第10章 对文件的输入输出

10.1 C文件的有关基本知识

10.1.1 什么是文件

- 文件有不同的类型，在程序设计中，主要用到两种文件：

  (1) 程序文件。包括源程序文件**(**后缀为**.c)**、目标文件**(**后缀为**.obj)**、可执行文件**(**后缀为**.exe)**等。这种文件的内容是程序代码。

10.1.1 什么是文件

- 文件有不同的类型，在程序设计中，主要用到两种文件：

  (2) 数据文件。文件的内容不是程序，而是供程序运行时读写的数据，如在程序运行过程中输出到磁盘(或其他外部设备)的数据，或在程序运行过程中供读入的数据。如一批学生的成绩数据，或货物交易的数据等。

- 本章主要讨论的是数据文件

10.1.1 什么是文件

- 在以前各章中所处理的数据的输入和输出，从终端的键盘输入数据，运行结果输出到终端显示器上
- 常常需要将一些数据输出到磁盘上保存起来，以后使用
- 这就要用到磁盘文件

10.1.1 什么是文件

- 操作系统把各种设备都统一作为文件处理
- 从操作系统的角度看，每一个与主机相联的输入输出设备都看作是文件。例如，
  - 终端键盘是输入文件
  - 显示屏和打印机是输出文件

10.1.1 什么是文件

- “文件”指存储在外部介质上数据的集合
  - 一批数据是以文件的形式存放在外部介质上的
  - 操作系统是以文件为单位对数据进行管理
  - 想找存放在外部介质上的数据，先按文件名找到所指定的文件，然后再从该文件读数据
  - 要向外部介质上存储数据也必须先建立一个文件（以文件名作为标志），才能向它输出数据

10.1.1 什么是文件

- 输入输出是数据传送的过程，数据如流水一样从一处流向另一处，因此常将输入输出形象地称为流(stream)，即数据流。流表示了信息从源到目的端的流动。

10.1.1 什么是文件

- 输入操作时，数据从文件流向计算机内存
- 输出操作时，数据从计算机流向文件
- 无论是用Word打开或保存文件，还是C程序中的输入输出都是通过操作系统进行的
- “流”是一个传输通道，数据可以从运行环境流入程序中，或从程序流至运行环境

10.1.1 什么是文件

- 从C程序的观点来看，无论程序一次读写一个字符，或一行文字，或一个指定的数据区，作为输入输出的各种文件或设备都是统一以逻辑数据流的方式出现的。C语言把文件看作是一个字符（或字节）的序列。一个输入输出流就是一个字符流或字节(内容为二进制数据)流。

10.1.1 什么是文件

- C的数据文件由一连串的字符（或字节）组成，而不考虑行的界限，两行数据间不会自动加分隔符，对文件的存取是以字符（字节）为单位的。输入输出数据流的开始和结束仅受程序控制而不受物理符号（如回车换行符）控制，这就增加了处理的灵活性。这种文件称为流式文件。

10.1.2 文件名

- 文件要有一个唯一的文件标识，以便用户识别和引用。
- 文件标识包括三部分：
  - (1)文件路径
  - (2)文件名主干
  - (3)文件后缀

10.1.2 文件名

文件路径表示文件在外存储设备上的位置。

文件路径　文件名主干　文件后缀

D: \CC\temp\file1.dat

- 表示file1.dat文件存放在D盘中的CC目录下的temp子目录下面

10.1.2 文件名

- 文件路径表示文件在外部存储设备中的位置。如：

文件名

D: \CC\temp\file1.dat

  - 表示file1.dat文件存放在D盘中的CC目录下的temp子目录下面

10.1.2 文件名

命名规则遵循标识符的命名规则

- 文件路径表示[illegible]备中的位置。如：

  D: \CC\temp\file1.dat

  - 表示**file1.dat**文件存放在**D**盘中的**CC**目录下的**temp**子目录下面

一般不超过3个字母（doc、txt、dat、c、cpp、obj、exe、ppt、bmp等）

- 文 的位置。如：

  D: \CC\temp\file1.dat

  - 表示**file1.dat**文件存放在**D**盘中的**CC**目录下的**temp**子目录下面

10.1.3 文件的分类

- 根据数据的组织形式，数据文件可分为**ASCII**文件和二进制文件。
  - 数据在内存中是以二进制形式存储的，如果不加转换地输出到外存，就是二进制文件
  - 如果要求在外存上以**ASCII**代码形式存储，则需要在存储前进行转换
  - **ASCII**文件又称文本文件，每一个字节放一个字符的**ASCII**代码

10.1.3 文件的分类

- 字符一律以ASCII形式存储
- 数值型数据既可以用ASCII形式存储，也可以用二进制形式存储
  - 如有整数10000，如果用ASCII码形式输出到磁盘，则在磁盘中占 5 个字节(每一个字符占一个字节)，而用二进制形式输出，则在磁盘上只占4个字节(用VC++ C时)

10.1.3 文件的分类

ASCII形式

00110001	00110000	00110000	00110000	00110000
（1）	（0）	（0）	（0）	（0）

二进制形式

00000000	00000000	00100111	00010000

（10000）

10.1.4 文件缓冲区

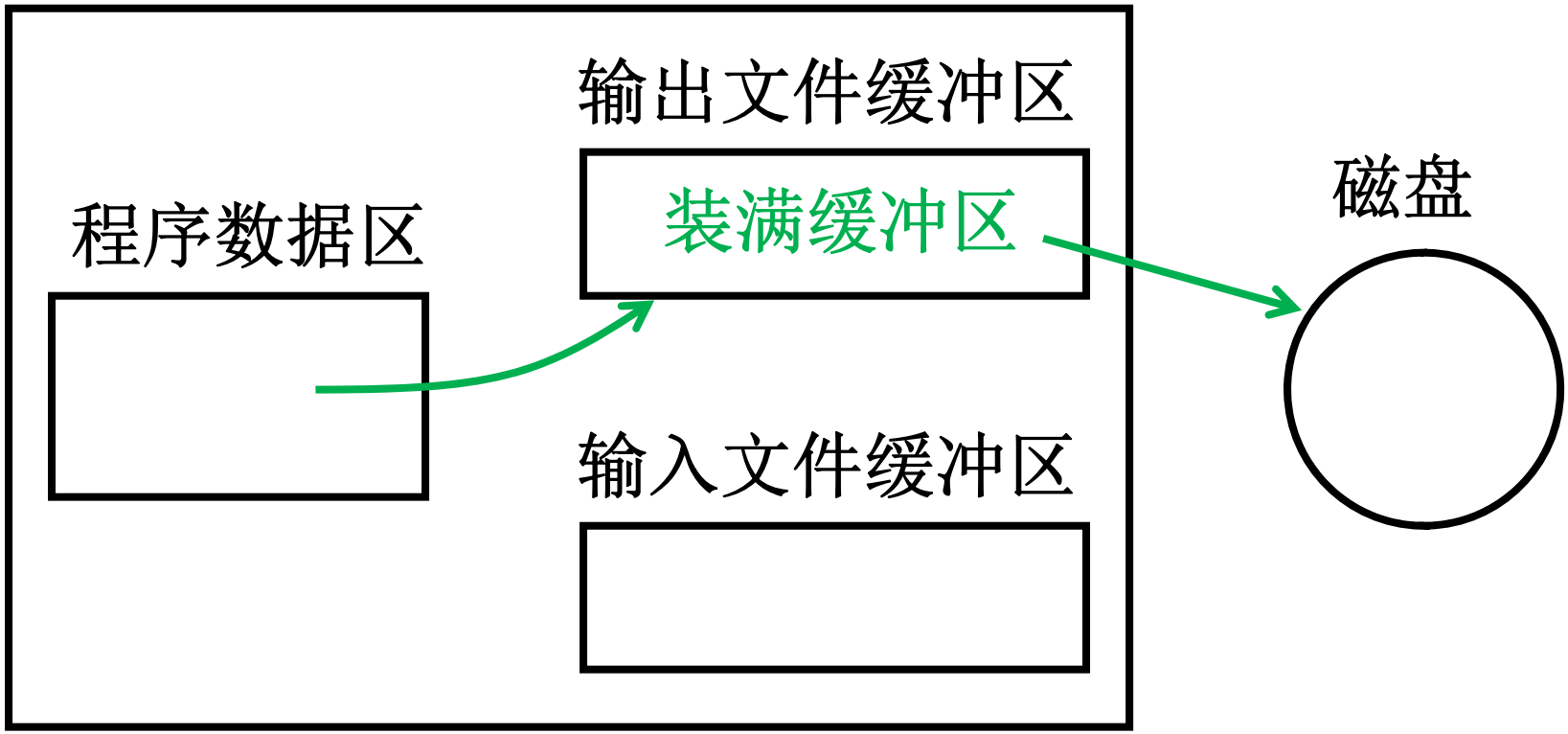
- ANSI C标准采用“缓冲文件系统”处理数据文件
- 所谓缓冲文件系统是指系统自动地在内存区为程序中每一个正在使用的文件开辟一个文件缓冲区

10.1.4 文件缓冲区

- 从内存向磁盘输出数据必须先送到内存中的缓冲区，装满缓冲区后才一起送到磁盘去
- 如果从磁盘向计算机读入数据，则一次从磁盘文件将一批数据输入到内存缓冲区（充满缓冲区），然后再从缓冲区逐个地将数据送到程序数据区（给程序变量）

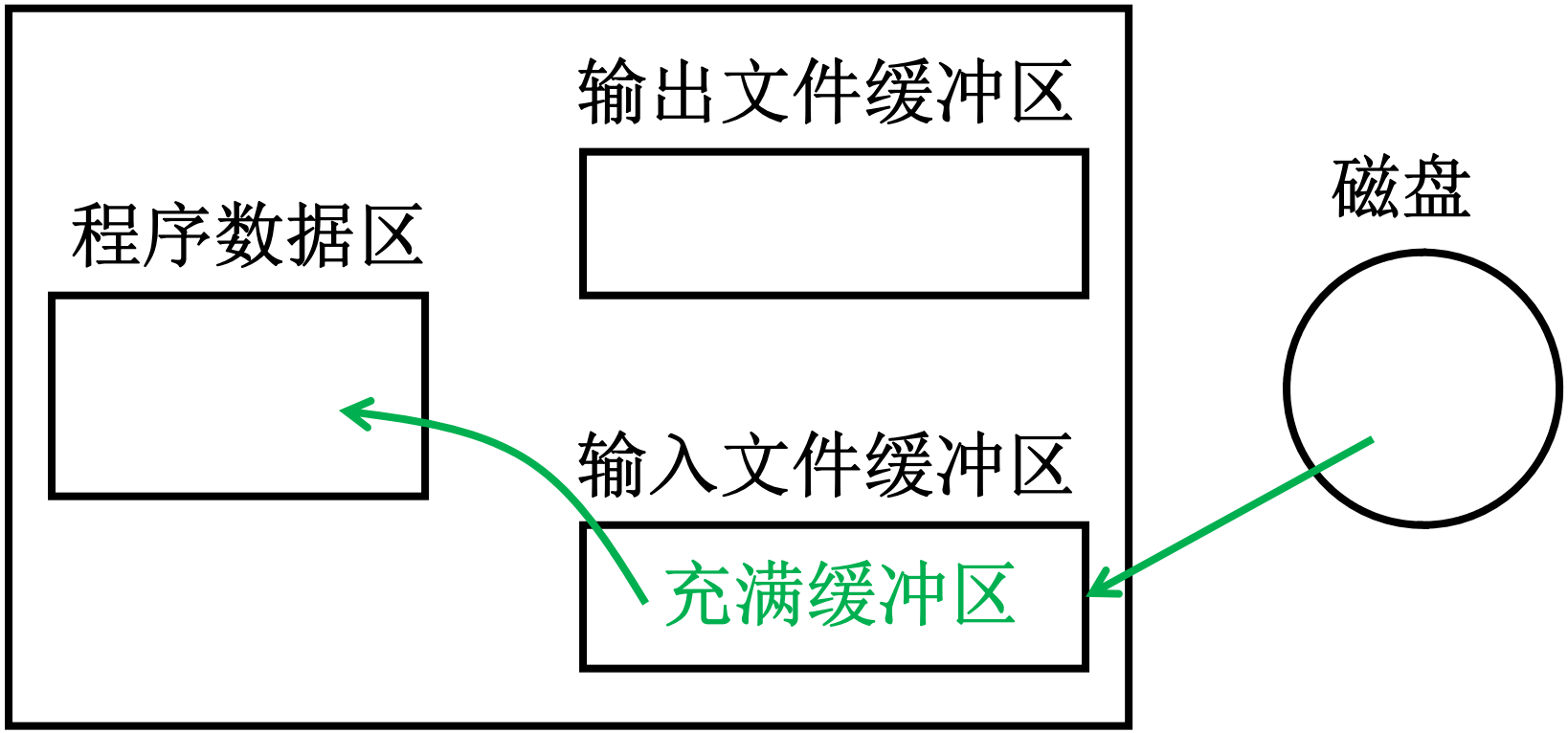
10.1.4 文件缓冲区

- 从内存向磁盘输出数据



10.1.4 文件缓冲区

- 从磁盘向计算机读入数据



10.1.5 文件类型指针

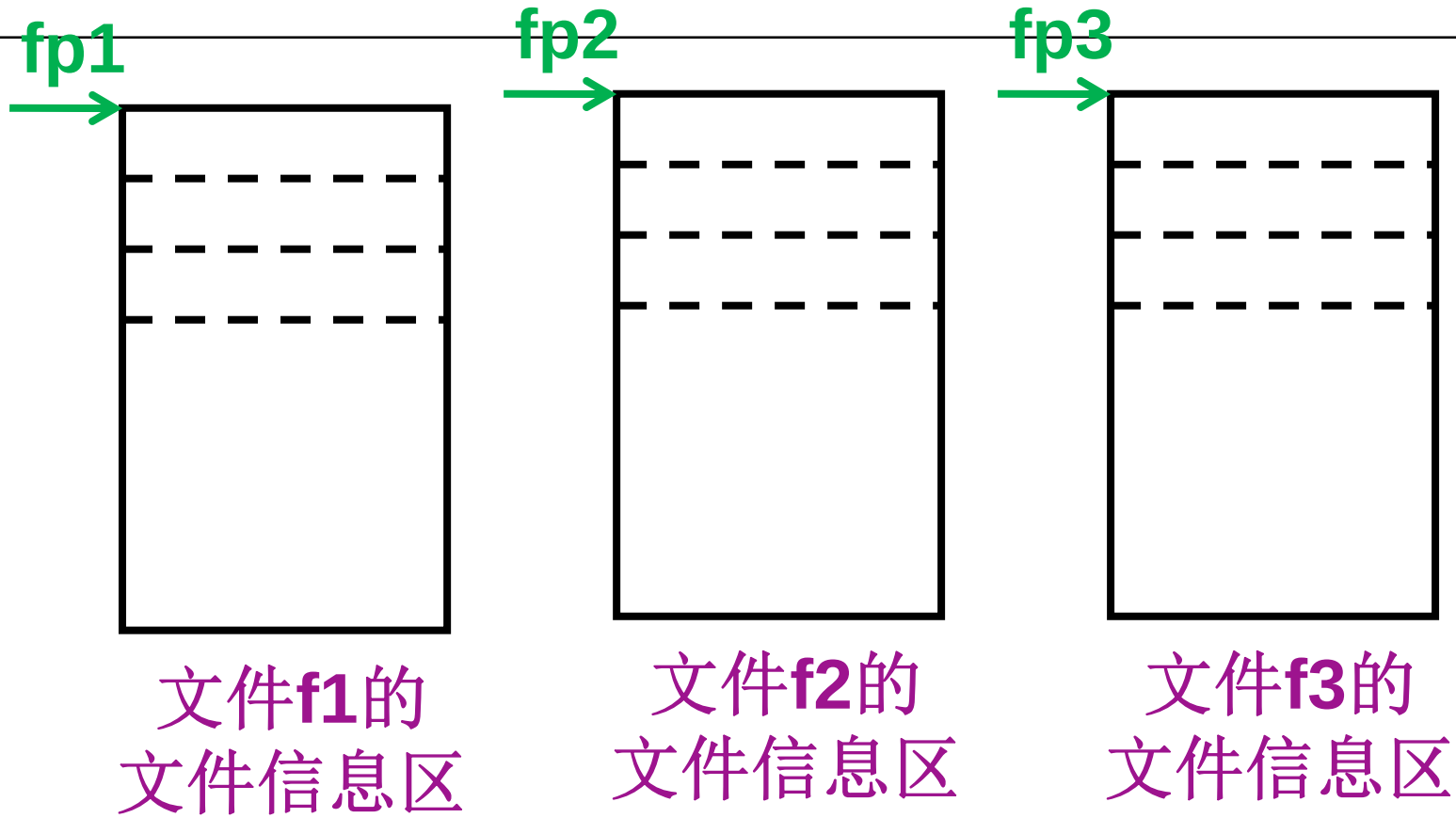
- 缓冲文件系统中，关键的概念是“文件类型指针”，简称“文件指针”
  - 每个被使用的文件都在内存中开辟一个相应的文件信息区，用来存放文件的有关信息（如文件的名字、文件状态及文件当前位置等）
  - 这些信息是保存在一个结构体变量中的。该结构体类型是由系统声明的，取名为**FILE**

10.1.5 文件类型指针

- 声明FILE结构体类型的信息包含在头文件“stdio.h”中
- 一般设置一个指向FILE类型变量的指针变量，然后通过它来引用这些FILE类型变量

10.1.5 文件类型指针

```
FILE   *fp1,*fp2,*fp3;
```



10.2 打开与关闭文件

10.2.1 用fopen函数打开数据文件

- 对文件读写之前应该“打开”该文件，在使用结束之后应“关闭”该文件。
- 所谓“打开”是指为文件建立相应的信息区(用来存放有关文件的信息)和文件缓冲区(用来暂时存放输入输出的数据)。

10.2.1 用fopen函数打开数据文件

- 在编写程序时，在打开文件的同时，一般都指定一个指针变量指向该文件，也就是建立起指针变量与文件之间的联系，这样就可以通过该指针变量对文件进行读写
- 所谓“关闭”是指撤销文件信息区和文件缓冲区

10.2.1 用fopen函数打开数据文件

- fopen函数的调用方式为：
  fopen(文件名,使用文件方式);
- 例如：
  fopen(“a1”,”r”);
  - 表示要打开名为“a1”的文件，使用文件方式为“读入”
  - fopen函数的返回值是指向a1文件的指针

10.2.1 用fopen函数打开数据文件

- 通常将fopen函数的返回值赋给一个指向文件的指针变量。如：

```
FILE *fp;
fp=fopen( “a1” ,” r” );
```

  - fp和文件a1相联系，fp指向了a1文件

10.2.1 用fopen函数打开数据文件

- 在打开一个文件时，通知编译系统以下3个信息：
  - ①需要访问的文件的名字
  - ②使用文件的方式（“读”还是“写”等）
  - ③让哪一个指针变量指向被打开的文件
- 使用文件方式参见教材表10.1。

- 说明：

  (1) 用“**r**”方式打开的文件只能用于向计算机输入而不能用作向该文件输出数据，而且该文件应该已经存在，并存有数据，这样程序才能从文件中读数据。

  - 不能用“**r**”方式打开一个并不存在的文件，否则出错。

- 说明：

(2) 用“w”方式打开的文件只能用于向该文件写数据（即输出文件），而不能用来向计算机输入。

- 如果原来不存在该文件，则在打开文件前新建立一个以指定的名字命名的文件。
- 如果原来已存在一个以该文件名命名的文件，则在打开文件前先将该文件删去，然后重新建立一个新文件。

- 说明：

  (3) 如果希望向文件末尾添加新的数据（不希望删除原有数据），则应该用“**a**”方式打开

  - 但此时应保证该文件已存在；否则将得到出错信息。
  - 打开文件时，文件读写标记移到文件末尾

- 说明：

(4) 用**r+**、**w+**、**a+**方式打开的文件既可以用来输入数据，也可以用来输出数据。

- 用**r+**方式时该文件应该已经存在。
- 用**w+**方式则新建立一个文件，先向此文件写数据，然后可以读此文件中的数据。
- 用**a+**方式打开的文件，原来的文件不被删去，文件读写位置标记移到文件末尾，可以添加，也可以读。

- 说明：

  (5) 如果打开失败，**fopen**函数将会带回一个出错信息。**fopen**函数将带回一个空指针值**NULL**

- 常用下面的方法打开一个文件：

```
if ((fp=fopen(“file1”,’r”))==NULL)
 {printf(“cannot open this file\n”);
  exit(0);
 }
```

终止正在执行的程序

- 说明：

  (6) C标准建议用表10.1列出的文件使用方式打开文本文件或二进制文件，但目前使用的有些C编译系统可能不完全提供所有这些功能

- 说明：

(7) 计算机输从**ASCII**文件读入字符时，遇到回车换行符，系统把它转换为一个换行符，在输出时把换行符转换成为回车和换行两个字符。在用二进制文件时，不进行这种转换，在内存中的数据形式与输出到外部文件中的数据形式完全一致，一一对应。

- 说明：

  (8) 程序中可以使用**3**个标准的流文件：标准输入流、标准输出流、标准出错输出流。

  - 系统已对这**3**个文件指定了与终端的对应关系
  - 标准输入流是从终端的输入
  - 标准输出流是向终端的输出
  - 标准出错输出流是当程序出错时将出错信息发送到终端

- 程序开始运行时系统自动打开这3个标准流文件。因此，程序编写者不需要在程序中用**fopen**函数打开它们。所以以前我们用到的从终端输入或输出到终端都不需要打开终端文件。

10.2.2 用fclose函数关闭数据文件

- 关闭文件用**fclose**函数。**fclose**函数调用的一般形式为囍

  fclose(文件指针**);** 囍

  例如：

  fclose（fp);

- 如果不关闭文件将会丢失数据。

10.3 顺序读写数据文件

- 在顺序写时，先写入的数据存放在文件中前面，后写入的数据存放在文件中后面
- 在顺序读时，先读文件中前面的数据，后读文件中后面的数据
- 对顺序读写来说，对文件读写数据的顺序和数据在文件中的物理顺序是一致的
- 顺序读写需要用库函数实现

10.3 顺序读写数据文件

10.3.1 怎样向文件读写字符

- 读写一个字符的函数

函数名	调用形式	功能	返回值
fgetc	fgetc(fp)	从fp指向的文件读入一个字符	读成功，带回所读的字符，失败则返回文件结束标志EOF(即-1)
fputc	fputc(ch,fp)	把字符ch写到文件指针变量fp所指向的文件中	写成功，返回值就是输出的字符；输出失败，则返回EOF（即-1）

例10.1 从键盘输入一些字符，逐个把它们送到磁盘上去，直到用户输入一个“#”为止。

- 解题思路：用**fgetc**函数从键盘逐个输入字符，然后用**fputc**函数写到磁盘文件即可。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ FILE *fp;
   char ch,filename[10];
   printf("请输入所用的文件名: ");
   scanf("%s",filename);
   if((fp=fopen(filename, "w" ))==NULL)
   {  printf("无法打开此文件\n");
      exit(0);
   }
   ch=getchar( );
```

用exit函数时加

输入文件名

只写

接收最后输入的回车符

```
    printf(“请输入一个字符串(以#结束)：");
    ch=getchar( );
    while(ch!= ‘#’ )
    {  fputc(ch,fp);
       putchar(ch);
       ch=getchar();
     }
    fclose(fp);
    putchar(10);
    return 0;
}
```

例10.2 将一个磁盘文件中的信息复制到另一个磁盘文件中。 今要求将上例建立的file1.dat文件中的内容复制到另一个磁盘文件file2.dat中。

- 解题思路：处理此问题的算法是：从file1.dat文件中逐个读入字符，然后逐个输出到file2.dat中。

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{ FILE *in,*out;
   char  ch,infile[10],outfile[10];
   printf("输入读入文件的名字:");
   scanf("%s",infile);
   printf("输入输出文件的名字:");
   scanf( “%s” ,outfile);
   if((in=fopen(infile, “r” ))==NULL)
   {p                                  件\n"); exit(0);}
   if                                  “w” ))==NULL)
   {printf(                            件\n"); exit(0); }
```

改为rb和wb，则复制一个二进制文件

```
while(!feof(in))
{   ch=fgetc(in);
    fputc(ch,out);
    putchar(ch);
}
putchar(10);
fclose(in);
fclose(out);
return 0;
}
```

检查当前读写位置
是否移到文件末尾

10.3.2 怎样向文件读写一个字符串

- 读写一个字符串的函数

函数名	调用形式	功能	返回值
fgets	fgets(str,n,fp)	从fp指向的文件读入长度为(n-1)的字符串，存放到字符数组str中	读成功，返回地址str，失败则返回NULL)
fputs	fputs(str,fp)	str所指向的字符串写到文件指针变量fp所指向的文件中	写成功，返回0；否则返回非0值

- 说明：

fgets函数的函数原型为：

char *fgets (char *str,int n,FILE *fp);

- 其作用是从文件读入一个字符串
- 调用时可以写成：

fgets(str,n,fp);

- 说明：
  - **fgets(str,n,fp);**中**n**是要求得到的字符个数，但实际上只读**n-1**个字符，然后在最后加一个’**\0**’字符，这样得到的字符串共有**n**个字符，把它们放到字符数组**str**中
  - 如果在读完**n-1**个字符之前遇到换行符“**\n**”或文件结束符**EOF**，读入即结束，但将所遇到的换行符“**\n**”也作为一个字符读入
  - 执行**fgets**成功，返回**str**数组首地址，如果一开始就遇到文件尾或读数据错，返回**NULL**

- 说明：

fputs函数的函数原型为：

int fputs (char *str, FILE *fp);

- **str**指向的字符串输出到**fp**所指向的文件中
- 调用时可以写成： **fputs("China" ,fp);**
- **fputs**函数中第一个参数可以是字符串常量、字符数组名或字符型指针
- 字符串末尾的**'\0'**不输出
- 输出成功，函数值为 0 ；失败，函数值为**EOF**

例10.3 从键盘读入若干个字符串，对它们按字母大小的顺序排序，然后把排好序的字符串送到磁盘文件中保存。

- 解题思路：为解决问题，可分为三个步骤：
  - 从键盘读入n个字符串，存放在一个二维字符数组中，每一个一维数组存放一个字符串；
  - 对字符数组中的n个字符串按字母顺序排序，排好序的字符串仍存放在字符数组中；
  - 将字符数组中的字符串顺序输出。


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{ FILE *fp;
   char  str[3][10],temp[10];
   int i,j,k,n=3;
   printf( "Enter strings:\n" );
   for(i=0;i<n;i++)
     gets(str[i]);
```

```
for(i=0;i<n-1;i++)
{ k=i;
   for(j=i+1;j<n;j++)
   if(strcmp(str[k],str[j])>0) k=j;
   if(k!=i)
   { strcpy(temp,str[i]);
      strcpy(str[i],str[k]);
      strcpy(str[k],temp);}
   }
```

```
if((fp=fopen( “D:\\CC\\string.dat”,
                “w” ))==NULL)
{printf("can't open file!\n"); exit(0);}
printf("\nThe new sequence:\n");
for(i=0;i<n;i++)
{ fputs(str[i],fp);
   fputs( “\n” ,fp);
   printf( “%s\n” ,str[i]);
}
return 0;
}
```

人为地输出一个’\n’

- 思考：
  - 从文件**string.dat**中读回字符串，并在屏幕上显示，应如何编写程序？

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ FILE *fp;  char  str[3][10];  int i=0;
   if((fp=fopen( “D:\\CC\\string.dat” ,
                                “r” ))==NULL)
   {printf("can't open file!\n");exit(0);}
   while(fgets(str[i],10,fp)!=NULL)
   { printf("%s",str[i]);   i++;   }
  fclose (fp);
  return 0;
}
```

不用人为地输出’\n’

10.3.3用格式化的方式读写文件

- 一般调用方式为：
  fprintf(文件指针,格式字符串,输出表列);
  fscanf (文件指针,格式字符串,输入表列);
  如：
  fprintf (fp,” %d,%6.2f” ,i,f);
  fscanf (fp,” %d,%f” ,&i,&f);

10.3.4 用二进制方式向文件读写一组数据

- 一般调用形式为:

  fread(buffer，size，count，fp);

  fwrite(buffer，size，count，fp);

10.3.4 用二进制方式向文件读写一组数据

- **buffer**：是一个地址
  - 对**fread**来说，它是用来存放从文件读入的数据的存储区的地址
  - 对**fwrite**来说，是要把此地址开始的存储区中的数据向文件输出
- **size**：要读写的字节数
- **count**：要读写多少个数据项
- **fp**：**FILE**类型指针

例10.4 从键盘输入10个学生的有关数据，然后把它们转存到磁盘文件上去。

- 解题思路：
  - 定义有10个元素的结构体数组，用来存放10个学生的数据
  - 从main函数输入10个学生的数据
  - 用save函数实现向磁盘输出学生数据
  - 用fwrite函数一次输出一个学生的数据

```
#include <stdio.h>
#define SIZE 10
struct Student_type
{ char name[10];
   int num;
   int age;
   char addr[15];
 }stud[SIZE];
```

```
void save( )
{ FILE *fp;   int i;
   if((fp=fopen("stu.dat","wb"))==NULL)
   { printf("cannot open file\n");
      return;
   }
   for(i=0;i<SIZE;i++)
      if(fwrite(&stud[i],
                sizeof(struct Student_type),
                     1,fp)!=1)
         printf("file write error\n");
   fclose(fp);
}
```

当前路径下的文件

10+4+4+15=33，实际上开辟36字节，是4的倍数

```
int main()
{ int i;
   printf( “enter data of students:\n");
   for(i=0;i<SIZE;i++)
     scanf("%s%d%d%s",
                 stud[i].name,&stud[i].num,
                  &stud[i].age,stud[i].addr);
   save( );
   return 0;
}
```

- 为了验证在磁盘文件“**stu.dat**”中是否已存在此数据，可以用以下程序从“**stu.dat**”文件中读入数据，然后在屏幕上输出。


```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
struct Student_type
{ char name[10];
   int num;
   int age;
   char addr[15];
}stud[SIZE];
```

```
int main( )
{int i;   FILE *fp;
  if((fp=fopen("stu.dat","rb"))==NULL)
  {printf("cannot open file\n"); exit(0); }
  for(i=0;i<SIZE;i++)
  {fread (&stud[i],sizeof(struct
                          Student_type),1,fp);
    printf ( "%-10s %4d %4d  %-15s\n" ,
              stud[i].name,stud[i].num,
                  stud[i]. age,stud[i].addr);
   }
   fclose (fp);
   return 0;
}
```

- 如果修改例10.4：从已有的二进制文件“stu.list”中，读入数据并输出到“stu.dat”文件中，应如何修改程序？
- 解题思路：
  - 编写load函数
  - main函数中再调用load函数

```
void load( )
{ FILE *fp;   int i;
   if((fp=fopen("stu_list","rb"))==NULL)
   {printf("cannot open infile\n"); return;}
   for(i=0;i<SIZE;i++)
      if(fread(&stud[i],sizeof(struct
                          student_type),1,fp)!=1)
      { if(feof(fp))
           {   fclose(fp);   return;     }
        printf("file read error\n");
      }
   fclose (fp);
}
```


```
int main()瀆
{ 瀆 load();瀆
        save(); 瀆
        return 0;
}
```

10.4 随机读写数据文件

- 对文件进行顺序读写比较容易理解，也容易操作，但有时效率不高
- 随机访问不是按数据在文件中的物理位置次序进行读写，而是可以对任何位置上的数据进行访问，显然这种方法比顺序访问效率高得多囍

10.4 随机读写数据文件

10.4.1 文件位置标记及其定位

10.4.2 随机读写

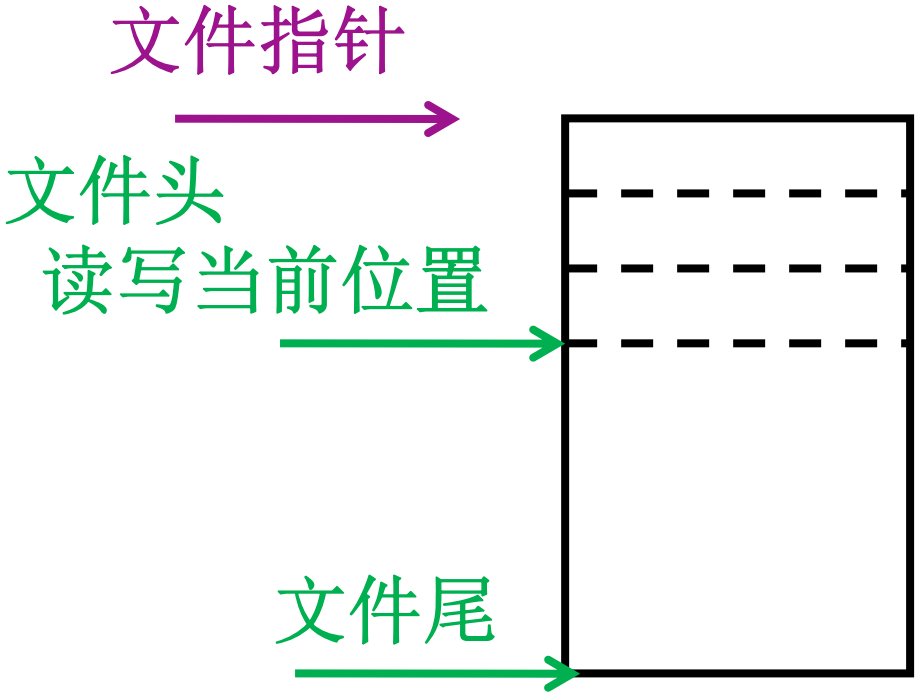
10.4.1 文件位置标记及其定位

1.文件位置标记

- 为了对读写进行控制，系统为每个文件设置了一个文件读写位置标记(简称文件标记)，用来指示“接下来要读写的下一个字符的位置”

10.4.1 文件位置标记及其定位

1.文件位置标记



10.4.1 文件位置标记及其定位

1.文件位置标记

- 一般情况下，在对字符文件进行顺序读写时，文件标记指向文件开头，进行读的操作时，就读第一个字符，然后文件标记向后移一个位置，在下一次读操作时，就将位置标记指向的第二个字符读入。依此类推，直到遇文件尾，结束

10.4.1 文件位置标记及其定位

1.文件位置标记

- 如果是顺序写文件，则每写完一个数据后，文件标记顺序向后移一个位置，然后在下一次执行写操作时把数据写入指针所指的位置。直到把全部数据写完，此时文件位置标记在最后一个数据之后

10.4.1 文件位置标记及其定位

1.文件位置标记

- 可以根据读写的需要，人为地移动了文件标记的位置。文件标记可以向前移、向后移，移到文件头或文件尾，然后对该位置进行读写——随机读写
- 随机读写可以在任何位置写入数据，在任何位置读取数据

10.4.1 文件位置标记及其定位

2. 文件位置标记的定位

- 可以强制使文件位置标记指向指定的位置
- 可以用以下函数实现：囍

(1)用rewind函数使文件标记指向文件开头囍

rewind函数的作用是使文件标记重新返回文件的开头，此函数没有返回值。

例10.5 有一个磁盘文件，内有一些信息。要求第一次将它的内容显示在屏幕上，第二次把它复制到另一文件上。

明德求新 尚用笃行

School of Software

- 解题思路：
  - 因为在第一次读入完文件内容后，文件标记已指到文件的末尾，如果再接着读数据，就遇到文件结束标志，**feof**函数的值等于**1**(真)，无法再读数据
  - 必须在程序中用**rewind**函数使位置指针返回文件的开头

```
#include<stdio.h>
int main()
{ FILE *fp1,*fp2;
   fp1=fopen( “file1.dat” , “r” );
   fp2=fopen( “file2.dat” , “w” );
   while(!feof(fp1))
        putchar(getc(fp1));
   putchar(10);
   rewind(fp1);
   while(!feof(fp1))
          putc(getc(fp1),fp2);
   fclose(fp1);   fclose(fp2);
   return 0;
}
```

10.4.1 文件位置标记及其定位

2. 文件位置标记的定位

- 可以强制使文件标记指向指定的位置
- 可以用以下函数实现：囍

(2) 用fseek函数改变文件标记

fseek函数的调用形式为：囍

fseek(文件类型指针,位移量,起始点) 囍

- 起始点0代表“文件开始位置”，1为“当前位置”，2为“文件末尾位置”

- C 标准指定的名字

起始点	名 字	用数字代表
文件开始位置	SEEK_SET	0
文件当前位置	SEEK_CUR	1
文件末尾位置	SEEK_END	2

- 位移量指以起始点为基点，向前移动的字节数。位移量应是**long**型数据(在数字的末尾加一个字母**L**)。
- **fseek**函数一般用于二进制文件。下面是**fseek**函数调用的几个例子：
  - **fseek (fp,100L,0);**
  - **fseek (fp,50L,1);**
  - **fseek (fp,-10L,2);**

10.4.1 文件位置标记及其定位

2. 文件位置标记的定位

- 可以强制使文件位置标记指向指定的位置
- 可以用以下函数实现：囍

(3) 用ftell函数测定文件位置标记的当前位置

ftell函数的作用是得到流式文件中文件位置标记的当前位置。

- 由于文件中的文件位置标记经常移动，人们往往不容易知道其当前位置，所以常用**ftell**函数得到当前位置，用相对于文件开头的位移量来表示。如果调用函数时出错（如不存在**fp**指向的文件），**ftell**函数返回值为**-1L**。例如：

```
i=ftell(fp);灌
if(i==-1L) printf( “error\n” );
```

