参数传递。

重点理解： 读取一组文本行，并把最长的文本行打印出来

```c
// 用到getline()和copy()两个自定义函数调用
// getline()：获取当前行长度
// copy()：将当前行所有元素复制，从from字符数组到to字符数组

# inlude <stdio.h>
# define MAXLINE    //行最大长度

int getline(char line[], int maxline);
void copy(char to[], char from[]);

int main()
{
    int len, max;   //当前行长度和最长行长度
    char line[MAXLINE], longest[MAXLINE];   //当前的输入行和最长的输入行

    max = 0;
    while ( (len = getline(line,MAXLINE)) > 0 )
    {
        if(len > max)
        {
            max = len;
            copy(longest,line);
        }
    }

    printf("%s",longest);

    return 0;
}

int getline(char s[], int lim)
{
    int c,i;

    for (i=0; i<lim-1 && (c=getchar()) != EOF && c != '\n'; i++)
        s[i] = c;
    if (c == '\n')
    {
        s[i] = c;
        i++;
    }
    s[i] = '\0';    //加入行结束标识符‘\0’
    return i;
}

void copy(char to[], char from[])
{
    int i;

    i = 0;
    while((to[i] = from[i]) != '\0')
        i++;
```

```
53 }
```

递归：

C语言中的函数可以递归调用，即函数可以直接或间接调用自身。

C预处理器

最常用的预处理器： # include（用于在编译期间把指定文件的内容包含进当前文件中）： `#include <文件名>`

define指令（用于以任意字符序列替代一个标记）

宏替换

`# define 名字 替换文本`，后续所有出现*名字*记号的地方都将被替换为*替换*文本，替换文本可以是任意字符串，#define指令定义的名字的作用域从其定义点开始，到编译的源文件的末尾处结束。

常用的如：

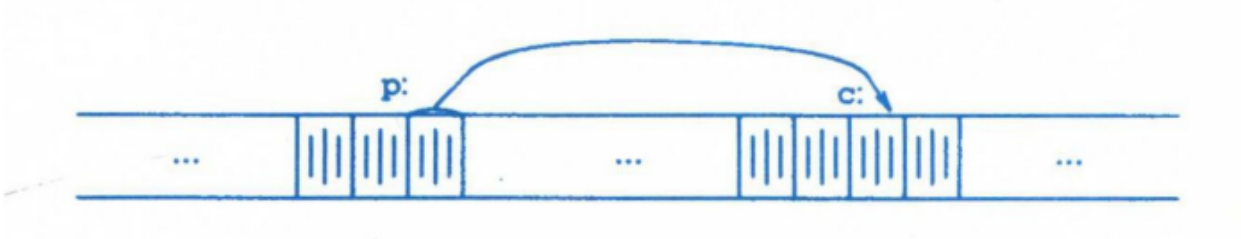
```
# define forever for(;;)    //无限循环
# define max(A,B) ((A) > (B) ? (A) : (B))   //取最大值
# define MaxSize 10
```

`# undef getchar`取消宏替换

5、指针与数组

- ***指针是一种保存变量地址的变量，够存放一个地址的一组存储单元（通常是2或4个字节）***
- 通常的机器都有一系列连续编号或编址的存储单元

加入c的类型是char，并且p是指向c的指针，则他们之间的关系如下：



一元运算符&可用于取一个对象的地址，语句如下：`p = &c;`

- 说明：将把c的地址赋值给变量p，我们称p为“指向”c的指针。地址运算符&只能应用于内存中的对象，即变量于数组元素。它不能作用于表达式、常量或register类型的变量。

一元运算符*是间接寻址间接引用运算符，当它作用于指针时，将访问指针所指向的对象。

假设x与y是整数，而ip是指向int类型的指针

```
int x = 1, y = 2, z[10];
int *ip;    // ip是指向int类型的指针

ip = &x;    // ip现在指向变量x
y = *ip;    // y等于x，即1
*ip = 0;    // x等于0
ip = &z[0]; // ip现在指向z[0]
```

指针的声明： `int *ip`

函数的声明同样可以用上述方式：

```
double *dp;
double atof (char *);    // dp和atof(s)的值都是double类型，且atof的参数是一个指向char类型的指针
```

