数据结构与程序设计
(Data Structure and Programming)

数据结构基础

复杂数据程序设计

北航计算机学院 晏海华

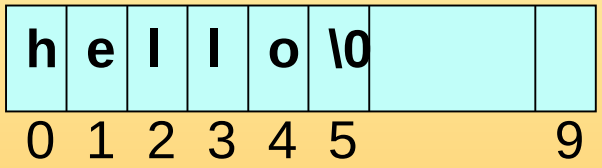
本章目标

- 回顾数组的定义与初始化；
- 掌握指针说明与指针运算；
- 掌握指针与数组的关系；
- 掌握指针作为函数参数；
- 掌握指针数组；
- 掌握结构的定义和使用；
- 了解自引用结构。
- 掌握文件的基本操作

数组的定义与初始化

- 数组初始化：可以在定义时初始化一个数组。下面是一些数组初始化实例：

```
double sales[5] = {12.25, 32.50, 16.90, 23, 45.68};
double sales[ ] = {12.25, 32.50, 16.90, 23, 45.68};
int list[5] = { 6, 5, 12 }; //相当于：  int list[5] = {6,5,12,0,0};
int list[5] = {0};  //最常用的初始化数据组元素为0的方式
char string[10] = “hello”;
char string[10] = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};
char string[ ] = “hello”;
```



注意：

1. 对于一个长度为N的数组，其数据范围（下标）为0~N-1，而不是1~N（这是C程序员常范的错误之一）。
2. 用字符串常量初始化一个字符数组时，数组长度应至少为字符个数多1。

数组作为函数参数

- **数组可以作为参数传递给函数。**实际上传递的是数组的首地址（即数组第一个元素的地址，将在指针部分说明），我们可以这样理解数组作为参数传递：形参数组与实参数组是一对共享同一数据区的数组，即它们是同一个数组，而不是对实参数组的拷贝。
- 非字符数组作为参数时，函数的定义形式：

```
void fun( int array[ ], int size)
{…}
```

- 数组作为参数时，函数的调用形式：

```
main()
{
    int a[10];
    ….
    fun(a, 10);
    ….
}
```

注意：定义数组形参时，数组长度可省略。对于非字符数组，还应在形参中指定数组元素个数。

注意：函数调用时，用数组名作实参。

字符数组

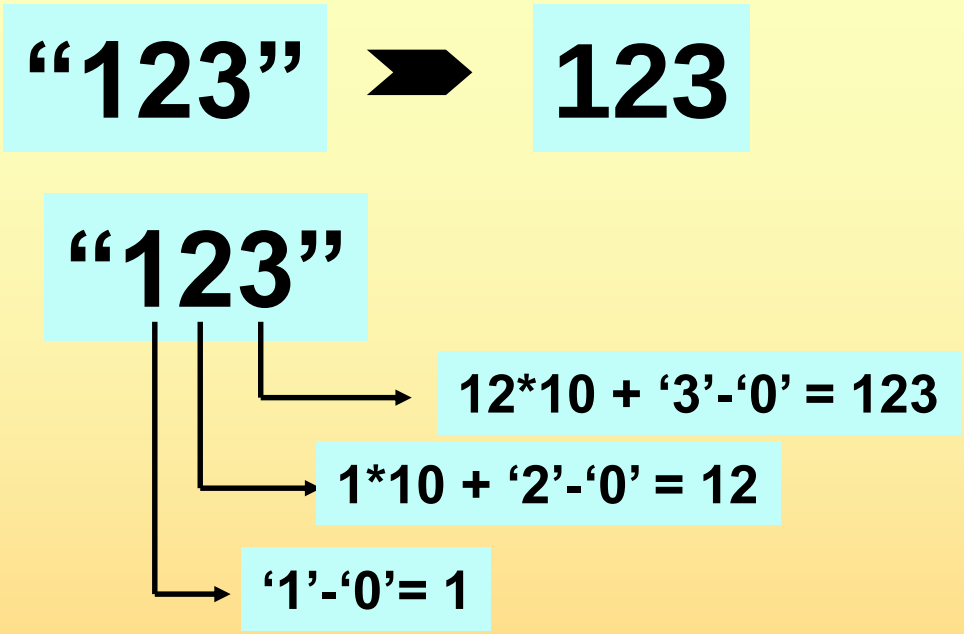
数组元素的类型是char。

```
char mes[ ] = “C Language”;
char line[100] = “Programming”;
```

字符数组有如下特点：

- 数组元素跟一般变量一样可赋值、比较、计算等。
- 数组下标也是从0 ~ N-1（N为数组长度）。
- 字符数组长度可以显式给出，也可以隐式得到。
- 由双引号括起来的的字符串常量具有静态字符串数组类型。
- 用字符串常量对数组初始化时，编译程序以\0作为结束这个数组。因此，**用字符数组来存放字符串时，数组长度要比字符串长度多1**。

问题1：将数字字符串转换成整数



方法分析

```
n = 0;
for(i=0; s[i]为数字字符; i++)
        n = 10*n + s[i] - ‘0’;
```

问题1：代码实现

```
int atoi(char s[ ])
{
    int i, n, sign;
    for(i=0; s[i] = = ' ' || s[i] = = '\n' || s[i] = = '\t'; i++)
        ;                /* skip white space */
    sign = 1;
    if(s[i] = = '+' || s[i] = = '-')
        sign = (s[i++] = = '+')?1:-1;
    for(n=0; s[i] >= '0' && s[i] <= '9'; i++)
        n = 10*n + s[i] – '0';
    return ( sign * n);
}
```

字符数组。

空语句

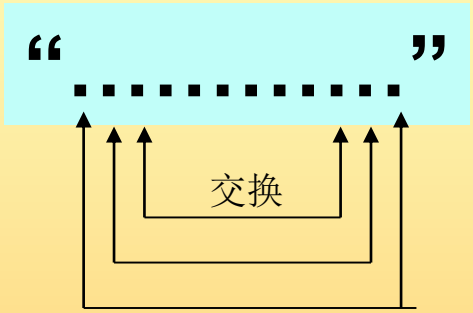
条件运算符：
<表达式1> ? <表达式2> : <表达式3>
先计算表达式1，若其值为非零，则整个表达式结果为表达式2的值，否则就为表达式3的值。

```
#include <stdio.h>
int atoi(char s[ ]);
int main()
{
    char s[20];
    scanf("%s", s);
    printf("%d\n", atoi(s));
    return 0;
}
```

读入一个以空白字符分隔的字符串到s中，以'\0'结束(即由非空字符组成的字符串)。

为何字符数组（字符串）作为函数参数传递时，通常不用传递数组长度？

问题2：将字符串颠倒



方法分析

字符数组作为函数参数传递时，不需要同时传递数组长度。因为字符数组中字符串是以'\0'结束的。

当有多个循环变量时，要用逗号隔开。
逗号表达式：如*e1,e2*顺序求e1和e2，以e2值作为整个表达式结果的值。如，
a = (t = 3, t+2);
结果为**5**

```
#include <stdio.h>
void reverse(char s[ ]);
int strlen(char s[ ]);
int main()
{
    char s[20];
    scanf("%s", s);
    reverse(s);
    printf("%s\n",s);
    return 0;
}
```

```
void  reverse(char s[ ])
{
    int c, i, j;
    for(i=0, j=strlen(s)-1; i < j ; i++,j--){
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

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

输出一个以'\0'结束的字符串。

提示：数组作为函数参数传递时，要不要同时传递数组长度，取决于函数中是否知道数组中元素的个数。

问题3：将一个整数转换成字符串*

例：将一个整型数转换成字符串。

```
void itoa(int n, char s[ ])
{
        int i, sign;
        if((sign = n) < 0)
                n = -n;
        i = 0;
        do {
                s[i++] = n%10 + '0';
        } while( (n /= 10 ) > 0);
        if(sign < 0)
                s[i++] = '-';
        s[i] = '\0';
        reverse(s);
}
```

```
#include <stdio.h>
void itoa(int n, char s[ ]);
int main()
{
    int n;
    char s[20];
    scanf("%d", &n);
    itoa(n,s);
    printf("%s\n",s);
    return 0;
}
```

常用标准字符串库函数*

使用strcpy、strcat函数之前，必须保证s有足够的空间容纳操作后的字符串！

- **#include <string.h> - 字符串处理函数**

```
int strlen(char s[]);    /*计算字符串长度，字符串[illegible]结果*/
char *strcpy(char s[], char t[]); /*将字符串t拷贝到字符串s中*/
char *strcat(char s[], char t[]);  /*将字符串t拷贝到字符串s尾部*/
int strcmp(char s[], char t[]);  /*比较两个字符串，若s>t，则返回大于0的数；若s<t，则返回小于0的数；若相等，返回0 */
```

- **#include <stdlib.h> - 实用函数**

```
int   atoi(char  s[ ]);      /* 将字符串转换成相应整数 */
```

- **#include <stdio.h> - 输入/输出函数**

```
getchar()/putchar(c)          /*按字符输入输出 */
scanf( “%s…)                  /*输入由非空字符组成的串*/
printf( “%s…)                 /*输出由‘\0’结束的串*/
gets(char s[])                /*输入由回车结束的串，回车本身不读入*/
puts(char s[])                /*输出由‘\0’结束的串，自动加回车*/
```

子曰：工欲善其事，必先利其器。…

问题4：文本查找

【问题描述】

从文件中查找包含给定字符串的行。

【输入形式】

从标准输入中分两行分别输入被查找的文件及要查找的字符串（中间不含空格）。

【输出形式】

在屏幕上输出文件中包含给定字符串的行。

【样例输入】

在键盘输入如下文件名及字符串：

```
test.txt
the
```

文件test.txt内容如下：

```
Now is the time
for all good
men to come to the aid
of their party
```

【样例输出】

屏幕输出为：

```
this is the time
men to come to the aid
of their party
```

简单文件操作

- 到目前为止，所有输入/输出操作均为标准输入/输出（即针对键盘及屏幕）。
- 实际应用多数都是针对文件输入/输出，如Office应用。

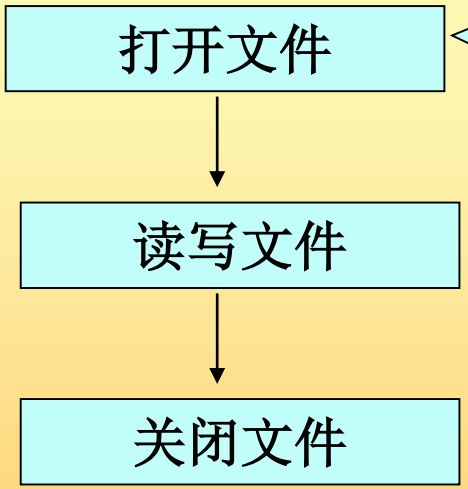
如何进行文件的读写？

文件输入/输出

首先在程序文件的头部应有如下语句：

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

- 文件输入/输出过程




```
FILE *in, *out;

in = fopen("input.txt", "r"); //为输入打开一个给定文件"input.txt"；打开方式"r"为以只读方式打开一个文件。
out = fopen("output.txt","w"); //为输出打开一个给定文件"output.txt"；打开方式"w"为打开一个只写文件。
```

```
c =fgetc(in);  //从文件（input.txt）中读入一个字符

fputc（c,out); //输出一个字符到文件（output.txt）中
fgets(s, n, in); //从 文件in上读入一行（最多读入n-1个字符），放入s 字符数组中。返回s或NULL。
 fputs( s, out); //把字符串s（不一定含\n）写入文件out中。返回非负数或EOF
fscanf(in, "%d", &score); //从文件in中读入一个整数
fprintf( out, "%d\n", score); //输出一个整数到文件out中
```

```
fclose(in); // 关闭文件input.txt
fclose(out); //关闭文件output.txt
```

示例：将一个文件内容拷贝到另一个文件

```
#include <stdio.h>
int main()
{
    int c;
  FILE *in, *out;

  in = fopen("input.txt", "r");
  out = fopen("output.txt","w");
    while ((c = fgetc(in)) != EOF)
      fputc(c, out);

  fclose(in);
  fclose(out);

    return 0;
}
```

为读写文件定义文件指针

打开文件
文件input.txt和output.txt位于与该执行程序.exe文件同一目录下（在VC中则与工程在同一目录下）

从文件input.txt中依次读一个字符

向文件output.txt中依次写一个字符

关闭两个打开的文件

问题4：问题分析

- **数据结构设计**：分析问题描述，显然需要三个字符数组变量，分别存放文件名、要查找的字符串及从文件中读入的行。

```
char   filename[32],  str[81], line[1024];
```

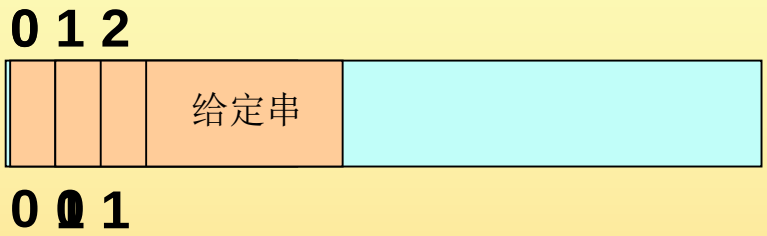
（一个文件名长度通常不超过32个字符；屏幕上一行通常显示80个字符；而1024是一般文件的最大物理行长度。当然这些取决于具体系统实现。）

- **数据输入**
  - 用scanf(“%s…)读入文件名和要查找的串。（中间不能有空格）
  - 从文件中读入一行最简单的方法是用fgets(…)函数。（为何不能用fscanf(fp, “%s…)
- **数据处理**：主要处理就是要从所读入的一行中查找给定的字符串（即从一个字符串中查找另一个字符串）。可用一个单独的函数index实现在一个字符串中查找另一个字符串。（体现模块化思想）

问题4：（字符串查找）算法设计

朴素字符串查找算法

- 设int index(char s[ ], char t[ ])函数用来在字符串s中查找字符串t。若找到则返回t在s中出现的位置，否则返回-1。其主要查找算法如下：



输入串

确定输入字符串s中查找起始位置

```
在字符串s中查找字符串t ：
for(i=0; s[i] != ‘\0’; i++)
    for(j=i,k=0; t[k] != ‘\0’; j++,k++)
        s[j]和t[k]进行比较
```

依次与给定串中每个字符比较。j为s中每次开始比较的位置。

主要算法分析

问题4：算法设计（续）

主函数算法如下：

```
char fileanme[32],s[81],line[1024];
            //分别用于存储文件名、查找串及文件中一行;
read 文件名和要查找的串到filename和s中;
以读方式打开文件filename;
while 文件中还有内容时读一行到line
    if index(line, s) >= 0
        输出line;
```

如何从文件中读入一行？

char *fgets(char s[ ], int n, FILE *fp)
从fp上读入一行（不超过n-1个字符），放入s 字符数组中。返回s或NULL

行输入/输出*

char *fgets(char s[ ], int n, FILE *fp)

从文件fp上最多读入n-1个字符，放入s 字符数组中。返回s或NULL。

int fputs(char s[ ], FILE *fp)

把字符串s（不一定含\n）写入文件fp中。返回非负数或EOF。

- fgets在s最后加换行字符（与gets不同），即’\n’作为一行的内容读入；
- fputs不在输出后加换行字符（与puts不同）；
- fgets能设置输入字符的最大个数，因此，相比函数gets，它更安全。

建议：在调用fgets时，n的值通常为字符串s的长度！

问题4：

使用scanf的缺点是不能输入带空格的字符串。可换成
gets(s);
来实现查找带空格的字符串（即查找一个句子）。

风格建议： 使用fopen时应判断其返回值！