10.4.2 随机读写

例10.6 在磁盘文件上存有10个学生的数据。要求将第1,3,5,7,9个学生数据输入计算机，并在屏幕上显示出来。

- 要求：从例10.4中建立的“stu.dat”中读入数据

10.4.2 随机读写

- 解题思路：
  - 按二进制只读方式打开文件
  - 将文件位置标记指向文件的开头，读入一个学生的信息，并把它显示在屏幕上
  - 再将文件标记指向文件中第3，5，7，9个学生的数据区的开头，读入相应学生的信息，并把它显示在屏幕上
  - 关闭文件


```
#include<stdio.h>
#include <stdlib.h>
struct St
{ char name[10];
   int num;
   int age;
   char addr[15];
}stud[10];
```

```
int main()
{ int i;  FILE *fp;
   if((fp=fopen( “stu.dat”, “rb” ))==NULL)
   { printf("can not open file\n"); exit(0); }
    for(i=0;i<10;i+=2)
    { fseek(fp,i*sizeof(struct St),0);
       fread(&stud[i], sizeof(struct St),1,fp);
       printf( “%-10s %4d %4d %-15s\n”,
                        stud[i].name,stud[i].num,
                        stud[i].age,stud[i].addr);
   }
   fclose(fp);  return 0;
}
```

10.5 文件读写的出错检测

1.ferror函数

- ferror函数的一般调用形式为囍

  ferror(fp); 囍

  - 如果返回值为0，表示未出错，否则表示出错
  - 每次调用输入输出函数，都产生新的ferror函数值，因此调用输入输出函数后立即检查
  - 调用fopen时，ferror的初始值自动置为0

10.5 文件读写的出错检测

2. clearerr函数

- 作用是使文件错误标志和文件结束标志置为0
- 调用一个输入输出函数时出现错误（ferror值为非零值），立即调用clearerr(fp)，使ferror(fp)值变0，以便再进行下一次检测
- 只要出现文件读写错误标志，它就一直保留，直到对同一文件调用clearerr函数或rewind函数，或任何其他一个输入输出函数

第1章 程序设计和C语言

1.1 什么是计算机程序

- 程序：一组计算机能识别和执行的指令
- 只要让计算机执行这个程序，计算机就会自动地、有条不紊地进行工作
- 计算机的一切操作都是由程序控制的，离开程序，计算机将一事无成

1.2 什么是计算机语言

- 计算机语言：人和计算机交流信息的、计算机和人都能识别的语言

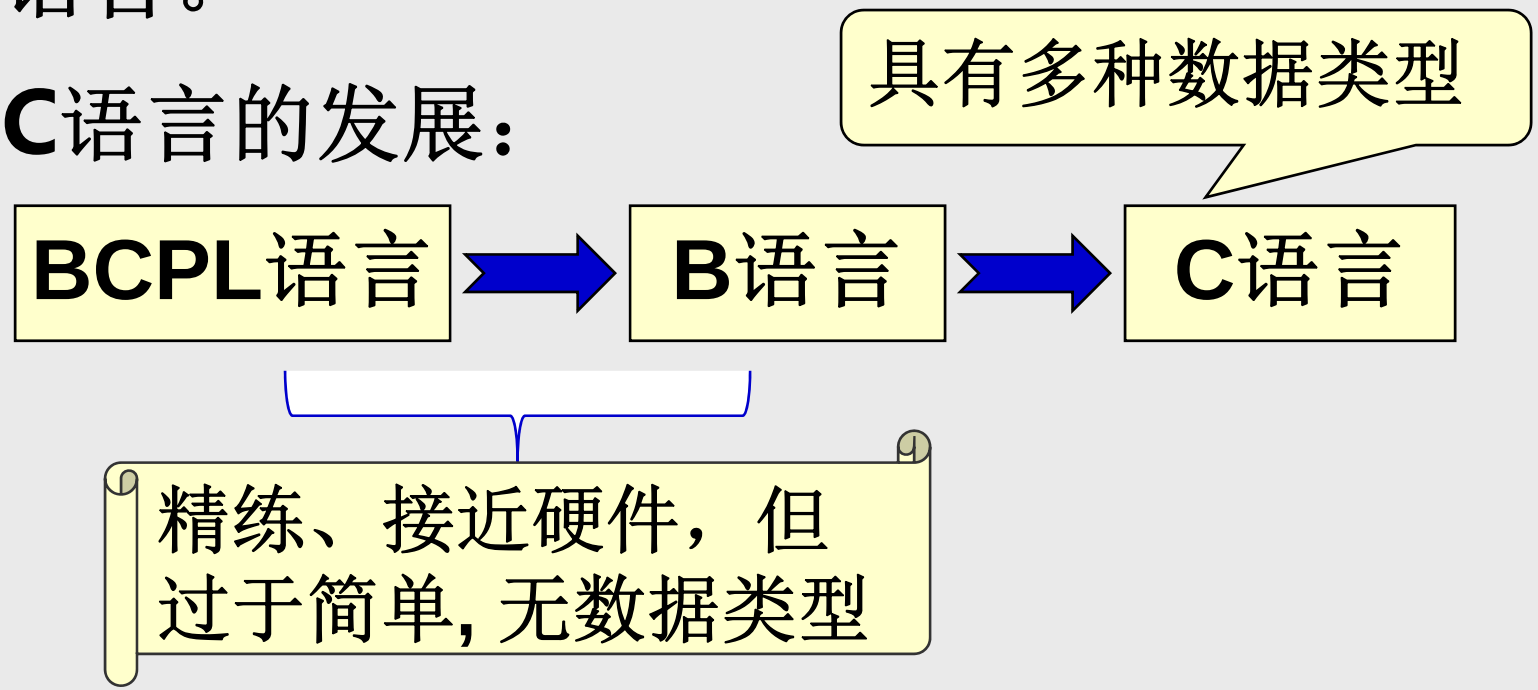
1.2 什么是计算机语言

低级语言

- 计算机语言发展阶段：
  - 机器语言（由0和1组成的指令）
  - 符号语言（用英文字母和数字表示指令）
  - 高级语言（接近于人的自然语言和数学语言）
    - 面向过程的语言
      （非结构化的语言、结构化语言）
    - 面向对象的语言

1.3 C语言的发展及其特点

- C语言是国际上广泛流行的计算机高级语言。
- C语言的发展：



1.3 C语言的发展及其特点

- 最初的C语言只是为描述和实现UNIX操作系统提供一种工作语言而设计的。

1.3 C语言的发展及其特点

- 1983年，美国国家标准协会(ANSI)成立了一个委员会，根据C语言问世以来各种版本对C语言的发展和扩充，制定了第一个C语言标准草案(’83 ANSI C)。

1.3 C语言的发展及其特点

- 1989年，ANSI公布了一个完整的C语言标准—ANSI X3.159-1989(常称ANSI C，或C89)。

1.3 C语言的发展及其特点

- 1990年，国际标准化组织ISO(International Standard Organization) 接受C89作为国际标准ISO/IEC 9899:1990，它和ANSI的C89基本上是相同的。

1.3 C语言的发展及其特点

- 1995年，ISO对C90作了一些修订，1999年，ISO又对C语言标准进行修订，在基本保留原来的C语言特征的基础上，针对应用的需要，增加了一些功能，尤其是C++中的一些功能，命名为ISO/IEC 9899:1999。

1.3 C语言的发展及其特点

- 2001、2004年先后进行了两次技术修正（TC1和TC2）。

ISO/IEC 9899:1999(及其技术修正)被称为 C99。

- C99是C89(及1995基准增补1)的扩充。

1.3 C语言的发展及其特点

- 本书的叙述以C99标准为依据（对C99新增加的功能作特别的说明）。
- 目前不同软件公司提供的各C语言编译系统多数并未完全实现C99建议的功能
- 本书中程序基本上都可以在目前所用的编译系统(如VC++ 6.0，Turbo C++ 3.0,GCC)上编译和运行。

1.3 C语言的发展及其特点

- C语言是一种用途广泛、功能强大、使用灵活的过程性(procedural)编程语言，既可用于编写应用软件，又能用于编写系统软件。因此C语言问世以后得到迅速推广。

1.3 C语言的发展及其特点

- C语言主要特点：
  - 语言简洁、紧凑，使用方便、灵活。
    - 只有37个关键字、9种控制语句
    - 程序书写形式自由，源程序短

1.3 C语言的发展及其特点

- C语言主要特点：
  - 运算符丰富。
    - 有34种运算符
    - 把括号、赋值、强制类型转换等都作为运算符处理
    - 表达式类型多样化

1.3 C语言的发展及其特点

- C语言主要特点：
  - 数据类型丰富。
    - 包括:整型、浮点型、字符型、数组类型、指针类型、结构体类型、共用体类型
    - C99又扩充了复数浮点类型、超长整型(long long)、布尔类型(bool)
    - 指针类型数据，能用来实现各种复杂的数据结构(如链表、树、栈等)的运算。

1.3 C语言的发展及其特点

- C语言主要特点：
  - 具有结构化的控制语句
    - 如if...else语句、while语句、do...while语句、switch语句、for语句
    - 用函数作为程序的模块单位，便于实现程序的模块化
    - C语言是完全模块化和结构化的语言

1.3 C语言的发展及其特点

- C语言主要特点：
  - 语法限制不太严格，程序设计自由度大。
    - 对数组下标越界不做检查
    - 对变量的类型使用比较灵活，例如，整型量与字符型数据可以通用
    - C语言允许程序编写者有较大的自由度，因此放宽了语法检查

1.3 C语言的发展及其特点

- C语言主要特点：
  - 允许直接访问物理地址，能进行位操作，可以直接对硬件进行操作
    - C语言具有高级语言的功能和低级语言的许多功能，可用来编写系统软件
    - 这种双重性，使它既是成功的系统描述语言，又是通用的程序设计语言

1.3 C语言的发展及其特点

- C语言主要特点：
  - 用C语言编写的程序可移植性好。
    - C的编译系统简洁，很容易移植到新系统
    - 在新系统上运行时，可直接编译“标准链接库”中的大部分功能，不需要修改源代码
    - 几乎所有计算机系统都可以使用C语言

1.3 C语言的发展及其特点

- C语言主要特点：
  - 生成目标代码质量高，程序执行效率高。

1.4最简单的C语言程序

1.4.1 最简单的C语言程序举例

1.4.2 C语言程序的结构

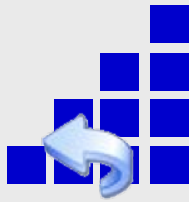
1.4.1 最简单的C语言程序举例

例1.1 要求在屏幕上输出以下一行信息。

This is a C program.

- 解题思路：

  在主函数中用**printf**函数原样输出以上文字。



1.4.1 最简单的C语言程序举例

```
#include <stdio.h>
int main( )
{
   printf (" This is a C program.\n" );
   return 0;
}
```

C程序必须有一个 main 函数

函数的名字，表示主函数

明德求新 尚用笃行

School of Software

软件学院

1.4.1 最简单的C语言程序举例

```
#include <stdio.h>
int main( )
{
   printf (" This is a C program.\n" );
   return 0;
}
```

主函数类型

1.4.1 最简单的C语言程序举例

```
#include <stdio.h>
int main( )
{
   printf (" This is a C program.\n" );
   return 0;
}
```

函数体

1.4.1 最简单的C语言程序举例

```
#include <stdio.h>
int main( )
{
    printf (" This is a C program.\n" );
    return 0;
}
```

输出函数

输出语句

1.4.1 最简单的C语言程序举例


```
#include <stdio.h>
int main( )
{
   printf (" This is a C program.\n" );
   return 0;
}
```

```
This is a C program.
Press any key to continue_
```

输出语句

```
#include <stdio.h>
int main( )
{
   printf (" This is a C program.\n" );
   return 0;
}
```

```
This is a C program.
Press any key to continue_
```

换行符

```
#include <stdio.h>
int main( )
{
   printf (" This is a C program.\n" );
   return 0;
}
```

当main函数执行结束前
将整数0作为函数值

```
#include <stdio.h>
int main( )
{
   printf (" This is a C program.\n" );
   return 0;
}
```

用到函数库中的输入输出函数时

表示语句结束

C语言允许用两种注释方式：

- //：单行注释
  - 可单独占一行
  - 可出现在一行中其他内容的右侧
- /*……*/：块式注释
  - 可包含多行

例1.2 求两个整数之和。

- 解题思路：
  - 设置3个变量
  - a和b用来存放两个整数
  - sum用来存放和数
  - 用赋值运算符“=”把结果传送给sum

```
#include <stdio.h>
int main( )
{
  int a,b,sum;
  a = 123;
  b = 456;
  sum =  a + b;
  printf(" sum is %d\n" ,sum);
  return 0;
}
```

定义整型变量a,b,sum

对变量a,b赋值

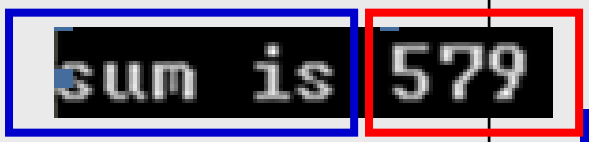
将a与b的和赋给sum

```
sum is 579
```

```
#include <stdio.h>
int main( )
{
  int a,b,sum;
  a = 123;
  b = 456;
  sum =  a + b;
  printf(" sum is %d\n" ,sum);
  return 0;
}
```

用sum的值替代

希望输出的字符



明德求新 尚用笃行

例1.3求两个整数中的较大者。

- 解题思路：
  - 用一个函数实现求两个整数中的较大者
  - 在主函数中调用此函数并输出结果

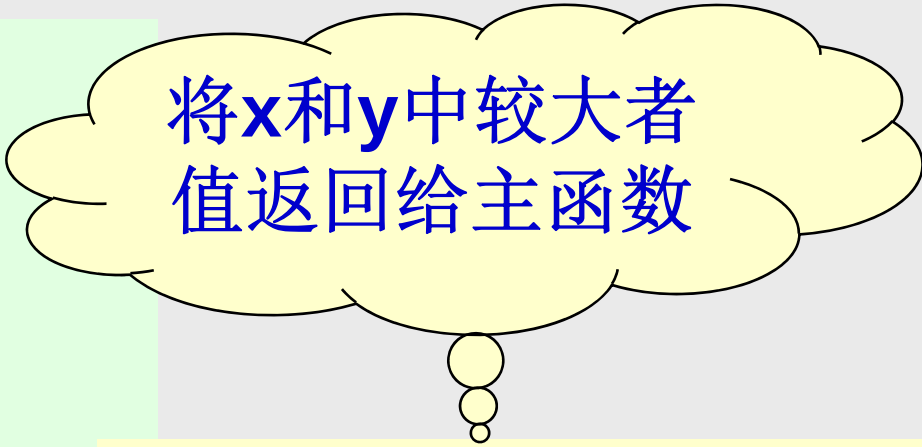
```c
#include <stdio.h>
int main( )
{
  int max(int x,int y);
  int a,b,c;
  scanf(" %d,%d" ,&a,&b);
  c = max(a,b);
  printf("max=%d\n",c);
  return 0;
}
```

主函数

max函数

```c
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

```
#include <stdio.h>
int main( )
{
  int max(int x,int y);
  int a,b,c;
  scanf(" %d,%d" ,&a,&b);
  c = max(a,b);
  printf("max=%d\n",c);
  return 0;
}
```




```
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

```
#include <stdio.h>
int main( )
{
  int max(int x,int y);
  int a,b,c;
  scanf(" %d,%d" ,&a,&b);
  c = max(a,b);
  printf("max=%d\n",c);
  return 0;
}
```

```
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

```
#include <stdio.h>
int main( )
{
  int max(int x,int y);
  int a,b,c;
  scanf(" %d,%d" ,&a,&b);
  c = max(a,b);
  printf("max=%d\n",c);
  return 0;
}
```

因max函数的定义在main函数之后，需声明

```
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

```
#include <stdio.h>
int main( )
{
  in                    .nt y);
  int a,b,c;
  scanf(" %d,%d" ,&a,&b);
  c = max(a,b);
  printf("max=%d\n",c);
  return 0;
}
```

输入函数

```
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

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

输入语句

```
8,5
```

```
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

```
#include <stdio.h>
int main( )
{
  int r...);
  int a,b,c;
  scanf(" %d,%d" ,&a,&b);
  c = max(a,b);
  printf("max=%
  return 0;
}
```

输入格式

输入的数据放到a,b中

a的地址

```
8,5
```

```
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

```
#include <stdio.h>
int main( )
{
  int max(int x,int y);
  int a,b,c;
  scanf("%d,%d",&a,&b);
  c = max(a,b);
  printf("max=%d\n",c);
  return 0;
}
```

调用max函数

```
8,5
```

```
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

School

```
#include <stdio.h>
int main( )
{
  int max(int x,int y);
  int a,b,c;
  8 anf(" %d,%d" ,&a,&b);
  c = max(a,b);
  printf("max=%d\n",c);
  return 0;
}
```

```
8,5
```

8 5

```
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

8

```
#include <stdio.h>
int main( )
{
  int max(int x,int y);
  int a,b,c;
  8 anf(" %d,%d" ,&a,&b);
  c = max(a,b);
  printf("max=%d\n",c);
  return 0;
}
```

```
8 ,5
max=8
```

8 5

```
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

8

```
#include <stdio.h>
int main( )
{
  int max(int x,int y);
  int a,b,c;
  scanf(" %d,%d" ,&a,&b);
  c = max(a,b);
  printf("max=%d\n",c);
  return 0;
}
```

实际参数

形式参数

```
int max(int x,int y)
{
  int z;
  if (x > y) z = x;
  else z = y;
  return(z);
}
```

```
8,5
max=8
```

School

1.4.2 C语言程序的结构

C语言程序的结构特点：

1.一个程序由一个或多个源程序文件组成

- 小程序往往只包括一个源程序文件
- 例1.1，例1.2只有一个函数
- 例1.3有两个函数

只包括一个源程序文件

1.4.2 C语言程序的结构

C语言程序的结构特点：

- 一个源程序文件中可以包括三个部分：
  - 预处理指令　#include <stdio.h>等
  - 全局声明　在函数之外进行的数据声明
  - 函数定义　每个函数用来实现一定的功能

1.4.2 C语言程序的结构

C语言程序的结构特点：

2.函数是C程序的主要组成部分

- 一个C程序是由一个或多个函数组成的
- 必须包含一个main函数（只能有一个）
- 每个函数都用来实现一个或几个特定功能
- 被调用的函数可以是库函数，也可以是自己编制设计的函数

1.4.2 C语言程序的结构

C语言程序的结构特点：

3.一个函数包括两个部分：

- 函数首部　　函数的第1行

```
int   max ( int  x,  int  y )
```

函数类型　函数名　参数类型　参数名

1.4.2 C语言程序的结构

C语言程序的结构特点：

3.一个函数包括两个部分：

- 函数首部

```
int   max（ int   x，  int   y ）
```

若函数无参，在括弧中写void或空括弧

int main(void)　或　int main()

1.4.2 C语言程序的结构

C语言程序的结构特点：

3.一个函数包括两个部分：

- 函数体
  - 声明部分
    - 定义在本函数中所用到的变量
    - 对本函数所调用函数进行声明
  - 执行部分：由若干个语句组成，指定在函数中所进行的操作

可以没有声明部分

1.4.2 C语言程序的结构

C语言程序的结构特点：

3.一个函数包括两个部分：

- 函数体

可以是空函数

```
void dump ( )
{       }
```

1.4.2 C语言程序的结构

C语言程序的结构特点：

4. 程序总是从main函数开始执行

5. C程序对计算机的操作由C语句完成

- C程序书写格式是比较自由的
  - 一行内可以写几个语句
  - 一个语句可以分写在多行上
- 为清晰起见，习惯上每行只写一个语句

1.4.2 C语言程序的结构

C语言程序的结构特点：

4. 程序总是从main函数开始执行
5. C程序对计算机的操作由C语句完成
6. 数据声明和语句最后必须有分号
7. C语言本身不提供输入输出语句
8. 程序应当包含注释，增加可读性

1.5 运行C程序的步骤与方法

1.上机输入和编辑源程序（.c文件）

2.对源程序进行编译（.obj文件）

3.进行连接处理（.exe文件）

4.运行可执行程序，得到运行结果

说明：以上过程参见教材中图1.1

附录A中有Visual C++ 6.0中编辑、编译、连接和运行C程序的方法

1.6 程序设计的任务

1.问题分析

- 对于接手的任务要进行认真的分析
- 研究所给定的条件
- 分析最后应达到的目标
- 找出解决问题的规律
- 选择解题的方法

1.6 程序设计的任务

1.问题分析

2.设计算法

- 设计出解题的方法和具体步骤

1.6 程序设计的任务

1.问题分析

2.设计算法

3.编写程序

4.对源程序进行编辑、编译和连接

5.运行程序，分析结果

- 结果错了，程序肯定错
- 结果对了，程序未必对

1.6 程序设计的任务

1.问题分析

2.设计算法

3.编写程序

4.对源程序进行编辑、编译和连接

5.运行程序，分析结果

6.编写程序文档

明德求新 尚用笃行

第2章 算法---程序的灵魂

School of Software

- 一个程序主要包括以下两方面的信息：

  (1) 对数据的描述。在程序中要指定用到哪些数据以及这些数据的类型和数据的组织形式

  - 这就是数据结构(data structure)

  (2) 对操作的描述。即要求计算机进行操作的步骤

  - 也就是算法(algorithm)

- 数据是操作的对象
- 操作的目的是对数据进行加工处理，以得到期望的结果
- 著名计算机科学家沃思(Nikiklaus Wirth)提出一个公式：

  算法 + 数据结构 = 程序

- 一个程序除了算法和数据结构这主要要素外，还应当采用结构化程序设计方法进行程序设计，并且用某一种计算机语言表示
- 算法、数据结构、程序设计方法和语言工具是一个程序设计人员应具备的知识

明德求新 尚用笃行

- 算法是解决“做什么”和“怎么做”的问题
- 程序中的操作语句，是算法的体现
- 不了解算法就谈不上程序设计

明德求新 尚用笃行

School of Software

2.1 什么是算法

2.2 简单的算法举例

2.3 算法的特性

2.4 怎样表示一个算法

2.5 结构化程序设计方法

2.1 什么是算法

- 广义地说，为解决一个问题而采取的方法和步骤，就称为“算法”
- 对同一个问题，可以有不同的解题方法和步骤
- 为了有效地进行解题，不仅需要保证算法正确，还要考虑算法的质量，选择合适的算法

2.1 什么是算法

- 计算机算法可分为两大类别：
  - 数值运算算法
  - 非数值运算算法
- 数值运算的目的是求数值解
- 非数值运算包括的面十分广泛，最常见的是用于事务管理领域

2.2简单的算法举例

例2.1 求1×2×3×4×5× …×1000

- 可以用最原始的方法进行。
  - 步骤1：先计算1×2，得到结果2。
  - 步骤2：将步骤1得到的乘积2再乘以3，得到结果6。
  - 步骤3：将6再乘以4，得24。
  - 步骤4：将24再乘以5，得120。这就是最后的结果。

太繁琐

2.2简单的算法举例

- 改进的算法：
  - 设变量p为被乘数
  - 变量i为乘数
  - 用循环算法求结果

2.2简单的算法举例

- S1：使p=1，或写成1⇒p
- S2：使i=2，或写成2⇒i
- S3：使p与i相乘，乘积仍放在变量p中，可表示为：p*i⇒p
- S4：使i的值加1，即i+1⇒i
- S5：如果i不大于5，返回重新执行S3；否则，算法结束
- 最后得到p的值就是 5!的值

若是1000，求什么？

若求1×3×5×7×9×11

- S1：使p=1，或写成1⇒p
- S2：使i=3 或写成2 3 i
- S3：使p与i相乘，乘积仍放在变量p中，可表示为：p*i⇒p
- S4：使i的值加2 即i+2⇒i
- S5：如果i不大于11返回重新执行S3；否则，算法结束
- 最后得到p的值就是11的值

相当于i ≤11

例2.2 有50个学生，要求将成绩在80分以上的学生的学号和成绩输出。

- 用n_i代表第i个学生学号，g_i表示第i个学生成绩

S1：$1 \Rightarrow i$

S2：如果$g_i \geq 80$，

则输出n_i和g_i，否则不输出

S3：$i+1 \Rightarrow i$

S4：如果$i \leq 50$，返回到步骤S2，继续执行，否则，算法结束

例2.3 判定2000—2500年中的每一年是否闰年，并将结果输出。

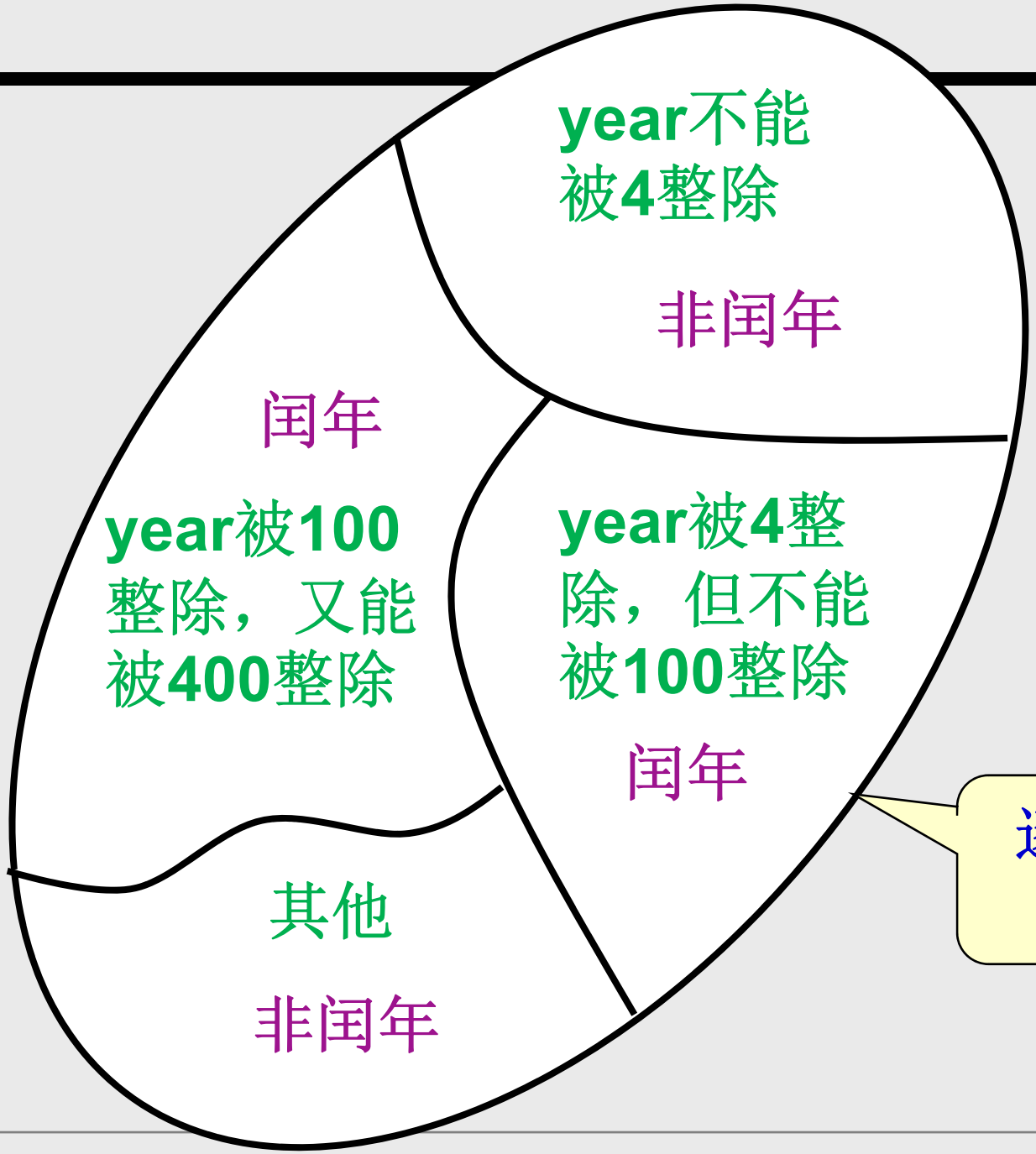
- 闰年的条件：

  (1)能被4整除，但不能被100整除的年份都是闰年，如2008、2012、2048年

  (2)能被400整除的年份是闰年，如2000年

  - 不符合这两个条件的年份不是闰年
  - 例如2009、2100年

- 设year为被检测的年份。算法表示如下：
  - S1：2000⇒year
  - S2：若year不能被4整除，则输出year 的值和“不是闰年”。然后转到S6
  - S3：若year能被4整除，不能被100整除，则输出year的值和“是闰年”。然后转到S6
  - S4：若year能被400整除，则输出year的值和“是闰年”，然后转到S6
  - S5: 其他情况输出year的值和“不是闰年”
  - S6：year+1⇒year
  - S7：当year≤2500时，转S2，否则停止


year不能
被4整除
非闰年
闰年
year被100
整除，又能
被400整除
year被4整
除，但不能
被100整除
闰年
其他
非闰年

逐渐缩小判断的范围

例2.4 求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$

- 规律：
  - ①第1项的分子分母都是1
  - ② 第2项的分母是2，以后每一项的分母子都是前一项的分母加1
  - ③ 第2项前的运算符为“-”，后一项前面的运算符都与前一项前的运算符相反

例2.4 求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$

sign—当前项符号
term—当前项的值
sum—当前各项的和
deno—当前项分母

- S1：sign=1
- S2：sum=1
- S3：deno=2
- S4：sign=(-1)*sign
- S5：term=sign*(1/deno)
- S6：sum=sum+term
- S7：deno=deno+1
- S8：若deno≤100返回S4；否则算法结束

-1

-1/2

1-1/2

3

满足，返回S4

例2.4 求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$

sign—当前项符号
term—当前项的值
sum—当前各项的和
deno—当前项分母

- S1：sign=1
- S2：sum=1
- S3：deno=2
- S4：sign=(-1)*sign
- S5：term=sign*(1/deno)
- S6：sum=sum+term
- S7：deno=deno+1
- S8：若deno≤100返回S4；否则算法结束

1

1/3

1-1/2+1/3

4

满足，返回S4

例2.4 求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$

- S1：sign=1
- S2：sum=1
- S3：deno=2
- S4：sign=(-1)*sign
- S5：term=sign*(1/deno)
- S6：sum=sum+term
- S7：deno=deno+1
- S8：若deno≤100返回S4；否则算法结束

99次循环后sum的值就是所要求的结果

例2.5 给出一个大于或等于3的正整数，判断它是不是一个素数。

- 所谓素数(prime)，是指除了1和该数本身之外，不能被其他任何整数整除的数
- 例如，13是素数，因为它不能被2，3，4，…，12整除。

- 判断一个数n(n≥3)是否素数：将n作为被除数，将2到(n-1)各个整数先后作为除数，如果都不能被整除，则n为素数

S1：输入n的值

S2：i=2　　（i作为除数）

S3：n被i除，得余数r

S4：如果r=0，表[illegible]则输出n“不是素数”，算法[illegible]否则执行S5

S5：i+1⇒i

S6：如果i≤n-1，返回S3；否则输出n“是素数”，然后结束。

可改为n/2 $\sqrt{n}$

2.3算法的特性

- 一个有效算法应该具有以下特点：

(1) 有穷性。一个算法应包含有限的操作步骤，而不能是无限的。

(2) 确定性。算法中的每一个步骤都应当是确定的，而不应当是含糊的、模棱两可的。

2.3算法的特性

- 一个有效算法应该具有以下特点：

(3) 有零个或多个输入。所谓输入是指在执行算法时需要从外界取得必要的信息。

(4) 有一个或多个输出。算法的目的是为了求解，“解” 就是输出。

- 没有输出的算法是没有意义的。

(5) 有效性。算法中的每一个步骤都应当能有效地执行，并得到确定的结果。

2.3算法的特性

- 对于一般最终用户来说:
  - 他们并不需要在处理每一个问题时都要自己设计算法和编写程序
  - 可以使用别人已设计好的现成算法和程序
  - 只需根据已知算法的要求给予必要的输入，就能得到输出的结果

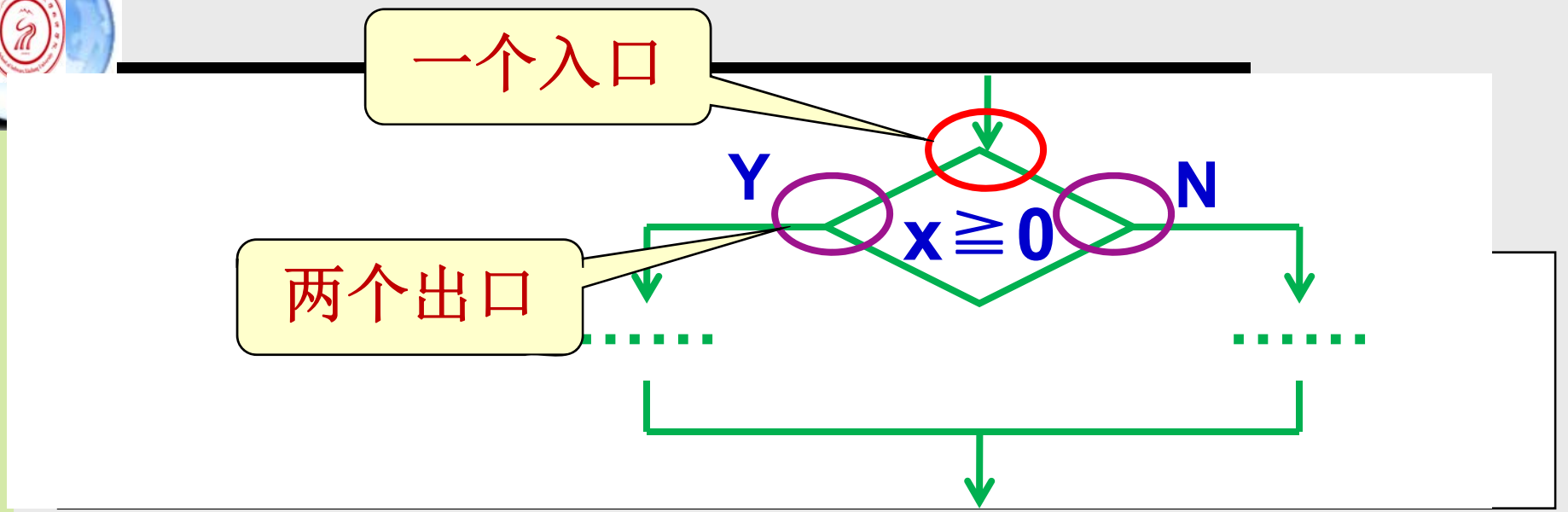
2.4怎样表示一个算法

- 常用的方法有：
  - 自然语言
  - 传统流程图
  - 结构化流程图
  - 伪代码
  - ……

2.4怎样表示一个算法

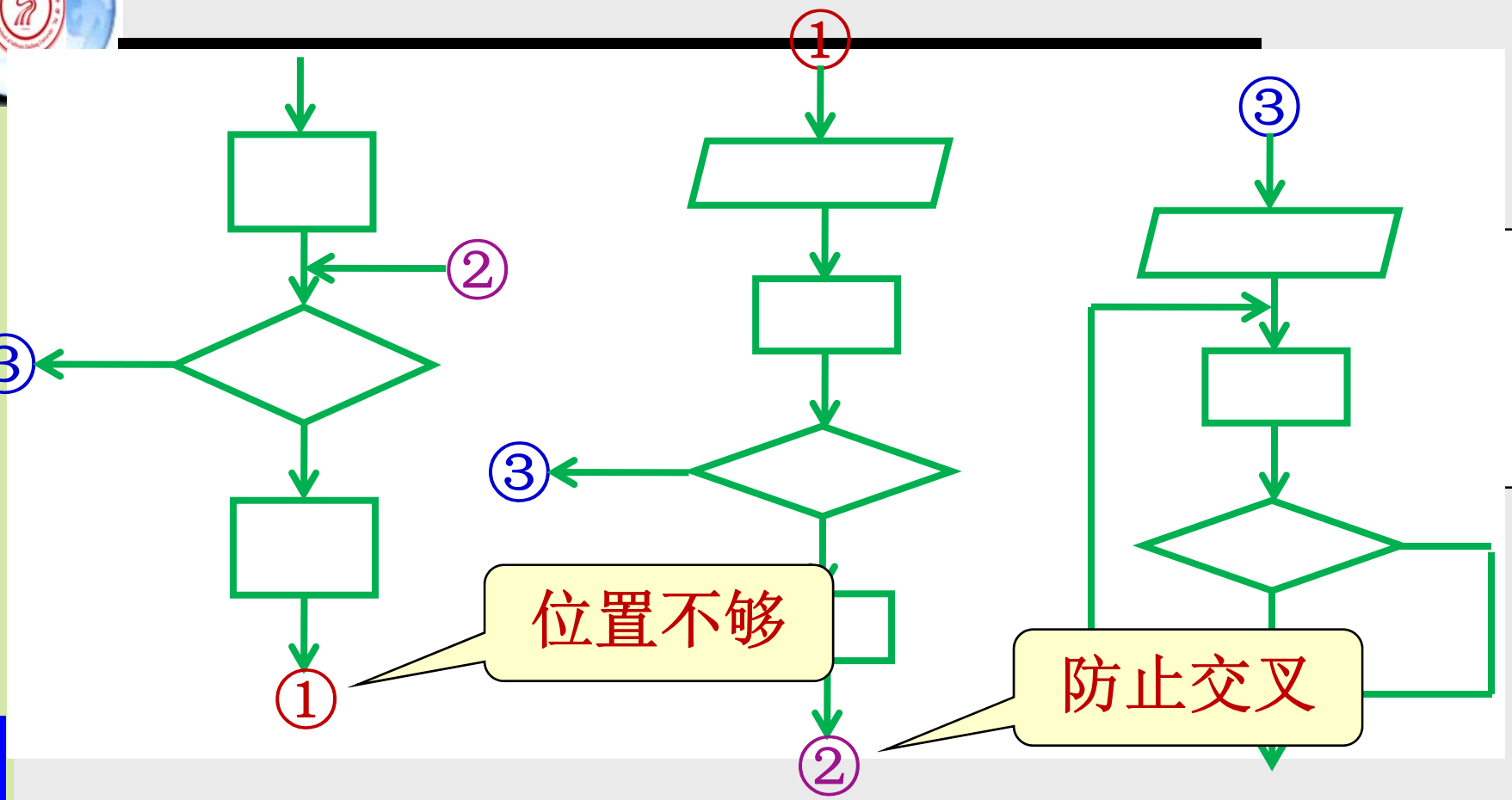
2.4.1 用自然语言表示算法

- 2.2节介绍的算法是用自然语言表示的
- 用自然语言表示通俗易懂，但文字冗长，容易出现歧义性
- 用自然语言描述包含分支和循环的算法，不很方便
- 除了很简单的问题外，一般不用自然语言