- 需要注意：每个指针都必须指向某种特定的数据类型（除void类型的指针可以存放指向任何类型的指针，但不能间接引用其自身）

指针的使用案例:

```
*ip = *ip + 10; // *ip可以直接当作变量使用

y = *ip + 1;    // 一元运算符 * 和 & 优先级比运算符的优先级别高
*ip += 1;

++*ip;
(*ip)++;

iq = ip;
```

- 语句(*ip)++中的圆括号是必须的，否则表达式将对ip进行加1运算，而不是对ip指向的对象进行加1运算，这是因为类似于*和++这样的**一元运算符遵循从右向左**的结合顺序
- 指针也是变量，所以在程序中直接使用，而不是通过间接引用的方法使用。

指针与函数参数

- C语言是以传值的方式将参数值传递给被调用函数的，因此被调用函数不能直接修改主调函数中变量的值。如下：

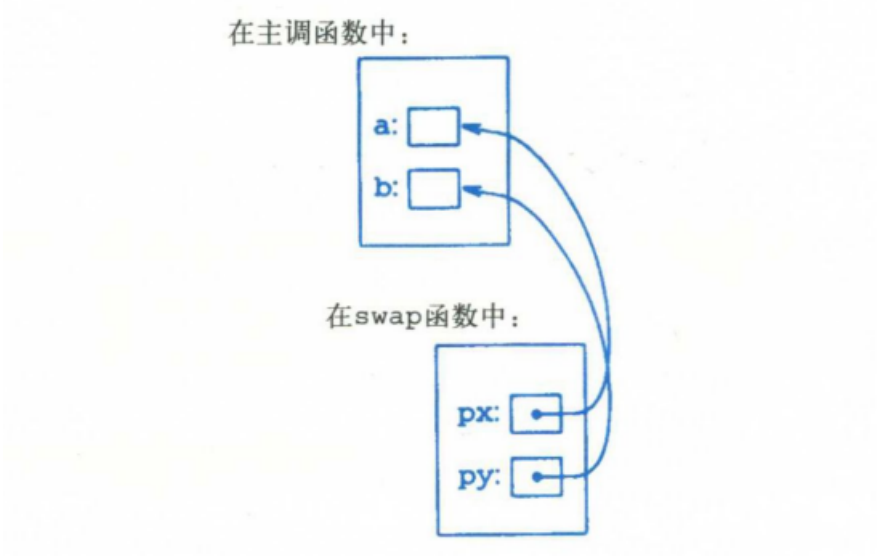
```
void swap(int x, int y)
{
    int = tmp;

    tmp = x;
    x = y;
    y = tmp;
}
```

则语句：`swap(a, b)`无法达到函数目的，因为**函数采用传值方式**，因此上述的swap函数不会影响调用它的例程中参数a和b的值，该**函数仅仅交换了a和b的副本值**，在文件执行结束后，副本值被删除。

考虑将上述文件修改为如下形式：

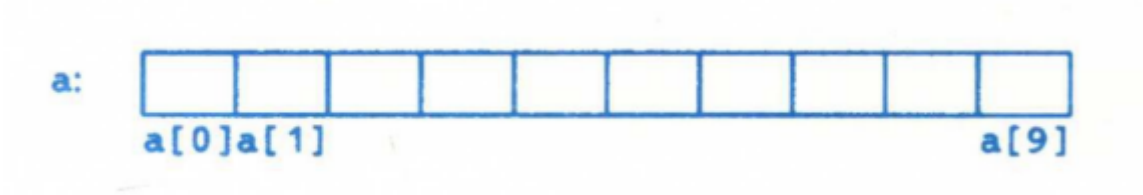
```
void swap(int *px, int *py)      // 对传入的两个指针，调用指针所指向的值
{
    int = tmp;

    tmp = *px;
    *px = *py;
    *py = tmp;
}

void mian()
{
    swap(&a, &b);    // 此处传递两个指针至被调用函数
}
```

具体实现框图如下：



指针与数组

指针声明：`int a[10]`，它定义了一个由10个对象组成的集合，这10个对象存储在相邻的内存区域内，名字分别为a[0], a[1], ... , a[9]



亦可用指针指向数组：

```
int *pa;
pa = &a[0];

x = *pa;    // 将数组元素a[0]中的内容赋值到变量x中
```

- 指针pa指向数组a的第0个元素，也就是说pa的值为数组元素a[0]的地址

如果pa指向数组中某个特定元素。那么，pa+1将指向下一个元素，pa+i将指向pa所指向数组元素之后的第i个元素；因此，如果指针pa指向a[0]，那么*(pa + 1)引用的是数组元素a[1]的内容，pa+i是数组元素a[i]的地址，*(pa + i)引用的是数组元素a[i]的内容。