```
if((fp=fopen(…))==NULL){
    打印错误信息;
    return 错误码;
}
```

```
#include <stdio.h>
#define MAXLINE  1024
int index(char s[ ], char t[ ]);
int main( )
{
    char filename[64], s[1], line[MAXLINE];
    FILE *fp;
    scanf("%s", filename);
    scanf("%s", s);
    if((fp = fopen(filename, "r")) == NULL){
        printf("Can't open file %s!\n", filename);
        return 1;
    }
    while(fgets(line, MAXLINE-1, fp) != NULL)
        if(index(line, s) >= 0)
            printf("%s", line);
    return 0;
}
```

```
int index(char s[ ], char t[ ])
{
    int i, j, k;
    for(i =0; s[i] != '\0'; i++){
        for(j=i,k=0;t[k]!='\0'&&s[j]==t[k]; j++,k++)
            ;
        if(t[k] == '\0')
            return ( i);
    }
    return ( -1);
}
```

由于打开一个文件操作可能失败(如打开一个读文件不存在)，因此，好的风格应判断fopen函数的返回值，进行错误处理。

注意：前面循环结束时有两种情况：
1. 找到相应子串，即t[k]=='\0'为真
2. 没有找到，即s[j] != t[k]

因此要依据t[k]=='\0'来判断查找是否成功。

读到文件结束时fgets返回NULL。

问题4：测试

当要查找的文件为”test.txt”,查找的串为”the”，
且文件test.txt中内容为：

Now is the time

for all good

men to come to the aid

of their party

则屏幕输出：

this is *the* time
men to come to *the* aid
of *the*ir party

其它特殊情况：
- 要查找的串在一行的头、尾
- 要查找的串在文件中不存在

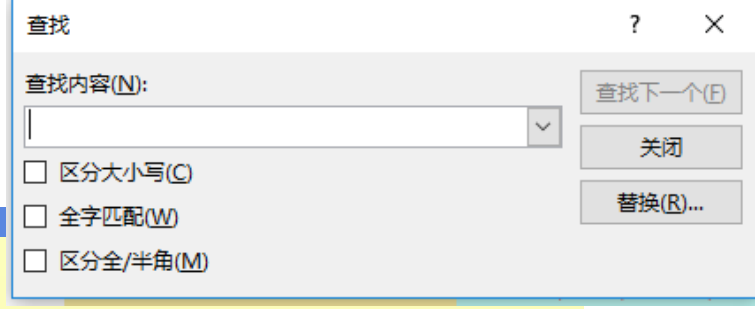
朴素字符串查找算法性能


```
int index(char s[ ], char t[ ])
{
        int i, j, k;
        for(i =0; s[i] != '\0'; i++){
                for(j=i,k=0;t[k]!='\0'&&s[j]==t[k];j++,k++)
                        ;
                if(t[k] == '\0')
                        return ( i);
        }
        return ( -1);
}
```

显然，在最坏情况下（被查找的串不存在），该算法语句的最大执行频度为：N*M。
N为源串的长度，M为要查找的串的长度。

时间复杂度为O(n*m)

问题4：思考1



- 问题4实现了大小写相关的字符串查找，即字符串”the”和”The”是不同字符串。请实现大小写无关的字符串查找。
- 算法分析：

在比较字符时，可将要比较字符均转换为小写或大写即可实现大小写无关查找。

设函数char tolower(char c)用于将字符c转换为相应小写字符，则上面index可改为：

```
int index(char s[ ], char t[ ])
{
        int i, j, k;
        for(i =0; s[i] != '\0'; i++){
                for(j=i,k=0;t[k]!='\0'&&tolower(s[j])== tolower(t[k]);
                        j++,k++)
                          ;
                if(t[k] == '\0')
                          return ( i);
        }
        return ( -1);
}
```

问题4：函数tolower实现

- 方法一：

```
char  tolower(char c)
{
   if( c >= ‘A’  && c<= ‘Z’ )
     return  ‘a’  -  ‘A’  + c;
   return c;
}
```

- 方法二：对于像tolower这样功能简单的函数，可以用宏函数来实现。

```
#define tolower(c)  (c>= ‘A’ &&c<= ‘Z’  ?  ‘a’ -
   ’ A’ +c:c)
```

预处理指令：define

- 用途一：**定义常量**，如：#define PI 3.14159
- 用途二：**定义宏函数**

宏定义还可带变元（参数）：

#define 标识符(参数1, 参数2, …)　　单词串

如：

```
#define max(A,B)    ((A)>(B)?(A):(B))
```

?:为条件运算符

于是语句x = max(p+q, r+s); 被替换为：

```
x = ((p+q) > (r+s) ? (p+q) : (r+s));
```

#define isupper(c) (c >='A' && c<='Z')?1:0

注意：

a. 宏定义名与参数间不能有空格，如max(A,B);
b. 参数应用括号括起来，如(A)>(B)?(A) : (B)

常用标准库函数：字符类型判断和转换*

- #include <ctype.h>

```
int   isalpha(int c)    是否是字母
int   isdigit(int c)    是否是数字
int   islower(int c)    是否是小写字母
int   isupper(int c)    是否是大写字母
int   isspace(int c)    是否是空白字符
int   tolower(int c)    将大写字母为小写字母
int   toupper(int c)    将小写字母为大写字母
...
```

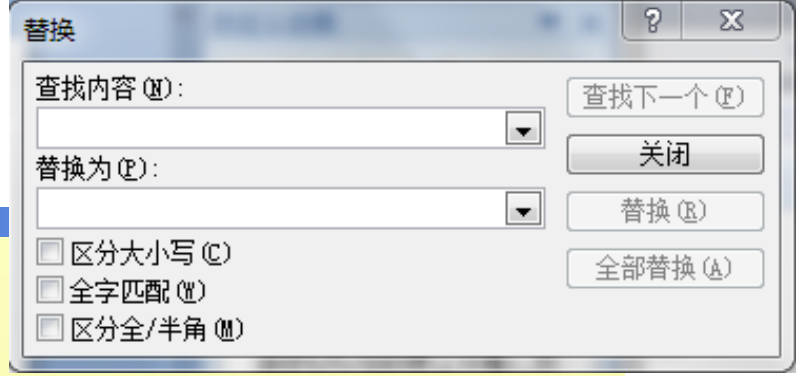
它们都是用宏函数实现的。

问题4：思考1（代码

```
#include <stdio.h>
#define MAXLINE  1000
#define tolower(c)  (c>='A'&&c<='Z' ? 'a'-'A'+c:c)
int index(char s[ ], char t[ ]);
int main( )
{
    char filename[64], s[81], line[MAXLINE];
    FILE *fp;
      scanf("%s", filename);
      scanf("%s", s);
      if((fp = fopen(filename, "r")) == NULL){
                printf("Can't open file %s!\n", filename);
                return 1;
      }
      while(fgets(line, 81, fp) != NULL)
       if(index(line, s) >= 0)
           printf("%s", line);
    return 0;
}
```

```
int index(char s[ ], char t[ ])
{
        int i, j, k;
        for(i =0; s[i] != '\0'; i++){
                for(j=i,k=0;t[k]!='\0'&&tolower(s[j])== tolower(t[k]);
                        j++,k++)
                          ;
                if(t[k] == '\0')
                          return ( i);
        }
        return ( -1);
}
```

问题4：思考2*



- 问题4.1中index只能查找的是子字符串的首次出现。请考虑如何查找子字符串的最后一次出现？
- 如果要查找一个字符串在一个文件中的所有出现（如给出所有出现的行列位置），如何实现？（注意，index只能查找子字符串首次出现，如果一行中含有多个要查找的字符串怎么办？）(类似Office软件中的查找功能)
- 在一个文件中查找给定串并用另一个替换串，如何实现？ (Office软件中的替换功能)
- 如何实现模糊查找（如UNIX命令grep），如要查找的串形式为：comp?ter, com*er (?单字符匹配，*多字符匹配）

问题4：思考3*

- 在上例中字符串查找算法用到了两层循环，有没有更好的算法？（如KMP算法，将在数据结构部分介绍）
- 另一种**字符串查找算法**（只须一层循环）

```
int index(char s[], char t[])
{
    int i=0, j=0;            //设置查找的起始位置
    while(s[i] != '\0' && t[j] != '\0') {
        if(s[i] == t[j]) {                //若字符相等，继续查找下一个字符
            i++; j++;
        }
        else {              //若字符不等，则s中退回到上次查找开始的下一个位置
            i = i-j+1;
            j = 0;
        }
    }
    if(t[j] == '\0')        //查找到字符串t，返回t在s中的起始位置
        return i-j;
    else
        return -1;
}
```

问题5：计算小岛面积

【问题描述】用一个二维方阵（最小为3X3，最大为50X50）表示一片海域。方阵中的元素只由0和1组成。1表示海岸线。计算由海岸线围起来的小岛面积（即：由1围起来的区域中0的个数）。如下图所示8X8方阵表示的小岛面积为9：

0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 1 0 1 0 0
0 0 1 0 0 0 1 0
0 1 0 0 0 1 0 0
0 1 0 1 0 1 0 0
0 1 1 0 1 0 0 0
0 0 0 0 0 0 0 0

上述方阵表示的海域满足下面两个要求：

1、小岛只有一个。

2、用1表示的海岸线是封闭的，但有可能是凸的，也有可能是凹的。

【输入形式】先从标准输入中输入方阵的阶数，然后从下一行开始输入方阵的元素（只会0或1）。

【输出形式】在标准输出上输出用整数表示的小岛面积。

【输入样例】

8
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 1 0 1 0 0
0 0 1 0 0 0 1 0
0 1 0 0 0 1 0 0
0 1 0 1 0 1 0 0
0 1 1 0 1 0 0 0
0 0 0 0 0 0 0 0

【输出样例】

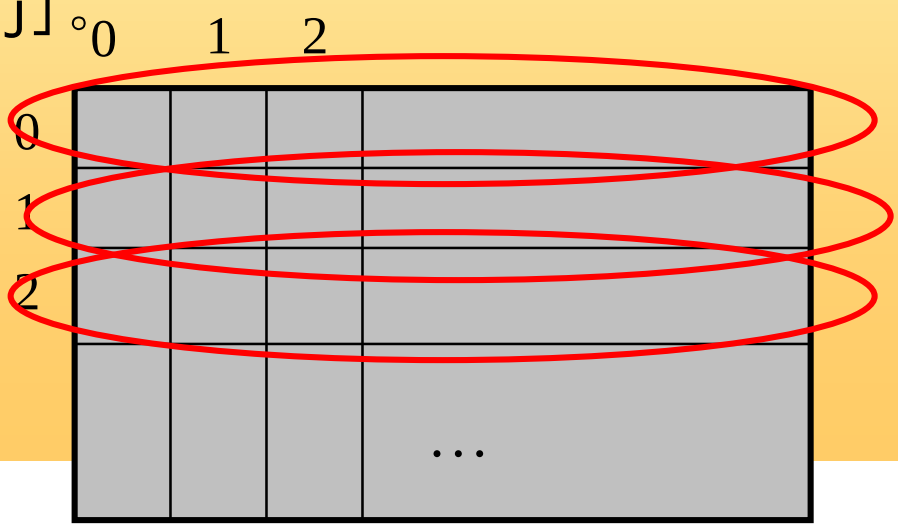
9

二维（多维）数组

- 二维（多维）数组

如， float y[4][3];

在C语言中，二维数组可以看作是一个元素为另一个一维数组的一维数组；三维数组可以看作元素为二维数组的一维数组，…。因此，在C语言中，下标变量应写作：y[i][j]，而不能写成：y[i, j]。



二维（多维）数组初始化

多维数组的初始化

```
int y[4][3] = {
        { 1, 3, 5 },
        { 2, 4, 6 },
        { 3, 5, 7 },
}
```

1	3	5
2	4	6
3	5	7
0	0	0

```
int y[4][3] = { 1, 3, 5, 2, 4, 6, 3, 5, 7 };
```

也同上，因为在C语言中，数组元素按行存贮。

```
int  y[4][3] = {
        {1}, {2}, {3}, {4}
}
int y[4][3] = {0}; /* 初始化为0 */
```

1	0	0
2	0	0
3	0	0
4	0	0

二维（多维）数组使用

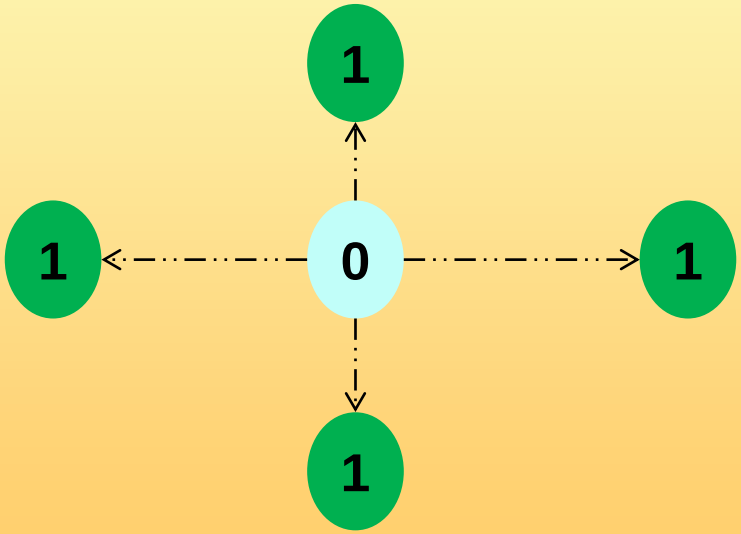
注意：如果把一个二维数组作为参数，则在函数定义中，形参数组的说明中必须指明列的数目，而行的数目可空着不写。以此类推，对于三维及以上数组，其作为参数传递时，形参说明中只能省略最内层的维数。

如：

```
main( )
{
    float a[4][3], b[3][4], c[4][4];
    …
    fun(a, b, c);
    …
}
void fun(float x[ ][3], float y[ ][4], float z[ ][4])
{
    …
}
```

问题5：算法分析

对于方阵中的任意一个元素0，如果其位于同一行上的两个1之间，并且位于同一列上的两个1之间，则该元素在1围起来的区域中（？）。



问题5：代码实现