起止框　输入输出框　判断框　处理框

流程线　连接点　注释框

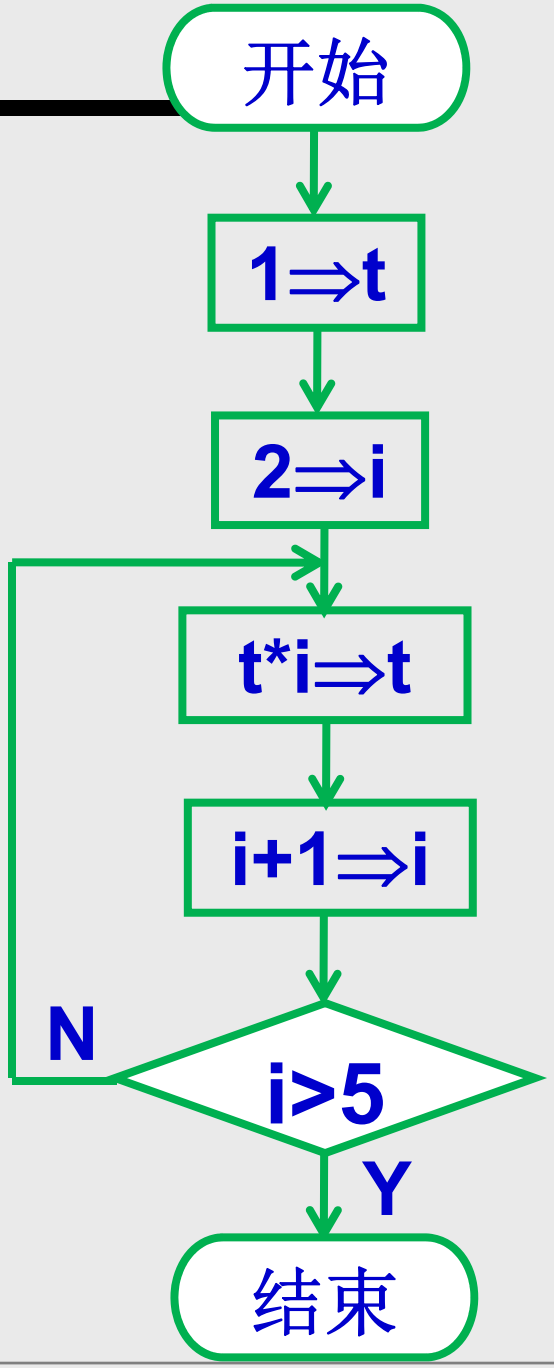


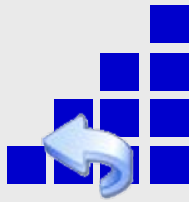
流程线　　连接点　　注释框

例2.6 将例2.1的算法用流程图表示。

求1×2×3×4×5

- 如果需要将最后结果输出:

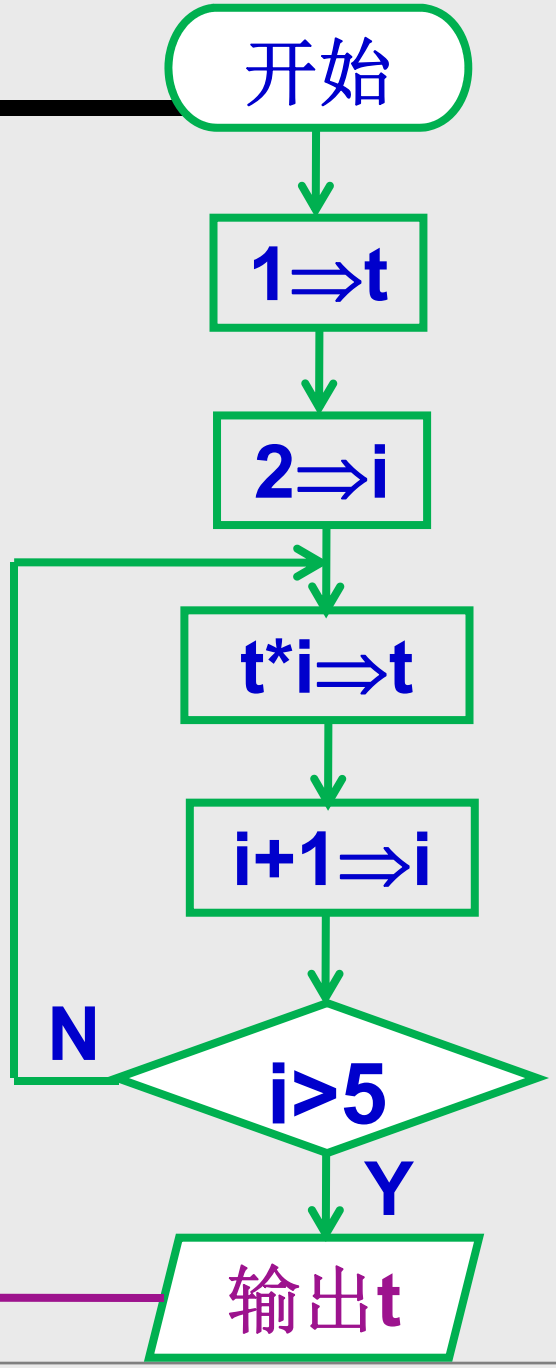




例2.6 将例2.1的算法用流程图表示。

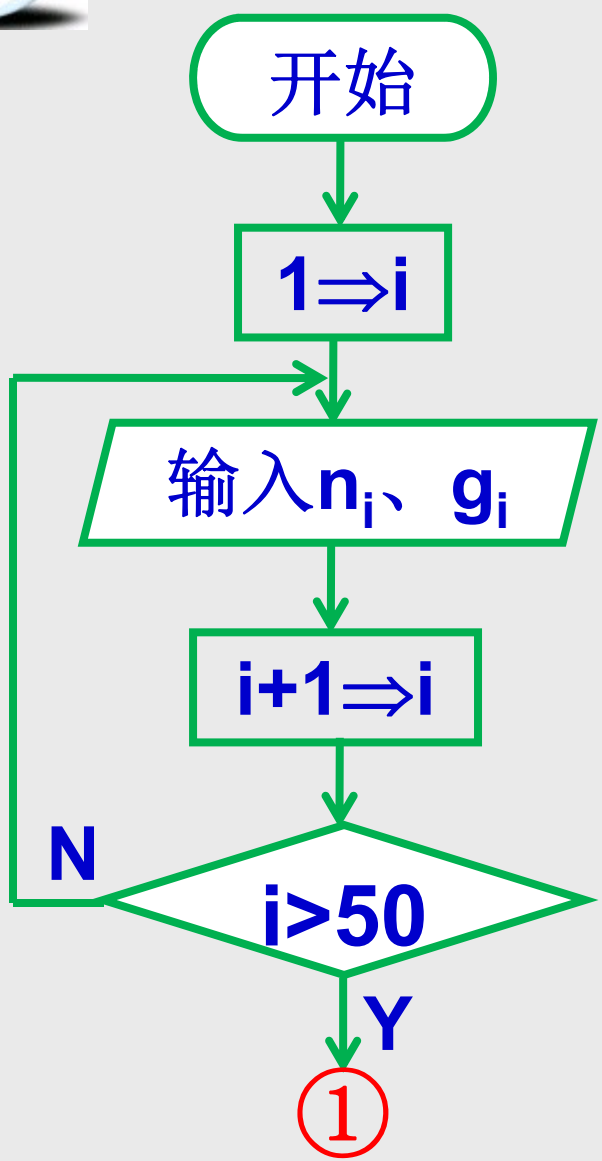
求1×2×3×4×5

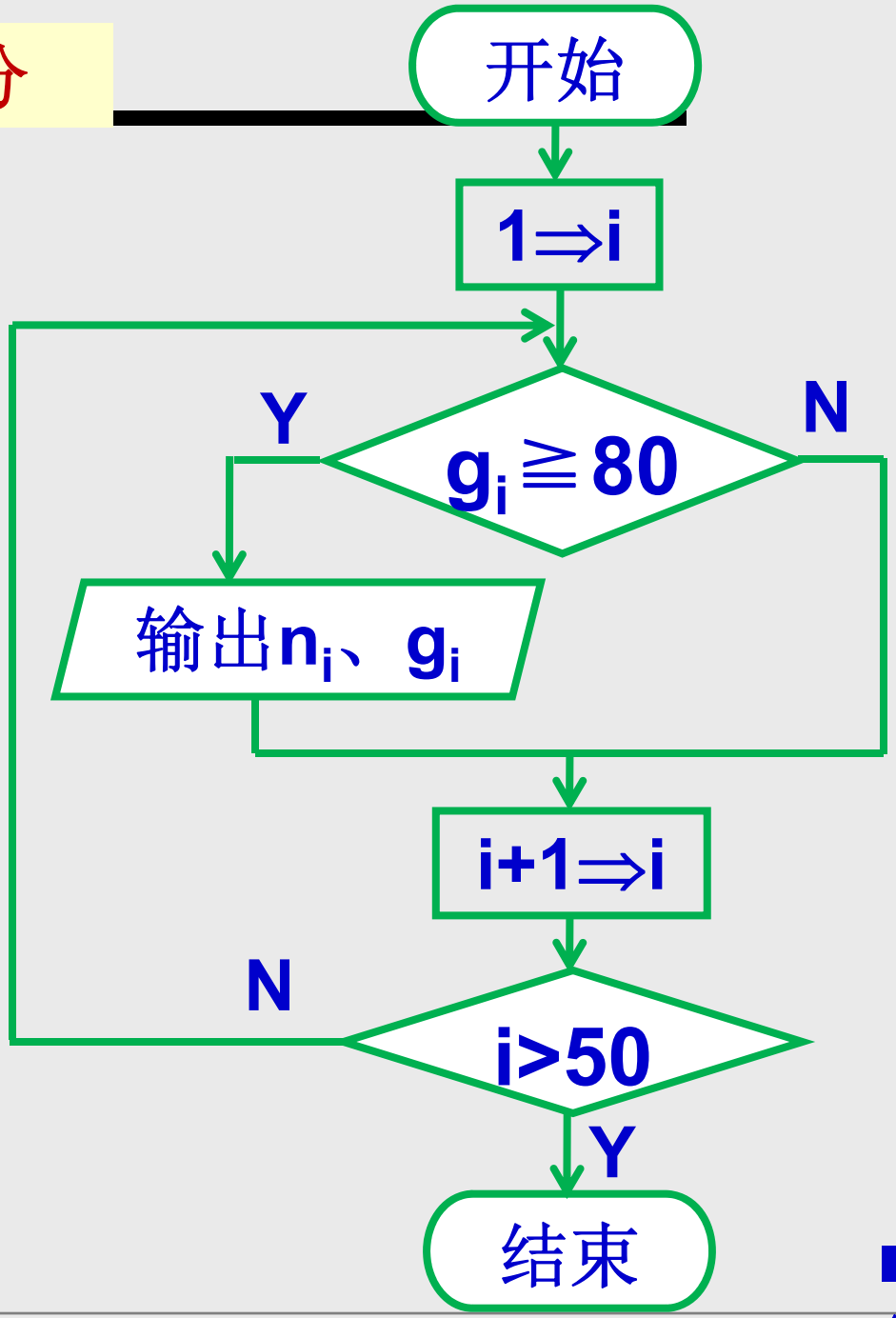
- 如果需要将最后结果输出:



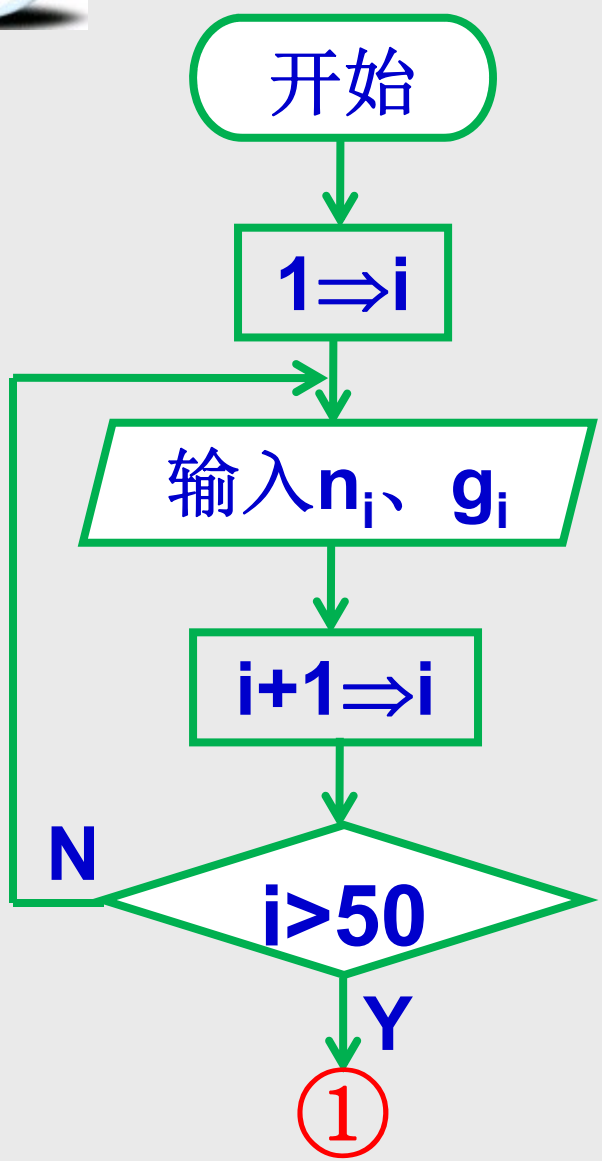
例2.7 例2.2的算法用流程图表示。有50个学生，要求将成绩在80分以上的学生的学号和成绩输出。

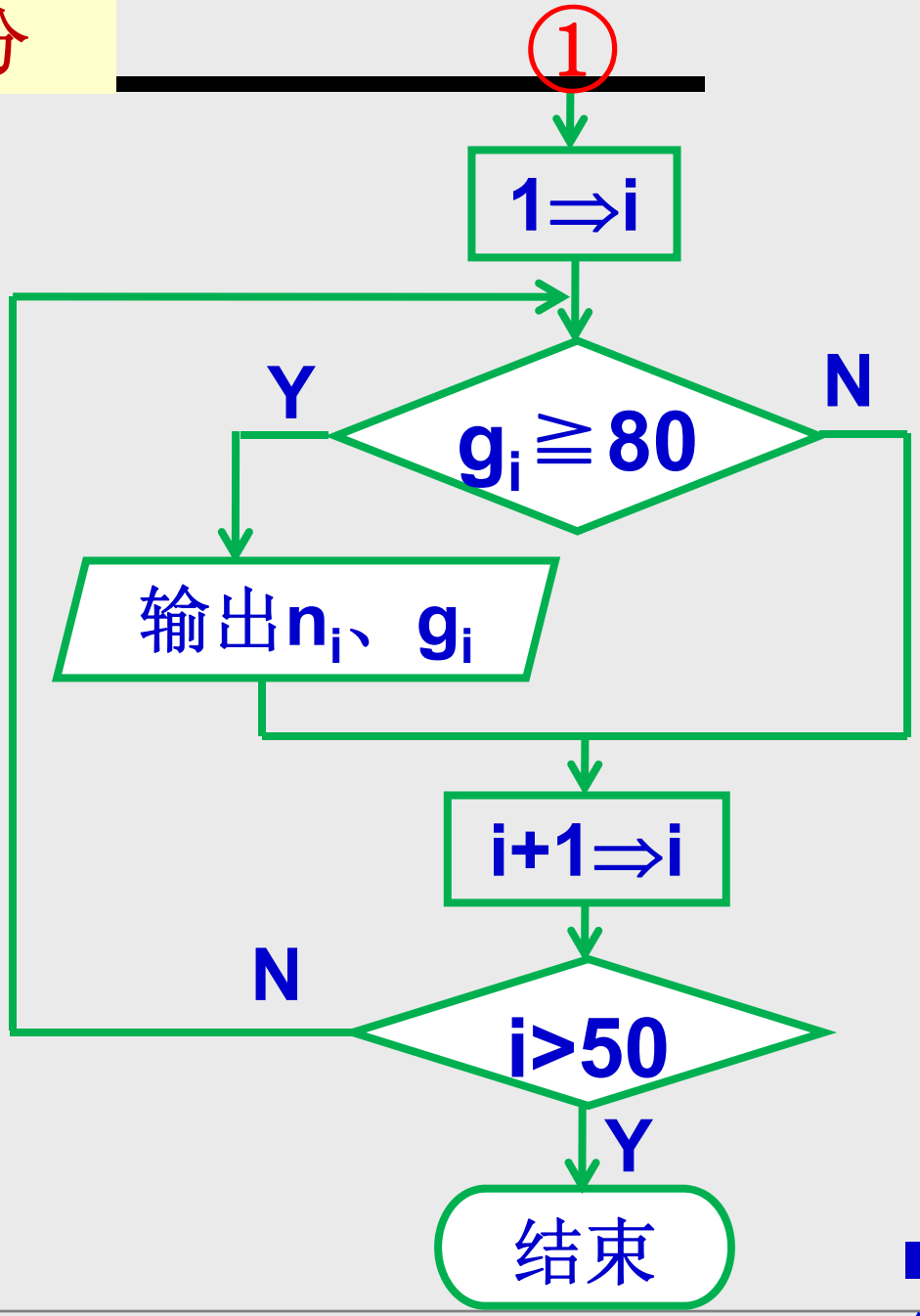
如果包括输入数据部分



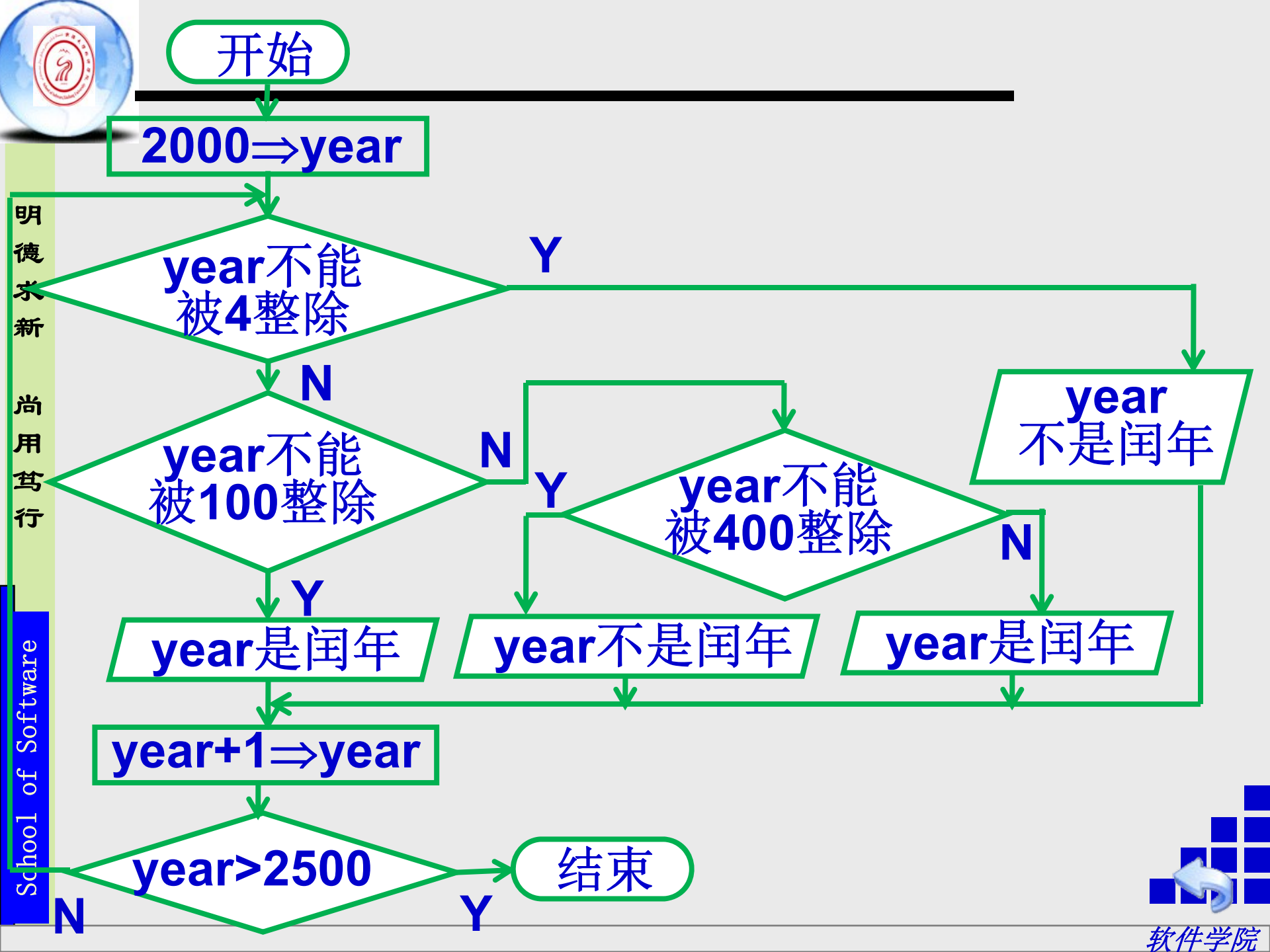


如果包括输入数据部分



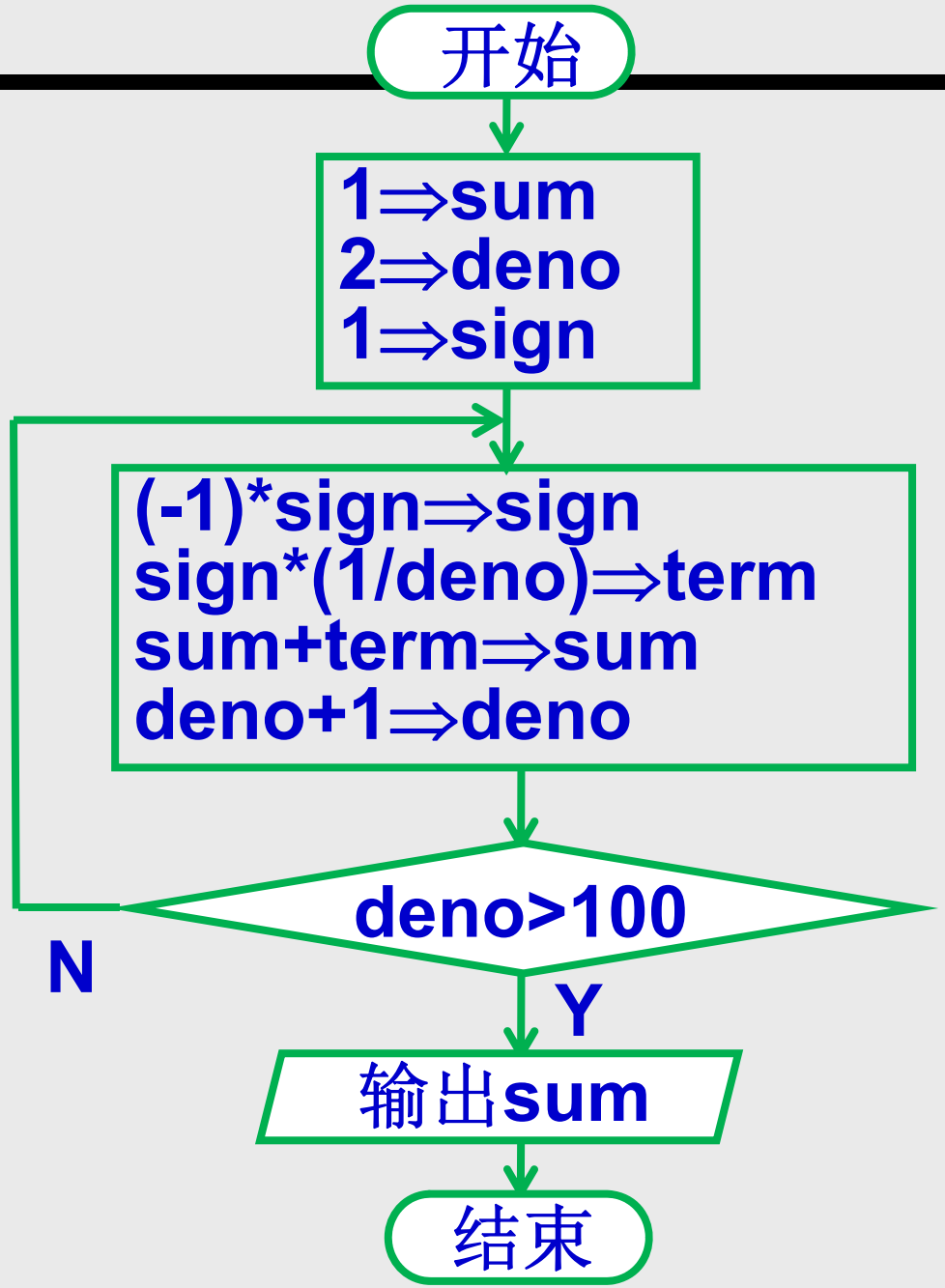


例2.8 例2.3判定闰年的算法用流程图表示。判定2000—2500年中的每一年是否闰年，将结果输出。

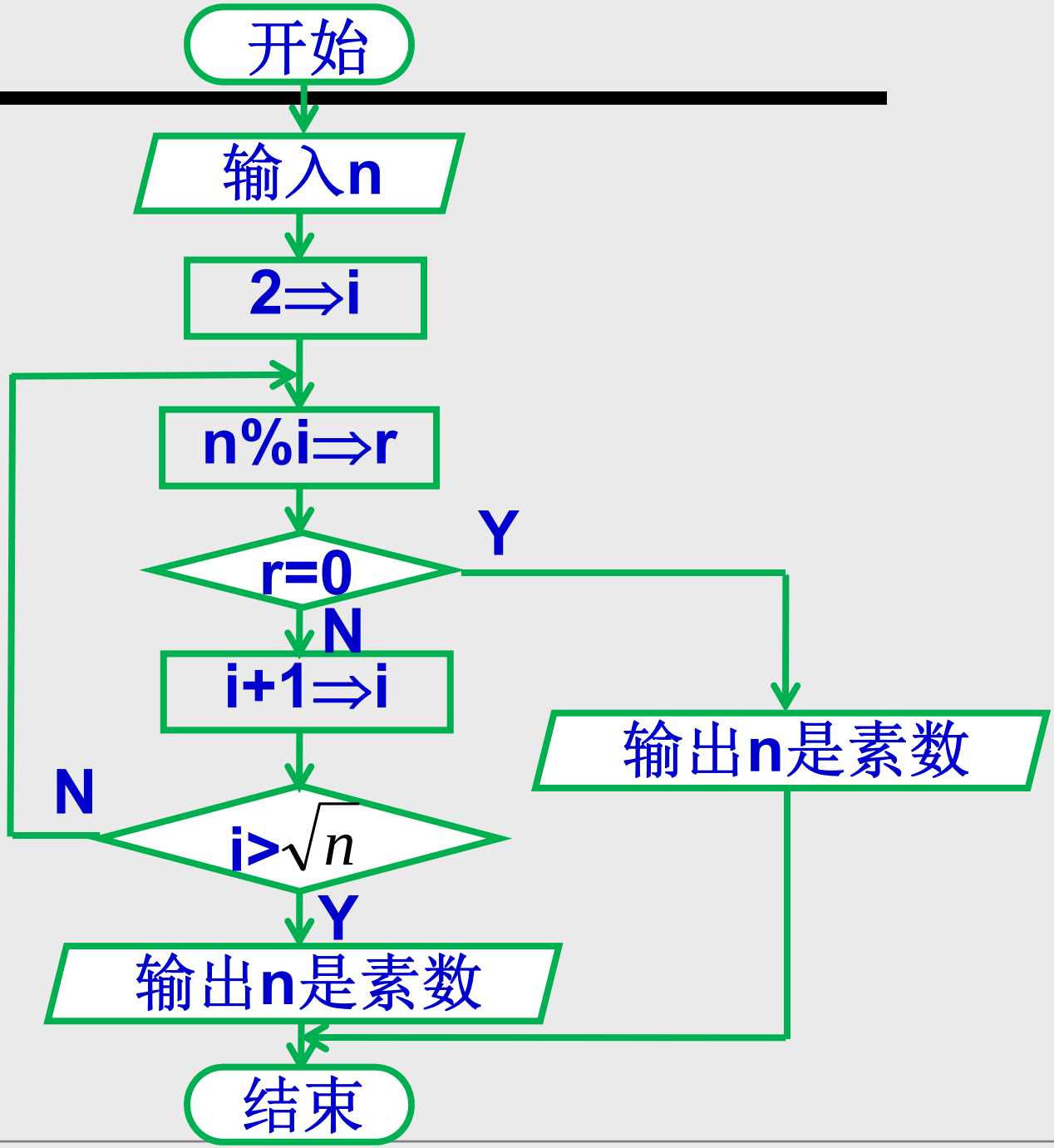

开始
2000⇒year
year不能被4整除
Y
N
year不能被100整除
N
Y
year不能被400整除
N
Y
year是闰年
year不是闰年
year是闰年
year不是闰年
year+1⇒year
year>2500
N
Y
结束

例2.9 将例2.4的算法用流程图表示。求

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$$


开始
1⇒sum
2⇒deno
1⇒sign
(-1)*sign⇒sign
sign*(1/deno)⇒term
sum+term⇒sum
deno+1⇒deno
deno>100
N
Y
输出sum
结束

例2.10 例2.5判断素数的算法用流程图表示。对一个大于或等于3的正整数，判断它是不是一个素数。


开始
输入n
2⇒i
n%i⇒r
r=0
Y
N
i+1⇒i
输出n是素数
N
i>√n
Y
输出n是素数
结束

- 通过以上几个例子可以看出流程图是表示算法的较好的工具
- 一个流程图包括以下几部分：
  - **(1)** 表示相应操作的框
  - **(2)** 带箭头的流程线
  - **(3)** 框内外必要的文字说明
- 流程线不要忘记画箭头，否则难以判定各框的执行次序