- 无论数组a中的元素的类型或者数组长度是多少，以上内容均成立，因为“指针加1”意味着，pa+1指向pa所指向的对象的下一个对象。

```
pa = &a[0];

pa = a;
```

- **数组名和指针运算之间具有密切的对应关系，在上述第一个表达式中，pa和a具有相同的值，因为数组名所代表的就是该数组最开始的一个元素的地址，所以第一个表达式亦可以写成第二个表达式的形式。**
- 在计算数组元素a[i]的值时，C语言实际上先将其转换为*(a+i)的形式，然后在进行求值，因此在程序中这两种形式是等价的
- 简而言之，一个数组和下标实现的表达式可等价的通过指针和偏移量实现
- 但是数组名和指针不同之处是，指针是一个变量，一个算数运算，但是数组名不是变量，例如a=pa和a++形式的语句是非法的

```c
// strlen函数：返回字符串s的长度

int strlen(char *s)
{
    int n;

    for (n = 0; *s != '\0'; s++)
        n++;

    return n;
}
```

- 因为s是一个指针，所以对其执行自增运算是合法的，执行s++运算不会影响到 strlen函数调用者的字符串。它仅对该指针在strlen函数中的私有副本进行自增运算。

类似的调用如下：

```c
strlen("hello world");  // 传入字符串常量
strlen(array);  // 字符数组
strlen(ptr);    // ptr是一个指向char类型对象的指针
```

上述的三种调用方式都可以正确执行

```c
char s[];

char *s;
```

- 上述两种表达方式是等价的，但是后者更佳，因为只管的表示了该参数是一个指针。

同样的，可以将子数组起始位置的指针传递给指针，这样，就将数组的一部分传递给函数了，被调用函数可以修改原始数组中的值

```c
f(&a[2]);

f(a+2);

// 声明可以如下面两种
f(int arry[])

f(int *arry)
```

- 由于指针也是变量，同样可以存储在数组中，成为**指针数组**，同样可以考虑 **指向指针的指针**。

多维数组：

- 不同中括号内括住不同行的值
- 行与行之间需要逗号' , '分隔

```
// 二维数组存储的是闰年和非闰年不同月份的长度，第一行是非闰年，第二行是闰年

char daytab[2][13] =
{
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
}

// 调用形式
data[i][j]  // 第i行第j列
```

6、结构

- 结构是一个或多个变量的集合，这些变量可能为不同的类型，为了处理的方便而将这些变量组织在一个名字之下
- 由于结构将一组相关的变量看作一个“单元”而不是各自独立的实体，因此结构有助于组织复杂的数据，特别是在大型的程序中

```
struct book
{
    char title[Maxtitle];
    char author[Maxautl];
};

struct point pt;    // 调用方式
pt.x = 3;
pt.y = 4;

// 或者如下方式
struct point maxpt = {320,  200};
```

- 关键字struct，表示接下来是一个结构体
- book是一个可选标记，用来引用该结构体的快速标记，如`struct book library`
- 花括号内，括起了结构体成员列表，以及每个成员变量，使用的都是自己的声明方式来描述
- 用分号来结束（不可省略）
- 调用方式：`结构名.成员` `结构名->成员`

结构体嵌套：

```c
struct point    // 点坐标
{
    int x;
    int y;
};

struct rect //矩形的表达方式
{
    struct point pt1;
    struct point pt2;
};

// 调用方式
struct rect screen;
screen.pt1.x = 3;
screen.pt2.x = 4;
```

- 如果传递给函数的结构很大，使用指针方式的效率通常比复制整个结构的效率要高。结构指针类似于普通变量指针

```c
struct point *pp

// 调用方式如下
(*pp).x;
(*pp).y;    // 一定要注意，此处的圆括号是必须的！！！
```

一元运算符 `sizeof 对象` `sizeof(类型名)`

类型定义（typedef）

```c
typedef int zhengshu;   // zhengshu此处定义为与int具有同等意义的名字，功能也完全相同

// 故可以将结构名简化
typedef struct point pd
```

联合

```c
struct book
{
    char title[Maxtitle];
    char author[Maxautl];
}book_here;
```

- 变量book_here足够强大，相当于以下程序

```
typedef struct book book_here;
```