```
#include<stdio.h>
int main()
{
    int a[50][50]={0}, i,j,k,c,n,area=0;
    scanf("%d",&n);
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    for(i=0;i<n;i++)
        for(j=0;j<n;j++) {
            c=0;
            if(a[i][j]==0) {
                for(k=i;k<n;k++)
                    if(a[k][j]==1) {
                        c++; break;
                    }
                for(k=i;k>=0;k--)
                    if(a[k][j]==1) {
                        c++; break;
                    }
                for(k=j;k<n;k++)
                    if(a[i][k]==1) {
                        c++; break;
                    }
                for(k=j;k>=0;k--)
                    if(a[i][k]==1) {
                        c++; break;
                    }
            }
            if(c==4)
                area++;
        }
    printf("%d",area);
    return 0;
}
```

指针

- 指针是用来确定另一个数据项地址的数据项。
- 指针变量是用来存放所指对象地址的变量。
- 在C语言中，允许指针指向任何类型的对象（可指向基本类型、构造类型），甚至可指向其它指针或指向函数。
- 在C语言里，当对象本身不能被直接传送的情况下，往往可以通过指针来进行传递。如函数的参数或返回结果通常是基本类型，但也可以是指向任何构造类型的指针。
- 指针应具有非零（无符号整数）值，如将0（通常#define NULL 0）赋给予指针，则该指针没有指向任何具体对象，即空指针。
- 在C语言中，指针使用得较多，指针用好了，可使程序表达能力大大加强，但用时需多加小心，要切实掌握指针的含义和用法，否则会使程序运行时乱套。

指针定义

指针的值与类型：
值：表示某个对象的位置（地址）
类型：表示那个位置上所存储对象的类型（如整数或浮点数）

- 指针变量的定义（说明）：

*<类型> *<变量>;*

指针是用所指对象类型来表征的。如：

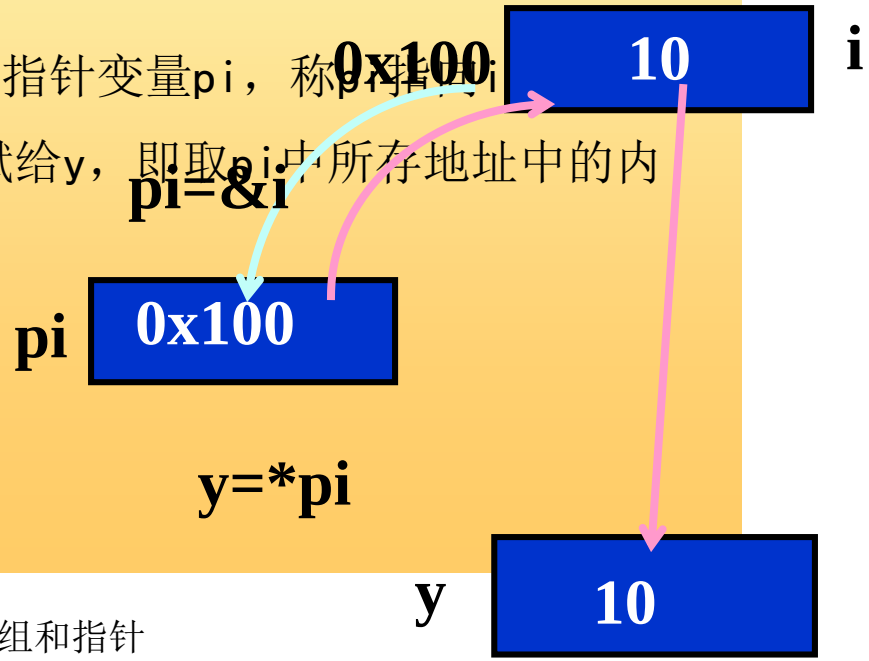
```
int   *px;                    /* 指向整型的指针 */
char  *pc;                    /* 指向字符型的指针 */
char  *acp[10];               /* 由指向字符的指针构成的数组，即指针数组 */
char  (*pac)[10];             /* 指向字符数组的指针，即数组指针 */
int  f( );                    /* 返回值为整型的函数 */
int  *fpi( );                 /* 返回值为指向整型的指针的函数，指针函数 */
int  (*pfi)(); /*指向一个返回值为整型的函数的指针，函数指针*/
```

指针运算符

- **单目（取地址）运算符 &：用来取变量或数组成员地址的运算符。**（获得某个变量的指针的运算符）
- **单目（间接引用，或递引用）运算符 *：用来取某地址中内容的运算符。**（获取指针所指对象的运算符）

例如：

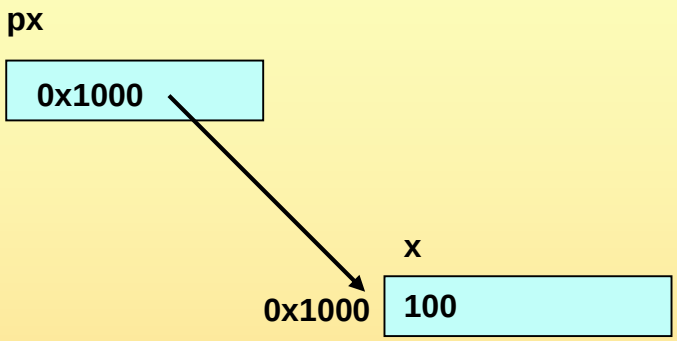
```
int  i = 10, y=20, *pi;
pi = &i;  /*将变量i的地址赋给指针变量pi，称pi指向i*/
y = *pi; /*取pi所指对象的值赋给y，即取pi中所存地址中的内容*/
```



指针和地址（续）

若定义：

```
int x = 100, y;
int *px;
```



则当：

```
px = &x;
```

则px指向具体对象x，*px则为px所指对象x的值，即100，以后凡是对x的引用，都可用*px来代替，如：

```
y = *px;
```

取px所指对象的值

指针和地址（续）

注意：使用任何指针变量之前必须先给它赋一个所指合法具体对象的地址值。

如下面用法是否有错？

1）
```
int x = 1;
int *px;
*px = x;
```
错！

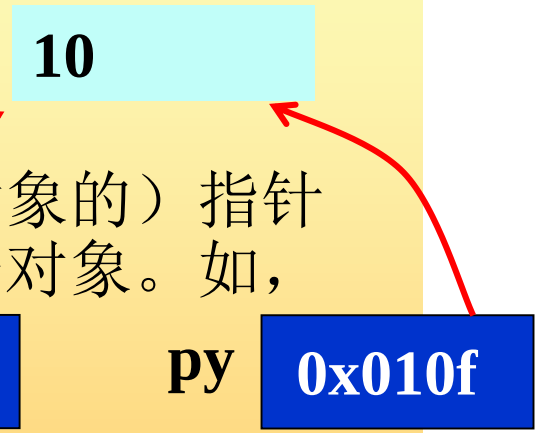
2）
```
char *string;
scanf("%s", string);
strcpy(string, "Hello");
```
错！

上面是在编写C程序时常犯的错误！

指针和地址（续）

- 如何使一个指针指向一个具体对象：
  - 通过&运算符使指针指向某个对象。如：

```
int n=10, *px;
px = &n;
```



  - 将另一个同类型的（已指向具体对象的）指针赋给它以获得值，两指针指向同一对象。如，

```
int n=10, *px,*py;
char str[10],*pstr;
px = &n; py = px;
pstr = str;
```

  - 使用malloc或alloc等函数给指针分配一个具体空间（动态存储分配）。如：

```
p = (char *)malloc(strlen(s)+1);
```

动态内存管理（malloc与free）*

- 在C中可以使用标准库函数malloc动态为指针变量申请一块内存空间（以字节为单位）（用于初始化指针变量），并返回该空间首地址。
- 。

  函数原型为：`void * malloc ( size_t  size );`
- 使用malloc初始化指针变量的常见用法：

```
char *s;
int *intptr;
s = (char *)malloc(32); /* s指向大小为32个字节（字符）的空间*/
s = (char *)malloc(strlen(p)+1);/* s指向能正好存放字符串p的空间*/
intptr = (int *)malloc(sizeof(int)*10);/* ptr指向能存放10个整型元
   素的空间*/
```

- 使用malloc申请到的动态空间在不用时应使用函数free释放。如，free(s);
- 使用malloc和free函数要用：

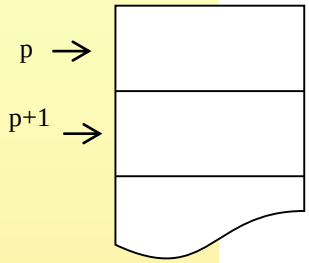
```
#include <stdlib.h>
```

运算符**sizeof**用来计算所在系统中某种类型或类型变量所占的长度（以字节为单位）。如：sizeof(int),sizeof(n),sizeof(double)
具体值取决于系统，通常int或int变量长度为4，double为8

指针运算

1. **指针和整型量可以进行加减。**

   若p为指针，则p+n和p-n是合法的，同样p++也是合法的，它们的结果同指针所指对象类型相关。如果p是指向数组某一元素的指针，则p+1及p++为数组下一元素的指针。



2. **当P1，和P2指向同一类型时，可以进行赋值。**

   如：py = px，则px，py指向同一对象。（注意与strcpy(py, px)的不同。）

```
int array[N],*p;
for(p=&array[0]; p<=&array[N-1];p++)
…
```

3. **两个指向同一类型的指针，可进行= = , > , <等关系运算，其实就是地址的比较。**

4. **两个指向同一数组成员的指针可进行相减，其结果为两指针间相差元素的个数。**

   如，p指向数组第一个元素，q指向数组最后一个元素,则q – p+1表示数组长度。

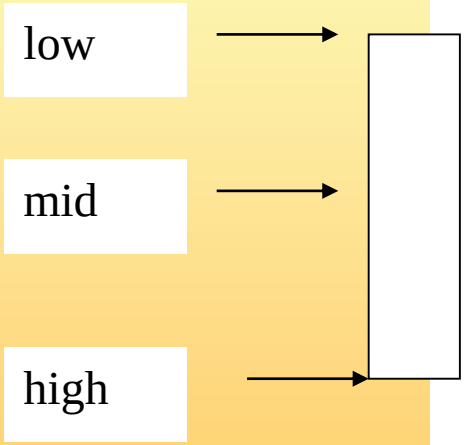
指针运算（续）

- 注意：两指针不能相加。

如右图，计算中间指针：

```
mid = (low + high ) /2
```

错!

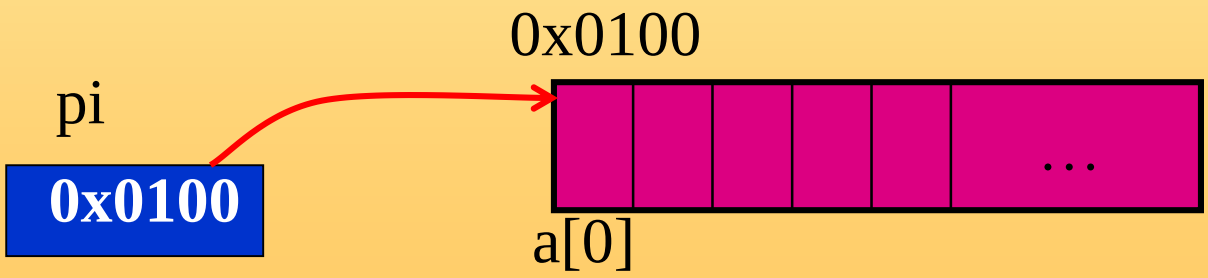


正确方法是：

```
mid = low + (high – low)/2
```

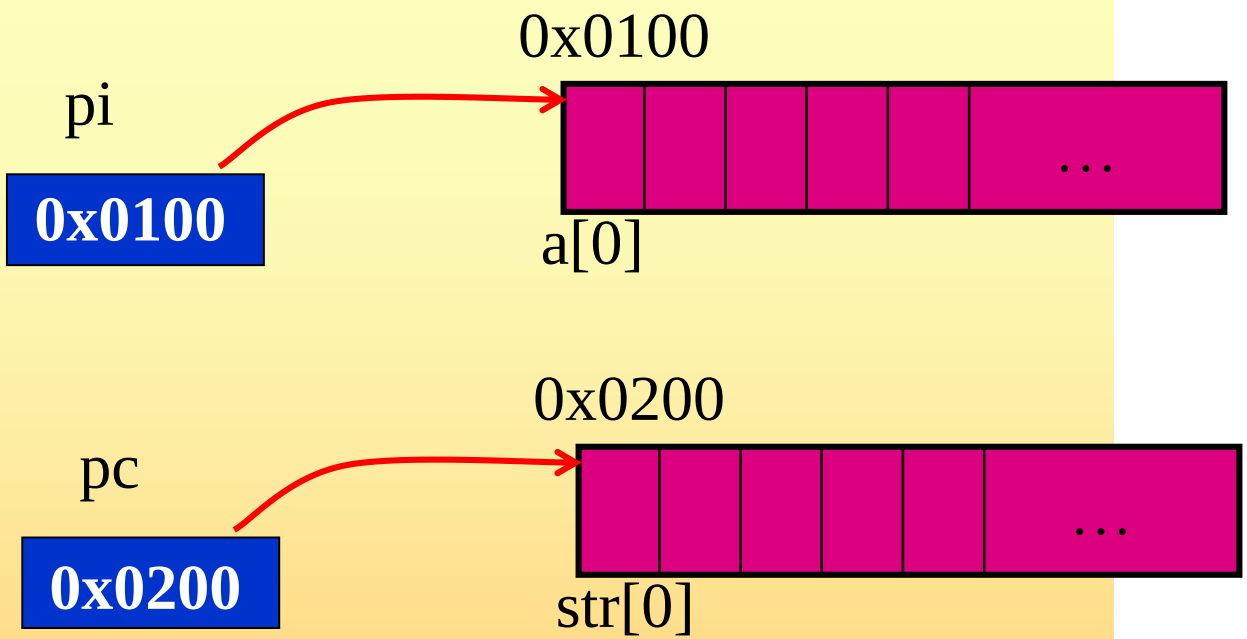
指针运算（续）

```
int a[5] = {0, 1, 2, 3, 4};
int *pi;
pi = &a[0];        /*这时pi指向a[0]*/
pi++;              /*这时pi指向a[1]，*pi为a[1], 即1*/
pi+=2;             /*这时pi指向a[3]，*pi为a[3], 即3*/
```



指针运算（续）

```
int  a[10];
int  *pi= &a[0];
char str[10];
char *pc= &str[0];
```



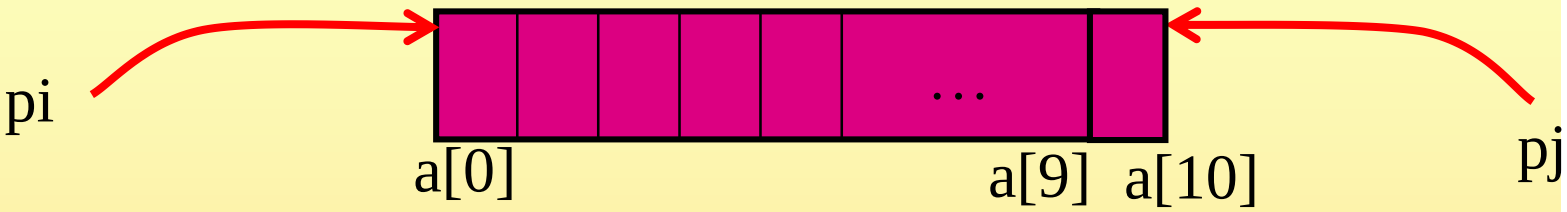