2.4.3 三种基本结构和改进的流程图

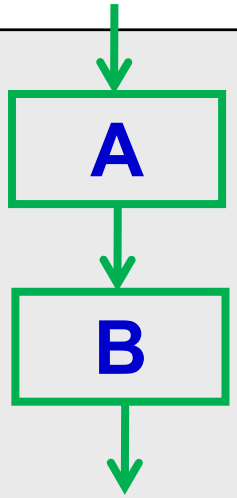
1.传统流程图的弊端

- 传统的流程图用流程线指出各框的执行顺序，对流程线的使用没有严格限制
- 使用者可以毫不受限制地使流程随意地转来转去，使人难以理解算法的逻辑

2.4.3 三种基本结构和改进的流程图

2.三种基本结构

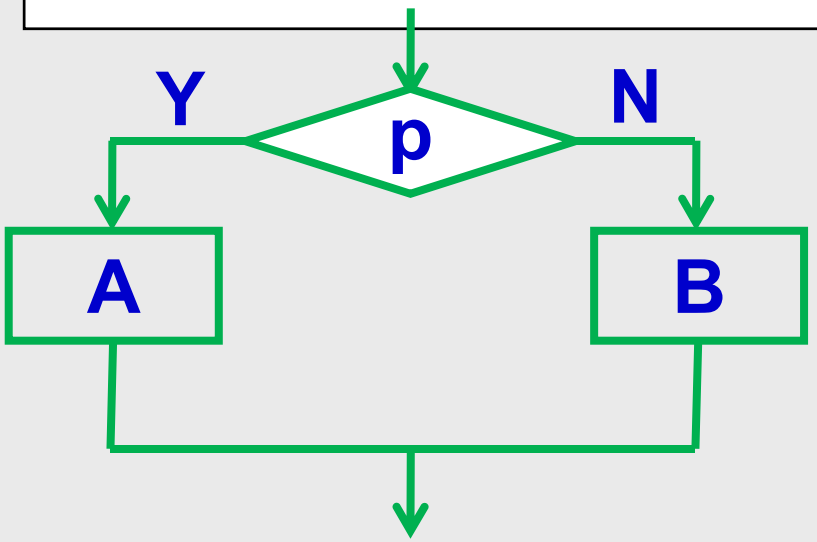
(1) 顺序结构

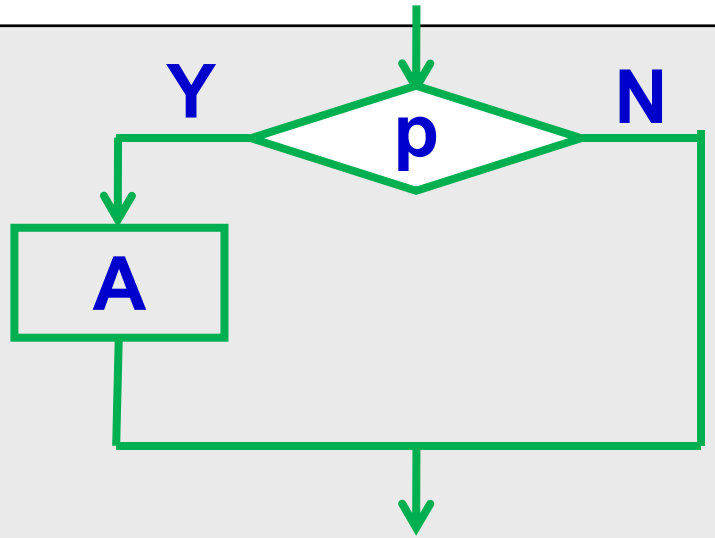


2.4.3 三种基本结构和改进的流程图

2.三种基本结构

(2) 选择结构





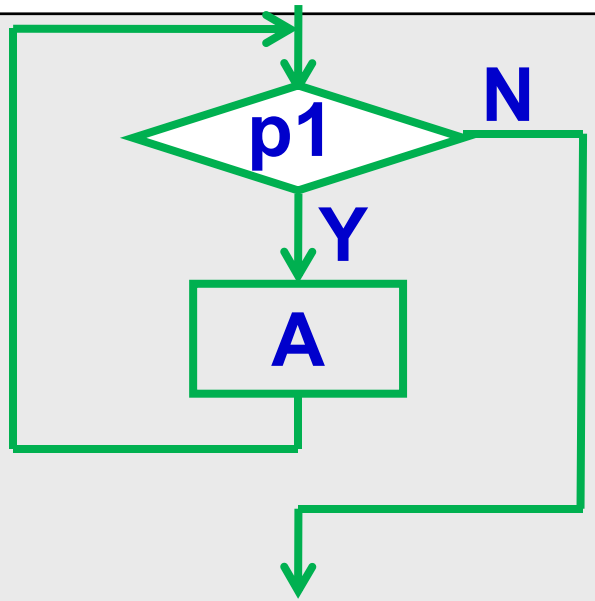
2.4.3 三种[illegible]改进的流程图

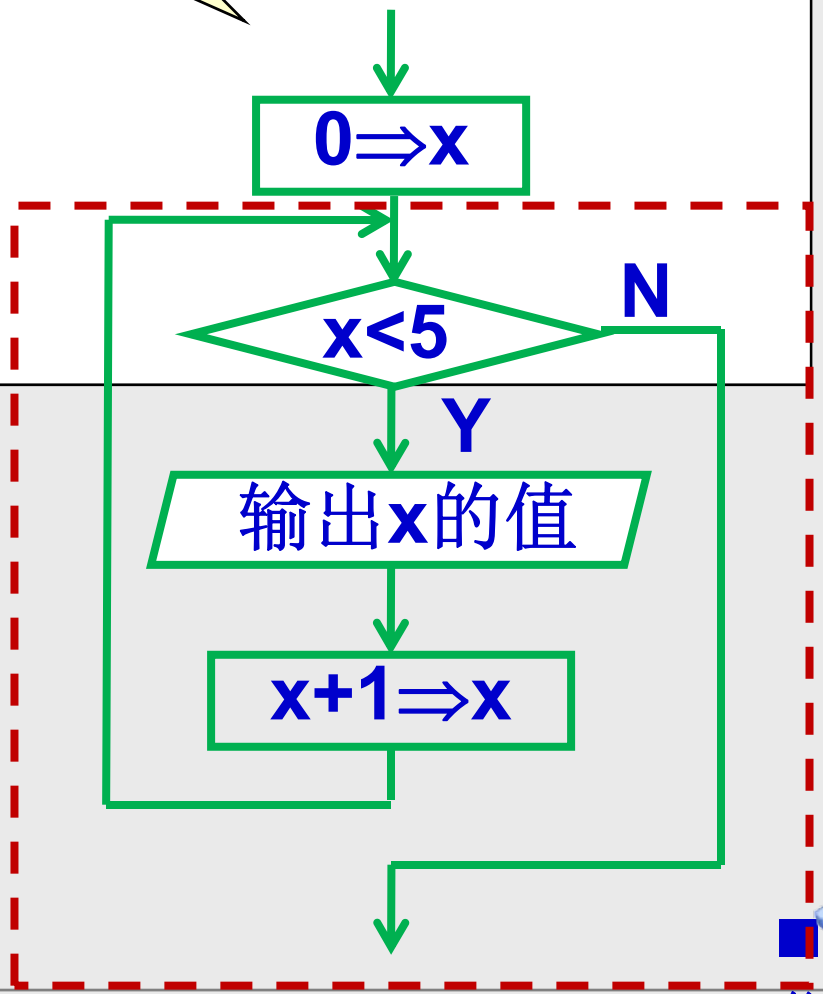
输出1,2,3,4,5

2.三种基本结构

(3) 循环结构

① 当型循环结构





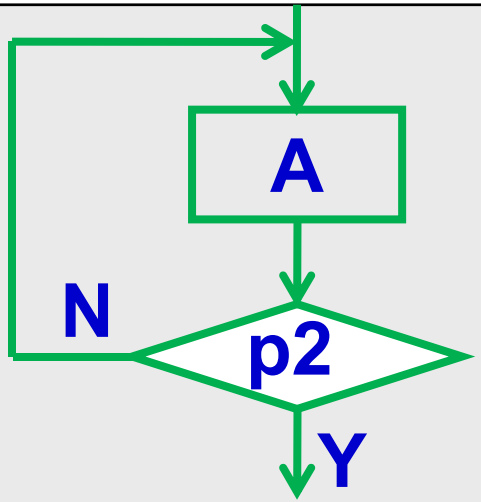
2.4.3 三种[illegible]改进的流程图

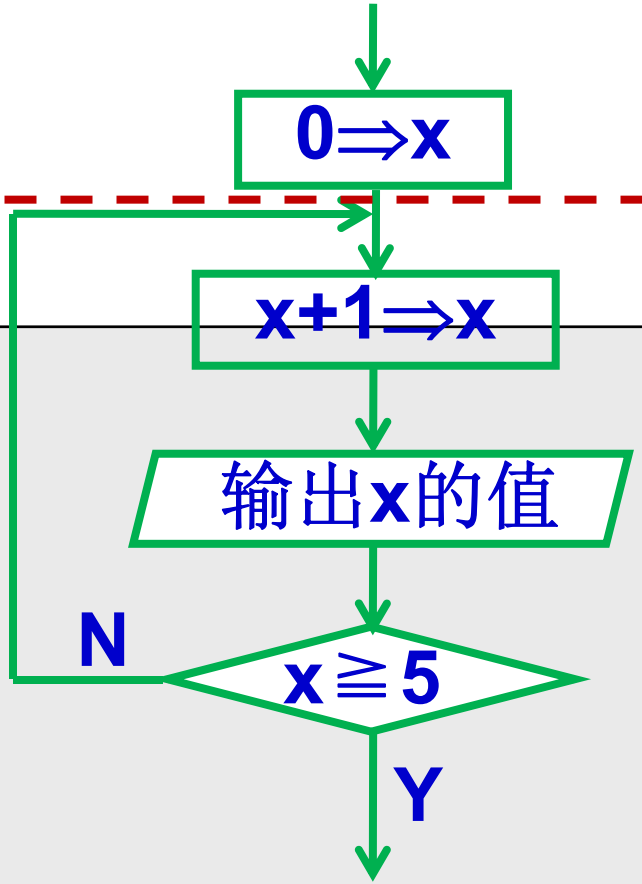
输出1,2,3,4,5

2.三种基本结构

(3) 循环结构

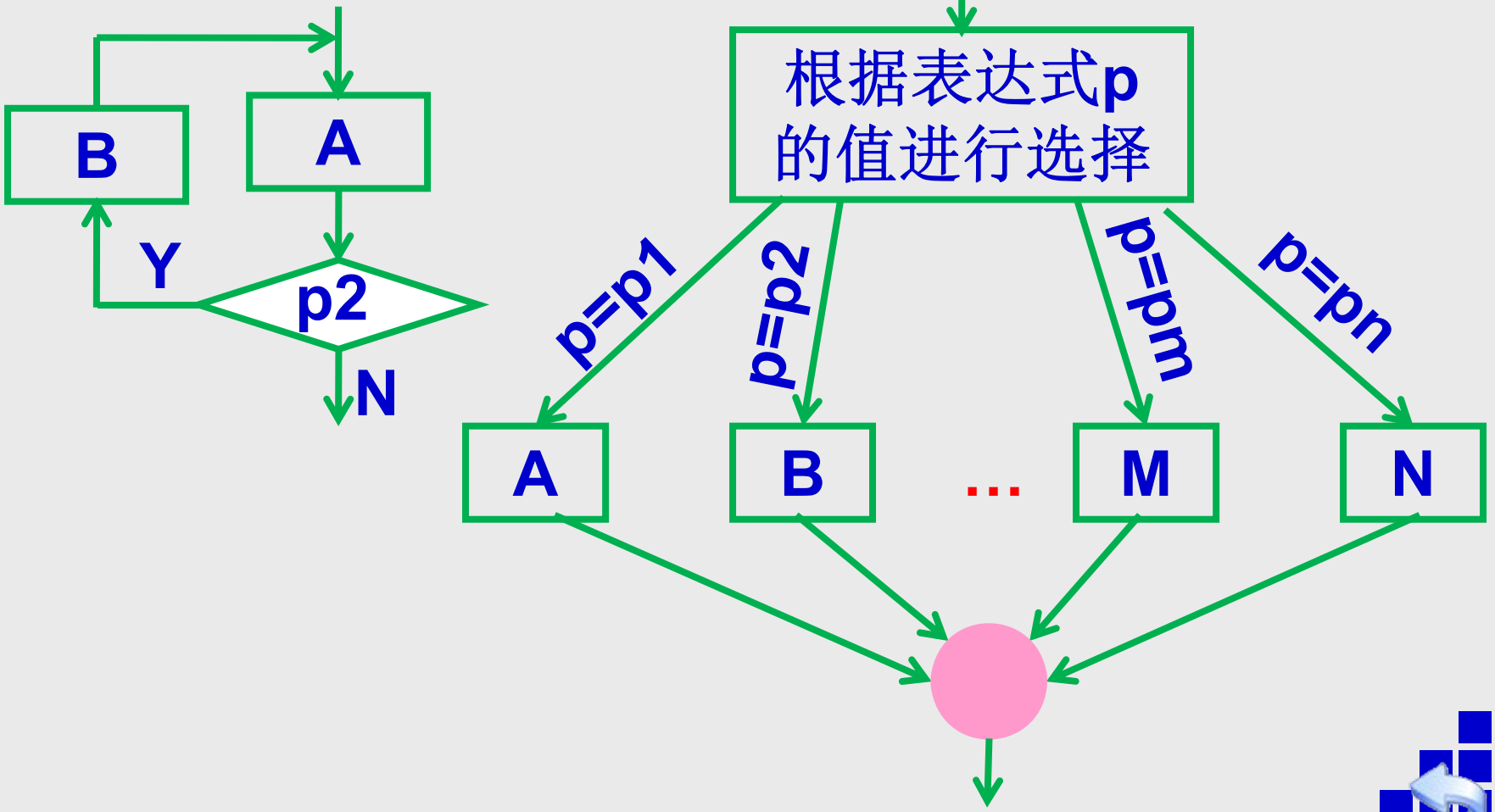
② 直到型循环结构





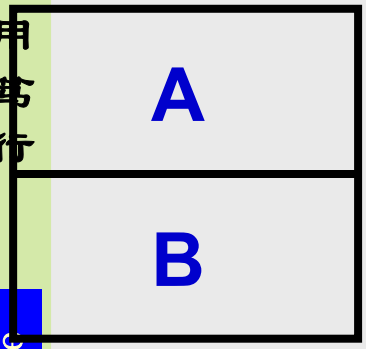
- 以上三种基本结构，有以下共同特点：
  - **(1)** 只有一个入口
  - **(2)** 只有一个出口
    - 一个判断框有两个出口
    - 一个选择结构只有一个出口
  - **(3)** 结构内的每一部分都有机会被执行到。也就是说，对每一个框来说，都应当有一条从入口到出口的路径通过它
  - **(4)** 结构内不存在“死循环”

- 由三种基本结构派生出来的结构：

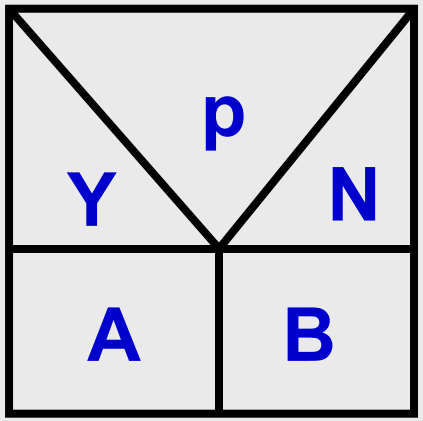


2.4.4 用N-S流程图表示算法

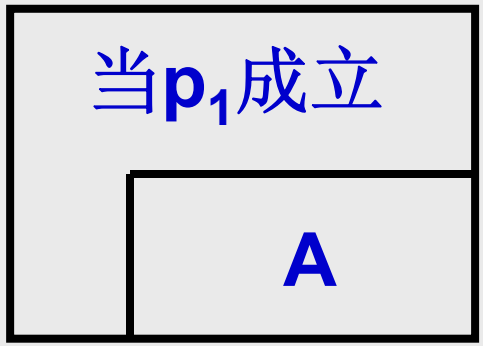
- N-S流程图用以下的流程图符号：



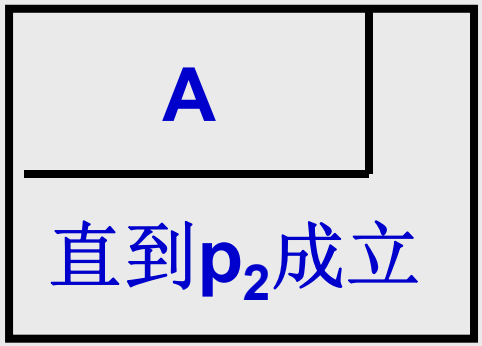
顺序结构



选择结构



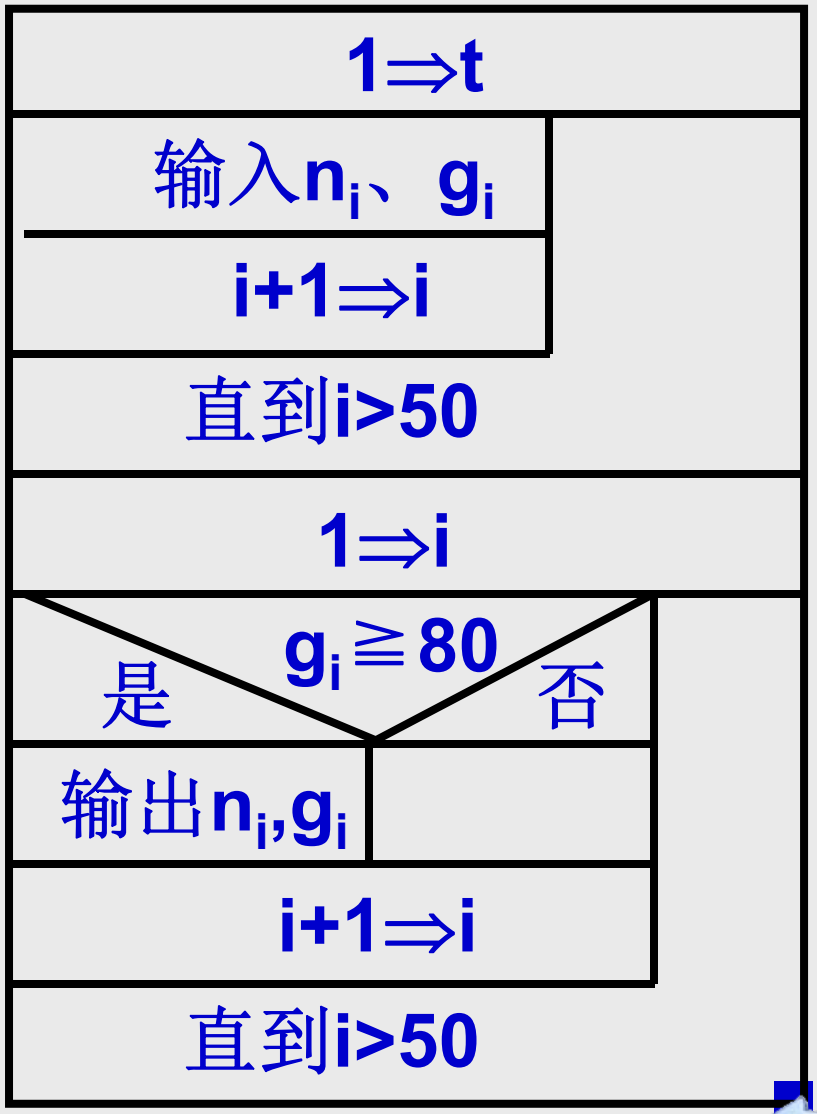
循环结构（当型）



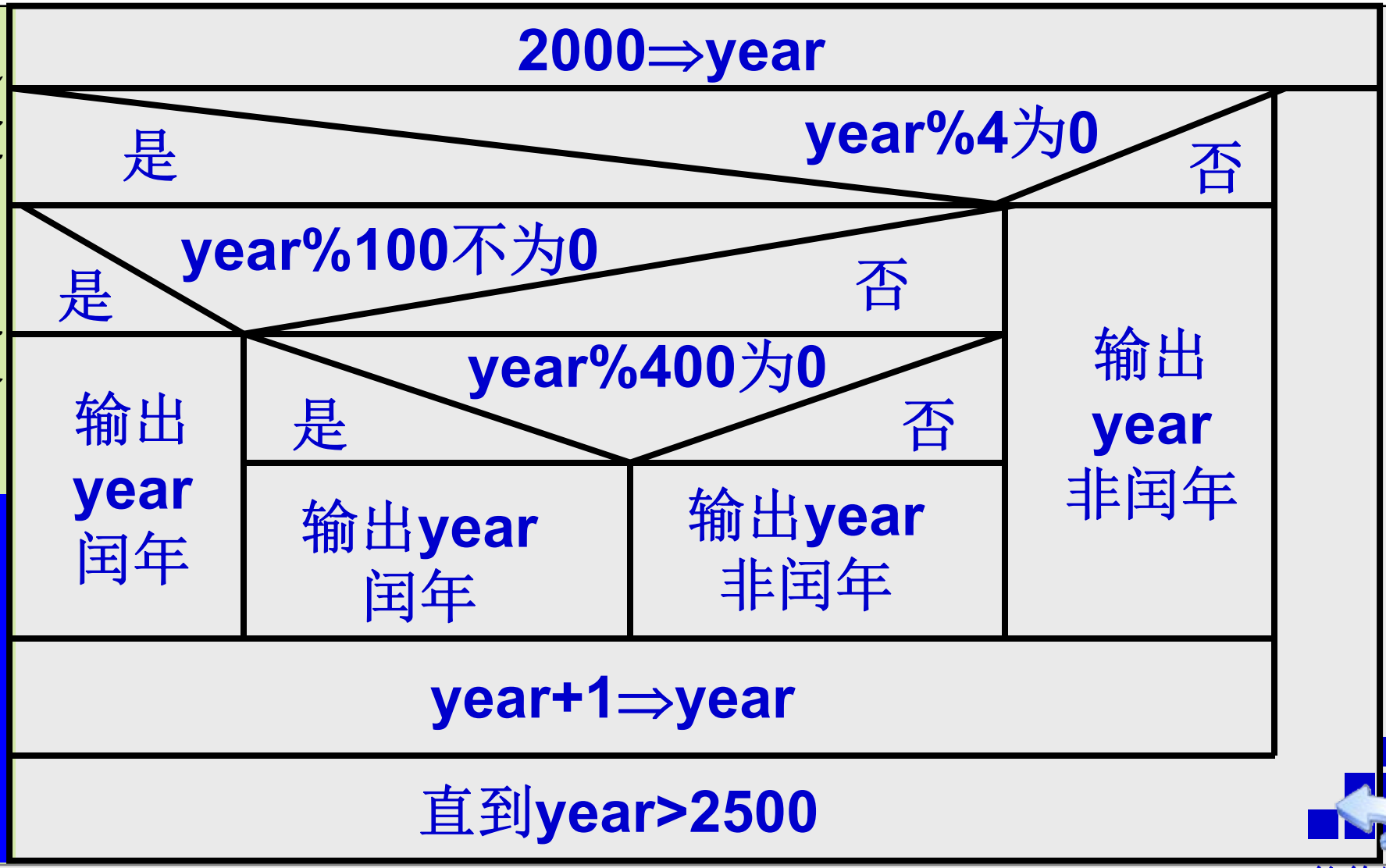
循环结构（直到型）

例2.11将例2.1的求5!算法用N-S图表示。

例2.12 将例2.2的算法用N-S图表示。将50名学生中成绩高于80分者的学号和成绩输出。



例2.13 将例2.3判定闰年的算法用N-S图表示



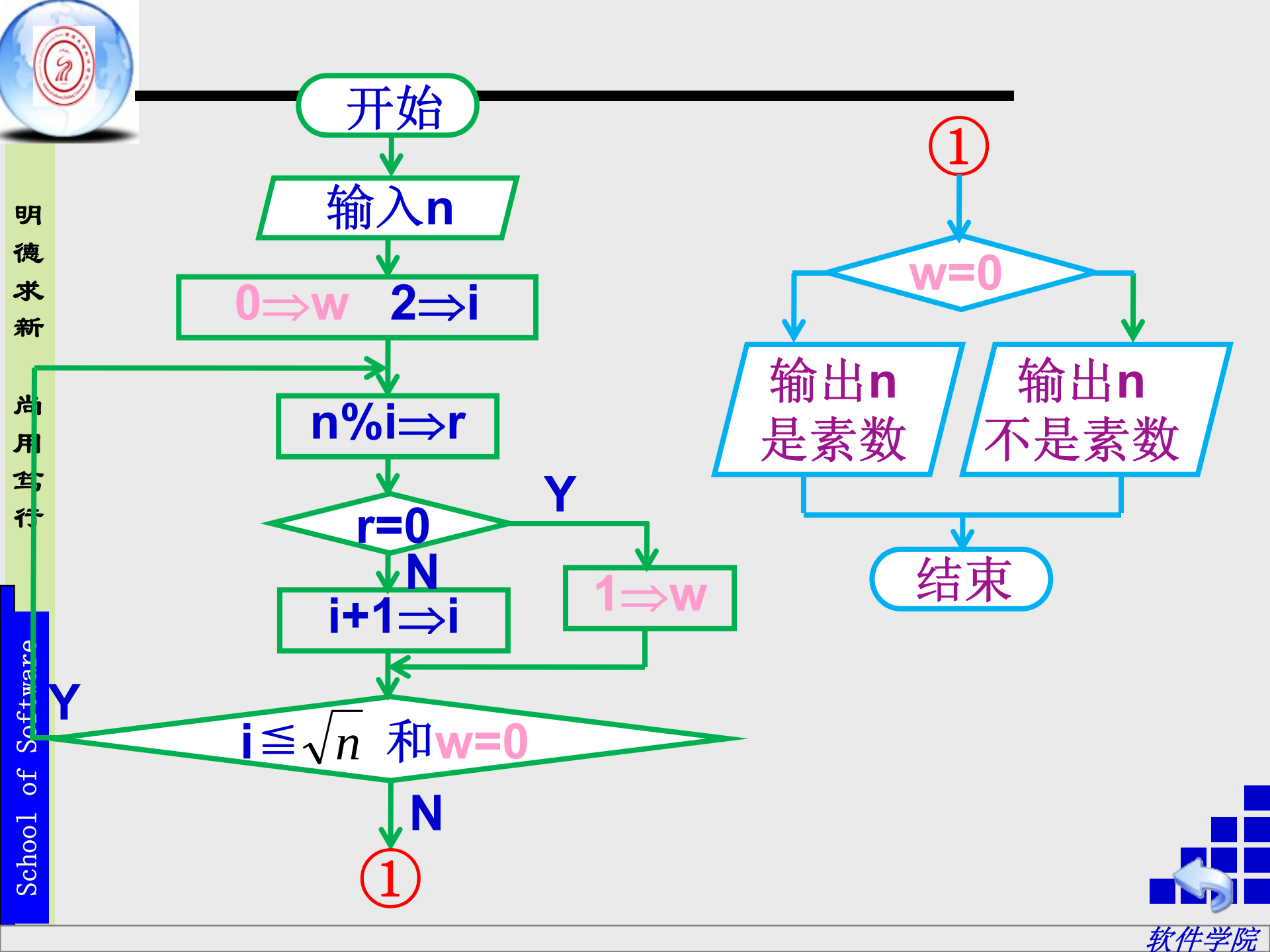
例2.14 将例2.4的算法用N-S图表示。求

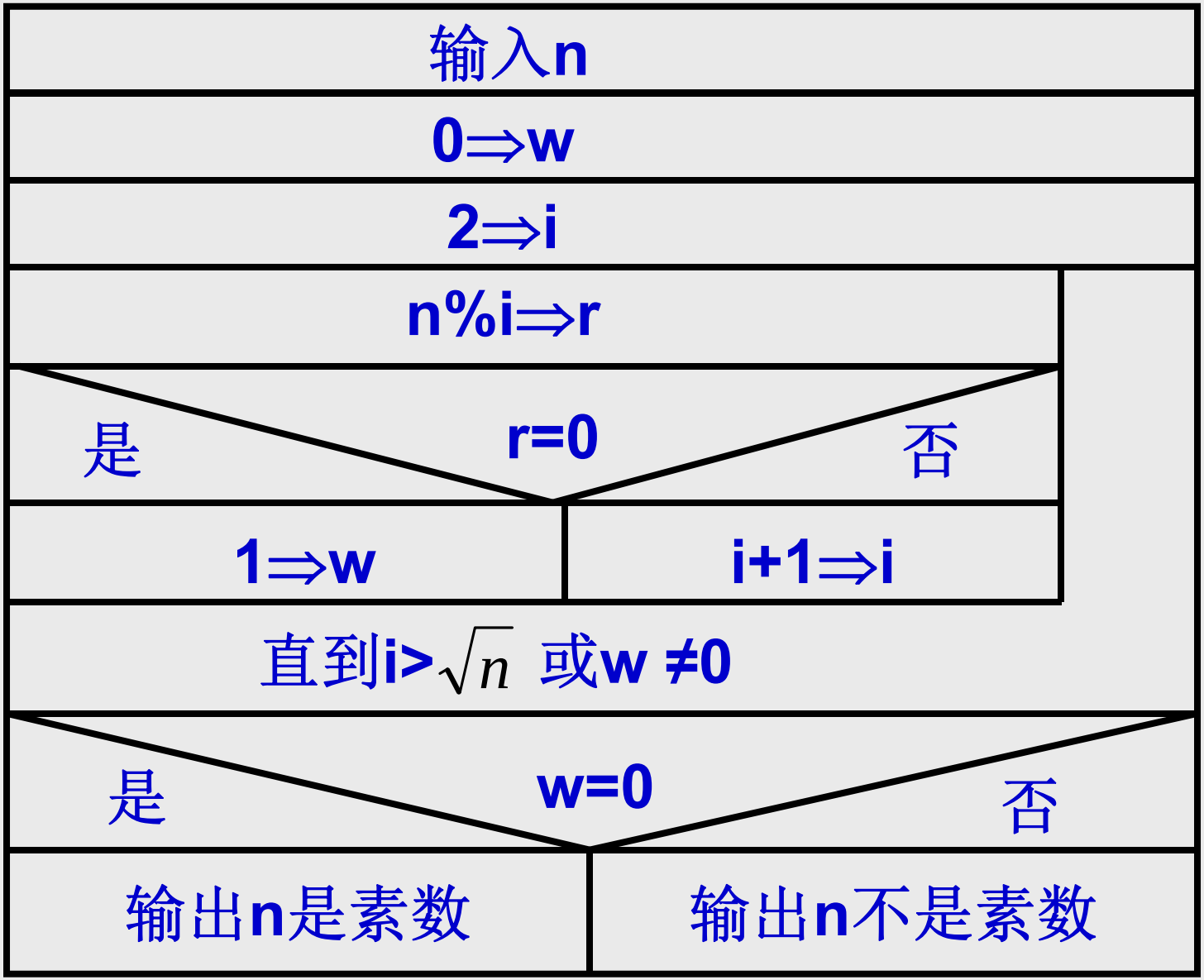
$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$$

1⇒sum
2⇒deno
1⇒sign
(-1)*sign⇒sign
sign*(1/deno)⇒term
sum+term⇒sum
deno+1⇒deno
直到deno>100
输出sum

例2.15 将例2.5判别素数的算法用N-S流程图表示。

- 例2.10的流程图不是由三种基本结构组成的
- 循环有两个出口，不符合基本结构的特点
- 无法直接用N-S流程图的三种基本结构的符号来表示
- 先作必要的变换


开始
输入n
0⇒w 2⇒i
n%i⇒r
r=0
Y
N
1⇒w
i+1⇒i
Y
i≦√n 和w=0
N
①
①
w=0
输出n
是素数
输出n
不是素数
结束


输入n
0⇒w
2⇒i
n%i⇒r
是
r=0
否
1⇒w
i+1⇒i
直到i>√n 或w ≠0
是
w=0
否
输出n是素数
输出n不是素数

- 一个结构化的算法是由一些基本结构顺序组成的
- 在基本结构之间不存在向前或向后的跳转，流程的转移只存在于一个基本结构范围之内
- 一个非结构化的算法可以用一个等价的结构化算法代替，其功能不变
- 如果一个算法不能分解为若干个基本结构，则它必然不是一个结构化的算法

2.4.5用伪代码表示算法

- 伪代码是用介于自然语言和计算机语言之间的文字和符号来描述算法
- 用伪代码写算法并无固定的、严格的语法规则，可以用英文，也可以中英文混用

例2.16 求5!。

```
begin                    (算法开始)
    1 ⇒ t
    2 ⇒ i
    while i≤5
    { t*i ⇒ t
      i+1 ⇒ i
    }
    print t
end                      (算法结束)
```

例2.17 求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$

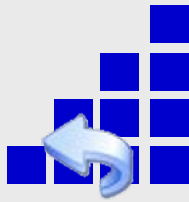
```
begin
   1 ⇒ sum
   2 ⇒ deno
   1 ⇒ sign
   while deno ≤ 100
   { (-1)*sign ⇒ sign
      sign*1/deno ⇒ term
      sum+term ⇒ sum
      deno+1 ⇒ deno
   }
   print sum
end
```

2.4.6用计算机语言表示算法

- 要完成一项工作，包括设计算法和实现算法两个部分。
- 设计算法的目的是为了实现算法。
- 不仅要考虑如何设计一个算法，也要考虑如何实现一个算法。

例2.18 将例2.16表示的算法（求5!）用C语言表示。


```
#include <stdio.h>
int main( )
{ int i,t;
   t=1;
   i=2;
   while(i<=5)
   {   t=t*i;
       i=i+1;
   }
   printf("%d\n",t);
   return 0;
}
```



例2.19 将例2.17表示的算法（求多项式

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$$

的值）用C语言表示。

```
#include <stdio.h>
int main( )
 {  int sign=1;
    double deno = 2.0,sum = 1.0, term;
    while (deno <= 100)
   {  sign = -sign;
      term = sign/deno;
      sum = sum+term;
      deno = deno+1;
    }
    printf ("%f\n",sum);
    return 0;
}
```

2.5结构化程序设计方法

- 结构化程序设计强调程序设计风格和程序结构的规范化，提倡清晰的结构。
- 结构化程序设计方法的基本思路是：把一个复杂问题的求解过程分阶段进行，每个阶段处理的问题都控制在人们容易理解和处理的范围内。

2.5结构化程序设计方法

- 采取以下方法保证得到结构化的程序：
  - （1） 自顶向下；
  - （2） 逐步细化；
  - （3） 模块化设计；
  - （4） 结构化编码。

第3章 最简单的C程序设计

3.1顺序程序设计举例

例3.1 有人用温度计测量出用华氏法表示的温度(如 F，今要求把它转换为以摄氏法表示的温度(如 C) 。

- 解题思路：找到二者间的转换公式

$$c = \frac{5}{9}(f - 32)$$

f代表华氏温度，c代表摄氏温度

3.1顺序程序设计举例

例3.1 有人用温度计测量出用华氏法表示的温度(如 F，今要求把它转换为以摄氏法表示的温度(如 C) 。

- 算法：

输入f的值
$c=\frac{5}{9}(f-32)$
输出c的值

N-S图

$\frac{5}{9}(f-32)$

明德求新 尚用笃行

School of Software

3.1顺序程序设计举例

```
#include <stdio.h>
int main ( )
{
   float f,c;
   f=64.0;
   c=(5.0/9)*(f-32);
   printf("f=%f\nc=%f\n",f,c);
   return 0;
}
```

定义f和c为单精度浮点型变量

指定f的值

计算c的值

输出f和c的值

```
f=64.000000
c=17.777778
```

软件学院

3.1顺序程序设计举例

例3.2 计算存款利息。有1000元，想存一年。有三种方法可选：

(1)活期，年利率为r1

(2)一年期定期，年利率为r2

(3)存两次半年定期，年利率为r3

请分别计算出一年后按三种方法所得到的本息和。

3.1顺序程序设计举例

- 解题思路：确定计算本息和的公式。

从数学知识可知：若存款额为**p0**，则：

活期存款一年后本息和为：

p1=p0(1+r1)

一年期定期存款，一年后本息和为：

p2=p0(1+r2)

两次半年定期存款，一年后本息和为：

$$p3 = p0(1+\frac{r3}{2})(1+\frac{r3}{2})$$

3.1顺序程序设计举例

- 算法：

输入p0,r1,r2,r3的值
计算p1=p0(1+r1)
计算p2=p0(1+r2)
计算$p3=p0(1+\frac{r3}{2})(1+\frac{r3}{2})$
输出p1,p2,p3

3.1顺序程序设计举例

```
#include <stdio.h>
int main ( )
 {float p0=1000, r1=0.0036,r2=0.0225,
                    r3=0.0198, p1, p2, p3;
   p1 = p0 * (1 +  r1);
   p2 = p0 * (1 +  r2);
   p3 = p0 * (1 +  r3/2) * (1 + r3/2);
   printf(" %f\n%f\n%f\n" ,p1, p2, p3);
   return 0;
 }
```

定义变量同时赋予初值

```
1003.599976
1022.500000
1019.898010
```


3.2 数据的表现形式及其运算

3.2.1 常量和变量

1.常量：在程序运行过程中，其值不能被改变的量

- 整型常量：如1000，12345，0，-345
- 实型常量
  - 十进制小数形式：如0.34　-56.79　0.0
  - 指数形式：如12.34e3 (代表12.34×10³)
- 字符常量：如′ ?′
  - 转义字符：如′ \n′
- 字符串常量：如" boy"
- 符号常量：#define PI 3.1416

3.2.1 常量和变量

2. 变量：在程序运行期间，变量的值是可以改变的

- 变量必须先定义，后使用
- 定义变量时指定该变量的名字和类型
- 变量名和变量值是两个不同的概念
- 变量名实际上是以一个名字代表的一个存储地址
- 从变量中取值，实际上是通过变量名找到相应的内存地址，从该存储单元中读取数据

3.2.1 常量和变量

3.常变量：const int a=3;

4.标识符：一个对象的名字

大小写字母是不同的字符

- C语言规定标识符只能由字母、数字和下划线3种字符组成，且第一个字符必须为字母或下划线
- 合法的标识符：如sum，average, _total, Class, day, BASIC, li_ling
- 不合法的标识符：M.D.John，￥123，＃33，3D64，a＞b

3.2.2 数据类型

- 所谓类型，就是对数据分配存储单元的安排，包括存储单元的长度(占多少字节)以及数据的存储形式
- 不同的类型分配不同的长度和存储形式

3.2.2 数据类型

C语言允许使用的数据类型：

- 基本类型
  - 整型类型
    - 基本整型
    - 短整型
    - 长整型
    - 双长整型
    - 字符型
    - 布尔型
  - 浮点类型
    - 单精度浮点型
    - 双精度浮点型
    - 复数浮点型

3.2.2 数据类型

C语言允许使用的数据类型：

- 基本类型
- 枚举类型
- 空类型
- 派生类型
  - 指针类型
  - 数组类型
  - 结构体类型
  - 共用体类型
  - 函数类型

算术类型

纯量类型

3.2.3 整型数据

1. 整型数据的分类

- 最基本的整型类型
  - 基本整型(int型)：占2个或4个字节
  - 短整型(short int)：VC++6.0中占2个字节
  - 长整型(long int)：VC++6.0中占4个字节
  - 双长整型(long long int)：C99新增的

3.2.3 整型数据

1. 整型数据的分类
2. 整型变量的符号属性
   - 整型变量的值的范围包括负数到正数
   - 可以将变量定义为“无符号”类型
   - 扩充的整形类型：

3.2.3 整型数据

扩充的整型类型：

- 有符号基本整型 [signed] int;
- 无符号基本整型 unsigned int;
- 有符号短整型 [signed] short [int];
- 无符号短整型 unsigned short [int];
- 有符号长整型 [signed] long [int];
- 无符号长整型 unsigned long [int]
- 有符号双长整型 [signed] long long [int];
- 无符号双长整型 unsigned long long [int]

3.2.4 字符型数据

- 字符是按其代码(整数)形式存储的
- C99把字符型数据作为整数类型的一种
- 字符型数据在使用上有自己的特点

3.2.4 字符型数据

1.字符与字符代码

大多数系统采用**ASCII**字符集

- 字母：**A ~Z，a ~z**
- 数字：**0～9**
- 专门符号：**29**个：**！” # & ‘ ( ) ***等
- 空格符：空格、水平制表符、换行等
- 不能显示的字符：空**(null)**字符(以‘**\0**’表示)、警告(以‘**\a**’表示)、退格(以‘**\b**’表示)、回车(以‘**\r**’表示)等

3.2.4 字符型数据

- 字符’1’和整数1是不同的概念：
  - 字符’1’只是代表一个形状为’1’的符号，在需要时按原样输出，在内存中以ASCII码形式存储，占1个字节

0 0 1 1 0 0 0 1

  - 整数1是以整数存储方式(二进制补码方式)存储的，占2个或4个字节

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

3.2.4 字符型数据

2.字符变量

- 用类型符char定义字符变量
  - char c = ’ ?’ ;

    系统把“?”的ASCII代码63赋给变量c
  - printf(” %d %c\n”,c,c);
  - 输出结果是:

    63 ?