```
pi++;
```

/*若int占4个字节, 此时pi增加一个单位, 即四个字节, 结果为0x0104*/

```
pc++;
```

/*若char占1个字节， 此时, pc增加一个单位,即一个字节,结果为0x0201*/

指针运算（续）




```
for (pi=&a[0],pj=&a[N-1]; pi<=pj; pi++)
    …
```

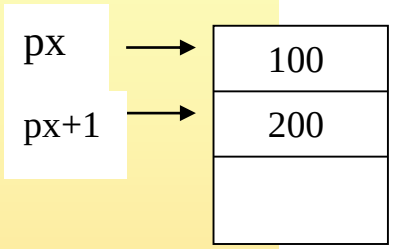
用来遍历一个数组

指针运算（续）

- 指针运算分析：
  - p++和p+1的区别；
  - y = *px + 1和y= *(px + 1)的区别；
  - y = (*px)++和y = *px ++的区别；

指针运算（续）

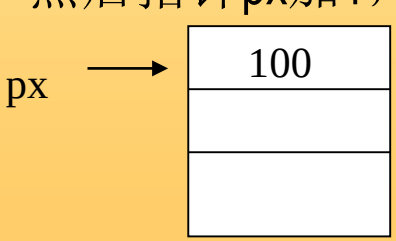
- p++结果为p指向下一元素；p+1结果为下一元素的指针，但p本身不变。
- *px+1为取px所指对象内容加1；*(px+1)为px指针加1，并取结果指针所指对象内容；如右图：

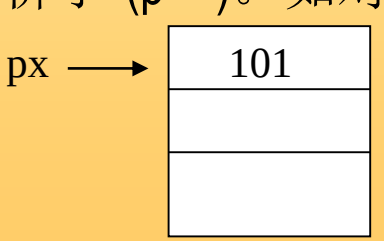


y= *px+1 = **101**

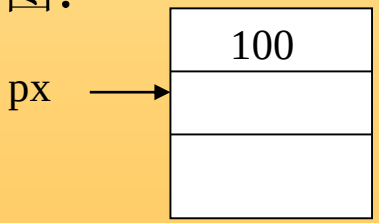
y= *(px+1) = **200**

- (*px)++为先取px所指对象内容进行运算，然后对其加1；*px++为先取px所指对象内容进行运算，然后指针px加1，其等价于*(p++)。如对下图：





Y=(*px)++ =**100**



Y=*px++ = **100**

问题6

- 问题：输出输入行中的最长行
- 数据结构设计
  - 设两个一维字符数组来存储新输入行及当前最长行
- 主算法设计

```
While(还有新输入行)
    If(新行比以前保存的最长行更长)
        保存新行及其长度;
输出所保存的最长行;
```

如何从标准输入中输入一行？如何判断输入结束？

```
while(gets(s)!=NULL)
    …
```

如何比较两个字符串长度大小？需要计算字符串长度：

```
int str_len(char s[ ]);
```

如何保存一个字符串？需要拷贝一个串至另一个串：

```
int str_copy(char s[], char t[]);
```

问题6：算法设计

- 计算字符串（即输入行）长度
  - 函数str_len(char s[])

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

- 保存字符串（即输入行）
  - 函数str_copy(char s[], char t[])

```
i = 0;
while ((s[i] =t[i]) != '\0')
    i++;
```

问题6：代码实现

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

void str_copy(char s[ ], char t[ ])
{
    int i = 0;
    while((s[i] =t[i] )!=  '\0' )    i++;
 }
```

问题6：代码实现（续）

```
/* c5_2.c */
#include <stdio.h>
#define MAXLINE              1024
int str_len(char s[ ]);
void str_copy(char s[ ], char t[ ]);
int main( )      /* find longest line */
{
        int len;                /* current line length */
        int max;                /* maximum length seen so far */
        char line[MAXLINE];     /* current input line */
        char save[MAXLINE]; /* longest line saved */
        max = 0;
        while( gets(line)  != NULL ){
                len = str_len(line);
                if( len > max ) {
                  max = len;
                  str_copy(save, line);
                }
        }
        if( max > 0)
                printf("%s", save);
        return 0;
}
```

char* gets(char s[ ])从标准输入中读入一行到数组s中，但换行符不读入，数组以’\0’结束。若输入结束或发生错误，则返回NULL

问题6：常见问题分析

- 一个错误的str_copy函数实现案例：

```
void str_copy(char s[], char t[])
{
    int i = 0;
    while(t[i] != '\0'){
       s[i] = t[i];
       i++;
    }
}
```

错误原因：字符串结束符（'\0'）没有拷贝到字符串s中。

- 一个错误的str_len函数实现案例：

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

错误原因：字符串长度多算了一个。

常用标准字符串库函数

使用strcpy、strcat函数之前，必须保证s有足够的空间容纳操作后的字符串！

- #include <string.h>

```
int strlen(char s[]);    /*计算字符串长度，字符串以\0结果*/
char *strcpy(char s[], char t[]); /*将字符串t拷贝到字符串s中*/
char *strcat(char s[], char t[]);  /*将字符串t拷贝到字符串s尾部*/
int strcmp(char s[], char t[]);  /*比较两个字符串,若s>t,则返回大于0
   的数;若s<t,则返回小于0的数;若相等, 返回0 */
```

下面是C提供的高效函数接口（不仅仅用于字符串）：

```
void *memcpy(void *s, void *t, int n);
void *memmove(void *s, void *t, int n);
int memcmp(void *s, void *t, int n);
```

子曰：工欲善其事，必先利其器。…

问题6：指针方式实现

- 可用指针方式实现问题6（输出输入行中的最长行）。
- 算法设计：

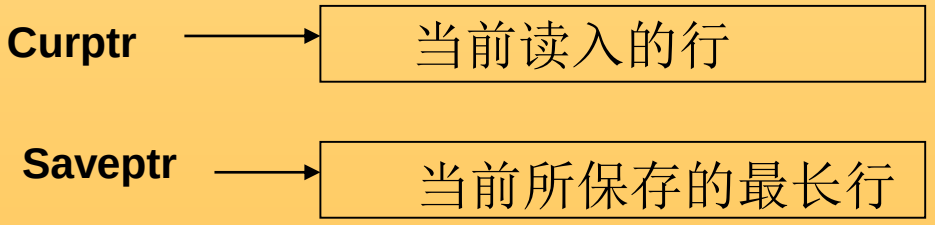
设指针变量Curptr和Saveptr分别指向当前行（新行）和当前最长行

While(还有新输入行)

If(Curptr所指向的行比Saveptr所指向的行长)

交换Curptr和Saveptr指针并保存新行长度;

输出Saveptr所指内容



问题6：代码实现（指针方式）（续）

```
#include <stdio.h>
#define MAXLINE              1024
int str_len(char s[]);
int main( )        /* find longest line */
{
      int len,   max;      /* current length and maxim
      char *curptr, *saveptr,*tmp; /* current  line
          */
      char  save1[MAXLINE], save2[MAXLINE];
      curptr = &save1[0];
      saveptr= &save2[0];
      max = 0;
      while( gets(curptr)  != NULL ){
            len = str_len(curptr);
            if( len > max ) {
                max = len;
                tmp = curptr;
                curptr = saveptr;
                saveptr = tmp;
            }
      }
      if( max > 0)
            printf( "%s" , saveptr);
      return 0;
}
```

初始化指针使其分别指向一个数组

保存新行长度

交换指向当前行和所保存行的指针

```
#include <stdio.h>
#define MAXLINE              1024
int str_len(char s[ ]);
void str_copy(char s[ ], char t[ ]);
int main( )      /* find longest line */
{
      int len;                  /* current line length */
      int max;                  /* maximum length seen so far */
      char line[MAXLINE];   /* current input line */
      char save[MAXLINE]; /* longest line saved */
      max = 0;
      while( gets(line)  != NULL ){
            len = str_len(line);
            if( len > max ) {
              max = len;
              str_copy(save, line);
            }
      }
      if( max > 0)
            printf("%s", save);
      return 0;
}
```

问题6：代码实现（指针方式）（续）

- 与数组实现方式相比，指针实现方式减少了每当发现新的更长行时所进行的字符数组拷贝（通过调用函数str_copy）。显然指针实现方式代码执行速度要快。

指针作为函数参数

- 在C中函数参数传递方式为“传值”。这种方式的最大好处是函数调用不会改变实参变量的值。如何通过函数调用来改变实参变量的值？如通过函数swap(a,b)来交换两个变量的内容。
- 可以通过将指针作为函数参数来改变实参内容。例如：

指针作为函数参数（续）

形参定义为指针

如何通过函数swap交换实参变量a和b？正确的做法应为：

```
void swap ( int *px, int *py)
{
    int temp;
    temp = *px;   /*间接取*/
    *px = *py;    /*间接取，间接存*/
    *py = temp;   /*间接存*/
}
 main( )
{
    int a =2, b = 3;
    swap （ &a, &b);
}
```

&a

实参传递的是变量的地址

在前面介绍函数时，说明不能通过调用下面函数swap(a,b)调用达到交换两个变量的值的目的。

```
void swap ( int x, int y)
{
      int temp;
       temp = x;
      x = y;
      y = temp;
}
main( )
{
   int a=2,b=3;
```

下面用法亦不能达到交换两个变量的值的目的。

```
void swap ( int *px, int *py)
{
      int *temp;
       temp = px;
      px = py;
      py = temp;
}
main( )
{
   int a=2,b=3;
   swap(&a,&b);
}
```

指针作为函数参数（续）

因此，***在一定要改变实参变量内容时，应把函数的形参显式地说明为指向实参变量（类型）的指针，相应地调用时应该用变量的地址值作为参数。***

提示：这也是为什么用scanf读int, char, double类型数据时要取变量地址。因为需要改变变量内容。

尽管C的函数参数和函数返回值一般应为基本类型，但它们却可以是指向任何类型（包括复杂的结构类型，甚至其它函数）的指针，这就大大扩充了C的功能和应用范围。

指针和数组

在C语言中，数组的名字就是指向该数组第一个元素（下标为0）的指针，即该数组第一个元素的地址，也即数组的首地址。

例如：

```
int a[10], x;
int *pa;
```

若：

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

则：

```
x = *pa;          x = a[0];          x = *a;
x = *(pa + 1);    x = a[1];          x = *(a+1);
x = *(pa+i);      x = a[i];          x = *(a+i);
```

其实pa = &a[0] 可以写成pa = a;

一般有：a[i] = *(a+i)

但特别注意：数组名和指针（变量）是有区别的，*前者是常量，而后者是变量。* 因此，尽管我们可写pa =a; 但决不能写：a = pa ; a++; pa = &a; 等。 （也就是说，数组一经定义，其首地址将不允许改变。）

指针和数组（续）

数组名可作为参数进行传递。当将数组名传给函数时，实际上所传递的是数组的开始地址。（即数组第一个元素的地址）

为什么要使用指针？

- 扩展了语言的功能，如通过传递指针来修改实参变量、或通过返回指针来返回数组等；
- 能够更方便地组织和操作数据，如，离散数据的组织和访问（链表，树等）；
- 能够提高程序的性能。

指针和数组（续）

```
char a[]="hello", *p="hello";
a[0] = 'b';          /*正确 */
*p = 'b';  /*错误，不能修改常量值*/
```

对于字符串常量，可以把它看成一个无名字符数组，C编译程序会自动为它分配一个空间来存放这个常量，字符串常量的值是指向这个无名数组的第一个字符的指针，其类型是字符指针。

所以，printf("a constant character string\n"); 传递给函数的是字符串第一个字符的指针。

注意：字符数组和字符指针使用时容易混淆。

例：

```
char  *char_ptr, word[20];
char_ptr = "point to me";

word = "you can 't do this";
```

正确， 把字符串常量第一个字符指针赋给指针变量。

正确做法为:strcpy(word, "…");

指针和数组（续）

假设：int values[100], *intptr = values, i;

表：指针与数组的关系

表达式	值
&values[0] values intptr	指向values数组第一个元素的指针
values[0] *values *intptr	values数组的第一个元素
&values[i] values+i intptr+i	指向values数组第i+1个元素的指针
valuse[i] *(values+i) *(intptr+i), intptr[i]	values数组的第i+1个元素

指针和数组（续）

在函数定义中形参形式char s[ ]和char *s完全等价，即指向某类型的指针与该类型没有指明长度的数组是同一回事。用哪个取决于在函数里表达式的写法。

例：用指针和数组两种方式实现strlen函数

1) 数组方式

```
int strlen( char s[ ])
{
    int n = 0;
    while(s[n] != '\0' )
       ++n;
    return (n);
}
```

2) 指针方式

```
int strlen(char *s)
{
    int n;
    for(n=0; *s != '\0' ; s++)
       n++;
    return (n);
}
```

```
main( )
{
    char st[100];
    scanf("%s", st);
    printf("%d\n", strlen(st));
}
```

或

```
main( )
{
    char *st;
    int l;
    st = "C Language";
    l = strlen(st);
    printf("length=%d\n", l);
}
```

建议：由于指针方式可读性明显不如数组方式，而且容易出错，对于初学者来说建议使用数组方式来访问元素。

指针和数组（续）

例：用指针和数组两种方式实现strcpy函数

1) 数组方式

```
void strcpy(char s[ ], char t[ ])
{
    int i = 0;
    while((s[i] = t[i]) != '\0' )
        i++;
}
```

2) 指针方式

```
void strcpy(char *s, char *t)
{
    while((*s = *t) != '\0' ){
        s++; t++;
    }
}
```

或

```
void strcpy(char *s, char *t)
{
    while(*s++ = *t++)
        ;
}
```

指针和数组（续）*

- 数组就是一个（常量）指针，数组的名就是指向数据第一个元素的指针。
- 数组可以按指针方式使用，如：

```
int a[10],*p;
for(i=0; i<n; i++) …*(a+i)…;
```

- 指针亦可按数组形式访问，如：

```
void fun(int p[]){
  …p[i]…
```

忠告：对于初学者来说，将指针用数组形式访问不仅可读性好，且不容易出错！

指针数组

- 单个字符串（人名、单词、文件名、文件中一行）我们知道可以用一个字符数组来保存，如：

  char word[32]= “Wednesday”

- 若要同时保存多个相关字符串（通讯录中的人名单、英文字典中单词表、文件中的多行...)?

  如何同时保存如下相关数据：

  Sunday, Monday, Tuesday, Wednesday,Thursday, Friday,Saturday

  方法：

  - 二维字符数组，如：char days[7][10];
  - 指针数组，如：char *days[7];

指针数组

- **指针数组就是由指针组成的数组，即该数组的每一个元素都是指向某一类型对象的指针。**

如：

```
char *lineptr[100];        //由字符指针构成的数组
int  *iptr[50];            //由整型指针构成的数组
```

指针数组（续）

- **指针数组与二维数组的区别：**

1) 二维数组：

```
char days[7][10] = {
        “Sunday”, “Monday”, “Tuesday”, “Wednesday”,
        “Thursday”, “Friday”, “Saturday”
};
```

存贮形式：

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
days[0]	S	u	n	d	a	y	\0			
days[1]	M	o	n	d	a	y	\0			
days[2]	T	u	e	s	d	a	y	\0		
days[3]	W	e	d	n	e	s	d	a	y	\0
days[4]	T	h	u	r	s	d	a	y	\0	
days[5]	F	r	i	d	a	y	\0			
days[6]	S	a	t	u	r	d	a	y	\0	

使用方式：

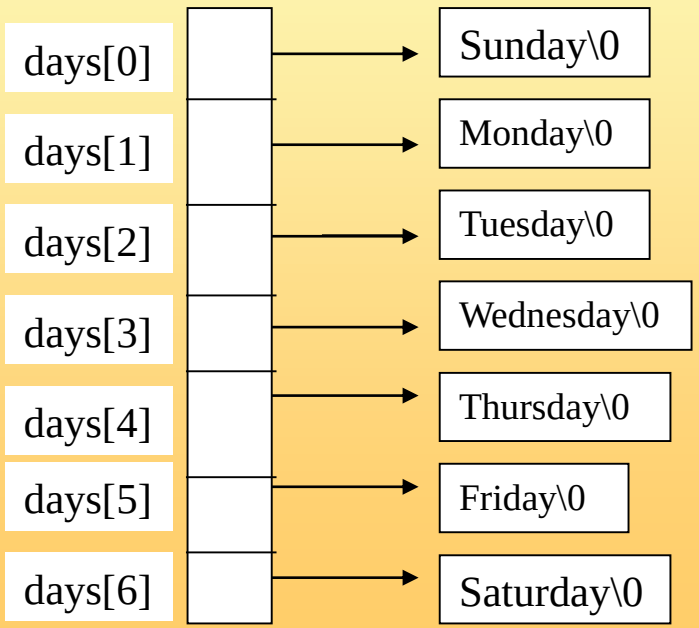
```
scanf(“%s”, days[i]);
gets(days[i]);
if(strcmp(days[i], “Friday”)==0)
    …
```

指针数组（续）

2）指针数组

```
char *days[7] = {
        “Sunday”,  “Monday”,
        “Tuesday”,  “Wednesday”,
        “Thursday”,  “Friday”,
        “Saturday”
};
```

存贮形式：



使用方式：

```
char buf[32],*days[7];
scanf(“%s”, buf);
days[i] = (char *)malloc(strlen(buf)+1);
strcpy(days[i], buf);

if(strcmp(days[i], buf)==0)
   …
```

指针数组（续）

- 比较上面两个例子，可以看出尽管二维字符数组与字符指针数组在存储形式上不同，但**它们在初始化形式以及使用方式上却是相同的**。

  例如，无论是指针数组，还是二维数组，下面两种形式访问的都是同一个元素，结果都是字符串”Friday”中的字符'y'。

  ***(days[5]+5) = days[5][5] = ‘y’**

  无论是二维数组还是指针数组，days[5]访问的都是字符串(即指向字符串的指针)”Friday”

- **使用指针数组来存放不同长度的字符串可以节省存贮空间，如，存放多个单词串、行。**例如，如果要保存从标准输入或文件中读入的行，字符指针数组是一个好的选择。因为读入的行可能长短差异很大。下面程序片段即为保存从标准输入中读入的多行：

指针数组（续）

```
/* read lines from input */
#define MAXLENGTH                512
#define MAXLINES                 1000
…
char *lineptr[MAXLINES], buf[MAXLENGTH];
int i;
…
i = 0;
while(gets(buf) != NULL){
        lineptr[i] = (char *)malloc(strlen(buf)+1);
        strcpy(lineptr[i], buf);
        i++;
}
…
```

指针数组（续）

例：将数字表示的月分转换成英文表示的月分（指针数组的初始化）。

```
char *month_name(int n)
{
    static char *name[ ] = {
      "illegal month",
      "January",
      "February",
      "March",
      "April",
      "May",
      "June",
      "July",
      "August",
      "September",
      "October",
      "November",
      "December"
    };
    return ((n<1 || n>12) ? name[0] : name[n]);
}
```

指针数组（续）

许多命令在执行时除了提供命令名之外，还要给出一定的参数，如在Windows命令行窗口中执行命令：

C>copy file1 file2

在此，file1和file2被称为**命令行参数**。在实际应用(特别是Linux下)时，经常会需要编写带命令行参数的程序。

- **命令行参数**

在C语言中，主函数main还可以带有参数，形式如下：

```
int main( int argc, char *argv[ ])
```

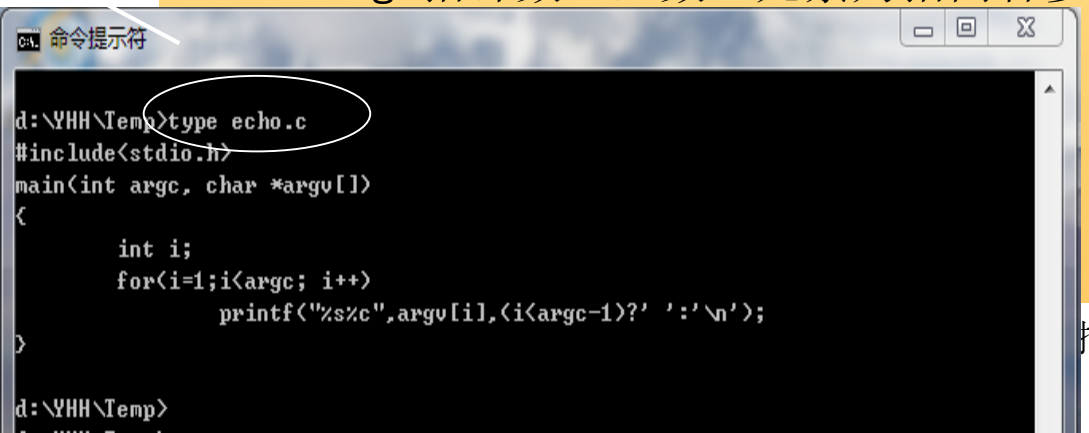
或

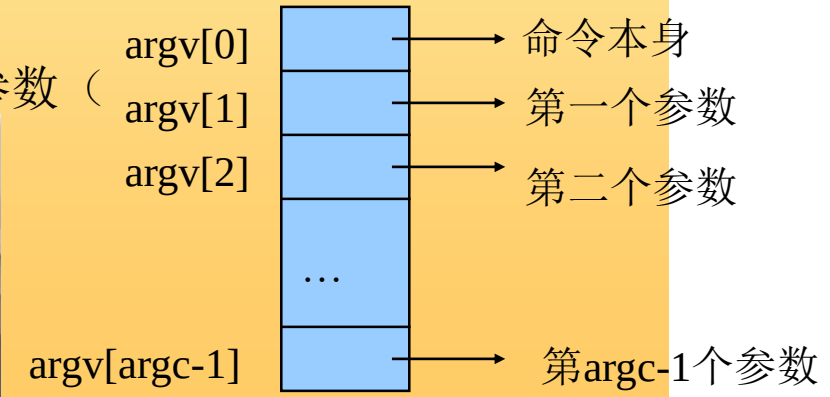
```
int main( int argc, char **argv)
```

其中：

命令及命令行参数

- argc包含命令本身在内的参数个数
- argv指针数组，数组元素为指向各参数（





问题7

- 问题：实现一个命令echo，其将命令后的正文串显示在屏幕上，如：

```
C> echo  hello world
```

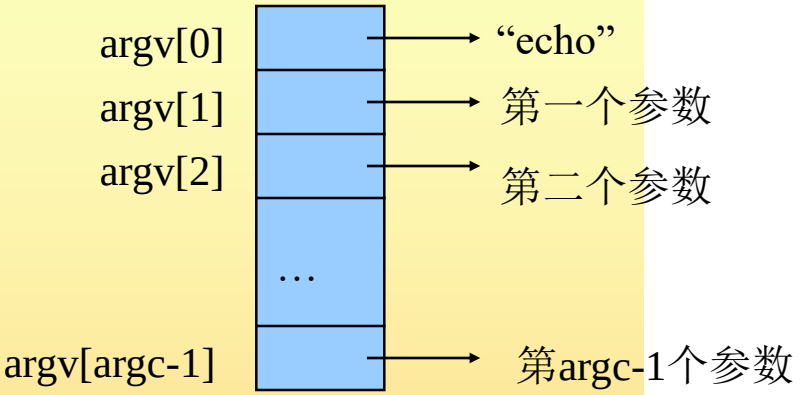
屏幕输出：

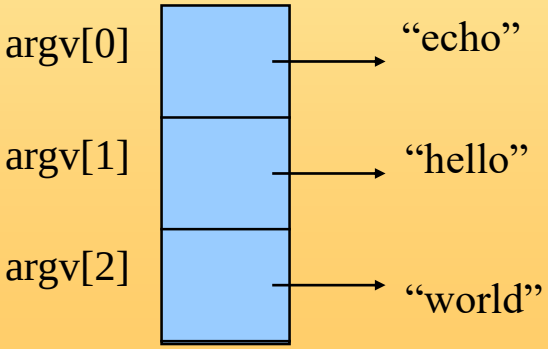
```
hello world
```

问题7：算法分析

- 从右图可知，使用下面循环就可输出所有命令行参数：

```
for(i=1; i<argc; i++)
    printf("%s ", argv[i]);
```





C> echo hello world

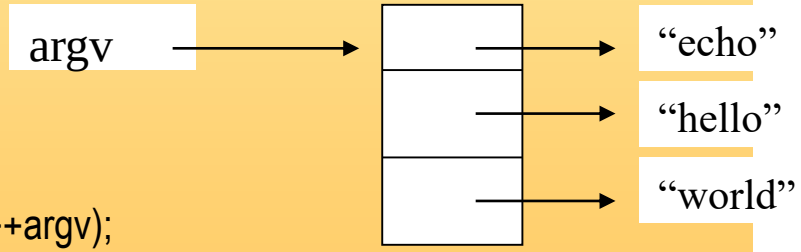
问题7：代码实现

- 实现一：

```
main( int argc, char *argv[ ])
{
    int i;
    for(i=1; i<argc; i++)
        printf("%s%c", argv[i], (i< argc-1)? ' ': '\n');
}
```

- 实现二：

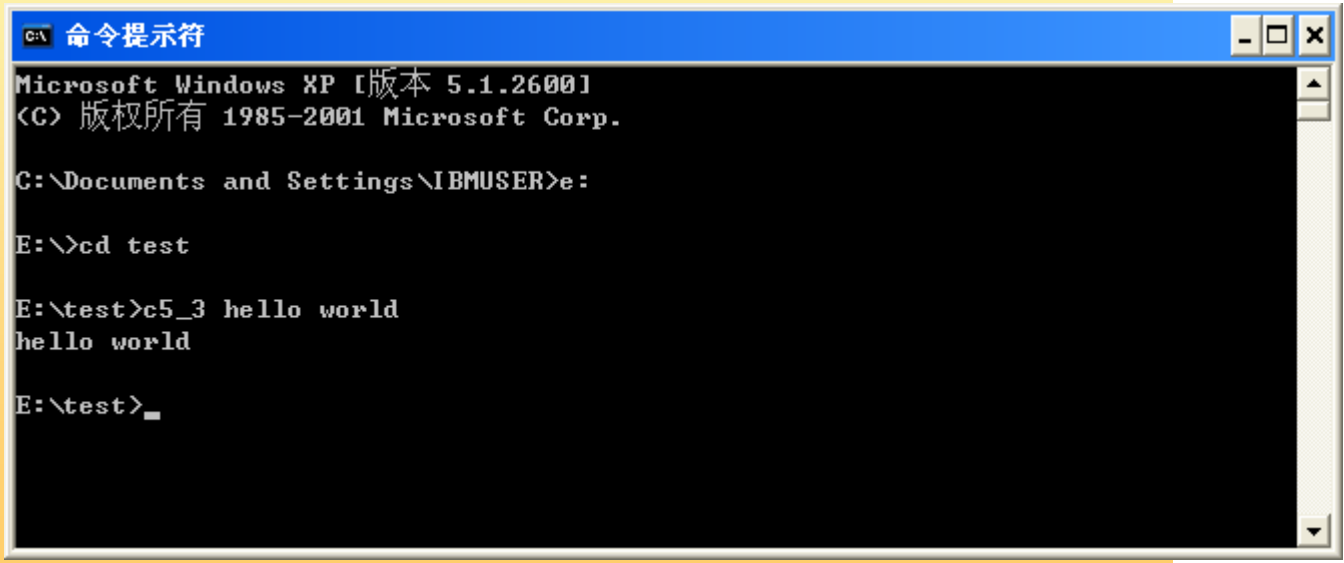
```
main(int argc, char *argv[ ])
{
    while(--argc > 0)
        printf((argc > 1)? "%s " : "%s\n", *++argv);
}
```