3.2.5 浮点型数据

浮点型数据是用来表示具有小数点的实数

- **float型(单精度浮点型)**
  - 编译系统为**float**型变量分配**4**个字节
  - 数值以规范化的二进制数指数形式存放

    参见主教材图**3.11**

3.2.5 浮点型数据

浮点型数据是用来表示具有小数点的实数

- **float型(单精度浮点型)**
- **double型(双精度浮点型)**
  - 编译系统为**double**型变量分配**8**个字节
  - **15**位有效数字
- **long double(长双精度)型**

3.2.6 怎样确定常量的类型

- 字符常量：由单撇号括起来的单个字符或转义字符
- 整型常量：不带小数点的数值
  - 系统根据数值的大小确定int型还是long型等
- 浮点型常量：凡以小数形式或指数形式出现的实数
  - C编译系统把浮点型常量都按双精度处理
  - 分配8个字节

3.2.7 运算符和表达式

明德求新 尚用笃行

School of Software

1.基本的算术运算符：

+ ： 正号运算符(单目运算符)

- ： 负号运算符(单目运算符)

* ： 乘法运算符

/ ： 除法运算符

% ： 求余运算符

+ ： 加法运算符

- ： 减法运算符

3.2.7 运算符和表达式

说明

- 两个整数相除的结果为整数
  - 如5/3的结果值为1，舍去小数部分
  - 如果除数或被除数中有一个为负值，舍入方向不固定。例如，-5/3，有的系统中得到的结果为-1，在有的系统中则得到结果为-2
  - VC++采取“向零取整”的方法

    如5/3=1，-5/3=-1，取整后向零靠拢
- % 运算符要求参加运算的运算对象(即操作数)为整数，结果也是整数。如8%3，结果为2

3.2.7 运算符和表达式

明德求新 尚用笃行

2. 自增、自减运算符：

- 作用是使变量的值 1 或减 1
  - **++i，--i：在使用i之前，先使i的值加（减）1**
  - **i++，i--：在使用i之后，使i的值加（减）1**

3.2.7 运算符和表达式

3. 算术表达式和运算符的优先级与结合性：

- 用算术运算符和括号将运算对象（也称操作数）连接起来的、符合C语法规则的式子，称为C算术表达式
- 运算对象包括常量、变量、函数等
- C语言规定了运算符的优先级和结合性

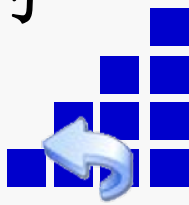
3.2.7 运算符和表达式

4.不同类型数据间的混合运算：

(1)+、-、*、/ 运算的两个数中有一个数为float或double型，结果是double型。系统将float型数据都先转换为double型，然后进行运算

(2) 如果int型与float或double型数据进行运算，先把int型和float型数据转换为double型，然后进行运算，结果是double型

(3)字符型数据与整型数据进行运算，就是把字符的ASCII代码与整型数据进行运算



3.2.7 运算符和表达式

例3.3 给定一个大写字母，要求用小写字母输出。

- 解题思路：
  - 关键是找到大、小写字母间的内在联系
  - 同一个字母，用小写表示的字符的ASCII代码比用大写表示的字符的ASCII代码大32

3.2.7 运算符和表达式

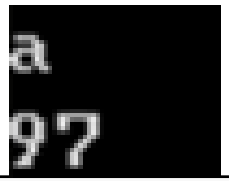
```
#include <stdio.h>
int main ( )
{
   char c1,c2;
   c1='A';
   c2=c1+32;
   printf("%c\n",c2);
   printf("%d\n",c2);
   return 0;
}
```

将字符'A'的ASCII代码65放到c1中

将65+32的结果放到c2中

用字符形式输出

用十进制形式输出



3.2.7 运算符和表达式

5. 强制类型转换运算符

- 强制类型转换运算符的一般形式为

  （类型名）（表达式）

  - **(double)a** （将 a 转换成**double**类型）
  - **(int) (x+y)** （将**x+y**的值转换成**int**型）
  - **(float)(5%3)**（将**5%3**的值转换成**float**型）
- 有两种类型转换
  - 系统自动进行的类型转换
  - 强制类型转换

3.2.7 运算符和表达式

6. C 运算符

(1) 算术运算符　　（+ - * / % ++ --）

(2) 关系运算符　　（＞ ＜ ＝＝ ＞＝ ＜＝ ！ ＝）

(3) 逻辑运算符　　（！ && ||）

(4) 位运算符　　（<< >> ~ | ^ &）

(5) 赋值运算符　　（＝及其扩展赋值运算符）

(6) 条件运算符　　（？ ： ）

3.2.7 运算符和表达式

6. C 运算符

(7) 逗号运算符 （，）

(8) 指针运算符 （*和&）

(9) 求字节数运算符 （sizeof）

(10) 强制类型转换运算符 （ (类型) ）

(11) 成员运算符 （.->）

(12) 下标运算符 （ [ ] ）

(13) 其他 （如函数调用运算符（））

3.3 C语句

3.3.1 C语句的作用和分类

3.3.2 最基本的语句----赋值语句

3.3.1 C语句的作用和分类

C语句分为以下5类：

(1) 控制语句： **if、switch、for、while、do…while、continue、break、return、goto等**

(2) 函数调用语句

(3) 表达式语句

(4) 空语句

(5) 复合语句

3.3.2 最基本的语句----赋值语句

- 在C程序中，最常用的语句是：
  - 赋值语句
  - 输入输出语句
- 其中最基本的是赋值语句

3.3.2 最基本的语句----赋值语句

例3.4 给出三角形的三边长，求三角形面积。

3.3.2 最基本的语句----赋值语句

- 解题思路：假设给定的三个边符合构成三角形的条件
- 关键是找到求三角形面积的公式
- 公式为：

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$

其中s=(a+b+c)/2

```
#include <stdio.h>
#include <math.h>
int main ( )
{ double a,b,c,s,area;
   a=3.67;
   b=5.43;
   c=6.21;
   s=(a+b+c)/2;
   area=sqrt(s*(s-a)*(s-b)*(s-c));
   printf("a=%f\tb=%f\t%f\n",a,b,c);
   printf("area=%f\n",area);
   return 0;
}
```

对边长a、b、c赋值

计算s

计算area

```
#include <stdio.h>
#include <math.h>
int main ( )
{ double a,b,c,s,area;
   a=3.67;
   b=5.43;
   c=6.21;
   s=(a+b+c)/2;
   area=sqrt(s*(s-a)*(s-b)*(s-c));
   printf("a=%f\tb=%f\t%f\n",a,b,c);
   printf("area=%f\n",area);
   return 0;
 }
```

调用数学函数加此行

数学函数，计算平方根

```
#include <stdio.h>
#include <math.h>
int main ( )
{ double a,b,c,s,area;
   a=3.67;
   b=5.43;
   c=6.21;
   s=(a+b+c)/2;
   area=sqrt(s*(s-a)*(s-b)*(s-c));
   printf("a=%f\tb=%f\t%f\n",a,b,c);
   printf("area=%f\n",area);
```

调用数学函数加此行

转义字符，使输出位置跳到下一个tab位置

```
a=3.670000      b=5.430000      6.210000
area=9.903431
```

明德求新 尚用笃行

School of Software

- 归纳总结：

1.赋值运算符

- “＝”是赋值运算符
- 作用是将一个数据赋给一个变量
- 也可以将一个表达式的值赋给一个变量

明德求新 尚用笃行

School of Software

- 归纳总结：

1.赋值运算符

2.复合的赋值运算符

  - 在赋值符“＝”之前加上其他运算符，可以构成复合的运算符
  - a ＋＝ 3　　等价于　a ＝ a ＋ 3

- 归纳总结：

1.赋值运算符

2.复合的赋值运算符

3.赋值表达式

  - 一般形式为：

    变量 赋值运算符 表达式

  - 对赋值表达式求解的过程：

    - 求赋值运算符右侧的“表达式”的值
    - 赋给赋值运算符左侧的变量

明德求新 尚用笃行

School of Software

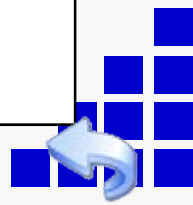
- 归纳总结：

1.赋值运算符

2.复合的赋值运算符

3.赋值表达式

  - 赋值表达式“a=3*5”的值为15，对表达式求解后，变量a的值和表达式的值都是15
  - “a=(b=5)”和“a=b=5”等价
  - “a=b”和“b=a”含义不同



明德求新 尚用笃行

归纳总结：

1.赋值运算符

2.复合的赋值运算符

3.赋值表达式

4.赋值过程中的类型转换

- 两侧类型一致时，直接赋值
- 两侧类型不一致，但都是算术类型时，自动将右侧的类型转换为左侧类型后赋值
- 定义变量时要防止数据溢出

归纳总结：

1.赋值运算符

2.复合的赋值运算符

3.赋值表达式

4.赋值过程中的类型转换

5.赋值表达式和赋值语句

- 赋值表达式的末尾没有分号，而赋值语句有分号
- 一个表达式可以包含赋值表达式，但决不能包含赋值语句

明德求新 尚用笃行

School of Software

- 归纳总结：

1.赋值运算符

2.复合的赋值运算符

3.赋值表达式

4.赋值过程中的类型转换

5.赋值表达式和赋值语句

6.变量赋初值

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

int a=3; 相当于 int a; a=3;

3.4 数据的输入输出

3.4.1 输入输出举例

例3.5 求锝$ax^2+bx+c=0$方程的根。a、b、c由键盘输入

设 $b^2-4ac>0$

3.4.1 输入输出举例

- 解题思路：首先要知道求方程式的根的方法。
- 由数学知识已知：如果 $b^2-4ac\geq 0$，则一元二次方程有两个实根：

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a} \qquad x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

若记 $p=\frac{-b}{2a}$ $q=\frac{\sqrt{b^2-4ac}}{2a}$

$$x_1=p+q$$
$$x_2=p-q$$

```
#include <stdio.h>
#include  <math.h>
int main ( )
{double a,b,c,disc,x1,x2,p,q;
  scanf("%lf%lf%lf",&a,&b,&c);
  disc=b*b-4*a*c;
  p=-b/(2.0*a);
  q=sqrt(disc)/(2.0*a);
  x1=p+q;   x2=p-q;
  printf("x1=%7.2f\nx2=%7.2f\n",x1,x2);
  return 0;
}
```

程序中调用数学函数sqrt

输入a,b,c的值

```
#include <stdio.h>
#include  <math.h>
int main ( )
{double a,b,c,disc,x1,x2,p,q;
  scanf("%lf%lf%lf",&a,&b,&c);
  disc=b*b-4*a*c;
  p=-b/(2.0*a);
  q=sqrt(disc)/(2.0*a);
  x1=p+q;   x2=p-q;
  printf("x1=%7.2f\nx2=%7.2f\n",x1,x2);
  return 0;
}
```

输入的是双精度型实数

```
#include <stdio.h>
#include  <math.h>
int main ( )
{double a,b,c,disc,x1,x2,p,q;
  scanf("%lf%lf%lf",&a,&b,&c);
  disc=b*b-4*a*c;
  p=-b/(2.0*a);
  q=sqrt(disc)/(2.0*a);
  x1=p+q;   x2=p-q;
  printf("x1=%7.2f\nx2=%7.2f\n",x1,x2);
  return 0;
}
```

```
1 3 2
```

自动转成实数后赋给a,b,c

要求输入3个实数

```
#include <stdio.h>
#include  <math.h>
int main ( )
{double a,b,c,disc,x1,x2,p,q;
  scanf("%lf%lf%lf",&a,&b,&c);
  disc=b*b-4*a*c;
  p=-b/(2.0*a);
  q=sqrt(disc)/(2.0*a);
  x1=p+q;   x2=p-q;
  printf("x1=%7.2f\nx2=%7.2f\n",x1,x2);
  return 0;
}
```

```
1 3 2
x1=   -1.00
x2=   -2.00
```

输出数据占7列，其中小数占2列

3.4.2 有关数据输入输出的概念

- 几乎每一个C程序都包含输入输出
- 输入输出是程序中最基本的操作之一

3.4.2 有关数据输入输出的概念

(1) 所谓输入输出是以计算机主机为主体而言的

- 从计算机向输出设备(如显示器、打印机等)输出数据称为输出
- 从输入设备（如键盘、磁盘、光盘、扫描仪等）向计算机输入数据称为输入

3.4.2 有关数据输入输出的概念

(2) C语言本身不提供输入输出语句

- 输入和输出操作是由**C**标准函数库中的函数来实现的
- **printf**和**scanf**不是C语言的关键字，而只是库函数的名字
- **putchar**、**getchar**、**puts**、**gets**

3.4.2 有关数据输入输出的概念

(3)在使用输入输出函数时，要在程序文件的开头用预编译指令

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

或

```
#include "stdio.h"
```

3.4.3 用printf函数输出数据

- 在C程序中用来实现输出和输入的，主要是printf函数和scanf函数
- 这两个函数是格式输入输出函数
- 用这两个函数时，必须指定格式

3.4.3 用printf函数输出数据

1.printf函数的一般格式

printf（格式控制，输出表列）

例如：

```
printf(" i=%d,c=%c\n" ,i,c);
```

格式声明

3.4.3 用printf函数输出数据

明德求新 尚用笃行

1.printf函数的一般格式

printf（格式控制，输出表列）

例如：

```
printf(" i=%d,c=%c\n" ,i,c);
```

普通字符

School of Software

3.4.3 用printf函数输出数据

1.printf函数的一般格式

printf（格式控制，输出表列）

例如：

```
printf(" i=%d,c=%c\n" ,i,c);
```

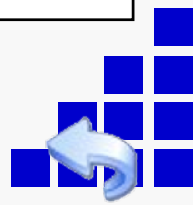
可以是常量、变量或表达式

3.4.3 用printf函数输出数据

2. 常用格式字符

- d 格式符。用来输出一个有符号的十进制整数
  - 可以在格式声明中指定输出数据的域宽

    printf(" %5d%5d\n",12,-345);
  - **%d输出int型数据**
  - **%ld输出long型数据**



3.4.3 用printf函数输出数据

明德求新 尚用笃行

School of Software

2. 常用格式字符

- c 格式符。用来输出一个字符

```
char ch=’a’;潼
printf(”%c”,ch);  或
printf(”%5c”,ch);
```

输出字符：a

3.4.3 用printf函数输出数据

2. 常用格式字符

- s 格式符。用来输出一个字符串

  printf（" %s"，" CHINA"）；潼

  输出字符串：CHINA

3.4.3 用printf函数输出数据

2. 常用格式字符

- **f格式符。用来输出实数，以小数形式输出**

  ①不指定数据宽度和小数位数，用**%f**

例**3.6** 用**%f**输出实数，只能得到 6 位小数。

```
double a=1.0;
printf(" %f\n" ,a/3);
```

灞

```
0.333333
```

3.4.3 用printf函数输出数据

2. 常用格式字符

- f格式符。用来输出实数，以小数形式输出

② 指定数据宽度和小数位数。用%m.nf

```
printf("%20.15f\n",1/3);
```


```
printf("%.0f\n" ,10000/3.0);
```

3.4.3 用printf函数输出数据

2. 常用格式字符

- f格式符。用来输出实数，以小数形式输出

② 指定数据宽度和小数位数。用%m.nf

```
float a;
a=10000/3.0;
printf("%f\n",a);
```

```
3333.333333
```

3.4.3 用printf函数输出数据

2. 常用格式字符

- **f**格式符。用来输出实数，以小数形式输出

  ③ 输出的数据向左对齐，用**%-m.nf**

3.4.3 用printf函数输出数据

2. 常用格式字符

- **f**格式符。用来输出实数，以小数形式输出
  - **float**型数据只能保证**6**位有效数字
  - **double**型数据能保证**15**位有效数字
  - 计算机输出的数字不都是绝对精确有效的

3.4.3 用printf函数输出数据

明德求新 尚用笃行

School of Software

2. 常用格式字符

- e格式符。指定以指数形式输出实数
  - %e，VC++给出小数位数为 6 位

    指数部分占5列

    小数点前必须有而且只有1位非零数字

printf(" %e",123.456);

输出：**1.234560 e+002**

3.4.3 用printf函数输出数据

2. 常用格式字符

- e格式符。指定以指数形式输出实数
  - %m.ne

    printf(" %13.2e",123.456);

    输出： 1.23e+002 (前面有4个空格)

3.4.4 用scanf函数输入数据

1. scanf 函数的一般形式

scanf（格式控制，地址表列）

含义同printf函数

3.4.4 用scanf函数输入数据

1. scanf 函数的一般形式舊

scanf（格式控制，地址表列）

可以是变量的地址，或字符串的首地址

3.4.4 用scanf函数输入数据

2. scanf函数中的格式声明

- 与printf函数中的格式声明相似
- 以%开始，以一个格式字符结束，中间可以插入附加的字符

```
scanf("a=%f,b=%f,c=%f",&a,&b,&c);
```

3.4.4 用scanf函数输入数据

3.使用scanf函数时应注意的问题甚

scanf(" %f%f%f" ,a,b,c); 错

scanf(" %f%f%f" ,&a,&b,&c); 对

对于

scanf("a=%f,b=%f,c=%f",&a,&b,&c);

1 3 2↙ 错

a=1,b=3,c=2↙ 对

a=1 b=3 c=2↙ 错

3.4.4 用scanf函数输入数据

3.使用scanf函数时应注意的问题囍

对于scanf(" %c%c%c" ,&c1,&c2,&c3);

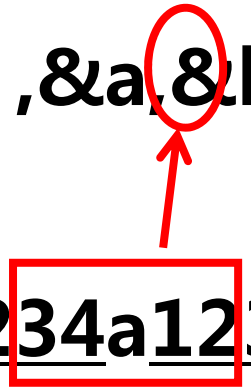
abc↙ 对

a b c↙ 错

对于scanf(" %d%c%f" ,&a,&b,&c);

若输入

1234a123o.26↙



3.4.4 用scanf函数输入数据

3.使用scanf函数时应注意的问题囍

对于scanf(" %c%c%c" ,&c1,&c2,&c3);

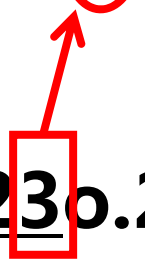
abc↙ 对

a b c↙ 错

对于scanf(" %d%c%f" ,&a,&b,&c);

若输入

1234a123b.26↙



3.4.4 用scanf函数输入数据

3.使用scanf函数时应注意的问题囍

对于scanf(" %c%c%c" ,&c1,&c2,&c3);

abc↙ 对

a b c↙ 错

对于scanf(" %d%c%f" ,&a,&b,&c);

若输入

1234a123b.26↙

3.4.5 字符数据的输入输出

1.用putchar函数输出一个字符

- 从计算机向显示器输出一个字符
- **putchar函数的一般形式为：**

  putchar(c) 囍

3.4.5 字符数据的输入输出

例3.8 先后输出BOY三个字符。

- 解题思路：
  - 定义3个字符变量，分别赋以初值B、O、Y
  - 用putchar函数输出这3个字符变量的值 囍

3.4.5 字符数据的输入输出

```
#include <stdio.h>
int main ( )
{
    char a='B',b='O',c='Y';
    putchar(a);          向显示器输出字符B
    putchar(b);
    putchar(c);
    putchar ('\n');      向显示器输出换行符
    return 0;
 }
```

```
BOY
```

3.4.5 字符数据的输入输出

```
#include <stdio.h>
int main ( )
{
    char a='B',b='O',c='Y';
    putchar(a);
    putchar(b);
    putchar(c);
    putchar ('\n');
    return 0;
 }
```

改为int a=66,b=79,c=89;

```
BOY
```

3.4.5 字符数据的输入输出

putchar（’\101’）　(输出字符A)潼

putchar（’\’’）　(输出单撇号字符’)

3.4.5 字符数据的输入输出

2. 用getchar函数输入一个字符

- 向计算机输入一个字符
- getchar函数的一般形式为：

```
getchar( )
```

3.4.5 字符数据的输入输出

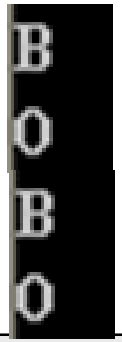
例3.9 从键盘输入BOY三个字符，然后把它们输出到屏幕。

- 解题思路：
  - 用3个getchar函数先后从键盘向计算机输入BOY三个字符
  - 用putchar函数输出

3.4.5 字符数据的输入输出

```
#include <stdio.h>
int  main ( )
{ char a,b,c;
   a=getchar();
   b=getchar();
   c=getchar();
   putchar(a); putchar(b); putchar(c);
   putchar('\n');
   return 0;
 }
```

输入一个字符，送给变量a



3.4.5 字符数据的输入输出

```
#include <stdio.h>
int  main ( )
{ char a,b,c;
   a=getchar();        putchar(getchar());
   b=getchar();
   c=getchar();
   putchar(a); putchar(b); putchar(c);
   putchar('\n');
   return 0;
 }
```

3.4.5 字符数据的输入输出

```
#include <stdio.h>
int  main ( )
{ char a,b,c;
                          putchar(getchar());
  b=getchar();            putchar(getchar());
  c=getchar();
                    putchar(b); putchar(c);
  putchar('\n');
  return 0;
 }
```

3.4.5 字符数据的输入输出

```
#include <stdio.h>
int  main ( )
{ char a,b,c;
                          putchar(getchar());
                          putchar(getchar());
  c=getchar();            putchar(getchar());
                                    putchar(c);
  putchar('\n');
  return 0;
 }
```

3.4.5 字符数据的输入输出

```
#include <stdio.h>
int  main ( )
{ char a,b,c;
                    putchar(getchar());
                    putchar(getchar());
                    putchar(getchar());

  putchar('\n');
  return 0;
 }
```

```
BOY
BOY
```

第4章 选择结构程序设计

4.1 选择结构和条件判断

- 在现实生活中，需要进行判断和选择的情况是很多的
  - 如果你在家，我去拜访你
  - 如果考试不及格，要补考
  - 如果遇到红灯，要停车等待
  - 周末我们去郊游
  - 70岁以上的老年人，入公园免票

4.1 选择结构和条件判断

- 在现实生活中，需要进行判断和选择的情况是很多的
- 处理这些问题，关键在于进行条件判断
- 由于程序处理问题的需要，在大多数程序中都会包含选择结构，需要在进行下一个操作之前先进行条件判断

4.1 选择结构和条件判断

- C语言有两种选择语句：

  (1)**if**语句，实现两个分支的选择结构

  (2)**switch**语句，实现多分支的选择结构

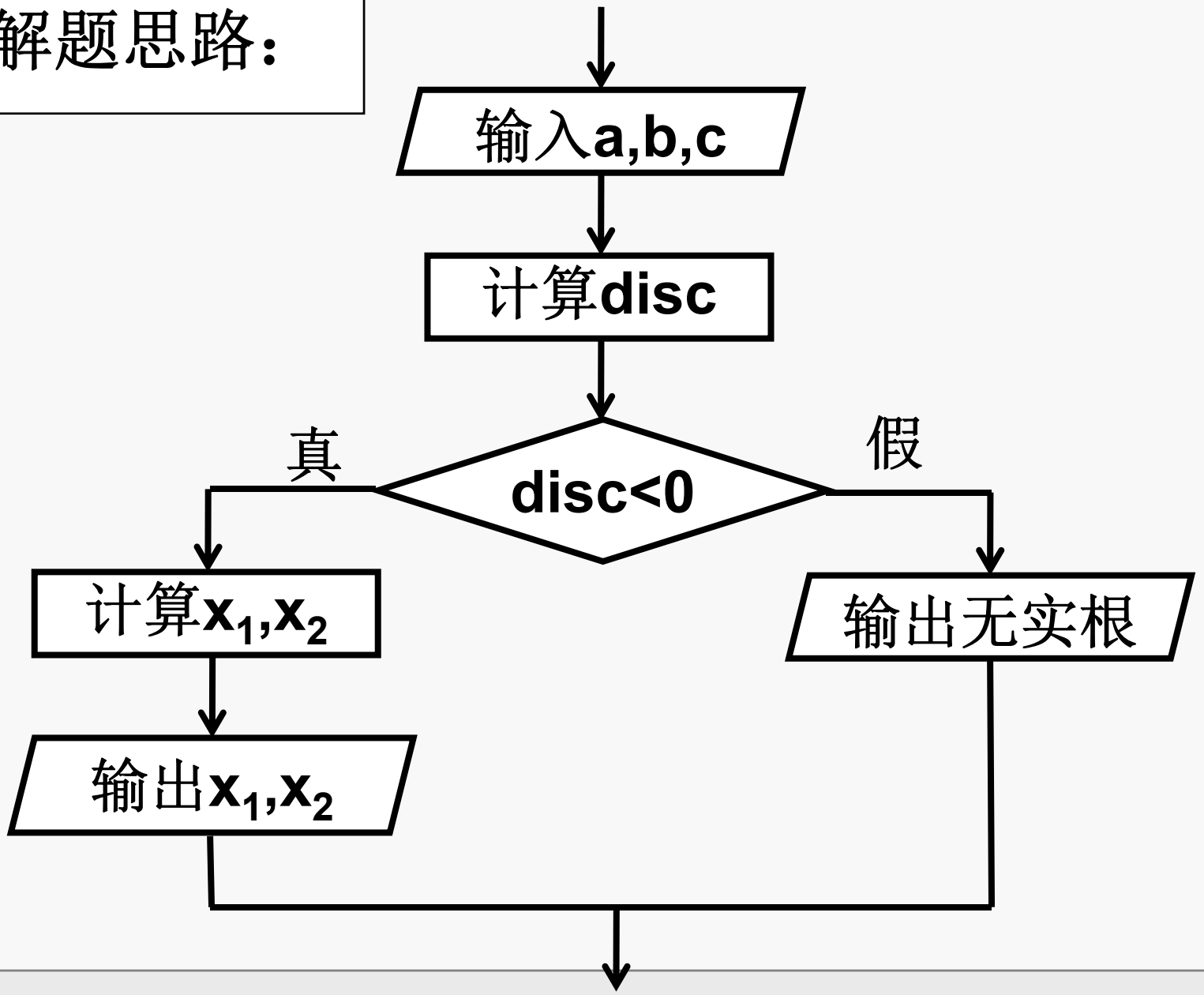
4.1 选择结构和条件判断

例4.1 在例3.5的基础上对程序进行改进。

题目要求是求 $ax^2+bx+c=0$ 方程的根。

由键盘输入a,b,c。假设a,b,c的值任意，并不保证 $b^2-4ac\geq 0$，需要在程序中进行判别，如果 $b^2-4ac\geq 0$，就计算并输出方程的两个实根，否则就输出“方程无实根”的信息。

解题思路：




```
#include <stdio.h>
#include  <math.h>
int main ( )
 {
    double a,b,c,disc,x1,x2,p,q;
    scanf("%lf%lf%lf",&a,&b,&c);
    disc=b*b-4*a*c;
```

6 3 1

计算b^2-4ac，disc的值变为-15

```
if (disc<0)          -15<0为真
   printf( “has not real roots\n” );
else
{  p=-b/(2.0*a);
    q=sqrt(disc)/(2.0*a);
    x1=p+q;
    x2=p-q;
    printf( “real roots:\nx1=%7.2f\n
                    x2=%7.2f\n” ,x1,x2);
 }
 return 0;
}
```

```
has not real roots
```

```
#include <stdio.h>
#include  <math.h>
int main ( )
 {
    double a,b,c,disc,x1,x2,p,q;
    scanf("%lf%lf%lf",&a,&b,&c);
    disc=b*b-4*a*c;
```

```
2 4 1
```

计算b^2-4ac，disc的值变为8

```
if (disc<0)          8<0为假
   printf( “has not real roots\n” );
else
{  p=-b/(2.0*a);          p的值变为-1
   q=sqrt(disc)/(2.0*a);          q的值变为0.71
   x1=p+q;          x1的值变为-0.29
   x2=p-q;          x2的值变为-1.71
   printf( “real roots:\nx1=%7.2f\n
                  x2=%7.2f\n” ,x1,x2);
}
return 0;
}
```

```
real roots:
x1=   -0.29
x2=   -1.71
```

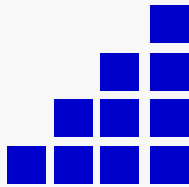
```
 if (disc<0)
   printf( “has not real roots\n” );
else
{  p=-b/(2.0*a);
   q=sqrt(disc)/(2.0*a);
   x1=p+q;
   x2=p-q;
   printf( “real roots:\nx1=%7.2f\n
                  x2=%7.2f\n” ,x1,x2);
 }
  return 0;
}
```

选择结构，用if语句实现的

```
    if (disc<0)
      printf( “has not real roots\n” );
    else
    {  p=-b/(2.0*a);
       q=sqrt(disc)/(2.0*a);
       x1=p+q;
       x2=p-q;
       printf( “real roots:\nx1=%7.2f\n
                        x2=%7.2f\n” ,x1,x2);
    }
    return 0;
}
```

复合语句

4.2 用if语句实现选择结构

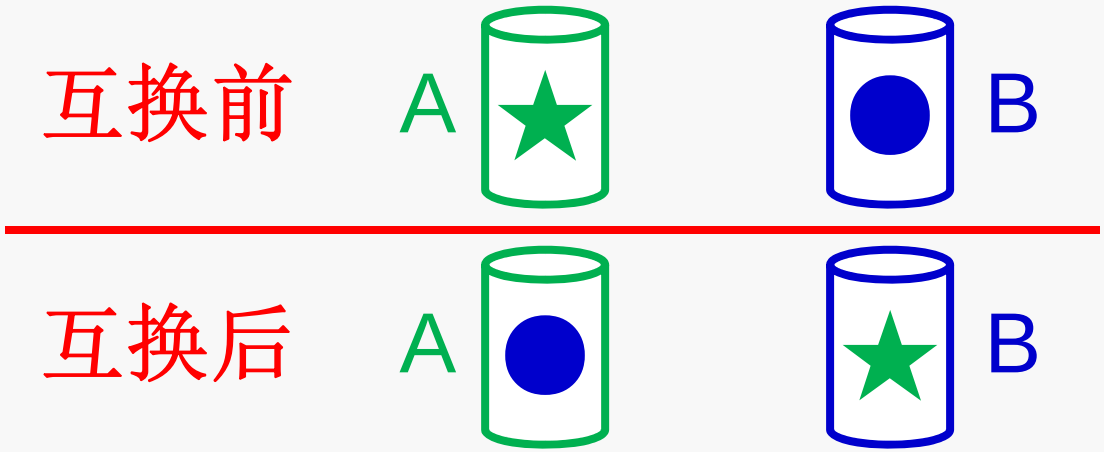


4.2.1 用if语句处理选择结构举例

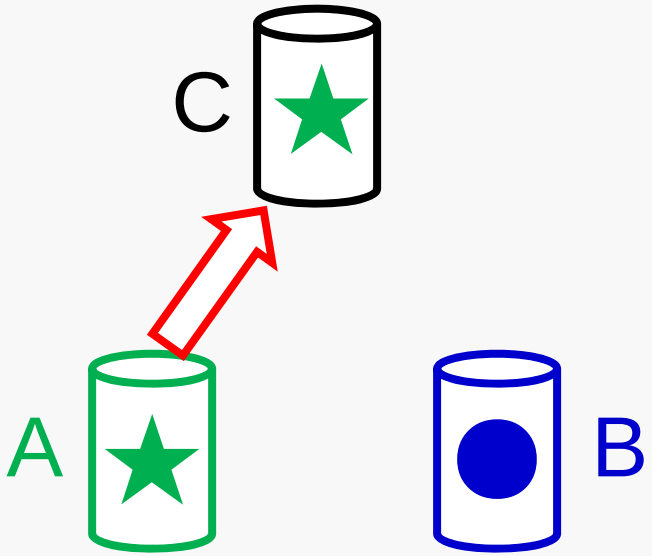
例4.2 输入两个实数，按代数值由小到大的顺序输出这两个数。

4.2.1 用if语句处理选择结构举例

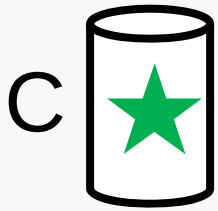
- 解题思路：
  - 只需要做一次比较，然后进行一次交换即可
  - 用if语句实现条件判断
  - 关键是怎样实现两个变量值的互换

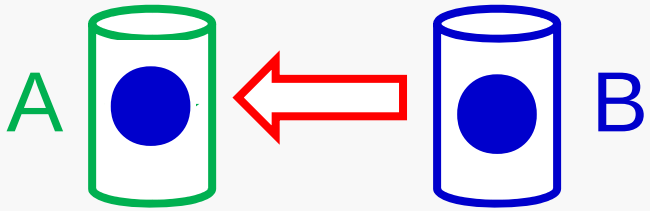


4.2.1 用if语句处理选择结构举例

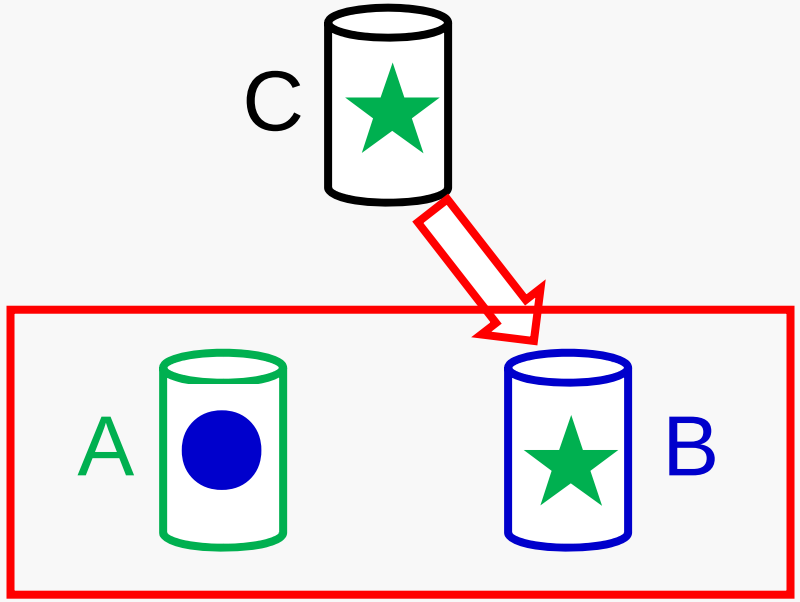


4.2.1 用if语句处理选择结构举例





4.2.1 用if语句处理选择结构举例



```
#include <stdio.h>
int main()
{ float a,b,t;
   scanf("%f,%f",&a,&b);
   if(a>b)
    {  t=a;
       a=b;
       b=t;
     }
   printf("%5.2f,%5.2f\n",a,b);
   return 0;
}
```

如果a>b

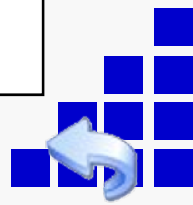
将a和b的值互换

```
3.6,-3.2
-3.20, 3.60
```


```
#include <stdio.h>
int main()
{ float a,b,t;
   scanf("%f,%f",&a,&b);
   if(a>b)
    {  t=a;
       a=b;
       b=t;
    }
   printf("%5.2f,%5.2f\n",a,b);
   return 0;
}
```

选择结构，用if语句实现的

例4.3 输入3个数a，b，c，要求按由小到大的顺序输出。



- 解题思路：可以先用伪代码写出算法：
  - if a>b，a和b对换　（a是a、b中的小者）
  - if a>c，a和c对换　（a是三者中最小者）
  - if b>c，b和c对换　（b是三者中次小者）
  - 顺序输出a，b，c

```
#include <stdio.h>
int main()
{ float a,b,c,t;
   scanf("%f,%f,%f",&a,&b,&c);
   if(a>b)
   {  t=a; a=b; b=t;  }
   if(a>c)
   {  t=a; a=c; c=t;  }
   if(b>c)
   {  t=b; b=c; c=t;  }
   printf("%5.2f,%5.2f,%5.2f\n",a,b,c);
   return 0;
}
```

如果 a>b，将a和b对换

a是a、b中的小者

```
#include <stdio.h>
int main()
{ float a,b,c,t;
   scanf("%f,%f,%f",&a,&b,&c);
   if(a>b)
   {  t=a; a=b; b=t;  }
   if(a>c)
   {  t=a; a=c; c=t;  }
   if(b>c)
   {  t=b; b=c; c=t;  }
   printf("%5.2f,%5.2f,%5.2f\n",a,b,c);
   return 0;
}
```

如果 a>c，将a和c对换

a是三者中的小者

```
#include <stdio.h>
int main()
{ float a,b,c,t;
   scanf("%f,%f,%f",&a,&b,&c);
   if(a>b)
   {  t=a; a=b; b=t;  }
   if(a>c)
   {  t=a; a=c; c=t;  }
   if(b>c)
   {  t=b; b=c; c=t;  }
   printf("%5.2f,%5.2f,%5.2f\n",a,b,c);
   return 0;
}
```

如果 b>c，将b和c对换

b是三者中的次小者

```
3,7,1
 1.00, 3.00, 7.00
```

4.2.2 if语句的一般形式

关系表达式
逻辑表达式
数值表达式

```
if (表达式) 语句1
[ else  语句2  ]
```

方括号内的部分为可选的

4.2.2 if语句的一般形式

```
if (表达式) 语句1
[ else  语句2 ]
```

简单的语句
复合语句
另一个if语句等

最常用的3种if语句形式：

1. if (表达式) 语句1　(没有else子句)
2. if (表达式) 语句1
   else 语句2　(有else子句)
3. if（表达式１）　语句１
   else if（表达式２）语句２
   else if（表达式３）语句３
   ⋮
   else if（表达式m）语句m
   else 语句m+1

(在else部分又嵌套了多层的if语句)

```
if(number > 500)  cost = 0.15;
else if (number > 300)  cost = 0.10;
else if (number > 100)  cost = 0.075;
else if (number > 50)   cost = 0.05;
else    cost=0;
```

等价于

```
if (number > 500)  cost = 0.15;
else
  if (number > 300)  cost = 0.10;
  else
    if (number > 100)  cost = 0.075;
    else
      if (number > 50)  cost = 0.05;
      else   cost = 0;
```

分号不能丢

- 说明：

(1)整个if语句可写在多行上，也可写在一行上
但都是一个整体，属于同一个语句

(2) “语句1”…“语句m”是if中的内嵌语句
内嵌语句也可以是一个if语句

(3) “语句1”…“语句m”可以是简单的语句，也可以是复合语句

4.3关系运算符和关系表达式

4.3.1关系运算符及其优先次序

4.3.2 关系表达式

4.3.1关系运算符及其优先次序

- 关系运算符：

  用来对两个数值进行比较的比较运算符

- C 语言提供 6 种关系运算符：

① < (小于) ② <= (小于或等于)

③ > (大于) ④ >= (大于或等于)

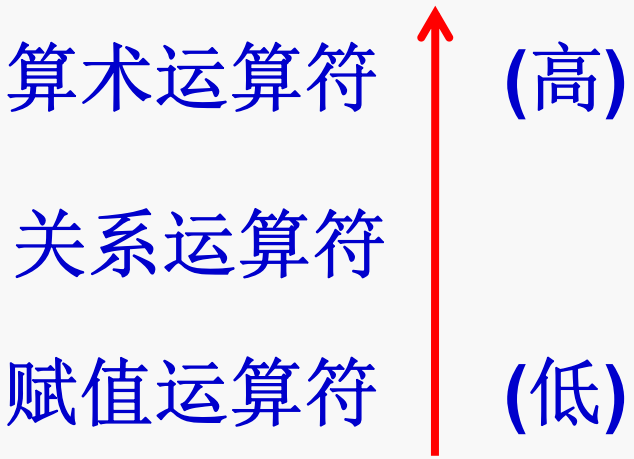
优先级相同 (高)

⑤ == (等于) ⑥ != (不等于)

优先级相同 (低)

4.3.1关系运算符及其优先次序

- 关系、算术、赋值运算符的优先级



4.3.1关系运算符及其优先次序

c>a+b	等效于	c>(a+b)
a>b==c	等效于	(a>b)==c
a==b<c	等效于	a==(b<c)
a=b>c	等效于	a=(b>c)

4.3.2 关系表达式

- 关系表达式
  - 用关系运算符将两个数值或数值表达式连接起来的式子
  - 关系表达式的值是一个逻辑值，即“真”或“假”
  - 在C的逻辑运算中，以“1”代表“真”，以“0”代表“假”

4.4 逻辑运算符和逻辑表达式

4.4.1 逻辑运算符及其优先次序

- 3种逻辑运算符：

  &&（逻辑与） ||（逻辑或） !（逻辑非）

- &&和||是双目(元)运算符
- !是一目(元)运算符
- 逻辑表达式
  - 用逻辑运算符将关系表达式或其他逻辑量连接起来的式子

4.4.1 逻辑运算符及其优先次序

- 判断年龄在13至17岁之内？

  age>=13 && age<=17

- 判断年龄小于12或大于65？

  age<12 || age>65

4.4.1 逻辑运算符及其优先次序

- 逻辑运算的真值表

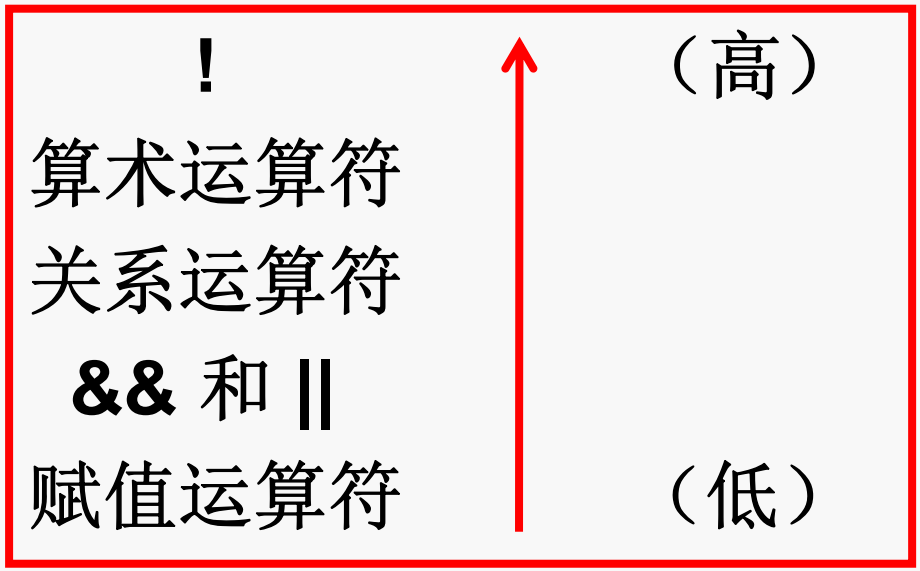
a	b	! a	! b	a && b	a || b
真	真	假	假	真	真
真	假	假	真	假	真
假	真	真	假	假	真
假	假	真	真	假	假