问题7：如何运行命令行程序

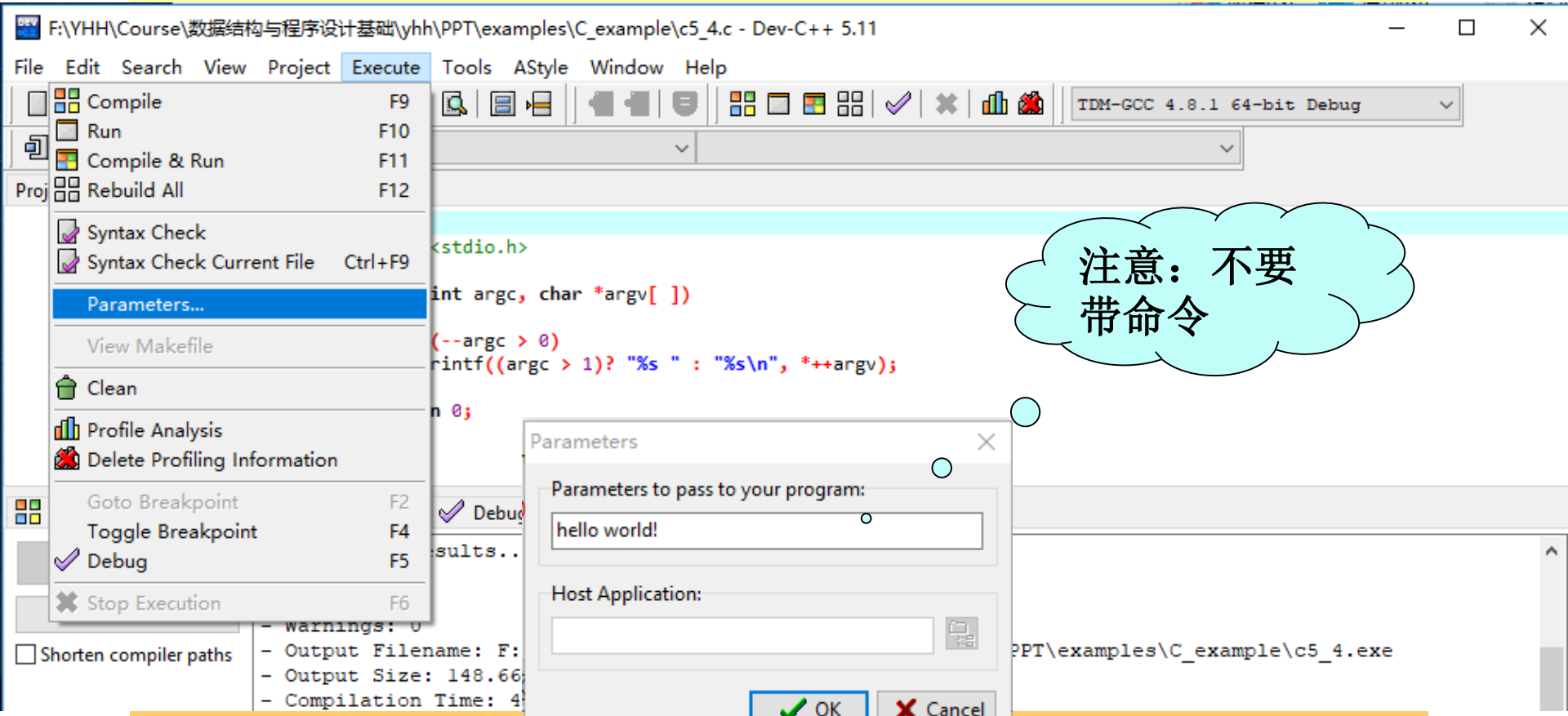
- **方式一：在命令窗口（DOS窗口）中直接运行；**

  若c5_4.exe执行文件在\test目录下，则从<开始>菜单中<附件>中找到<命令提示符>，并执行。然后，转到test目录下执行c5_4.exe文件。



问题7：如何运行命令行程序（续）

- 在DEV C++环境下运行



函数指针*

- 函数指针

  即指向函数的指针。

  函数指针说明形式为：

  *类型　　（*标识符）（ ）；*

  例：int (*fp)( );　　注意：与int *fp( )；的不同

  对函数指针赋值，可通过赋值语句或参数传递。

  函数指针 = 函数名；

  （在C语言中，函数名是作为该函数的指针来处理，也就是说函数名就是指向函数的指针）

函数指针*（续）

例：

```
int leapyear( int year);
main( )
{
    int (*fnptr)( ),result;
    fnptr = leapyear;
    result = (*fnptr)(2000);     /* 与调用leapyear(2000)完全等价*/
}

int leapyear( int year)
{
    if(((year % 4 == 0) && (year % 100 != 0)) || (year % 400) == 0)
      return ( 1);
    else
      return ( 0);
}
```

说明：在C中，通过函数指针调用一个函数时，(*fnptr)(2000)与fnptr(2000)用法等价。

典型错误案例分析


```
int main()
{
    char *string,c;
    scanf("%s\n",string);
    scanf("%c",&c);
    insert(*string,c);
    return 0;
}
```

典型错误案例分析（续）

```
#include <stdio.h>
char *insert(char *string,char c)
{
    int i;
    char s[50];
    for(i=0;*string<c;i++)
        s[i]=*string++;
    s[i++]=c;
    for(;*string!='\0';i++)
        s[i]=*string++;
    return s;
}
void main()
{
    char s1[50],c;
    scanf("%s\n",s1);
    scanf("%c",&c);
    printf("%s",insert(s1,c));
}
```

定义了一个局部数组

该循环没有复制\0。字符数组s没有\0

不能返回一个局部数组。因为它的生存期为当前函数。

或:

```
char *insert(char *string,char c)
{
    int i;
    char s[50], *s1=string;
    for(i=0;*s1<c;i++)
        s[i]=*s1++;
    s[i++]=c;
    for(;(s[i]=*s1++)!='\0';i++)
        ;
    for(i=0;(string[i]=s[i])!='\0';i++)
        ;
    return string;
}
```

典型错误案例分析（续）

```
#include<stdio.h>  //作业单词排序
#include<stdlib.h>
#include<string.h>
int main (int argc,char *argv[])
{
    int n,i=0,j=0,k,t,m;
    char c;
    char s[1024][100];
    FILE *in ,*out ;
    in = fopen (argv[1],"r");
    out = fopen (argv[2],"w");
    while ((c=fgetc(in) )!= EOF)
    {
        if (c == ‘ ’)
        {
        i++;
        j=0;
        }
        else
        s[i][j++] = c;
    }
 …..
```

单词间只能有一个空格,多个出错

读下一个单词，单词无’\0’

```
while(fscanf(in, “%s”, s[i]) != -1)
    i++;
```

结构（struct）

- 在实际应用中大量应用软件（如产品管理软件、学籍管理软件、人事管理软件...）经常需要处理诸如产品单、学生记录表、员工记录表（如下）...等数据
- 在C中用何数据结构来存储一组相关数据（记录、表）？

ID	姓名	单位	住址	工资	出生年月		
					年	月	日
1	张三	北航计算机学院	北航家属楼xxx	3000	1980	5	18
…	…	…	…	…	…	…	…

问题8

- 问题：编写 一个程序，统计输入中C语言每个关键字的出现次数。

问题8：问题分析

- 按目前掌握的知识，需要设两个数组分别存储C关键字列表和相应出现次数：

```
char *keyworklist[N];   /*存储关键表*/
int count[N];           /*存储关键出现次数*/
```

其中N为C中关键字的数目

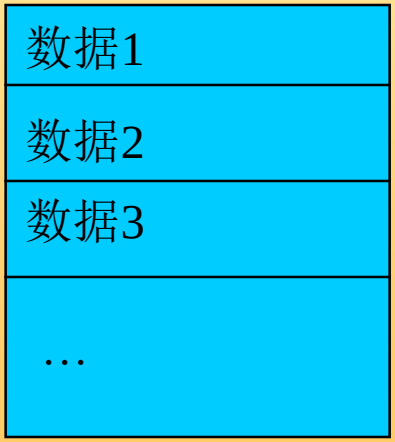
- 这样处理相互关联的数据（如某一关键字和其出现次数是相关的）的问题：
  - 数据的处理不方便，如插入/删除一个关键字及出现次数要分别操作两个数组。
- 如何组织不同类型的相关数据？

结构

结构说明

- . 结构（struct）的表示和含义

结构是由若干分量组成的一种构造类型。然而，组成结构的各个分量可以具有不同的类型（这和数组情况相异），并且，对结构变量的访问必须通过它的分量名字（亦称为成员名），而不像数组是通过下标来访问它的成员。



结构说明（续）

说明形式：

```
struct    结构类型名 {
        成员类型   成员名;
        成员类型   成员名;
        ...
        ...
};
```

结构说明（续）

如下面为用以表示日期相关信息的结构说明：

```
struct  date {
    int day;
    int month;
    int year;
    int yearday;
    char mon_name[4];
};
```

结构名

结构成员

上面的结构说明，只是定义了一个结构的模板（template）或称为结构的框架，而并未定义结构的对象，也不为它分配存储空间。有了这样的结构模板说明后，一个结构变量可定义为：

```
struct  date d1, d2;
```

注意：在此，**关键字struct和结构名date都不可少，可以把struct date一起看作是某种类型说明符（结构类型）。**

结构变量说明（续）

在说明结构模板时，一般都要有结构名，但也可不用结构名，直接把结构模板和变量定义（或说明）放在一起。下面是结构变量的几种说明方式：

1）无结构名

```
struct {
  …
} s1, s2;
```

一般适用于说明本地变量。

2）有结构名

```
struct date {
  …
} ;
struct date s1, s2;
```

通常用来说明外部结构变量，或需要在多个函数中用到的相同的结构的变量。

3）使用typedef

```
typedef struct {
  …
} DATE;
```

类型定义

则变量定义为：DATE d, *pd, ad[10];

使用typedef定义结构类型名后，结构变量的定义（或说明）就更简洁了。

类型定义（typedef）

类型定义的语法格式为：

typedef *原类型名* *新类型名*

如：

```
typedef  int  LENGTH;
typedef  char  *STRING;
```

变量说明为：

```
LENGTH  len, maxlen;
STRING  lineptr[LINES], alloc( );
```

这与如下直接说明等价：

```
int  len, maxlen;
char  *lineptr[LINES], *alloc( );
```

typedef常见用法：

1. 一些安全关键的软件中需要在程序中明确运行环境的数据类型长度，如：

```
typedef int INT32;
typedef short INT16
INT32 port0,port1;
…
```

2. 用来定义结构类型，如FILE就是一个用typedef定义的结构类型。
3. 数据结构中用来定义一个链表结点类型：

```
typedef struct node {
    int n;
    struct node *next;
} *Nodeptr;
Nodeptr list, p;
```

实际等价于：

```
struct node *list,*p;
```

类型定义（续）

必须强调，typedef说明均不产生新的数据类型，也不定义存储单元，它只是给已有的类型又增添了新的类型名，没有产生新的语义，即用这种方法所说明的变量与明确指出说明的那些变量有相同的性质。

类型定义的必要性：

- 将程序参数化，便于移植；
- 为程序提供较好的说明信息，

类型定义的一个常见用法是用来定义
常用的文件类型FILE，就是结构类型

```
struct _iobuf {
   char *_ptr;
   int _cnt;
   char *_base;
   int _flag;
   int _file;
   int _charbuf;
   int _bufsiz;
   char *_tmpfname;
 };
 typedef struct _iobuf FILE;
```

结构说明（续）：结构嵌套

- 结构成员可以具有各种类型，当然它也可以是其它的结构类型，即结构可以嵌套定义。如下面为描述个人信息（**姓名、住址、单位、薪水、生日**）的结构说明：

```
struct person {
        int ID;
        char name[32];
        char address[64];
        char department[64];
        double salary;
        struct date birthdate;
};
struct person table[100];
```

ID	姓名	单位	住址	工资	出生年月		
					年	月	日
1	张三	北航计算机学院	北航家属楼xxx	3000	1980	5	18
…	…	…	…	…	…	…	…

结构说明（续）

- 注意：*结构成员的类型不能是该结构本身，因为它无法确定此结构的边界。但**它可以是指向本身结构的指针。***例如：

```
struct keyword {
    char  *name;
    int   count;
    struct  keyword  next;
} *base;
```

错误

```
struct keyword {
    char  *name;
    int   count;
    struct  keyword  *next;
} *base;
```

正确

结构变量初始化

- 结构变量在定义时可以初始化，如：

```
struct date d = { 27, 12, 1984, 361,
   “Dec” };
```

- 结构变量亦可通过整体赋值来初始化，如：

```
struct date d1,d2 = { 27, 12, 1984, 361,
   “Dec” };

d1 = d2;
```

结构成员的引用

通过

结构变量名. 成员名

来访问结构成员，如：

若定义了结构变量：struct person emp;

则给emp的出生年、月赋值可写成：

```
emp.birthdate.year  = 1970;
emp.birthdate.month = 8;
```

如定义：struct date *pd, d; （pd是指向struct date的指针）

则：pd = &d （指向一个已有结构变量）

或：pd = (struct date *) malloc(sizeof(struct date)); （指向一个新的结构空间）

注意：与其它指针变量一样，定义了pd并不表示pd有了它所指对象。

在C中，运算符->专门用于存取指向结构的指针所指对象成员，如：

```
pd->year = 1958;
```

strcpy(pd->mon_name, “OCT”); 等等。

实际上，pd->year与(*pd).year 是完全等价的，这是由于在C语言中指向结构的指针使用非常频繁，因此，特为此设立了一个新运算符（->）。

结构成员的引用（续）

- 结构变量可进行如下操作：
  - 访问结构成员，如：emp.name
  - 作为一个整体复制、赋值，如：

```
struct person e1, emp ;
…
e1 = emp;
```

  - 取地址运算（&），如：

```
struct date d, *pd;
pd = &d;
```

结构成员的引用（续）

例：复数（加）运算

```
#include <stdio.h>
struct complex {
    float  real;
    float  imag;
};
struct complex addComplex(struct complex c1, struct complex c2);
int main()
{
    struct complex c1, c2, c3;
    scanf("%f %f %f %f", &c1.real, &c1.imag,&c2.real, &c2.imag);
    c3 = addComplex(c1, c2);
    printf("(%.2f, %.2f) + (%.2f, %.2f) = (%.2f, %.2f)\n", c1.real,
   c1.imag, c2.real, c2.imag, c3.real, c3.imag);
    return 0;
}
```

结构成员的引用（续）

```
struct complex addComplex(struct complex c1, struct complex c2)
{
    struct complex tmp;
    tmp.real = c1.real + c2.real;
    tmp.imag = c1.imag + c2.imag;
    return tmp;
}
```

结构成员的引用（续）*

例：给出下面程序的运行结果。

```
#include <stdio.h>
struct {
    int  x;
    char *y;
} *p, A[ ] = {
    1,  “for”,
    2,  “while”,
    3,  “do_while”,
    4,  “switch”
};
```

结果：

2 3 3 2 d d o _

3, for

2, while

3, _while

4, switch

```
main( )
{
    int i;
    p = A;
    printf(“%d”, ++p->x);
    printf(“%d”, ++(p->x));
    printf(“%d”, (p++)->x);
    printf(“%d”, p++->x);
    printf(“%c”, *p->y);
    printf(“%c”, *p->y++);
    printf(“%c”, *(p->y++));
    printf(“%c”, *p++->y);
    for(i=0; i<4; i++)
        printf(“\n%d, %s”, A[i].x, A[i].y);
}
```

结构数组

ID	姓名	单位	住址	工资	出生年月		
					年	月	日
1	张三	北航计算机学院	北航家属楼xxx	3000	1980	5	18
…	…	…	…	…	…	…	…

- 当数组中的每一个元素都是同一结构类型的变量时，则称该数组为结构数组。例如:

```
struct person  {
        int ID;
       char name[32];
       char address[64];
       char department[64];
       double salary;
       struct date birthdate;
};
struct person emplist[100];
```

结构数组通常用来实现表单（如员工表、学生记录表、产品表等）这类数据结构。

问题8：问题分析

- 问题：编写 一个程序，统计输入中C语言每个关键字的出现次数。
- 定义一个结构说明用以表示关键字与其出现次数：

```
struct Key {
    char *keyword;
    int count;
};
```

将关键字有序存放能提高关键字的查找效率。

- 关键字表的组织：使用一个有序的结构数组来存放关键字表及关键字出现次数：

```
struct Key Keytab[ ] = {
    “auto”, 0,
    “break”,0,
    “case”, 0,
    …
    “while”, 0
};
```

问题8：算法设计

- 主要算法：

While（ 仍有新单词读入）

If(在关键字表中查找并找到输入的单词)

相应关键字次数加1;

输出关键字及出现次数;

设函数
char getWord(char word[],int lim)
从标准输入中读入一个长度不超过lim-1的单词，并返回单词类型。
为何不用scanf的%s来读?
如：
while(scanf("%s", word) > 0)…

设函数
struct Key *binary(char *word, struct Key tab[ ], int n)
在关键字表tab中查找单词word是否存在。如果找到，则返回其出现位置。n为关键字表的长度（关键字个数）。

设函数
void printKey(struct Key tab[ ], int n)
输出关键字及出现次数。

顺序查找算法

- 在有序数据集中查找指定元素的最简单方法是顺序查找, 即指定数据依次与数据集中数据比较, 直到找到或查到数据集结束。

折半查找算法（binary search）

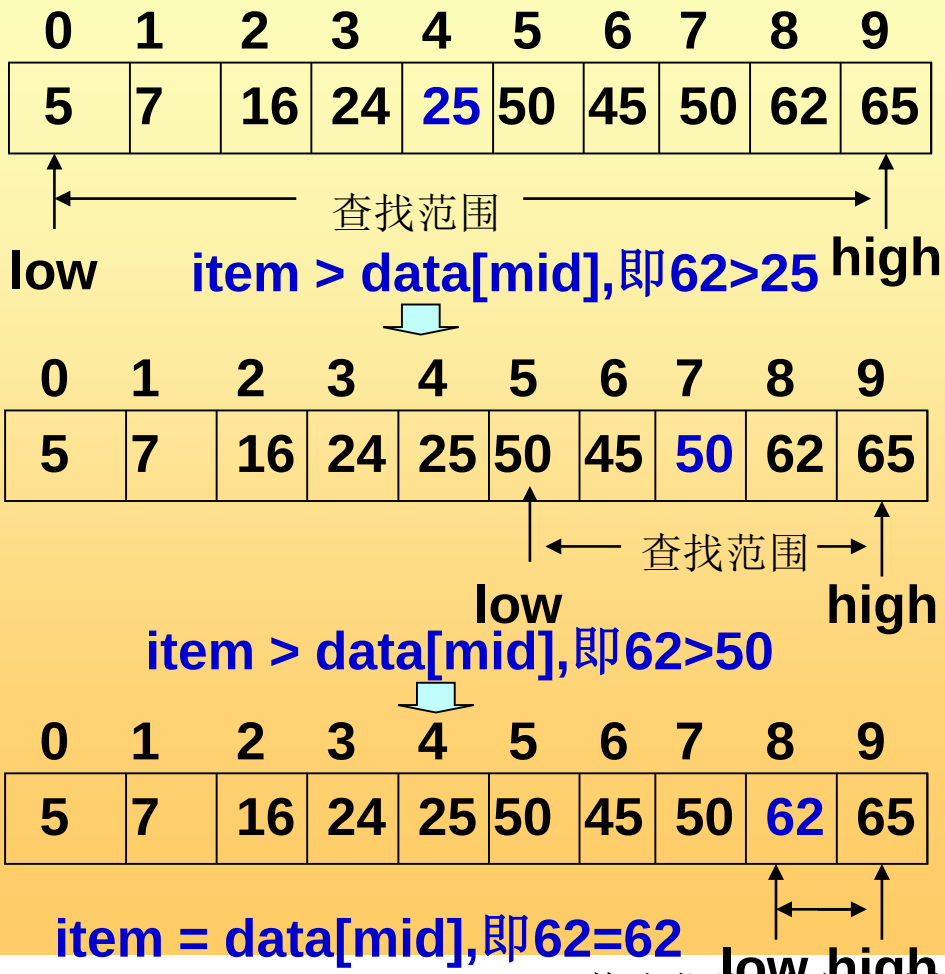
- 在有序数据集中查找指定数据项最常用及最快的算法是折半查找算法。
- 假设数据集按由小到大排列，折半查找算法的核心思想是：

1. 将要查找的有序数据集的中间元素与指定数据项相比较；
2. 如果指定数据项小于该中间元素，则将数据集的前半部分指定为要查找的数据集，然后转步骤1；
3. 如果指定数据项大于该中间元素，则将数据集的后半部分指定为要查找的数据集，然后转步骤1；
4. 如果指定数据项等于中间元素，则查找成功结束。
5. 最后如果数据集中没有元素再可进行查找，则查找失败。

下面以在一个有序整型数据集中查找给定整数为例来说明折半查找。

折半查找算法（续）

例：在下面有序数据集中查找数据项62



item = 62(查找项)
low = 0(查找范围开始)
high = 9(查找范围结束)
mid = (low+high)/2=4(查找范围中间)

low = mid+1=5
high = 9
mid = (low+high)/2=7

low = mid+1=8
high = 9
mid = (low+high)/2=8

折半查找算法（续）

在有序整数数组中查找给定元素的折半查找算法如下:

```
int bsearch(int item, int array[ ], int len)
{
    int low=0, high=len-1, mid;
    while(low <= high){
        mid = (high + low) / 2;
        if(( item < array[mid])
            high = mid – 1;
        else if ( item > array[mid])
            low = mid + 1;
        else
            return (mid);
    }
    return -1;
}
```

问题8：代码实现

```
/*c5_5.c
#include <stdio.h>
struct Key {
      char   *keyword;
      int   keycount;
} Keytab[ ] = {
       "auto",  0,
       "break",  0,
       "case",  0,
      ...
       "while",  0
};
 #define  MAXWORD  20
#define  NKEYS  (sizeof(Keytab) / sizeof(struct  Key))
#define  LETTER  'a'
#define  DIGIT  '0'
struct  Key  *binary(char  *word,  struct Key  tab[ ],  int  n);
char  getword(char  *w,  int lim);
char  type( int c);
void printKey(struct Key  tab[ ],  int  n);
```

关键字表长度（即关键字个数，或数组元素个数）

问题8：代码实现（续）

```
int main( )            /* count  C keyword */
{
    int t;
    char word[MAXWORD];
    struct Key *p;

    while((t = getword(word, MAXWORD)) != EOF)
        if( t == LETTER)
            if(( p = binary(word, Keytab, NKEYS)) != NULL)
                p->keycount++;
    printKey(keytab, NKEYS);
    return 0;
}
```

问题8：代码实现（续）

```
struct Key *binary(char *word, struct Key tab[ ], int n)
{
    int cond;
    struct Key *low = &tab[0];
    struct Key *high = &tab[n-1];
    struct Key *mid;

    while(low <= high){
        mid = low + (high - low) / 2;
        if((cond = strcmp(word, mid->keyword)) < 0)
            high = mid – 1;
        else if ( cond > 0)
            low = mid + 1;
        else
            return (mid);
    }
    return (NULL);
}
```

折半查找

注意：基于指针运算的折半查找算法在计算中间元素位置时与前面的不同。

问题8：代码实现（续）

```
char getword(char *w, int lim)
{
    int c, t;

    if(type(c = *w++ = getchar( )) != LETTER){
        *w = '\0';
        return ( c);
    }
    while(--lim > 0) {
        t = type(c = *w++ = getchar( ));
        if( t != LETTER && t != DIGIT){
          ungetc(c);
          break;
        }
    }
    *(w-1) = '\0';
    return ( LETTER);
}
```

如何从文件中识别出标识符？

```
int  getword（char s[], FILE *fp){
{
   int c,  i = 0;
   while(type(c=fgetc(fp))！ = LETTER)
      if(c == EOF) return c;
      else continue;
   s[i++] = c;
   while((c=fgetc(fp))!=EOF）{
       if(type(c)!=LETTER||type(c)!=DIGTH)
           break;
       s[i++] = c;
   }
   s[i]='\0';
   return 1;
}
```

问题8：代码实现（续）

```
char type(int c)          /* return type of ASCII character */
{
    if( c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z' )
        return ( LETTER );
    else if ( c >= '0' && c <= '9' )
        return ( DIGIT );
    else return (c);
}
void printKey(struct Key  tab[ ],  int  n)
{
    struct Key *p;
    for (p=tab,  p < tab+n;  p++)
        if (p->keycount > 0)
            printf("%4d%s\n", p->keycount, p->keyword);
}
```

问题A*：学生信息排序

【问题描述】
从文件sorelist.txt中读入最多不超过50个学生的学生信息（包括空格隔开的学号、姓名、成绩信息，以学号从低到高排序），分别按姓名和成绩排序输出学生信息到文件sorelist_sort.txt中。
【输入形式】
从文件scorelist.txt中读入最多不超过50个学生的学生信息：
第一行为学生人数；
后面每一行为空格隔开的学生学号、姓名、成绩，其中学号和成绩为整数，姓名为字符串，不超过5位英文字符。
【输出形式】
以姓名顺序（按字典顺序）及成绩顺序（从高到低）将学生信息分别输出到文件scorelist_sort.txt中，中间用一空行分隔。每行输出一位学生的信息，其中学号占3位，姓名（英文）占6位，成绩占5位。

问题A*（续）

【输入样例】文件scorelist.txt中内容为：

```
4
1 Li  86
2 Zhao 90
3 Wang 87
4 Zhang 56
```

【输出样例】 程序运行后，文件scorelist_sort.txt中内容为：

```
1 Li  86
3 Wang  87
4 Zhang  56
2 Zhao  90

2 Zhao  90
3 Wang  87
1 Li  86
4 Zhang  56
```

问题A*：代码实现

```c
/* c5_5.c */
#include <stdio.h>
#include <string.h>
struct Student {
    int no;
    char name[6];
    int score;
};
void sortbyName(struct Student array[], int n);
void sortbyScore(struct Student array[], int n);
void print(FILE *fp, struct Student array[], int n);
int main()
{
    FILE *in, *out;
    struct Student   info[51];
    int i,n;
    if((in=fopen("scorelist.txt","r")) == NULL){
        printf("Cann't Open file scorelist.txt!\n");
        return 1;
    }
    if((out=fopen("scorelist_sort.txt","w")) == NULL){
        printf("Cann't Open file scorelist_sort.txt!\n");
        return 1;
    }
    fscanf(in,"%d",&n);
    for(i=0;i<n;i++)
        fscanf(in, "%d%s%d", &info[i].no, info[i].name,&info[i].score);
    sortbyName(info, n);
    print(out, info, n);
    sortbyScore(info, n);
    print(out,info, n);
    fclose(in);
    fclose(out);
    return 0;
}
```

问题A*：代码实现（续）

```
void sortbyName(struct Student array[], int n)
{
    int i, j;
    struct Student tmp;
    for(i=0; i<n; i++)
        for(j=i; j<n; j++){
            if(strcmp(array[i].name,array[j].name)>0){
                tmp = array[i];
                array[i] = array[j];
                array[j] = tmp;
            }
        }
}
```

```
void sortbyScore(struct Student array[], int n)
{
    int i, j;
    struct Student tmp;
    for(i=0; i<n; i++)
        for(j=i; j<n; j++){
            if(array[i].score < array[j].score){
                tmp = array[i];
                array[i] = array[j];
                array[j] = tmp;
            }
        }
}
```

```
void print(FILE *fp, struct Student array[], int n)
{
    int i;
    for(i=0;i<n;i++)
        fprintf(fp, "%3d%6s%5d\n",array[i].no, array[i].name, array[i].score);
            fprintf(fp, "\n");
}
```

问题A*：思考

- 在按姓名排序时，如果姓名相同，则按成绩排序（由高到低）；在按成绩排序时，如果成绩相同，则按姓名排序（字典序）。上述程序能否实现？若不能实现，如何修改？

自引用结构

- **问题1：谁是幸存者？**

  海盗们曾经玩一种游戏：每当捕获一艘船，船上船员只有一人能幸存。规则如下：船上船员分别编上号，站成一圈；然后从1号船员开始循环数数，每数到13，该船员将被推到海里，直到剩下一个船员。谁是最后的幸存者？

- **问题2：显示文件最后N行。**
- **问题3：实现任意两个的多项式相加。**

由于打印前我们无法知道文件的总行数，如何在程序中组织这类数据？

自引用结构（续）

- （动态）数据的组织与存储方式
  - 顺序组织，如数组
    - 需要连续空间
    - 数据项的插入或删除操作需要移动大量数据
  - 非顺序（动态）组织，如链表，二叉树等
    - 不需要连续空间
    - 数据项的插入或删除操作非常简单

如何构造？

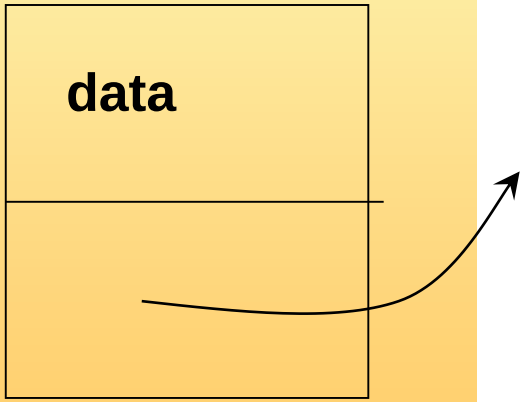
自引用结构

优点：
1）动态管理（增减数据）
2）不需要连续空间

自引用结构（续）

- **自引用结构其成员分为两部分：**
  1. 各种实际数据成员；
  2. 一个或几个指向自身结构的指针；

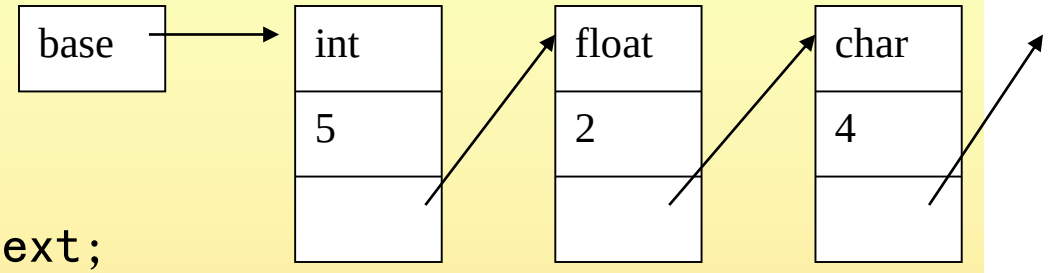
```
struct Type {
    data_member;        // 如 int n;
    struct Type *link;
};
```



自引用结构（续）

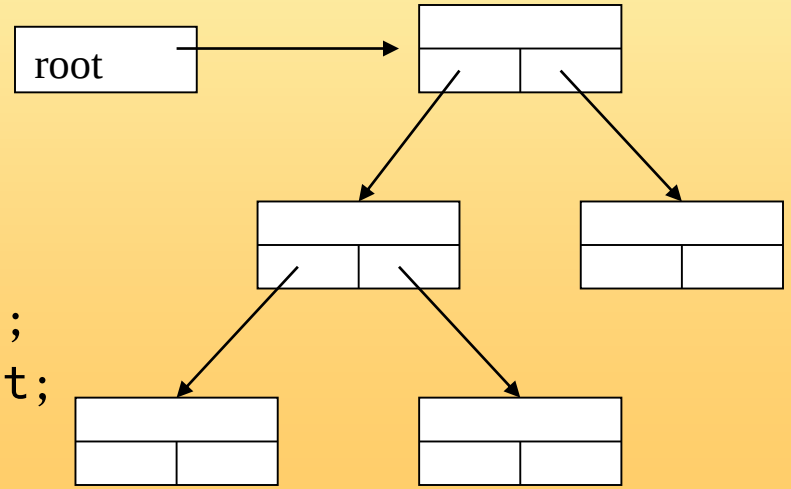
- 链表结构

```
struct  word {
    char *name;
    int count;
    struct word *next;
} *base;
```