4.4.1 逻辑运算符及其优先次序

- 逻辑运算符的优先次序

  ! → && → ||　　（!为三者中最高）

- 与其他运算符的优先次序



4.4.2 逻辑表达式

- 逻辑表达式的值应该是逻辑量“真”或“假”
- 编译系统在表示逻辑运算结果时
  - 以数值1代表“真”，以0代表“假”
- 但在判断一个量是否为“真”时
  - 以0代表“假”，以非0代表“真”
- 注意：将一个非零的数值认作为“真”

4.4.2 逻辑表达式

(1) 若a=4，则!a的值为0

(2) 若a=4，b=5，则a && b的值为1

(3) a和b值分别为4和5，则a||b的值为1

(4) a和b值分别为4和5，则!a||b的值为1

(5) 4 && 0 || 2的值为1

4.4.2 逻辑表达式

- 修改后的逻辑运算真值表

a	b	! a	! b	a && b	a || b
非0	非0	0	0	1	1
非0	0	0	1	0	1
假	非0	1	0	0	1
假	0	1	1	0	0

4.4.2 逻辑表达式

- 判别某一年是否闰年，用逻辑表达式表示
- 闰年的条件是符合下面二者之一：

  ①能被4整除，但不能被100整除，如2008

  ②能被400整除，如2000

  - (year % 4 == 0 && year 100 !=0)

    || year % 400 == 0

  - 如果表达式值为1，则闰年；否则为非闰年

4.4.3 逻辑型变量

- 这是C99所增加的一种数据类型
- 可以将关系运算和逻辑运算的结果存到一个逻辑型变量中，以便于分析和运算

4.5 条件运算符和条件表达式

- 有一种**if**语句，当被判别的表达式的值为“真”或“假” 时，都执行一个赋值语句且向同一个变量赋值
- 如：

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

条件运算符

```
max = (a > b) ? a : b;
```


4.5 条件运算符和条件表达式

- 有一种**if**语句，当被判别的表达式的值为“真”或“假” 时，都执行一个赋值语句且向同一个变量赋值
- 如：

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

```
max = (a > b) ? a : b;
```

条件表达式

4.5 条件运算符和条件表达式

- 条件表达式的一般形式为

  表达式1？表达式2：表达式3

4.5 条件运算符和条件表达式

- 条件运算符的执行顺序：
  - 求解表达式1
  - 若为非0（真）则求解表达式2，此时表达式2的值就作为整个条件表达式的值
  - 若表达式1的值为0（假），则求解表达式3，表达式3的值就是整个条件表达式的值

4.5 条件运算符和条件表达式

- 条件运算符优先于赋值运算符
- 条件运算符的结合方向为“自右至左”
- 以下为合法的使用方法：
  - a>b ? (max=a):(max=b);
  - a>b ? printf(“%d” ,a): printf(“%d” ,b);

4.5 条件运算符和条件表达式

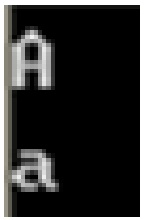
例4.4 输入一个字符，判别它是否大写字母，如果是，将它转换成小写字母；如果不是，不转换。然后输出最后得到的字符。

4.5 条件运算符和条件表达式

- 解题思路：用条件表达式来处理，当字母是大写时，转换成小写字母，否则不转换

4.5 条件运算符和条件表达式

```
#include <stdio.h>
int main()
{
  char ch;
  scanf("%c",&ch);
  ch=(ch>='A' && ch<='Z  ')?(ch+32):ch;
  printf("%c\n",ch);
  return 0;
}
```



4.6 选择结构的嵌套

- 在if语句中又包含一个或多个if语句称为if语句的嵌套
- 一般形式：

```
if( )
    if( ) 语句1
    else  语句2
else
    if( ) 语句3
    else  语句4
```

else总是与它上面最近的未配对的if配对

内嵌if

4.6 选择结构的嵌套

- 在if语句中又包含一个或多个if语句称为if语句的嵌套

```
if ()
{
    if () 语句1
}
else语句2
```

内嵌if

{ }限定了内嵌if范围

4.6 选择结构的嵌套

例4.5有一函数:

$$y=\begin{cases}-1 & (x<0)\\ 0 & (x=0)\\ 1 & (x>0)\end{cases}$$

编一程序，输入一个x值，要求输出相应的y值。

4.6 选择结构的嵌套

- 解题思路：
  - 用if语句检查x的值，根据x的值决定赋予y的值
  - 由于y的可能值不是两个而是三个，因此不可能只用一个简单的(无内嵌if)的if语句来实现

4.6

```
scanf("%d",&x);
if(x<0)    y = -1;
if(x==0)   y = 0;
if(x>0)    y = 1;
printf("x=%d,y=%d\n",x,y);
```

- 解题思路

  (1) 先后用3个独立的if语句处理：

  输入x

  若 x < 0, 则y =-1

  若 x = 0, 则y = 0

  若 x > 0, 则y = 1

  输出x和y

```
-5
x=-5,y=-1
```

4.6

```
scanf("%d",&x);
if(x<0)  y=-1;
else
    if(x==0) y=0;
    else  y=1;
printf("x=%d,y=%d\n",x,y);
```

- 解题思路

(2) 用一个

输入x

若x < 0, 则y = -1

否则

若 x = 0, 则y = 0

否则　y = 1

输出x和y

```
-5
x=-5,y=-1
```

```
5
x=5,y=1
```

明德求新 尚用笃行

4.6

- 解题思路

  (2) 用一

  输入x

  若x < 0 则y = -1

```
scanf("%d",&x);
if(x<0)  y=-1;
else
    if(x==0) y=0;
    else  y=1;
printf("x=%d,y=%d\n",x,y);
```

提倡内嵌if放在else中

```
scanf("%d",&x);
if (x>=0)
    if (x>0) y=1;
    else     y=0;
else       y=-1;
printf("x=%d,y=%d\n",x,y);
```

4.7 用switch语句实现多分支选择结构

- switch语句用来实现多分支选择结构
  - 学生成绩分类

    85分以上为‘A’等

    70～84分为‘B’等

    60～69分为‘C’等

    ……
  - 人口统计分类

    按年龄分为老、中、青、少、儿童

4.7 用switch语句实现多分支选择结构

例4.6 要求按照考试成绩的等级输出百分制分数段，A等为85分以上，B等为70～84分，C等为60～69分 ，D等为 60分以下 。成绩的等级由键盘输入。

4.7 用switch语句实现多分支选择结构

- 解题思路：
  - 判断出这是一个多分支选择问题
  - 根据百分制分数将学生成绩分为4个等级
  - 如果用if语句，至少要用3层嵌套的if，进行3次检查判断
  - 用switch语句进行一次检查即可得到结果

```
#include <stdio.h>
int main()
{ char grade;
   scanf("%c",&grade);
   printf("Your score:");
   switch(grade)
  { case 'A': printf("85～100\n");break;
     case 'B': printf("70～84\n");break;
     case 'C': printf("60～69\n");break;
     case 'D': printf("<60\n");break;
     default:  printf("enter data error!\n");
   }
   return 0;
 }
```

```
A
Your score:85～100
```

值为A

```
#include <stdio.h>
int main()
{ char grade;
   scanf("%c",&grade);
   printf("Your score:");
   switch(grade)
  { case 'A': printf("85～100\n");break;
     case 'B': printf("70～84\n");break;
     case 'C': printf("60～69\n");break;
     case 'D': printf("<60\n");break;
     default:  printf("enter data error!\n");
   }
   return 0;
 }
```

不能少

```
#include <stdio.h>
int main()
{ char grade;
   scanf("%c",&grade);
   printf("Your score:");
   switch(grade)
  { case 'A': printf("85～100\n");break;
     case 'B': printf("70～84\n");break;
     case 'C': printf("60～69\n");break;
     case 'D': printf("<60\n");break;
     default:  printf("enter data error!\n");
   }
   return 0;
 }
```

```
C
Your score:60～69
```

值为C

```
#include <stdio.h>
int main()
{ char grade;
   scanf("%c",&grade);
   printf("Your score:");
   switch(grade)
  { case 'A': printf("85～100\n");break;
     case 'B': printf("70～84\n");break;
     case 'C': printf("60～69\n");break;
     case 'D': printf("<60\n");break;
     default:  printf("enter data error!\n");
   }
   return 0;
 }
```

值为F

```
F
Your score:enter data error!
```

```
#include <stdio.h>
int main()
{ char grade;
   scanf("%c",&grade);
   printf("Your score:");
   switch(grade)
  { case 'A': printf("85～100\n");break;
     case 'B': printf("70～84\n");break;
     case 'C': printf("60～69\n");break;
     case 'D': printf("<60\n");break;
     default:  printf("enter data error!\n");
   }
   return 0;
 }
```

此行位置有问题，
应如何修改？

```
F
Your score:enter data error!
```

- switch语句的作用是根据表达式的值，使流程跳转到不同的语句
- switch语句的一般形式：

```
switch（表达式）
{ case 常量1 ： 语句1
  case 常量2 ： 语句2
    ┆   ┆     ┆
  case 常量n ： 语句n
  default    ： 语句n+1
}
```

整数类型(包括字符型)

- switch语句的作用是根据表达式的值，使流程跳转到不同的语句
- switch语句的一般形式：

```
switch（表达式）
{ case 常量1：语句1
  case 常量2：语句2
    ┆    ┆      ┆
  case 常量n：语句n
  default   ：语句n+1
}
```

不能相同

```
A
Your score:85~100
70~84
60~69
<60
enter data error!
```

```
scanf("%c",&grade);
printf("Your score:");
switch(grade)
{ case 'A': printf("85～100\n");break;
  case 'B': printf("70～84\n");break;
  case 'C': printf("60～69\n");break;
  case 'D': printf("<60\n");break;
  default:  printf("enter data error!\n");
}
```

```
 scanf("%c",&grade);
 printf("Your score:");
 switch(grade)
{ case 'A': printf("85～100\n");break;
   case 'B': printf("70～84\n");break;
   case 'C': printf("60～69\n");break;
   case 'D': printf("<60\n");break;
   default:  printf("enter data error!\n");
 }
```

```
 scanf("%c",&grade);
 printf("Your score:");
 switch(grade)
{ case 'A ‘:
   case 'B ‘:
   case 'C': printf("60～69\n");break;
   case 'D': printf("<60\n");break;
   default:  printf("enter data error!\n");
 }
```

```
A
Your score:60~69
```

例4.7 编写程序，用switch语句处理菜单命令。

- 解题思路：在许多应用程序中，用菜单对流程进行控制，例如从键盘输入一个’A’或’a’字符，就会执行A操作，输入一个’B’或’b’字符，就会执行B操作，等等。

```
#include <stdio.h>
int main()
{ void action1(int,in
   char ch;  int a=15,
   ch=getchar();
   switch(ch)
   { case 'a':
      case  ‘A’ : action1(a,b);break;
      case 'b':
      case  ‘B’ : action2(a,b);break;
      default:  putchar( ‘\a’ );
   }
   return 0;
}
```

```
void action1(int x,int y)
{
   printf("x+y=%d\n",x+y);
}
```

输入a或A

调用action1函数，执行A操作

```
#include <stdio.h>
int main()
{ void action1(int,i
   char ch;  int a=15
   ch=getchar();
   switch(ch)
   { case 'a':
      case  'A' : action1(a,b);break;
      case 'b':
      case  'B' : action2(a,b);break;
      default:  putchar( '\a' );
   }
   return 0;
}
```

```
void action2(int x,int y)
{
  printf("x*y=%d\n",x*y);
}
```

输入b或B

调用action2函数，执行B操作

```
#include <stdio.h>
int main()
{ void action1(int,int),action2(int,int);
   char ch;  int a=15,b=23;
   ch=getchar();
   switch(ch)
   { case 'a':
      case  ‘A’ : action1(a,b);break;
      case 'b':
      case  ‘B’ : action2(a,b);break;
      default:  putchar( ‘\a’ );
   }
   return 0;
}
```

输入其他字符

发出警告

- 这是一个非常简单的示意程序
- 实际应用中，所指定的操作可能比较复杂：
  - **A**：输入全班学生各门课的成绩
  - **B**：计算并输出每个学生各门课的平均成绩
  - **C**：计算并输出各门课的全班平均成绩
  - **D**：对全班学生的平均成绩由高到低排序并输出
- 可以按以上思路编写程序，把各**action**函数设计成不同的功能以实现各要求

4.8选择结构程序综合举例

例4.8 写一程序，判断某一年是否闰年。

- 解题思路：在前面已介绍过判别闰年的方法
- 本例用不同的方法编写程序

4.8选择结构程序综合举例

- 用变量leap代表是否闰年的信息。若闰年，令leap=1；非闰年，leap=0。最后判断leap是否为1（真），若是，则输出“闰年”信息
- 参见教材图4.13

```
#include <stdio.h>
int main()
{int year,leap;
  printf("enter year:"); scanf("%d",&year);
  if (year%4==0)
     if(year%100==0)
        if(year%400==0)    leap=1;
        else    leap=0;
     else    leap=1;
  else    leap=0;
  if (leap)     printf("%d is ",year);
  else  printf("%d is not ",year);
  printf("a leap year.\n");
  return 0;
}
```

标志变量

与if (leap!=0)含义相同

```
2012
2012 is a leap year.
```

```
2100
2100 is not a leap year.
```

```
{int year,leap;
  printf("enter year:"); scanf("%d",&year);
  if (year%4==0)
     if(year%100==0)
        if(year%400==0)    leap=1;
        else    leap=0;
     else    leap=1;
  else    leap=0;
  if (leap)     printf("%d is ",year);
  else  printf("%d is not ",year);
  printf("a leap year.\n");
  return 0;
}
```

采取锯齿形式

```
#include <stdio.h>
int main()
{int year,leap;
  printf("enter year:"); scanf("%d",&year);
  if (year%4==0)
     if(year%100==0)
        if(year%400==0)    leap=1;
        else    leap=0;
     else    leap=1;
  else    leap=0;
  if (leap)      printf("%d is ",year);
  else
  pri
  ret
}
```

```
if(year%4!=0)  leap=0;
else if (year%100!=0)  leap=1;
else if(year%400!=0)   leap=0;
else     leap=1;
```

```
#include <stdio.h>
int main()
{int year,leap;
  printf("enter year:"); scanf("%d",&year);
  if (year%4==0)
     if(year%100==0)
        if(year%400==0)    leap=1;
        else    leap=0;
     else    leap=1;
  else    leap=0;
  if((year%4==0 && year%100!=0)
                    || (year%400==0))
     leap=1;
  else
     leap=0;
}
```

例4.9 求 $ax^2+bx+c=0$ 方程的解。

- 解题思路：处理以下各情况
  - ① a ＝ 0，不是二次方程
  - ② $b^2-4ac=0$，有两个相等实根
  - ③ $b^2-4ac>0$，有两个不等实根。
  - ④ $b^2-4ac<0$，有两个共轭复根。

    应当以**p+qi**和**p-qi**的形式输出复根

    其中，**p=-b/2a**，**q=(**$\sqrt{b^2-4ac}$**)/2a**
- 参见教材图**4.14**

```
#include <stdio.h>
#include <math.h>
int main()
{
  double a,b,c,disc,x1,x2,realpart,
                                imagpart;
  scanf("%lf,%lf,%lf",&a,&b,&c);
  printf("The equation ");
  if(fabs(a)<=1e-6)        实型不能用if (a==0)
     printf("is not a quadratic\n");
```

```
 else
{disc=b*b-4*a*c;
 if(fabs(disc)<=1e-6)
   printf("has two equal roots:%8.4f\n",
                                  -b/(2*a));
 else
```

先算disc，以减少重复计算

不能用if (disc==0)

```
if(disc>1e-6)
{x1=(-b+sqrt(disc))/(2*a);
  x2=(-b-sqrt(disc))/(2*a);
  printf("has distinct real roots:%8.4f
                          and %8.4f\n",x1,x2);
}
 else
```

```
    { realpart=-b/(2*a);
      imagpart=sqrt(-disc)/(2*a);
      printf(" has complex roots:\n");
      printf("%8.4f+%8.4fi\n  "
                          ,realpart,imagpart);
      printf("%8.4f-%8.4fi\n",
                          realpart,imagpart);
    }
1,2,1
The equation has two equal roots: -1.0000
  return 0;
}
```

```
{ realpart=-b/(2*a);
   imagpart=sqrt(-disc)/(2*a);
   printf(" has complex roots:\n");
   printf("%8.4f+%8.4fi\n  "
                    ,realpart,imagpart);
   printf("%8.4f-%8.4fi\n",
                     realpart,imagpart);
```

```
1,2,2
The equation  has complex roots:
 -1.0000+  1.0000i
 -1.0000-  1.0000i
```

```
}
```

```
    { realpart=-b/(2*a);
      imagpart=sqrt(-disc)/(2*a);
      printf(" has complex roots:\n");
      printf("%8.4f+%8.4fi\n  "
                          ,realpart,imagpart);
      printf("%8.4f-%8.4fi\n",
                          realpart,imagpart);
    }
2,6,1
The equation has distinct real roots: -0.1771 and  -2.8229
    return 0;
}
```

例4.10 运输公司对用户计算运输费用。路程(s km）越远，每吨□千米运费越低。

□ 标准如下：

s < 250	没有折扣
250≤s < 500	2%折扣
500≤s < 1000	5%折扣
1000≤s < 2000	8%折扣
2000≤s < 3000	10%折扣
3000≤s	15%折扣

- 解题思路：
  - 设每吨每千米货物的基本运费为p，货物重为w，距离为s，折扣为d
  - 总运费f的计算公式为f=p×w×s×(1-d)

- 折扣的变化规律（参见教材图4.15）：
  - 折扣的“变化点”都是250的倍数
  - 在横轴上加一种坐标c，c的值为s/250
  - c代表250的倍数
  - 当c<1时，表示s<250，无折扣
  - 1≤c<2时，表示250≤s<500，折扣d=2%
  - 2≤c<4时，d=5%；4≤c<8时，d=8%；8≤c<12时，d=10%；c≥12时，d=15%

```
#include <stdio.h>
int main()
{
   int c,s;
   float p,w,d,f;
   printf("please enter
    price,weight,discount:");
   scanf("%f,%f,%d",&p,&w,&s);
   if(s>=3000)  c=12;
   else         c=s/250;
```

输入单价、重量、距离

```
switch(c)
{ case 0:   d=0; break;
   case 1:   d=2; break;
   case 2:
   case 3:   d=5; break;
   case 4:
   case 5:
   case 6:
   case 7:   d=8; break;
   case 8:  case 9:  case 10:
   case 11:  d=10; break;
   case 12:  d=15; break;
}
```

```
    f = p * w * s * (1 - d / 100);
    printf( “freight=%10.2f\n” ,f);
    return 0;
}
```

```
please enter price,weight,discount:100,20,300
freight= 588000.00
```

第5章 循环结构程序设计

5.1为什么需要循环控制

- 在日常生活中或是在程序所处理的问题中常常遇到需要重复处理的问题
  - 要向计算机输入全班50个学生的成绩
  - 分别统计全班50个学生的平均成绩
  - 求30个整数之和
  - 教师检查30个学生的成绩是否及格

5.1为什么需要循环控制

- 例如：全班有50个学生，统计各学生三门课的平均成绩。

输入学生1的三门课成绩，并计算平均值后输出

```
scanf(“%f,%f,%f”,&s1,&s2,&s3);
aver=(s1+s2+s3)/3;
printf(“aver=%7.2f”,aver);
```

输入学生2的三门课成绩，并计算平均值后输出

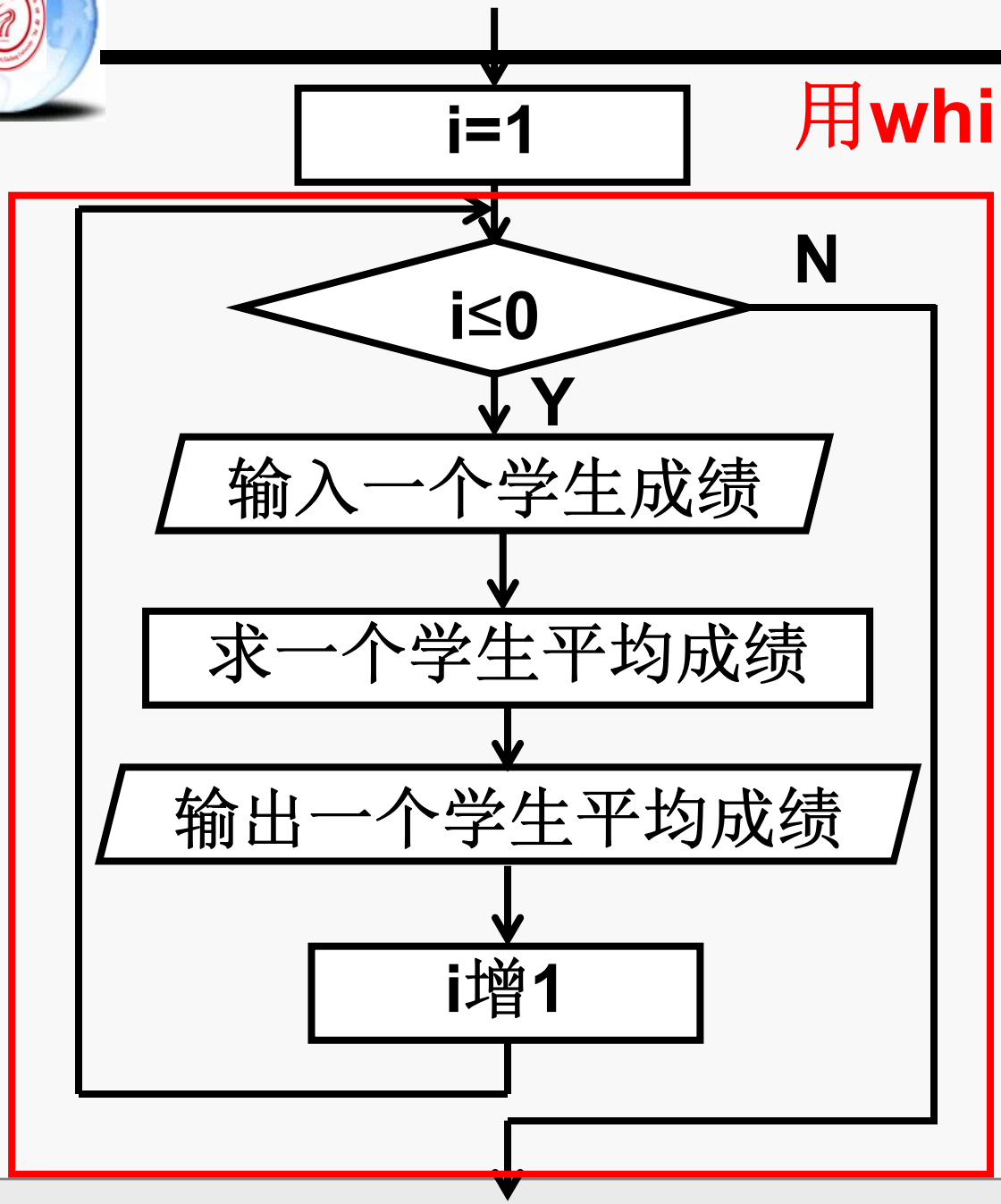
```
scanf(“%f,%f,%f”,&s1,&s2,&s3);
aver=(s1+s2+s3)/3;
printf(“aver=%7.2f”,aver);
```

要对50个学生进行相同操作　重复50次

- 大多数的应用程序都会包含循环结构
- 循环结构和顺序结构、选择结构是结构化程序设计的三种基本结构，它们是各种复杂程序的基本构造单元

5.2用while语句实现循环

- 全班有50个学生，统计各学生三门课的平均成绩。



用while循环结构实现

```
while(i<=50)
{ scanf…… ;
  aver=…… ;
  printf…… ;
  i++;
}
```

while语句的一般形式如下：

while (表达式) 语句

循环体

while语句的一般形式如下：

while (表达式) 语句

循环条件表达式

“真”时执行循环体语句
“假”时不执行

while循环的特点是：
先判断条件表达式，后执行循环体语句

例5.1求1+2+3+…+100，即 $\sum_{n=1}^{100} n$

- 解题思路：
  - 这是累加问题，需要先后将100个数相加
  - 要重复100次加法运算，可用循环实现
  - 后一个数是前一个数加1而得
  - 加完上一个数i后，使i加1可得到下一个数


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

不能少

复合语句

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

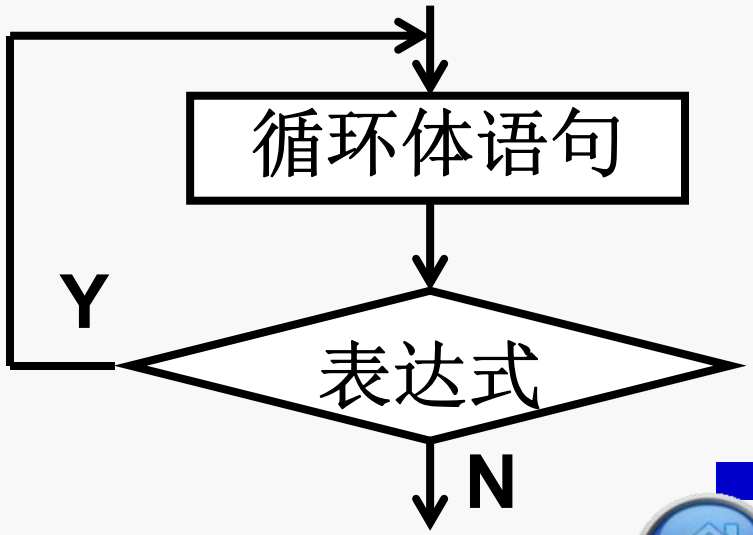
不能丢，否则循环永不结束

```
sum=5050
```

5.3用do---while语句实现循环

- do---while语句的特点：先无条件地执行循环体，然后判断循环条件是否成立
- do---while语句的一般形式为：

```
do
    语句
while (表达式);
```



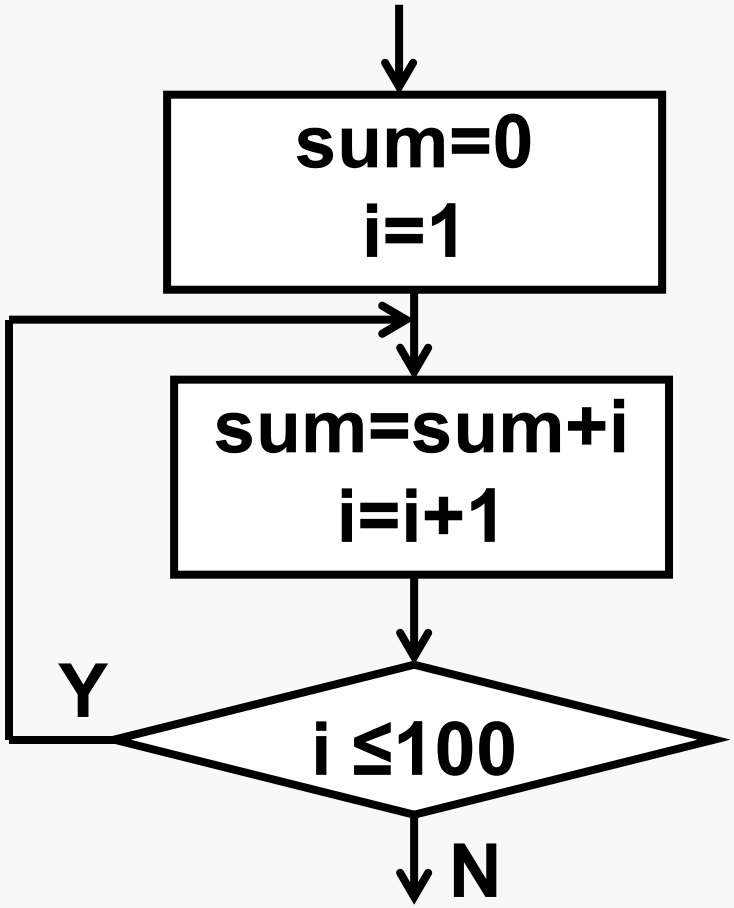
5.3用do---while语句实现循环

例5.2 用do…while语句求：

1+2+3+…+100，即 $\sum_{n=1}^{100} n$

5.3用do---while语句实现循环

- 解题思路：




```
i=1;   sum=0;
do
 {
    sum=sum+i;
    i++;
 }while(i<=100);
```


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

```
sum=5050
```


例5.3 while和do---while循环的比较。

当while后面的表达式的第一次的值为“真”时，两种循环得到的结果相同；否则不相同

```
in
pr
scanf(“%d”,&i);
while(i<=10)
{
  sum=sum+i;
  i++;
}
printf(“sum=%d\n",sum);
```

```
i=?1
sum=55
```

```
i=?11
sum=0
```

```
scanf(“%d”,&i);
do
{
   sum=sum+i;
   i++;
}while(i<=10);
printf(“sum=%d\n",sum);
```

```
i=?1
sum=55
```

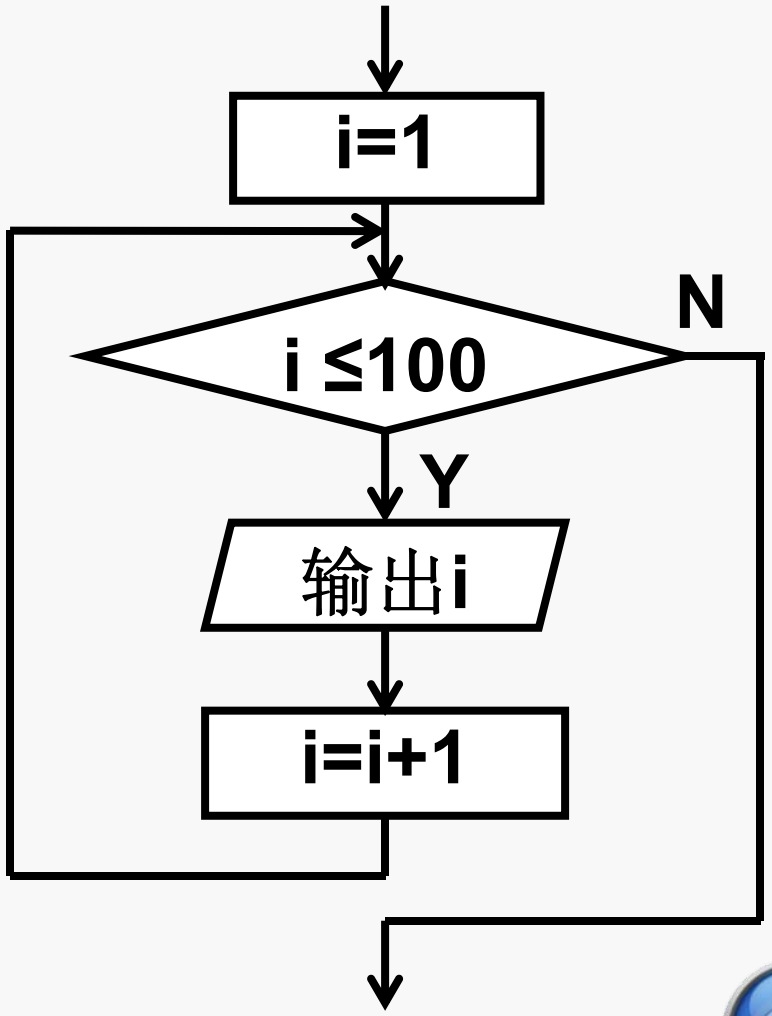
```
i=?11
sum=11
```

5.4用for 语句实现循环

- for语句不仅可以用于循环次数已经确定的情况，还可以用于循环次数不确定而只给出循环结束条件的情况
- for语句完全可以代替while语句

5.4用for 语句实现循环

```
for (i=1;i<=100;i++)
{
    printf("%d ", i );
}
```



5.4用for 语句实现循环

- for语句的一般形式为

  for(表达式1；表达式2；表达式3)

  语句

设置初始条件，只执行一次。可以为零个、一个或多个变量设置初值执行

5.4用for 语句实现循环

- for语句的一般形式为

  for(表达式1；表达式2；表达式3)

  语句

循环条件表达式，用来判定是否继续循环。在每次执行循环体前先执行此表达式，决定是否继续执行循环

5.4用for 语句实现循环

- for语句的一般形式为

  for(表达式1；表达式2；表达式3)

  语句

作为循环的调整器，例如使循环变量增值，它是在执行完循环体后才进行的

5.4用for 语句实现循环

- for语句的执行过程：

(1) 先求解表达式1

(2) 求解表达式2，若其值为真，执行循环体，然后执行下面第(3)步。若为假，则结束循环，转到第(5)步

(3) 求解表达式3

(4) 转回上面步骤(2)继续执行

(5) 循环结束，执行for语句下面的一个语句

5.4用for 语句实现循环

```
for(i=1;i<=100;i++)
    sum=sum+i;
```

等价于

```
i=1;
while(i<=100)
{
    sum=sum+i;
    i++;
}
```

用for语句更简单、方便

5.4用for 语句实现循环

for(表达式1; 表达式2; 表达式3)
语句

一个或两个或三个
表达式均可以省略

5.4用for 语句实现循环

```
for (sum=0 ;  i<=100;  i++)
        sum=sum+i;
```

与循环变量无关
合法

5.4用for 语句实现循环

```
for(sum=0,i=1 ; i<=100; i++)
        sum=sum+i;
```

```
for(i=0,j=100 ; i<=j;  i++,j--  )
      k=i+j;
```

逗号表达式
合法

5.4用for 语句实现循环

```
for(i=0; (c=getchar())!='\n'; i+=c)
        ;

for(     ; (c=getchar())!='\n';    )
        printf("%c",  c);
```

合法

5.5循环的嵌套

- 一个循环体内又包含另一个完整的循环结构，称为循环的嵌套
- 内嵌的循环中还可以嵌套循环，这就是多层循环
- 3种循环(while循环、do…while循环和for循环)可以互相嵌套

5.6几种循环的比较

(1)一般情况下，3种循环可以互相代替

(2) 在while和do---while循环中，循环体应包含使循环趋于结束的语句。

(3) 用while和do---while循环时，循环变量初始化的操作应在while和do---while语句之前完成。而for语句可以在表达式1中实现循环变量的初始化。

5.7 改变循环执行的状态

5.7.1 用break语句提前终止循环

5.7.2 用continue语句提前结束本次循环

5.7.3 break语句和continue语句的区别

5.7.1 用break语句提前终止循环

- break语句可以用来从循环体内跳出循环体，即提前结束循环，接着执行循环下面的语句

5.7.1 用break语句提前终止循环

例5.4 在全系1000学生中，征集慈善募捐，当总数达到10万元时就结束，统计此时捐款的人数，以及平均每人捐款的数目。

5.7.1 用break语句提前终止循环

- 编程思路：
  - 循环次数不确定，但最多循环1000次
    - 在循环体中累计捐款总数
    - 用if语句检查是否达到10万元
    - 如果达到就不再继续执行循环，终止累加
  - 计算人均捐款数

5.7.1 用break语句提前终止循环

- 编程思路：
  - 变量amount，用来存放捐款数
  - 变量total，用来存放累加后的总捐款数
  - 变量aver，用来存放人均捐款数
  - 定义符号常量SUM代表100000

```
#include <stdio.h>
#define SUM 100000
int main()
{  float amount,aver,total;   int i;
    for (i=1,total=0;i<=1000;i++)
   { printf("please enter amount:");
      scanf("%f",&amount);
      total= total+amount;
      if (total>=SUM) break;
   }
   aver=total / i ;
   printf( “num=%d\naver=%10.2f\n “
                                        ,i,aver);
   return 0;
}
```

指定符号常量SUM代表100000

```
#include <stdio.h>
#define SUM 100000
int main()
{  float amount,aver,total;   int i;
    for (i=1,total=0;i<=1000;i++)
   { printf("please enter amount:");
      scanf("%f",&amount);
      total= total+amount;
      if (total>=SUM) break;
   }
   aver=total / i ;
   printf( “num=%d\naver=%10.2f\n “
                                        ,i,aver);
   return 0;
}
```

应该执行1000次

```
#include <stdio.h>
#define SUM 100000
int main()
{  float amount,aver,total;   int i;
    for (i=1,total=0;i<=1000;i++)
   { printf("please enter amount:");
      scanf("%f",&amount);
      total= total+amount;
      if (total>=SUM) break;
   }
   aver=total / i ;
   printf( “num=%d\naver=%10.2f\n “
                                   ,i,aver);
   return 0;
}
```

达到10万，提前结束循环

```
#include <stdio.h>
#define SUM 100000
int main()
{  float amount,aver,tota
    for (i=1,total=0;i<=10
   { printf("please enter a
      scanf("%f",&amount);
      total= total+amount;
      if (total>=SUM) break;
   }
   aver=total / i;
   printf( "num=%d\naver=%10.2f\n "
                                  ,i,aver);
   return 0;
}
```

```
please enter amount:12000
please enter amount:24600
please enter amount:3200
please enter amount:5643
please enter amount:21900
please enter amount:12345
please enter amount:23000
num=7
aver=   14669.71
```

实际捐款人数

```
#include <stdio.h>
#define SUM 100000
int main()
{  float amount,aver,total;   int i;
    for (i=1,total=0;i<=1000;i++)
   { printf("please enter amount:");
      scanf("%f",&amount);
      total= total+amount;
      if (total>=SUM) break;
   }
   aver=total / i ;
   printf( "num=%d (.....
                                  ,i,aver);
   return 0;
}
```