- 二叉树结构

```
struct tnode {
    char *word;
    int count;
    struct  tnode *left;
    struct  tnode *right;
} *root;
```



自引用结构（续）

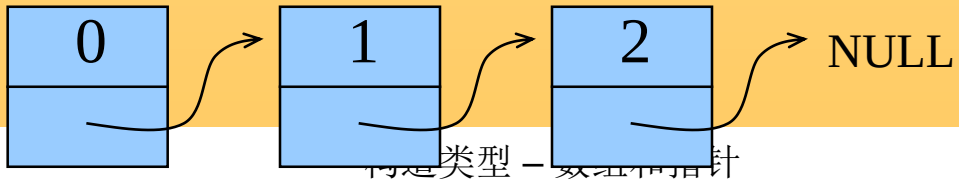
下面通过示例说明如何构造、使用一个链表：

链表结构：

```
struct link {
    int n;
    struct link *next;
};
```

1) 创建链表：

```
struct link *first=NULL, *p,*q;
for(i=0; i< 10; i++){
    q = (struct link *)malloc(sizeof(struct link));
    q->n = i;
    q->next = NULL;
    if(first == NULL)
        first = p = q;
    else {
        p->next = q;
        p = p->next;
    }
}
```

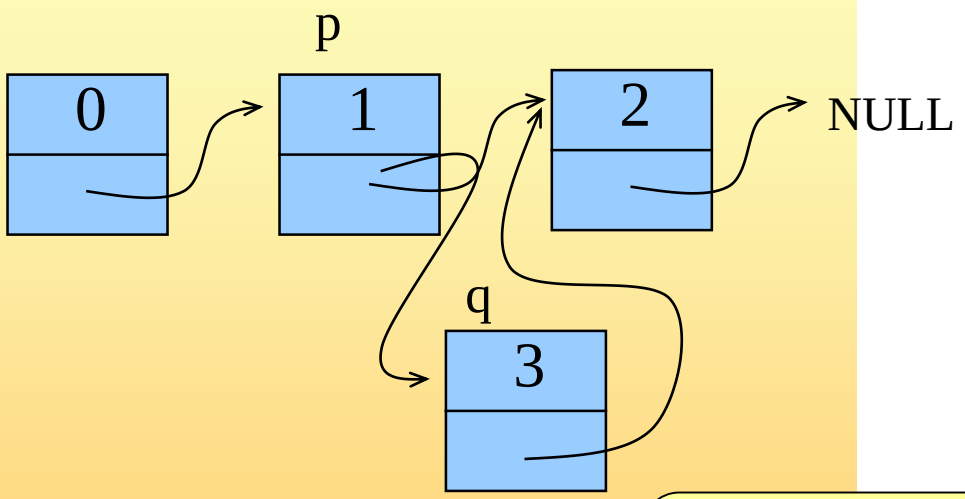


自引用结构（续）

2）插入一个节点：

```
q->next = p->next;
p->next = q;
```

（在p后插入q）

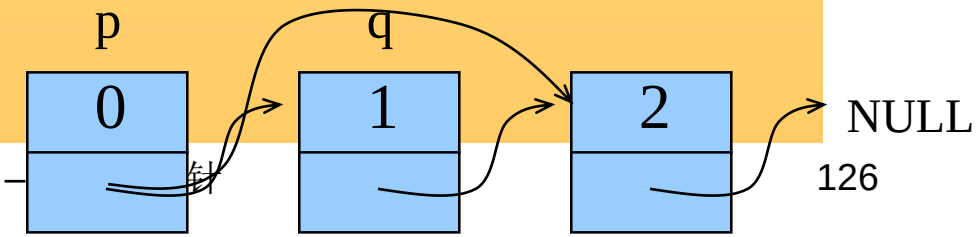


3）删除一个节点：

```
q = p->next;
p->next = p->next->next;
free(q);
```

（删除p的下一节点）

为什么首先要执行：
q = p->next ？



问题9

- **问题：命令tail用来显示一个文件的最后n行。其格式为：**

  tail [-n] filename

  其中：

  -n ：n表示需要显示的行数，省略时n的值为10。

  filename ：给定文件名。

  如，命令tail -20 example.txt 表示显示文件example.txt的最后20行。实现该程序，该程序应具有一定的错误处理能力，如能处理非法命令参数和非法文件名。

问题9：问题分析

若从命令行输入:

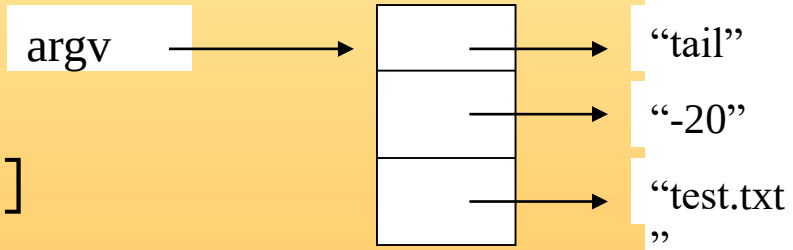
tail -20 test.txt

以显示文件test.txt的最后20行。

- **如何得到需要显示的行数和文件名？**
  - 使用命令行参数

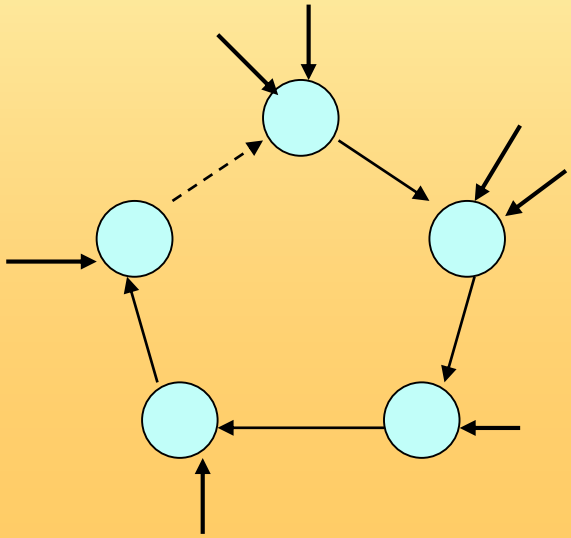
    ```
    int main(int argc, char *argv[])
    ```

  - 行数 `n = atoi(argv[1]+1)`
  - 文件名 `filename = argv[2]`
- **如何得到最后n行？**



问题9：算法设计

- **方法一：使用n个节点的循环链表。链表中始终存放最近读入的n行。**
  1. 首先创建一个空的循环链表；
  2. 然后再依次读入文件的每一行挂在链表上，最后链表上即为最后n行。

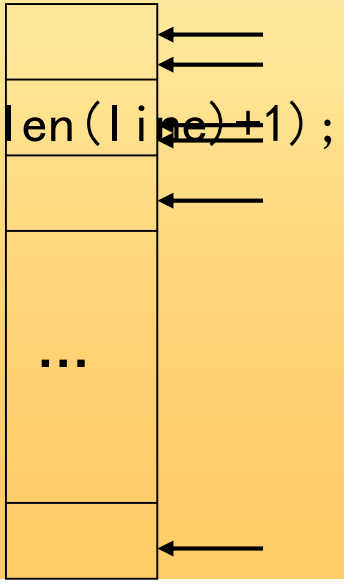


问题9：算法设计（续）

- 方法二：使用一个n个元素的指针数组。
  - 依次读入每一行，然后循环挂到指针数组上。

```
char *lineptr[N]; /*存入所读入的行*/
char line[LEN]; /*当前读入行*/
int i; /*读入的行数*/

lineptr[i % n] = (char *)malloc(strlen(line)+1);
strcpy(lineptr[i%n], line);
```



问题9：算法设计（续）

- **方法三：两次扫描文件。**
  - 第一遍扫描文件，用于统计文件的总行数N；
  - 第二遍扫描文件时，首先跳过前面N-n行，只读取最后n行。
- **如何开始第二遍扫描？**

  fseek(fp, 0, SEEK_SET)； -- 将文件读写位置移至文件头

  或（关闭后）再次打开同一个文件

请同学们考虑是否还有其它方法？甚至更好的方法？

如：我们可设计这样两个函数：

fbgetc(fp)；//从文件尾开始，每次倒读一个字符

fbgets(buf,size,fp)；//从文件尾开始每次倒读一行

问题9：代码实现（循环链表）


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define DEFLINES  10
#define MAXLEN    81
struct Node {
  char *line;
  struct Node *next;
};
```

命令行输入中没有指定打印行数时，获取文件名，此时行数为缺省10。如 tail test.txt

```
int main(int argc, char *argv[ ])
{
    char curline[MAXLEN],*filename;
    int n = DEFLINES, i;
    struct Node *first, *ptr;
    FILE *fp;
    if( argc == 3 && argv[1][0] == '-') {
        n = atoi(argv[1]+1);
        filename = argv[2];
    }
    else if( argc == 2)
        filename = argv[1];
    else {
        printf("Usage: tail [-n] filename\n");
        return (1);
     }
```

命令行输入中指定打印行数时，获取行数及文件名。如tail -20 test.txt

问题9：代码实现

```
if((fp = fopen(filename, "r")) == NULL){
    printf(" Cann't open file: %s !\n", filename);
    return (-1);
  }
  first = ptr = (struct Node *)malloc(sizeof ( struct Node));
  first->line = NULL;
  for(i=1; i<n; i++){
    ptr->next = (struct Node *)malloc(sizeof ( struct Node));
    ptr = ptr->next;
    ptr->line = NULL;
  }
  ptr->next = first;
  ptr = first;
```

创建循环链表

将链表的最后一个节点指向头节点，以构成一个循环链表。

问题9：代码实现

```
while(fgets(curline, MAXLEN, fp) != NULL){
    if(ptr->line != NULL)      /*链表已经满了，需要释放掉不需要的行*/
        free(ptr->line);
    ptr->line = (char *) malloc ( strlen(curline)+1);
    strcpy(ptr->line, curline);
    ptr = ptr->next;
 }
 for(i=0; i<n; i++) {
    if(ptr->line != NULL)
        printf("%s",ptr->line);
    ptr = ptr->next;
 }
 fclose(fp);
 return 0;
}
```

测试考虑点：

准备一个包含内容（如11~20行）的正文文件test.txt

1）tail -5 test.txt　　(正常)

2）tail test.txt　　(正常)

3）tail -30 test.txt　　(非正常)

4）tail -0 test.txt　　(非正常)

5）tail -1 test.txt　　(边界)

动态数组结构（链表、树等）的应用

提示：动态数组结构，如链表、树等特别合适于数组项数不确定的数据的组织。

如上例中多项式的项数就不确定。

忠告

一定要理解和会正确使用链式（自引用结构）数据结构来解决问题！

（数据结构课程中要大量应用它们）

联合（union）

- 联合是一种数据类型，该类型变量可以在不同时间内维持定义它的不同类型和不同长度的对象，也就是说提供单独的变量，以便合理地保存几种类型中的任何一类变量。

定义形式：

union 联合名 {分量表} 联合变量名;

联合（union）（续）

例：

```
union  v_tag {
    int  ival;
    float  fval;
    char  *pval;
} uval;
```

下面是联合的使用：

```
if(utype = = INT)
    printf( “%d\n”, uval.ival);
else if (utype = = FLOAT)
    printf( “%f\n”, uval.fval);
else if(utype = = STRING)
    printf( “%s\n”, uval.pval);
else
    printf( “bad type\n” );
```

[注意]：使用联合，用法必须一致，即取出的类型必须是最近存入的类型。因此，在使用联合时，要记住当前存于联合中的类型是什么，不允许存入是一种类型，而取出是另一种类型。

联合（union）（续）

联合可以出现在结构和数组中，数组和结构也可以出现在联合中，下例是在编译中常见到的符号表：

例：

```
struct {
    char  *name;
    int   flags;
    int   utype;
    union {
      int ival;
      float fval;
      char *pval;
    } uval;
} symtab[NSYM];
```

使用：symtab[i].uval.ival

```
int main()
{
   int n=5;
   char c='a';
   double pi = 3.14159;
   …
}
```

Name	Flags	Type	Value
n		INT	5
pi		DOUBLE	3.14159
c		CHAR	'a'
…	…	…	…

联合（union）（续）

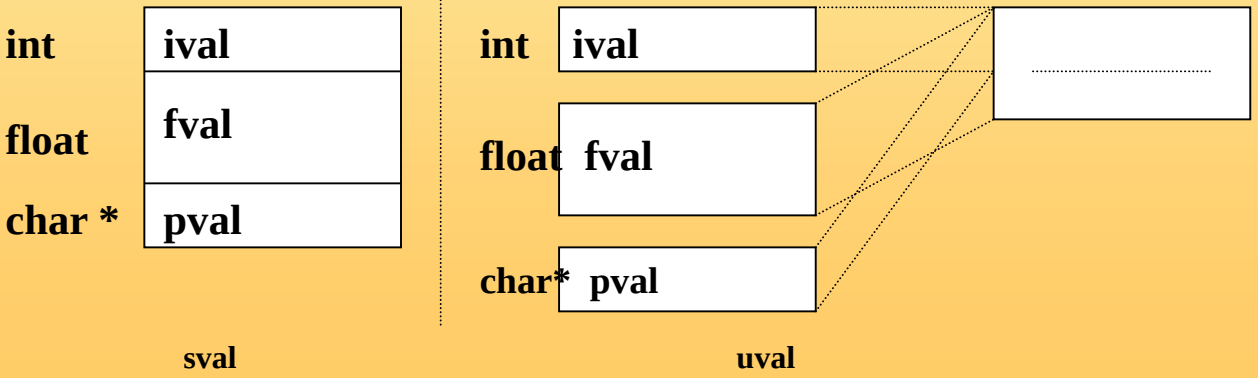
- **结构与联合的异同:**

假设我们定义如下具有相同内容的结构和联合：

```
strcut v_tag {
        int  ival;
        float  fval;
        char  *pval;
} sval;
```

```
union  v_tag {
        int  ival;
        float  fval;
        char  *pval;
} uval;
```

则struct v_tag和union v_tag的物理存贮形式可参见下图：



联合（union）（续）

- **结构与联合的异同：**
  - 在定义或说明形式上，union和struct很类似，但在任何时刻联合只允许联合中说明的某一成员留在联合里。
  - 结构由多个成员（分量）所组成，而联合只有一个成员，只不过该成员的名字和类型可以在规定的几个里选定一个。
  - 因此，联合可以看作是一个特殊的结构，其所有成员在结构中的位移量都是0，**当对联合变量分配存贮空间时，应保证它能容纳最大的一个成员的大小**，而存贮空间的边界应能适于联合中的所有可选成员的类型。（见上图）
  - 对联合的初始化只能对应于它的第一个可选成员。
  - 对联合成员的引用方式完全和结构的情况一样。

本讲结束！