只能用于循环语句和switch语句之中，而不能单独使用

5.7.2 用continue语句提前结束本次循环

- 有时并不希望终止整个循环的操作，而只希望提前结束本次循环，而接着执行下次循环。这时可以用**continue**语句

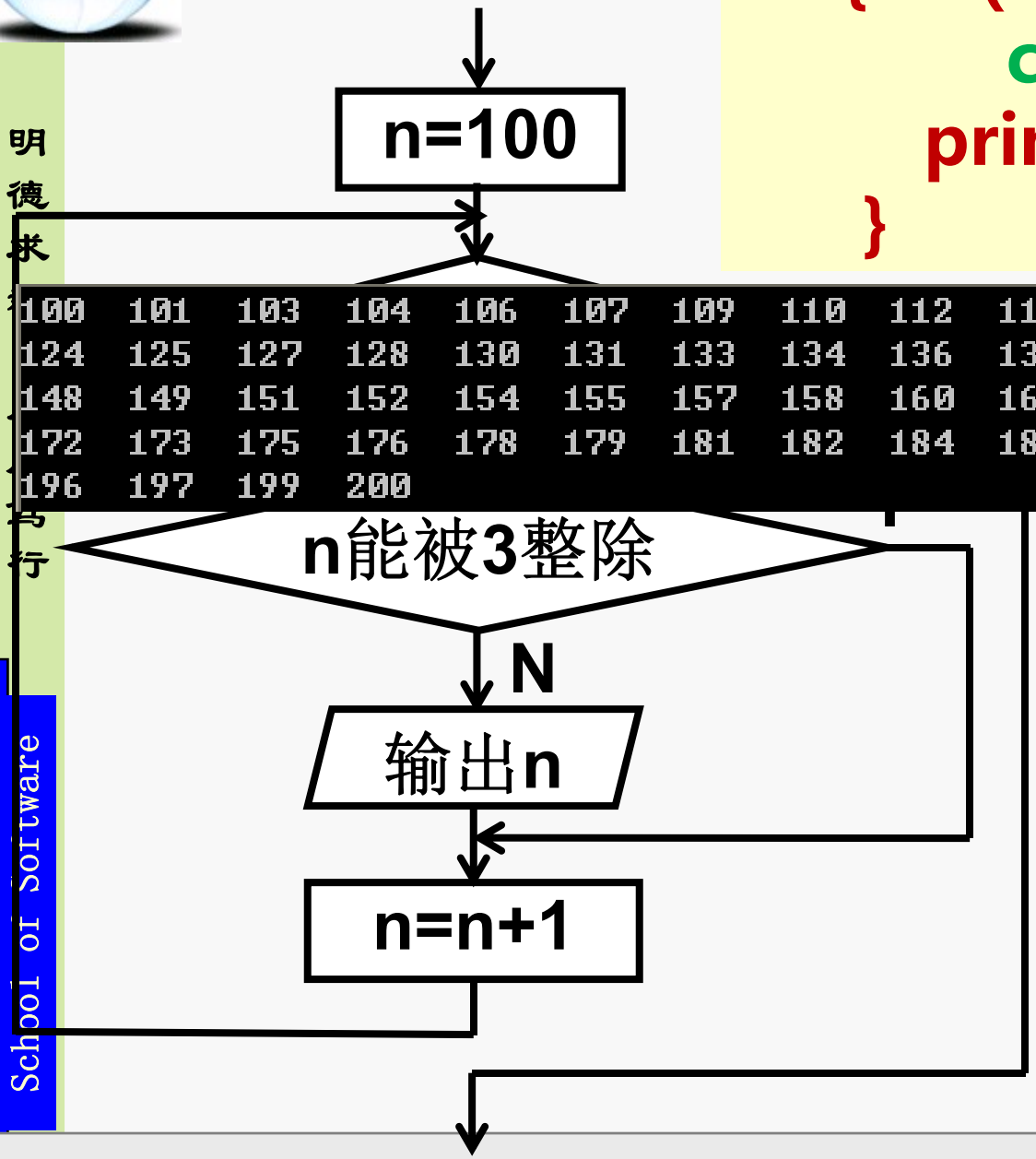
5.7.2 用continue语句提前结束本次循环

例5.5 要求输出100～200之间的不能被3整除的数。

- 编程思路：
  - 对100到200之间的每一个整数进行检查
  - 如果不能被3整除，输出，否则不输出
  - 无论是否输出此数，都要接着检查下一个数(直到200为止)。

```
for(n=100;n<=200;n++)
   {  if (n%3==0)
         continue;
      printf("%d  ",n);
   }
```

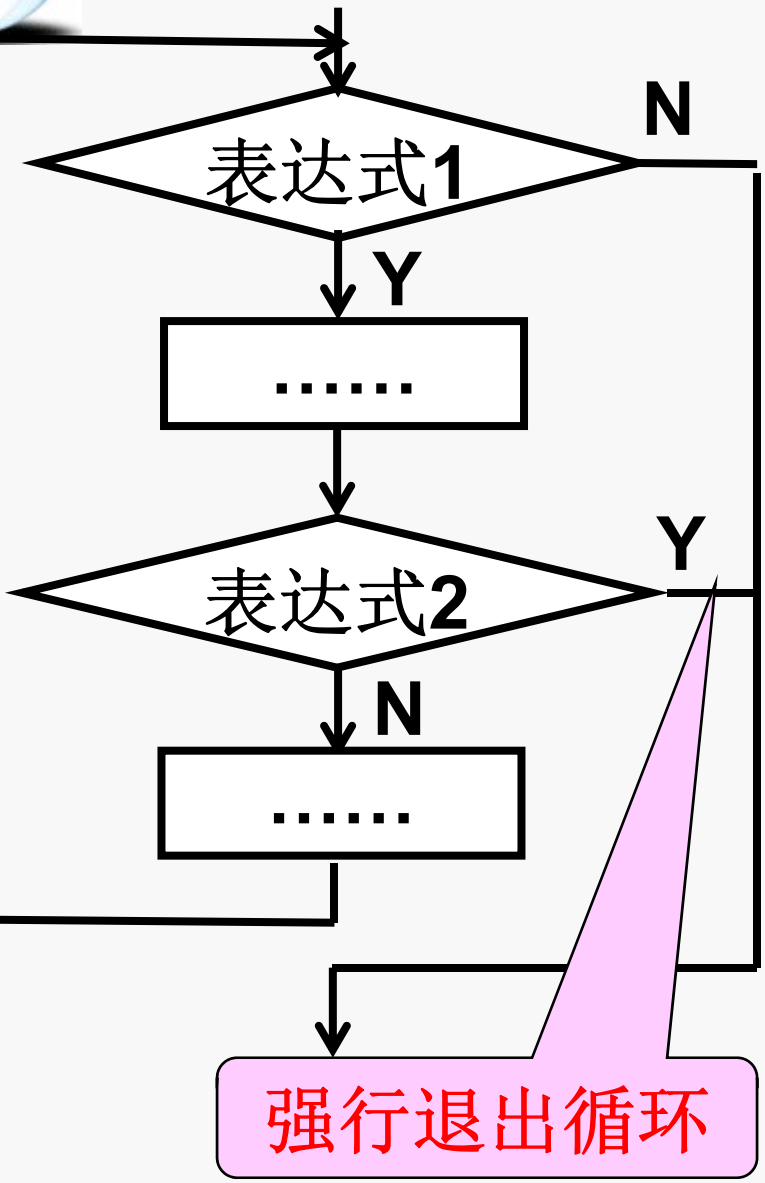
```
100  101  103  104  106  107  109  110  112  113  115  116  118  119  121  122
124  125  127  128  130  131  133  134  136  137  139  140  142  143  145  146
148  149  151  152  154  155  157  158  160  161  163  164  166  167  169  170
172  173  175  176  178  179  181  182  184  185  187  188  190  191  193  194
196  197  199  200
```



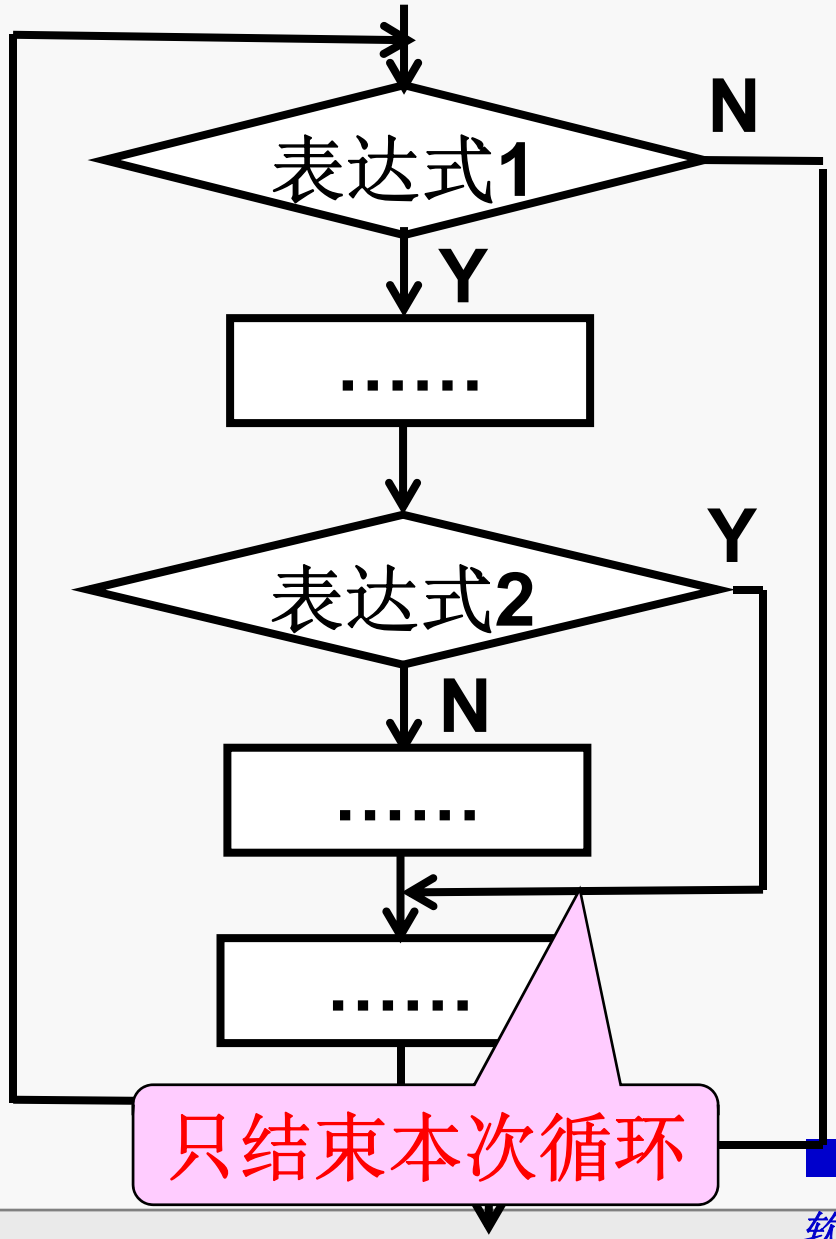
5.7.3 break语句和continue语句的区别

- **continue**语句只结束本次循环，而不是终止整个循环的执行
- **break**语句结束整个循环过程，不再判断执行循环的条件是否成立

break语句



continue语句



例5.6 输出以下4*5的矩阵。

```
1   2   3   4   5
2   4   6   8  10
3   6   9  12  15
4   8  12  16  20
```

明德求新 尚用笃行

- 解题思路：
  - 可以用循环的嵌套来处理此问题
  - 用外循环来输出一行数据
  - 用内循环来输出一列数据
  - 按矩阵的格式(每行5个数据)输出

```
#include <stdio.h>
int main()
{  int i,j,n=0;
    for (i=1;i<=4;i++)
        for (j=1;j<=5;j++,n++)
        {  if (n%5==0) printf ( "\n" );
           printf ("%d\t",i*j);
        }
    printf("\n");
    return 0;
}
```

累计输出数据的个数

控制一行内输出5个数据

双重循环

```
#include <stdio.h>
int main()
{  int i,j,n=0;
   for (i=1;i<=4;i++)
      for (j=1;j<=5;j++,n++)
      {  if (n%5==0) printf ( "\n" );
         printf ("%d\t",i*j);
      }
   printf("\n");
   return 0;
}
```


```
#include <stdio.h>
int main()
{  int i,j,n=0;
   for (i=1;i<=4;i++)
       for (j=1;j<=5;j++,n++)
       {  if (n%5==0) printf ( "\n" );
          printf ("%d\t",i*j);
       }
    printf("\n");
    return 0;
}
```

控制输出4行

```
#include <stdio.h>
int main()
{  int i,j,n=0;
    for (i=1;i<=4;i++)
       for (j=1;j<=5;j++,n++)
       {  if (n%5==0) printf ( "\n" );
          printf ("%d\t",i*j);
       }
     printf("\n");
     return 0;
}
```

控制每行中输出5个数据


```
#include <stdio.h>
int main()
{  int i,j,n=0;
    for (i=1;i<=4;i++)
       for (j=1;j<=5;j++,n++)
       {  if (n%5==0) printf ( "\n" );
          printf ("%d\t",i*j);
       }
    printf("\n");
    return 0;
}
```

i=1时

j由1变到5

i*j的值是1,2,3,4,5


```
#include
int main
{  int i,j,n=0;
    for (i=1;i<=4;i++)
       for (j=1;j<=5;j++,n++)
       {  if (n%5==0) printf ( "\n" );
          printf ("%d\t",i*j);
       }
    printf("\n");
    return 0;
}
```

如何修改程序，不输出第一行的空行？

j也由1变到5
i*j的值是2,4,6,8,10

```
1       2       3       4       5
2       4       6       8       10

4       8       12      16      20
```

```
#include <s
int main()
{  int i,j,n=0;
    for (i=1;i<=4;i++)
       for (j=1;j<=5;j++,n++)
       {  if (n%5==0) printf ( "\n" );
            if (i==3 && j==1) break;
          printf ("%d\t",i*j);
       }
    printf("\n");
    return 0;
}
```

遇到第3行第1列，终止内循环

```
1       2       3       4       5
2       4       6       8       10
6       9       12      15
4       8       12      16      20
```

原来第3行第1个数据3没有输出

```
#include
int main()
{
    int i,j,n=1;
    for (i=1;i<=4;i++)
        for (j=1;j<=5;j++,n++)
        {  if (n%5==0) printf ( "\n" );
            if (i==3 && j==1) continue;
            printf ("%d\t",i*j);
        }
    printf("\n");
    return 0;
}
```

5.8循环程序举例

例5.7用 $\frac{\pi}{4} \approx 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$ 公式求 π 的近似值，直到发现某一项的绝对值小于 10^{-6} 为止(该项不累计加)。

5.8循环程序举例

- 解题思路：
  - 求π近似值的方法很多，本题是一种
  - 其他方法：

$$\pi \approx \frac{22}{7}$$

$$\frac{\pi^2}{6} \approx \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{n^2}$$

$$\frac{\pi}{2} = \frac{2\times 2}{1\times 3} \times \frac{4\times 4}{3\times 5} \times \frac{6\times 6}{5\times 7} \times \cdots \times \frac{(n-1)^2}{n\times(n+2)}$$

5.8循环程序举例

$$\frac{\pi}{4} \approx 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$$

- 每项的分子都是1
- 后一项的分母是前一项的分母加2
- 第1项的符号为正，从第2项起，每一项的符号与前一项的符号相反

$$\frac{1}{n} \Rightarrow -\frac{1}{n+2}$$

5.8循环程序举例

sign=1,pi=0,n=1,term=1
当term ≥10^{-6}
pi=pi+term
n=n+1
sing=-sign
term=sign/n
pi=pi*4
输出pi

```
#include <stdio.h>
#include <math.h>
int main()
{  int sign=1; double pi=0,n=1,term=1;
   while(fabs(term)>=1e-6)
   {  pi=pi+term;
      n=n+2;
      sign=-sign;
      term=sign/n;
   }
   pi=pi*4;
   printf("pi=%10.8f\n",pi);
   return 0;
}
```

求绝对值的函数

```
pi=3.14159065
```

只保证前5位小数是准确的

```
#include <stdio.h>
#include <math.h>
int main()
{  int sign=1; double pi=0,n=1,term=1;
   while(fabs(term)>=1e-6)
   {  pi=pi+term;
      n=n+2;
      sign=-sign;
      term=sign/n;
   }
   pi=pi*4;
   printf("pi=%10.8f\n",pi);
   return 0;
}
```

改为1e-8

```
pi=3.14159065
```

```
pi=3.14159263
```

例5.8 求费波那西(Fibonacci)数列的前40个数。这个数列有如下特点：第1、2两个数为1、1。从第3个数开始，该数是其前面两个数之和。即:

$$\begin{cases} F_1 = 1 & (n = 1) \\ F_2 = 1 & (n = 2) \\ F_n = F_{n-1} + F_{n-2} & (n \geq 3) \end{cases}$$

- 这是一个有趣的古典数学问题：
  - 有一对兔子，从出生后第3个月起每个月都生一对兔子。
  - 小兔子长到第3个月后每个月又生一对兔子。
  - 假设所有兔子都不死，问每个月的兔子总数为多少？

第几个月	小兔子对数	中兔子对数	老兔子对数	兔子总数
1	1	0	0	1
2	0	1	0	1
3	1	0	1	2
4	1	1	1	3
5	2	1	2	5
6	3	2	3	8
7	5	3	5	13
⋮	⋮	⋮	⋮	⋮


```
f1=1,f2=1
输出f1,f2
For i=1 to 38
    f3=f1+f2
    输出f3
    f1=f2
    f2=f3
```


```
#include <stdio.h>
int main()
 { int f1=1,f2=1,f3;  int i;
    printf("%12d\n%12d\n",f1,f2);
    for(i=1; i<=38; i++)
    {  f3=f1+f2;
        printf("%12d\n",f3);
        f1=f2;
        f2=f3;
    }
    return 0;
}
```

代码可改进

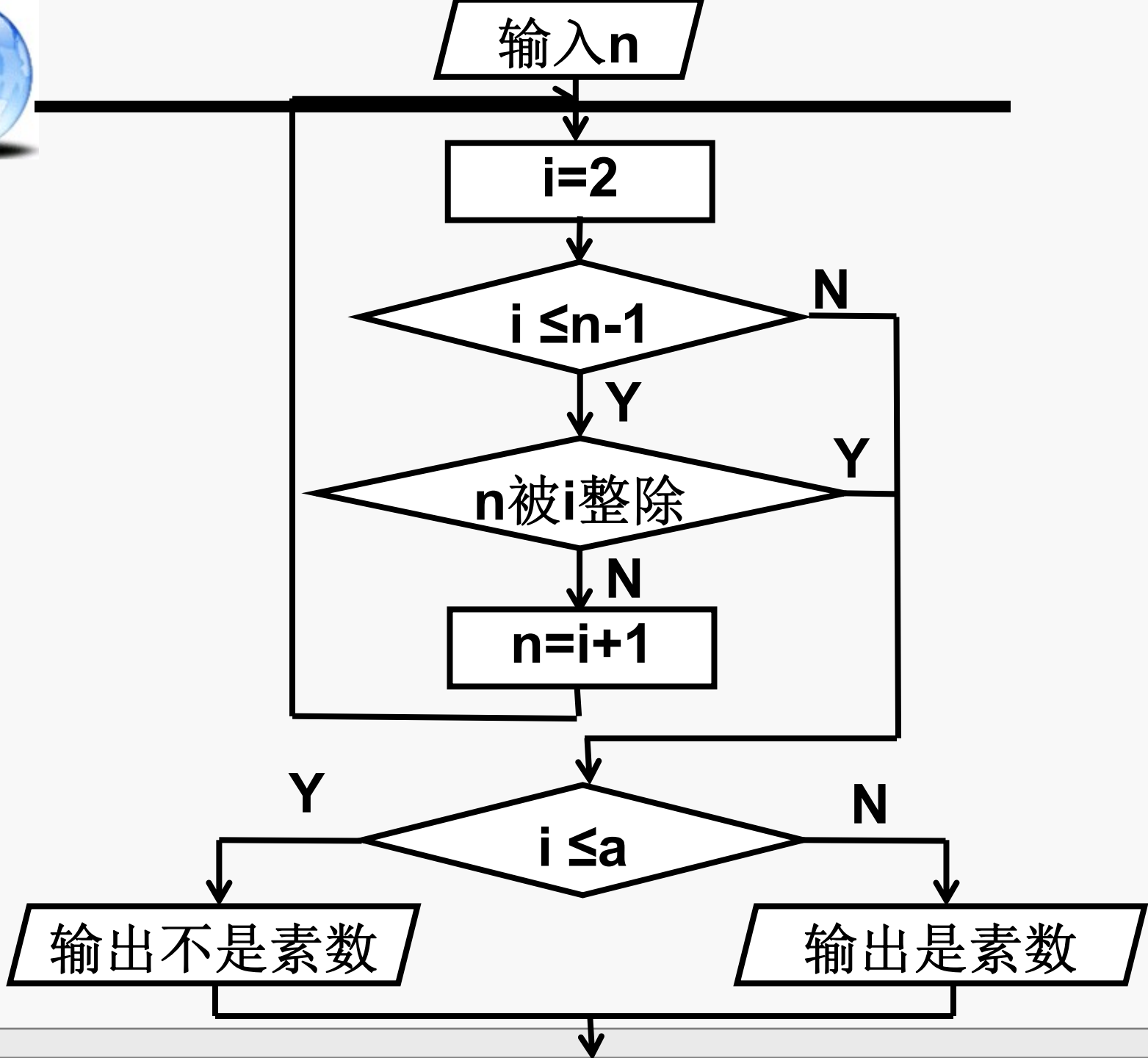
```
           1
           1
           2
           3
           5
           8
          13
          21
          34
          55
          89

         ...
```

```
#include <stdio.h>
int main()
 {  int f1=1,f2=1;  int i;
     for(i=1; i<=20; i++)
    { printf("%12d %12d ",f1,f2);
       if(i%2==0) printf("\n");
```

```
           1            1            2            3
           5            8           13           21
          34           55           89          144
         233          377          610          987
        1597         2584         4181         6765
       10946        17711        28657        46368
       75025       121393       196418       317811
      514229       832040      1346269      2178309
     3524578      5702887      9227465     14930352
    24157817     39088169     63245986    102334155
```

例5.9输入一个大于3的整数n，判定它是否素数(prime，又称质数)。

- 解题思路：
  - 让n被i整除(i的值从2变到n-1)
  - 如果n能被2～(n-1)之中任何一个整数整除，则表示n肯定不是素数，不必再继续被后面的整数除，因此，可以提前结束循环
  - 注意：此时i的值必然小于n


输入n
i=2
i ≤n-1
N
Y
n被i整除
Y
N
n=i+1
Y
i ≤a
N
输出不是素数
输出是素数

```
#include <stdio.h>
int main()
 { int n,i;
    printf( “n=?"); scanf("%d",&n);
    for (i=2;i<=n-1;i++)
        if(n%i==0) break;
    if(i<n) printf("%d is not\n",n);
    else printf("%d is\n",n);
    return 0;
}
```

```
n=?17
17 is
```

```
n=?327
327 is not
```

```
#include <stdio.h>
int main()
 { int n,i;
    printf( "n=?");  scanf("%d",&n);
    for (i=2;i<=n-1;i++)
        if(n%i==0) break;
    if(i<n) printf("%d is not\n",n);
    else printf("%d is\n",n);
    return 0;
}
```

$\sqrt{n}$

k=sqrt(n);

```
#include <stdio.h>
int main()
 { int n,i,k;
    printf( “n=?");  scanf("%d",&n);
    for (i=2; i<=k; i++)
        if(n%i==0) break;
    if(i<n) printf("%d is not\n",n);
    else printf("%d is\n",n);
    return 0;
}
```

#include <math.h>

k=sqrt(n);


```
#include <stdio.h>
int main()
 { int n,i,k;
    printf( "n=?");  scanf("%d",&n);
    for (i=2; i<=k; i++)
        if(n%i==0) break;
    if(i<=k) printf("%d is not\n",n);
    else printf("%d is\n",n);
    return 0;
}
```

#include <math.h>

k=sqrt(n);

例5.10 求100～200间的全部素数。

- 解题思路：
  - 使用例5.9的算法
  - 在例5.9程序中只要增加一个外循环，先后对100～200间的全部整数一一进行判定即可

```
……
for(n=101;n<=200;n=n+2)
{ k=sqrt(n);
   for (i=2;i<=k;i++)
       if (n%i==0) break;
   if (i>=k+1)
   {  printf("%d ",n);
       m=m+1;
   }
   if(m%10==0) printf( “\n” );
}
……
```

只对奇数进行检查

控制每行输出10个数据

例5.11 译密码。为使电文保密，往往按一定规律将其转换成密码，收报人再按约定的规律将其译回原文。

- 非字母字符保持原状不变
- 输入一行字符，要求输出其相应的密码

- 解题思路：问题的关键有两个：

(1) 决定哪些字符不需要改变，哪些字符需要改变，如果需要改变，从 c=getchar();

- 处理的方法是：输入一个字符给字符变量c，先判定它是否字母(包括大小写)，若不是字母，不改变c的值；若是字母，则还要检查它是否’W’到’Z’的范围内(包括大小写字母)。如不在此范围内，则使变量c的值改变为其后第4个字母。如果在’W’到’Z’的范围内，则应将它转换为A～D(或a～d)之一的字母。

- 解题思路：问题的关键有两个：

(1) 决定哪些字符不需要改变，哪些字符需要改变，如果需要改变，应改为哪个字符

```
if((c>='a' && c<='z') || (c>='A' && c<='Z'))
```

先判定它是否字母(包括大小写)，若不是字母，不改变c的值；若是字母，则还要检查它是否’W’到’Z’的范围内(包括大小写字母)。如不在此范围内，则使变量c的值改变为其后第4个字母。如果在’W’到’Z’的范围内，则应将它转换为A～D(或a～d)之一的字母。

- 解题思路：问题的关键有两个：

(1) 决定哪些字符不需要改变，哪些字符需要改变，如果需要改变，应改为哪个字符

```
if(c>='W' && c<='Z' || c>='w' && c<='z')
    c=c+4-26;
else  c=c+4;
```

否‘W’到‘Z’的范围内(包括大小写字母)。如不在此范围内，则使变量c的值改变为其后第4个字母。如果在‘W’到‘Z’的范围内，则应将它转换为A～D(或a～d)之一的字母。

- 解题思路：问题的关键有两个：

(2) 怎样使c改变为所指定的字母？

- 办法是改变它的ASCII值
- 例如字符变量c的原值是大写字母’A’，想使c的值改变为’E’，只需执行“c=c+4”即可，因为’A’的ASCII值为65，而’E’ 的ASCII值为69，二者相差4

```
char c;
c=getchar();
while(c!= ‘\n’ )
{ if((c>= ‘a’ && c<= ‘z’ ) || (c>= ‘A’
   &&
                                  c<= ‘Z’ ))
    { if(c>='W' && c<='Z' || c>='w' &&
                                    c<='z')
        c=c-22;
      else  c=c+4;
     }
    printf("%c",c);
    c=getchar();
}
```

可以改进程序

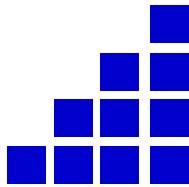
```
China!
Glmre!
```

```
char c;
while((c=getchar())!= '\n' )
{ if((c>= 'A' && c<= 'Z' ) || (c>= 'a'
  &&
                                c<= 'z' ))
  { c=c+4;
    if(c>= 'Z' && c<= 'Z' +4 || c> 'z' )
        c=c-26;
   }
   printf("%c",c);
}
```

不能少

明德求新 尚用笃行

第6章 利用数组处理批量数据



明德求新 尚用笃行

- 前几章使用的变量都属于基本类型，例如整型、字符型、浮点型数据，这些都是简单的数据类型。
- 对于有些数据，只用简单的数据类型是不够的，难以反映出数据的特点，也难以有效地进行处理。

- 如果有1000名学生，每个学生有一个成绩，需要求这1000名学生的平均成绩。
- 用$s_1, s_2, s_3, \ldots\ldots, s_{1000}$表示每个学生的成绩，它们之间在联系。
- C语言用方括号中的数字表示下标，如用s[15]表示

数组名

- 数组是一组有序数据的集合。数组中各数据的排列是有一定规律的，下标代表数据在数组中的序号
- 用一个数组名和下标惟一确定数组中的元素
- 数组中的每一个元素都属于同一个数据类型

6.1 怎样定义和引用一维数组

6.2 怎样定义和引用二维数组

6.3 字符数组

6.1怎样定义和引用一维数组

6.1.1怎样定义一维数组

- 一维数组是数组中最简单的
- 它的元素只需要用数组名加一个下标，就能惟一确定
- 要使用数组，必须在程序中先定义数组

6.1.1怎样定义一维数组

- 定义一维数组的一般形式为：

  类型符 数组名[常量表达式];

- 数组名的命名规则和变量名相同

如 **int a[10];**

数组名

6.1.1怎样定义一维数组

- 定义一维数组的一般形式为：

  类型符 数组名[常量表达式];

- 数组名的命名规则和变量名相同

如 **int a[10];**

数组长度

6.1.1怎样定义一维数组

- 定义一维数组的一般形式为：

  类型说明符 数组名[常量表达式];

  每个元素的数据类型

- 数组名的命名规则和变量名相同

如 int a[10];

10个元素：a[0],a[1],a[2],...,a[9]

a[0]	a[1]	a[2]	a[3]	...	a[7]	a[8]	a[9]

6.1.1怎样定义一维数组

- 定义一维数组的一般形式为：

  类型符 数组名[常量表达式];

int a[4+6]; 合法

int n=10;

int a[n];

不合法

6.1.2 怎样引用一维数组元素

- 在定义数组并对其中各元素赋值后，就可以引用数组中的元素
- 注意：只能引用数组元素而不能一次整体调用整个数组全部元素的值

6.1.2 怎样引用一维数组元素

- 引用数组元素的表示形式为：

  数组名［下标］

如a[0]=a[5]+a[7]-a[2*3]　合法

```
int n=5,a[10];
a[n]=20;
```

合法

6.1.2 怎样引用一维数组元素

例6.1 对10个数组元素依次赋值为0,1,2,3,4,5,6,7,8,9，要求按逆序输出。

- 解题思路：
  - 定义一个长度为10的数组，数组定义为整型
  - 要赋的值是从0到9，可以用循环来赋值
  - 用循环按下标从大到小输出这10个元素

```
#include <stdio.h>
int main()
 {  int i,a[10];
    for (i=0; i<=9;i++)
        a[i]=i;
    for(i=9;i>=0; i--)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
 }
```

使a[0]～a[9]的值为0～9

a[0]a[1]a[2]a[3]a[4]a[5]a[6]a[7]a[8]a[9]

0	1	2	3	4	5	6	7	8	9

```
#include <stdio.h>
int main()
 {  int i,a[10];
    for (i=0; i<=9;i++)
        a[i]=i;
    for(i=9;i>=0; i--)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
 }
```

```
9 8 7 6 5 4 3 2 1 0
```

先输出a[9]，最后输出a[0]

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	1	2	3	4	5	6	7	8	9

6.1.3一维数组的初始化

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

  int a[10]={0,1,2,3,4,0,0,0,0,0};
- int a[10]={0,0,0,0,0,0,0,0,0,0};相当于

  int a[10]={0};
- int a[5]={1,2,3,4,5};可写为

  int a[ ]={1,2,3,4,5};

6.1.4一维数组程序举例

例6.2 用数组处理求Fibonacci数列问题

- 解题思路：
  - 例5.8中用简单变量处理的，缺点不能在内存中保存这些数。假如想直接输出数列中第25个数，是很困难的。
  - 如果用数组处理，每一个数组元素代表数列中的一个数，依次求出各数并存放在相应的数组元素中

```
#include <stdio.h>
int main()
 {  int i;  int f[20]={1,1};
     for(i=2;i<20;i++)
         f[i]=f[i-2]+f[i-1];
     for(i=0;i<20;i++)
    {   if(i%5==0) printf( "\n" );
         printf( "%12d" ,f[i]);
```

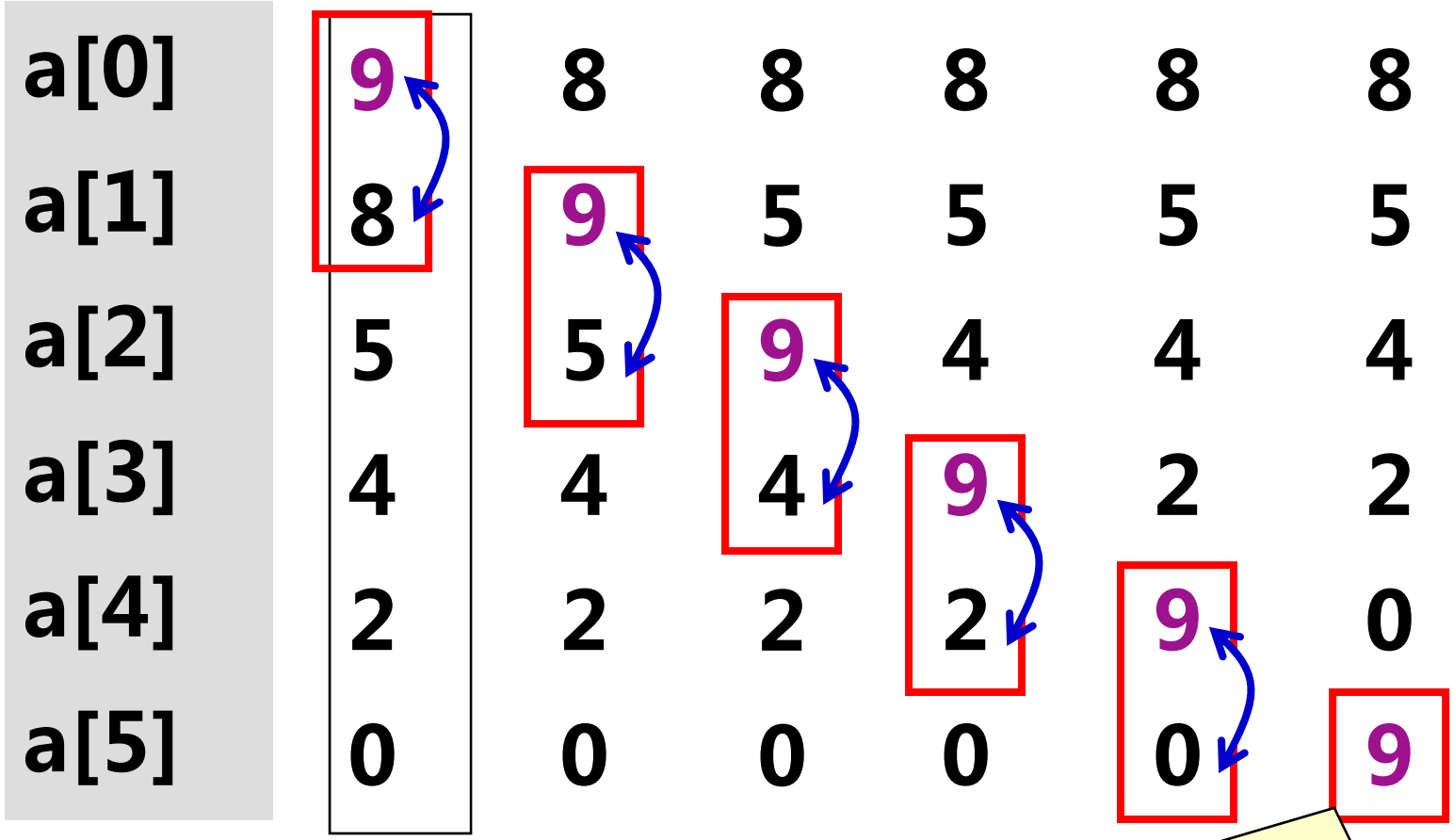
```
           1           1           2           3           5
           8          13          21          34          55
          89         144         233         377         610
         987        1597        2584        4181        6765
```

```
}
```

例6.3 有10个地区的面积，要求对它们按由小到大的顺序排列。

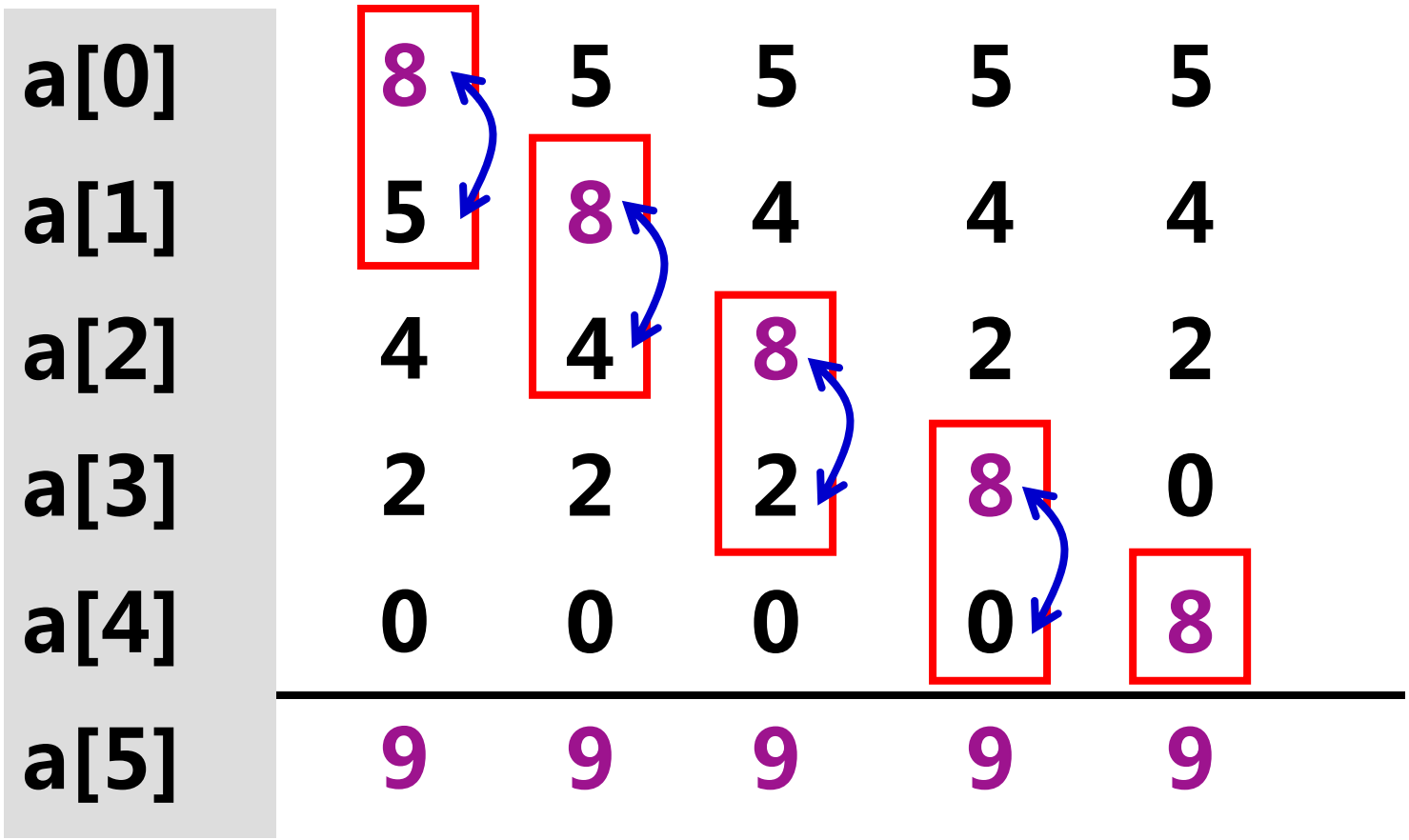
- 解题思路：
  - 排序的规律有两种：一种是“升序”，从小到大；另一种是“降序”，从大到小
  - 把题目抽象为：“对n个数按升序排序”
  - 采用起泡法排序

```
for(i=0;i<5;i++)
    if (a[i]>a[i+1])
    { t=a[i];a[i]=a[i+1];a[i+1]=t; }
```

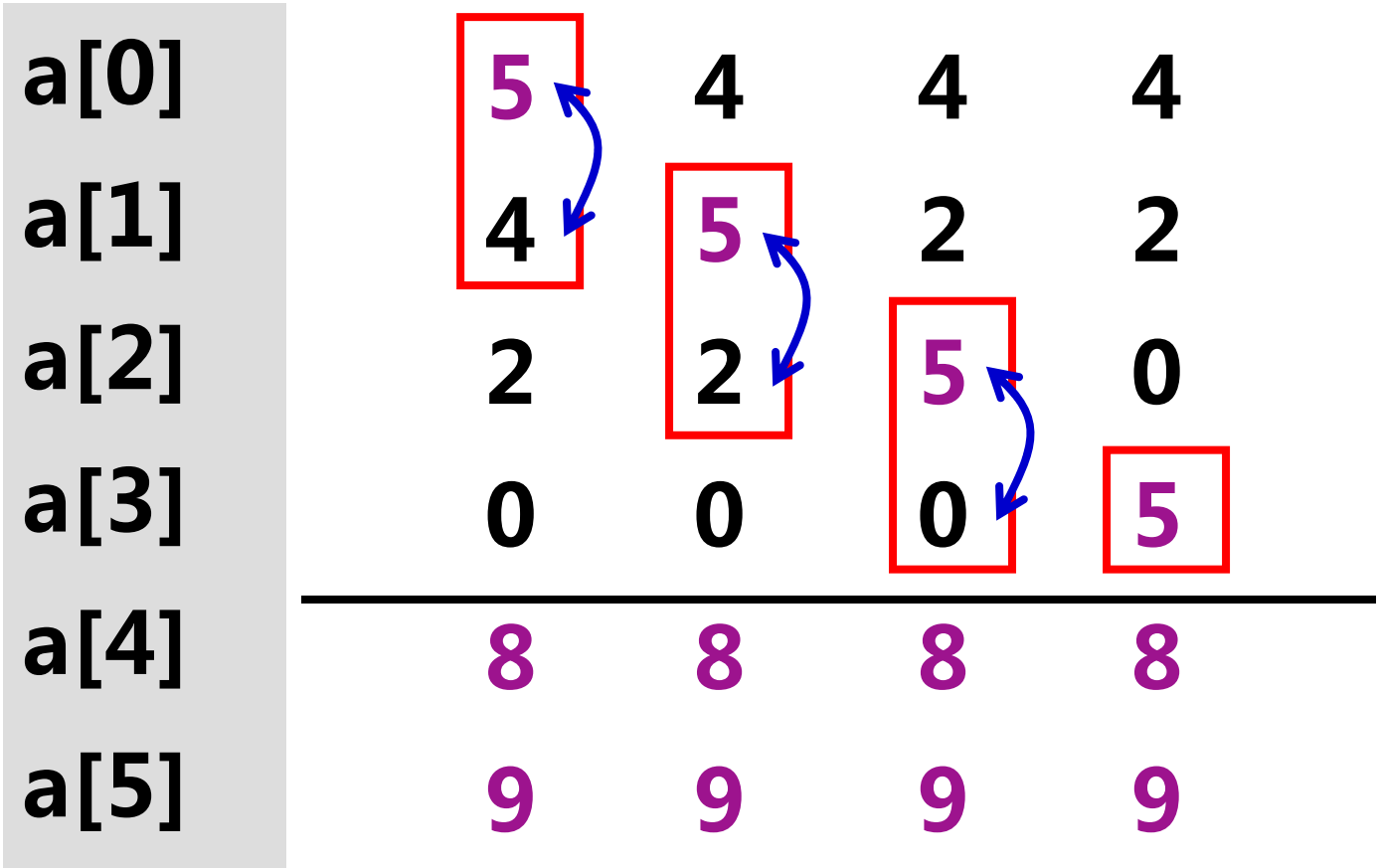


大数沉淀，小数起泡

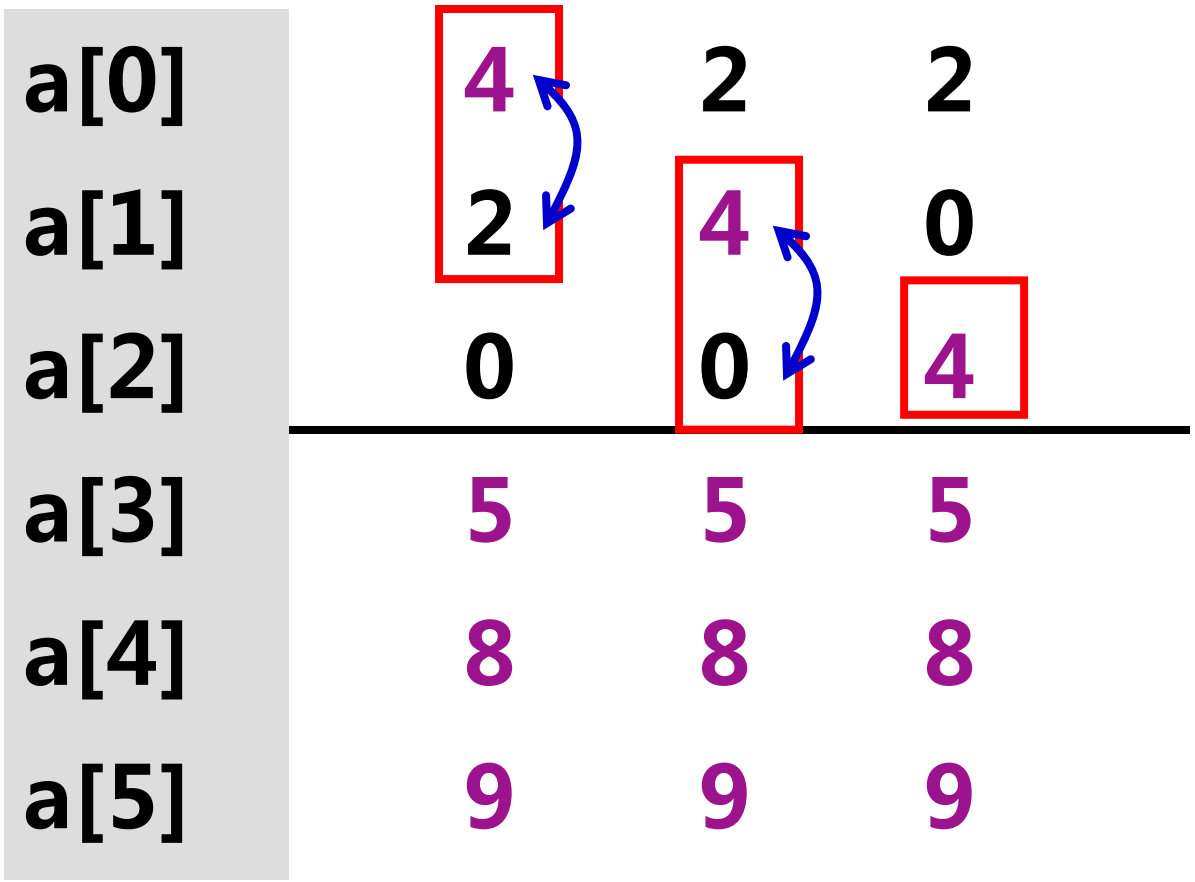
```
for(i=0;i<4;i++)
    if (a[i]>a[i+1])
    { t=a[i];a[i]=a[i+1];a[i+1]=t; }
```



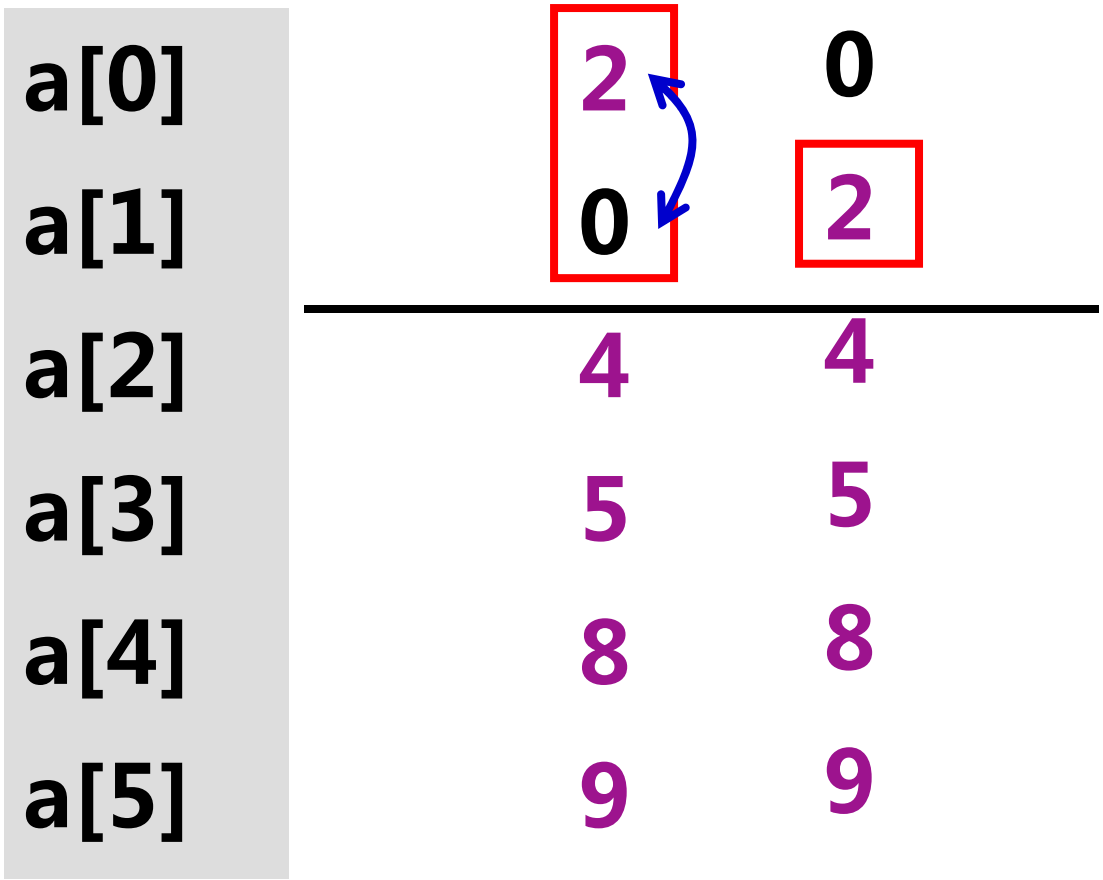
```
for(i=0;i<3;i++)
    if (a[i]>a[i+1])
    { t=a[i];a[i]=a[i+1];a[i+1]=t; }
```



```
for(i=0;i<2;i++)
    if (a[i]>a[i+1])
    { t=a[i];a[i]=a[i+1];a[i+1]=t; }
```



```
for(i=0;i<1;i++)
    if (a[i]>a[i+1])
    { t=a[i];a[i]=a[i+1];a[i+1]=t; }
```



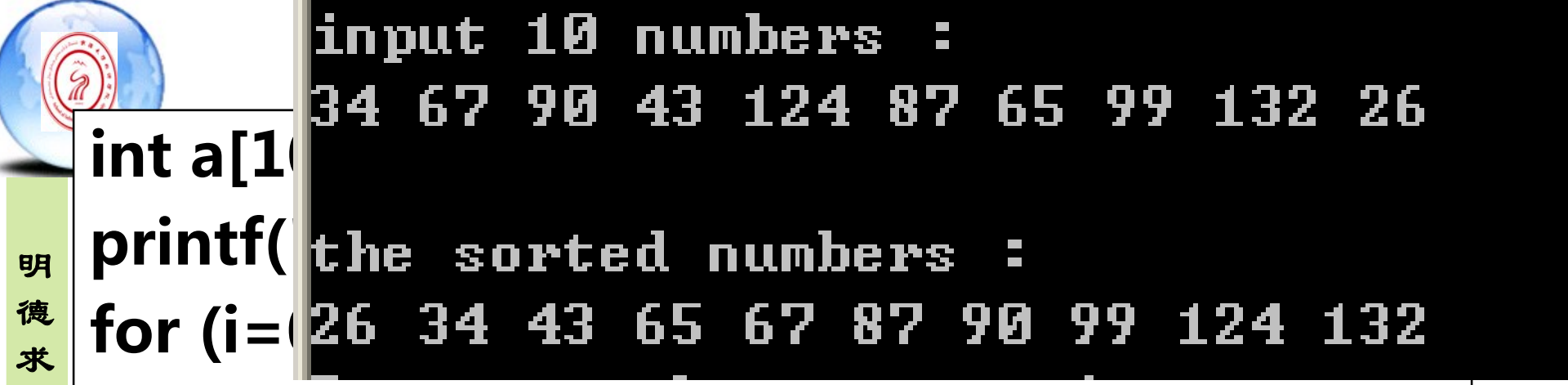
```
for(i=0;i<5;i++)
    if (a[i]>a[i+1])
    { ……}
```

```
for(i=0;i<4;i++)
    if (a[i]>a[i+1])
    { ……}
```

……

```
for(i=0;i<1;i++)
    if (a[i]>a[i+1])
    { ……}
```

```
for(j=0;j<5;j++)
    for(i=0;i<5-j;i++)
        if (a[i]>a[i+1])
        { ……}
```




```
int a[1
printf(
for (i=
printf("\n");
for(j=0;j<9;j++)
   for(i=0;i<9-j;i++)
     if (a[i]>a[i+1])
       {t=a[i];a[i]=a[i+1];a[i+1]=t;}
printf("the sorted numbers :\n");
for(i=0;i<10;i++)  printf("%d ",a[i]);
printf("\n");
```

6.2 怎样定义和引用二维数组

明德求新 尚用笃行

	队员1	队员2	队员3	队员4	队员5	队员6
1分队	2456	1847	1243	1600	2346	2757
2分队	3045	2018	1725	2020	2458	1436
3分队	1427	1175	1046	1976	1477	2018

```
float pay[3][6];
```

6.2 怎样定义和引用二维数组

6.2.1怎样定义二维数组

- 二维数组定义的一般形式为

  类型符 数组名[常量表达式][常量表达式];

  如：**float a[3][4],b[5][10];**

- 二维数组可被看作是一种特殊的一维数组：

  它的元素又是一个一维数组

- 例如，把a看作是一个一维数组，它有3个元素：

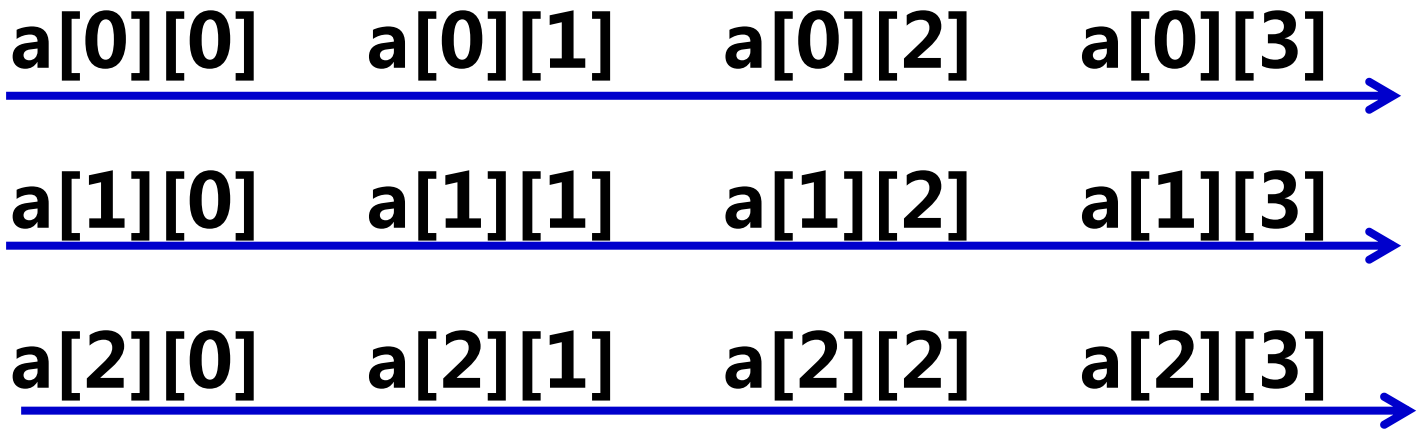
  a[0]、a[1]、a[2]

- 每个元素又是一个包含4个元素的一维数组

明德求新 尚用笃行

a[0]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1]	a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

逻辑存储



内存中的存储顺序

6.2.2怎样引用二维数组的元素

- 二维数组元素的表示形式为：

  数组名［下标］［下标］

- b[1][2]=a[2][3]/2　合法
- for(i=0;i<m;i++)

  printf(“%d,%d\n”,a[i][0],a[0][i]);合法

6.2.3二维数组的初始化

```
int a[3][4]={{1,2,3,4},{5,6,7,8},
                       {9,10,11,12}};
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
int a[3][4]={{1},{5},{9}};等价于
int a[3][4]={{1,0,0,0},{5,0,0,0},
                                {9,0,0,0}};
int a[3][4]={{1},{5,6}};相当于
int a[3][4]={{1},{5,6},{0}};
```

6.2.3二维数组的初始化

int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};

等价于：

int a[ ][4]={1,2,3,4,5,6,7,8,9,10,11,12};

int a[][4]={{0,0,3},{ },{0,10}};合法

6.2.4二维数组程序举例

例6.4 将一个二维数组行和列的元素互换，存到另一个二维数组中。

$$a=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \Longrightarrow b=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$$

6.2.4二维数组程序举例

- 解题思路：
  - 可以定义两个数组：数组a为2行3列，存放指定的6个数
  - 数组b为3行2列，开始时未赋值
  - 将a数组中的元素a[i][j]存放到b数组中的b[j][i]元素中
  - 用嵌套的for循环完成

```
#include <stdio.h>
int main()
 {  int a[2][3]={{1,2,3},{4,5,6}};
    int b[3][2],i,j;
    printf("array a:\n");
    for (i=0;i<=1;i++)
    { for (j=0;j<=2;j++)
      {  printf("%5d",a[i][j]);
         b[j][i]=a[i][j];
       }
      printf("\n");
     }
```

处理a的一行中各元素

处理a中某一列元素

输出a的各元素

a元素值赋给b相应元素

```
    printf("array b:\n");
    for (i=0;i<=2;i++)
    {  for(j=0;j<=1;j++)
            printf("%5d",b[i][j]);
        printf("\n");
    }
    return 0;
}
```

输出b的各元素

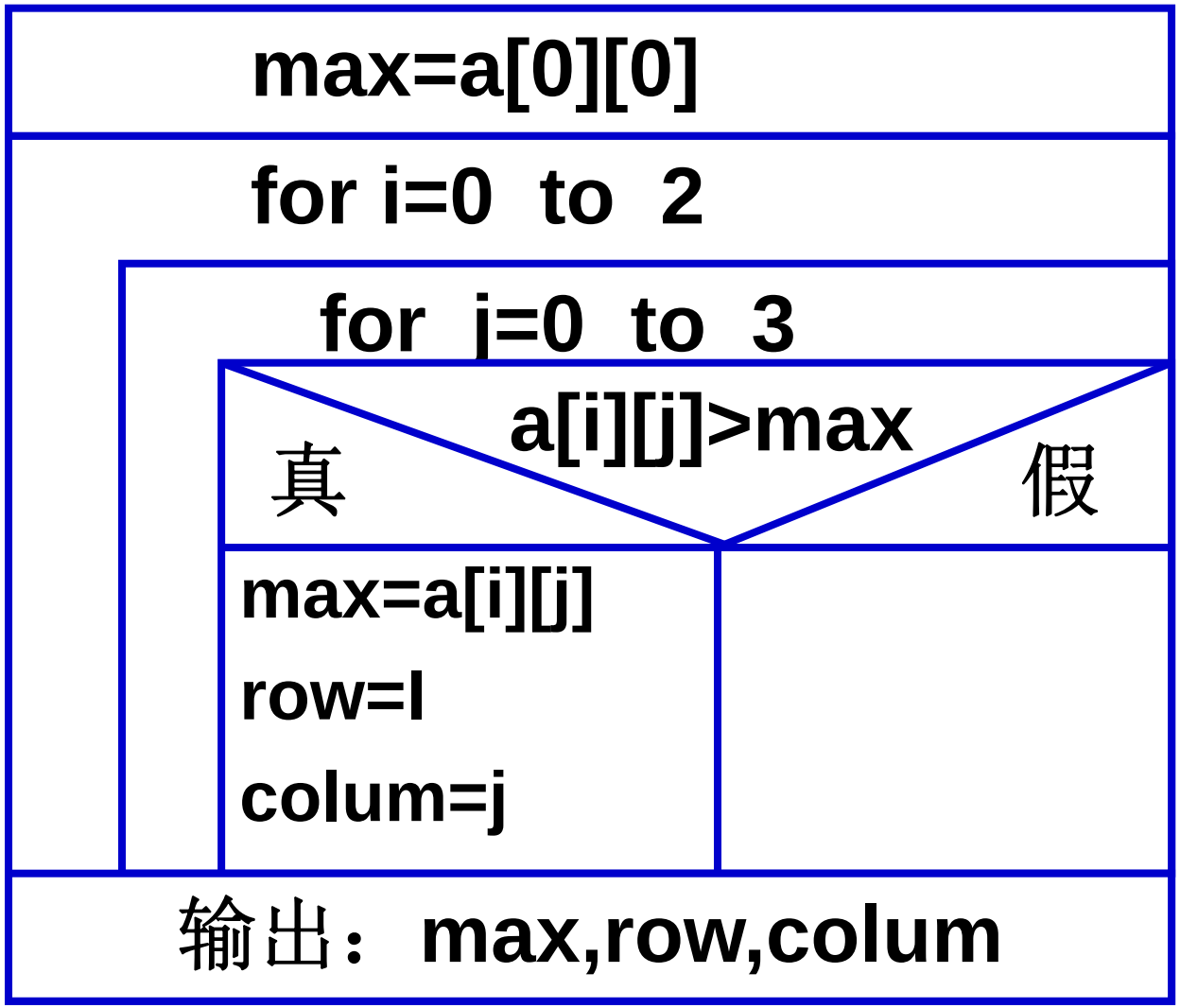
```
array a:
    1    2    3
    4    5    6
array b:
    1    4
    2    5
    3    6
```

例6.5 有一个3×4的矩阵，要求编程序求出其中值最大的那个元素的值，以及其所在的行号和列号。

- 解题思路：采用“打擂台算法”
  - 先找出任一人站在台上，第2人上去与之比武，胜者留在台上
  - 第3人与台上的人比武，胜者留台上，败者下台
  - 以后每一个人都是与当时留在台上的人比武，直到所有人都上台比为止，最后留在台上的是冠军

例6.5 有一个3×4的矩阵，要求编程序求出其中值最大的那个元素的值，以及其所在的行号和列号。

- 解题思路：采用“打擂台算法”
  - 先把a[0][0]的值赋给变量max
  - max用来存放当前已知的最大值
  - a[0][1]与max比较，如果a[0][1]>max，则表示a[0][1]是已经比过的数据中值最大的，把它的值赋给max，取代了max的原值
  - 以后依此处理，最后max就是最大的值


max=a[0][0]
for i=0 to 2
for j=0 to 3
a[i][j]>max
真
假
max=a[i][j]
row=I
colum=j
输出： max,row,colum

```
max=10
row=2
colum=1
```

……

```
int i,j,row=0,colum=0,max;
int a[3][4]={{1,2,3,4},{9,8,7,6},
                       {-10,10,-5,2}};
max=a[0][0];
for (i=0;i<=2;i++)
  for (j=0;j<=3;j++)
    if (a[i][j]>max)
    {  max=a[i][j];  row=i;  colum=j; }
printf("max=%d\nrow=%d\n
        colum=%d\n",max,row,colum);
```

记最大值

记列号

……

6.3 字符数组

6.3.1怎样定义字符数组

- 用来存放字符数据的数组是字符数组
- 字符数组中的一个元素存放一个字符
- 定义字符数组的方法与定义数值型数组的方法类似

6.3.1怎样定义字符数组

```
char c[10];
c[0]=' I' ;  c[1]=' ' ;
c[2]=' a' ;  c[3]=' m' ;
c[4]=' ' ;  c[5]=' h' ;
c[6]=' a' ; c[7]=' p' ;
c[8]=' p' ; c[9]=' y' ;
```

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]
I		a	m		h	a	p	p	y

6.3.2字符数组的初始化

明

```
char c[10]={' I' ,' ' ,' a' ,' m' ,'
   ' ,' h' ,' a' ,' p' ,' p' ,' y' };
```

c[0]c[1]c[2]c[3]c[4]c[5]c[6]c[7]c[8]c[9]

I		a	m		h	a	p	p	y

```
char c[10]={' c' ,'
   ' ,' b' ,' r' ,' o' ,' d' ,' r' ,' a' ,' m' };
```

c[0]c[1]c[2]c[3]c[4]c[5]c[6]c[7]c[8]c[9]

c		p	r	o	g	r	a	m	\0

School of Software

软件学院

6.3.2字符数组的初始化

明

```
char diamond[5][5]={{' ' ,' ' ,' ' ,' ' ,'*' },
                    {' ' ,' ' ,'*' ,' ' ,' ' ,' ' ,'*' },
                    {' ' ,'*' ,' ' ,' ' ,' ' ,' ' ,
' ' ,' ' ,'*' },
                    {' ' ,' ' ,'*' ,' ' ,' ' ,' ' ,'*' },
                    {' ' ,' ' ,' ' ,' ' ,'*' } };
```

6.3.3怎样引用字符数组中的元素

例6.6 输出一个已知的字符串。

- 解题思路：
  - 定义一个字符数组，并用“初始化列表”对其赋以初值
  - 用循环逐个输出此字符数组中的字符

6.3.3怎样引用字符数组中的元素

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

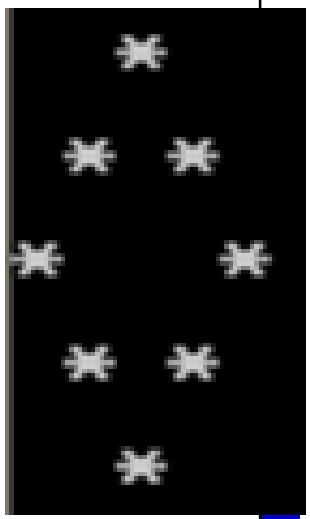
```
I am a student.
```

6.3.3怎样引用字符数组中的元素

例6.7 输出一个菱形图。

- 解题思路：
  - 定义一个字符型的二维数组，用“初始化列表”进行初始化
  - 用嵌套的for循环输出字符数组中的所有元素。

```
#include <stdio.h>
int main()
{ char diamond[][5]={{' ',' ','*'},
                    {' ','*',' ','*'},{'*',' ',' ',' ','*'},
                     {' ','*',' ','*'},{' ',' ','*'}};
  int i,j;
  for (i=0;i<5;i++)
    {for (j=0;j<5;j++)
         printf("%c",diamond[i][j]);
     printf("\n");
    }
  return 0;
}
```



6.3.4字符串和字符串结束标志

- 在C语言中，是将字符串作为字符数组来处理的
- 关心的是字符串的有效长度而不是字符数组的长度
- 为了测定字符串的实际长度，C语言规定了字符串结束标志‘ \0’

6.3.4字符串和字符串结束标志

- ‘\0’代表ASCII码为0的字符
- 从ASCII码表可以查到，ASCII码为0的字符不是一个可以显示的字符，而是一个“空操作符”，即它什么也不做
- 用它作为字符串结束标志不会产生附加的操作或增加有效字符，只起一个供辨别的标志

6.3.4字符串和字符串结束标志

```
char c[]={" I  am  happy" };
```

可写成

```
char c[]=" I  am  happy" ;
```

相当于

```
char c[11]={" I  am  happy" };
```

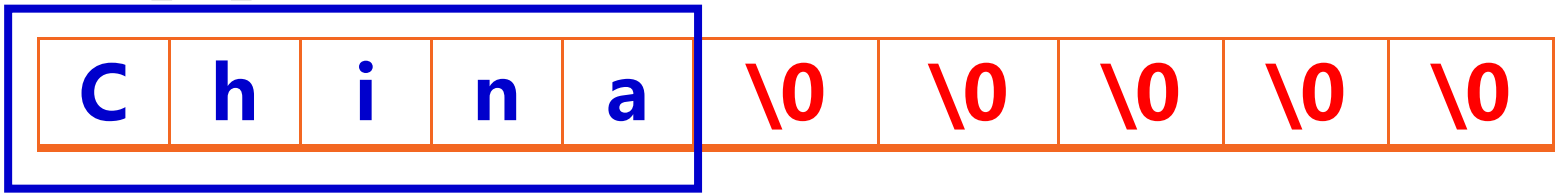
6.3.4字符串和字符串结束标志

char c[10]={" China"};

可写成

char c[10]=" China";

从c[5]开始，元素值均为\0



只显示

printf(" %s",c);

6.3.5 字符数组的输入输出

- 字符数组的输入输出可以有两种方法：
  - 逐个字符输入输出（%c）
  - 整个字符串一次输入输出（%s）
- 输出的字符中不包括结束符’\0’
- 用%s输出字符串时，printf函数中的输出项是字符数组名，不是数组元素名

6.3.5 字符数组的输入输出

- 如果一个字符数组中包含多个’\0’，则遇第一个’\0’时输出就结束
- 可以用**scanf**函数输入一个字符串
- **scanf**函数中的输入项**c**是已定义的字符数组名，输入的字符串应短于已定义的字符数组的长度

6.3.5 字符数组的输入输出

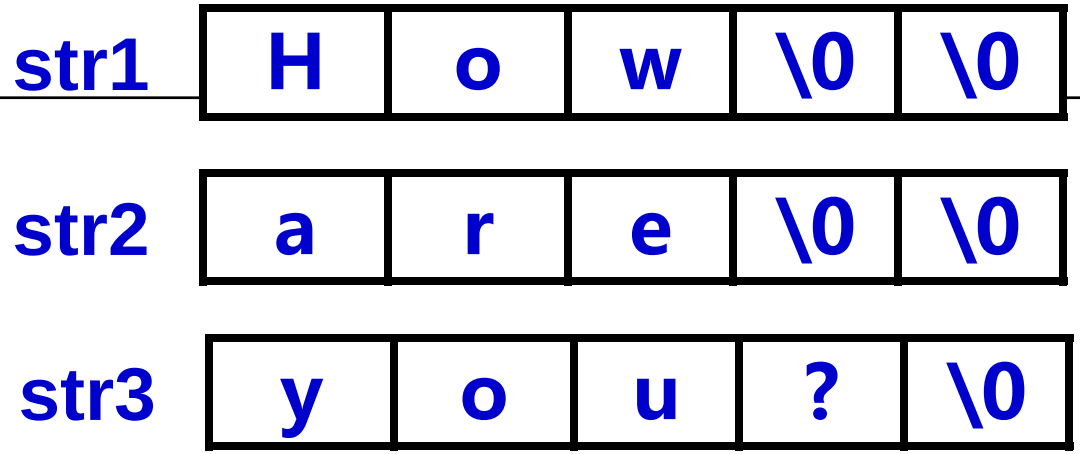
```
char c[6];
scanf(" %s" ,c);   China↙
```

系统自动在China后面加一个' \0'

6.3.5 字符数组的输入输出

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

How are you? ↙



6.3.6 善于使用字符串处理函数

- 在C函数库中提供了一些用来专门处理字符串的函数，使用方便

6.3.6 善于使用字符串处理函数

1.puts函数----输出字符串的函数

- 其一般形式为：

  puts (字符数组)

- 作用是将一个字符串输出到终端

```
char str[20]=”China”;
puts(str);
```

输出China

6.3.6 善于使用字符串处理函数

2. gets函数----输入字符串的函数

- 其一般形式为：

  gets(字符数组)

- 作用是输入一个字符串到字符数组

```
char str[20];
gets(str);
 Computer↙
```

6.3.6 善于使用字符串处理函数

3. strcat函数----字符串连接函数

- 其一般形式为：

  strcat(字符数组1，字符数组2)

- 其作用是把两个字符串连接起来，把字符串2接到字符串1的后面，结果放在字符数组1中

使用字符串函数时,在程序开头用#include <string.h>

6.3.6 善于使用字符串处理函数

3. strcat函数----字符串连接函数

```
char str1[30]=" People";
char str2[]=" China";
printf(" %s", strcat(str1,str2));
```

要足够大

输出：PeopleChina

6.3.6 善于使用字符串处理函数

4. strcpy和strncpy函数-字符串复制

- strcpy一般形式为：

  strcpy(字符数组1,字符串2)

- 作用是将字符串2复制到字符数组1中去

```
char str1[10],str2[]=”China”;
strcpy(str1,str2);
```

str1	C	h	i	n	a	\0	\0	\0	\0	\0

6.3.6 善于使用字符串处理函数

明德求新 尚用笃行

School of Software

4. strcpy和strncpy函数-字符串复制

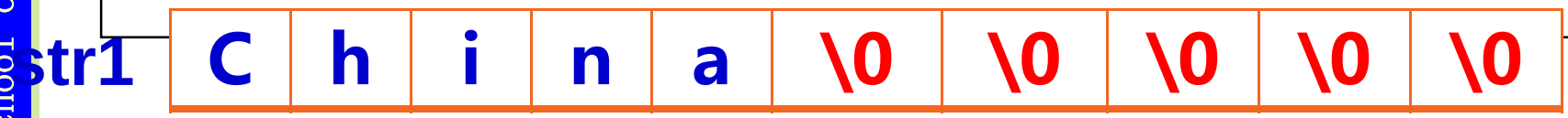
- strcpy一般形式为：

  strcpy(字符数组1,字符串2)

- 作用是将字符串2复制到字符数组1中去

```
char str1[10],str2[]=”China”;
strcpy(str1,str2);
```

要足够大



6.3.6 善于使用字符串处理函数

明德求新 尚用笃行

School of Software

4. strcpy和strncpy函数-字符串复制

- **strcpy一般形式为：**

  strcpy(字符数组1,字符串2)

- **作用是将字符串2复制到字符数组1中去**

```
char str1[10],str2[]=”China”;
strcpy(str1,str2);
```

数组名形式

6.3.6 善于使用字符串处理函数

4. strcpy和strncpy函数-字符串复制

- **strcpy一般形式为：**

  strcpy(字符数组1,字符串2)

- 作用是将字符串2复制到字符数组1中去

```
char str1[10],str2[]=”China”;
strcpy(str1,str2);
```

数组名或字符串常量

6.3.6 善于使用字符串处理函数

4. strcpy和strncpy函数-字符串复制

- **strcpy一般形式为：**

  strcpy(字符数组1,字符串2)

- 作用是将字符串2复制到字符数组1中去

char str1[10],str2[]=”China”;

strcpy(str1,str2); 相当于

strcpy(str1,”China”);

6.3.6 善于使用字符串处理函数

4. strcpy和strncpy函数-字符串复制

char str1[10],str2[]=”China”;

str1=”China”; 错误

str1=str2; 错误

6.3.6 善于使用字符串处理函数

4. strcpy和strncpy函数-字符串复制

- 可以用strncpy函数将字符串2中前面n个字符复制到字符数组1中去
- strncpy(str1，str2，2);
  - 作用是将str2中最前面2个字符复制到str1中，取代str1中原有的最前面2个字符
  - 复制的字符个数n不应多于str1中原有的字符

6.3.6 善于使用字符串处理函数

5. strcmp函数----字符串比较函数

- 其一般形式为

  strcmp(字符串1，字符串2)

- 作用是比较字符串1和字符串2
- **strcmp(str1,str2);**
- **strcmp(" China"," Korea");**
- **strcmp(str1," Beijing");**

6.3.6 善于使用字符串处理函数

5. strcmp函数----字符串比较函数

- 字符串比较的规则是：将两个字符串自左至右逐个字符相比，直到出现不同的字符或遇到’\0’为止
- 如全部字符相同，认为两个字符串相等
- 若出现不相同的字符，则以第一对不相同的字符的比较结果为准

6.3.6 善于使用字符串处理函数

5. strcmp函数----字符串比较函数

"A" < "B"　　"a" > "A"

"computer" > "compare"

"these" > "that"　"1A" > "$20"

"CHINA" > "CANADA"

"DOG" < "cat"

"Tsinghua" > "TSINGHUA"

6.3.6 善于使用字符串处理函数

5. strcmp函数----字符串比较函数

- 比较的结果由函数值带回
  - 如果字符串1=字符串2，则函数值为0
  - 如果字符串1>字符串2，则函数值为一个正整数
  - 如果字符串1<字符串2，则函数值为一个负整数

6.3.6 善于使用字符串处理函数

5. strcmp函数----字符串比较函数

```
if(str1>str2)  printf(" yes ");  错误
if(strcmp(str1,str2)>0)
    printf(" yes ");    正确
```

6.3.6 善于使用字符串处理函数

6. strlen函数----测字符串长度的函数

- 其一般形式为：

  strlen (字符数组)

- 它是测试字符串长度的函数
- 函数的值为字符串中的实际长度

6.3.6 善于使用字符串处理函数

6. strlen函数----测字符串长度的函数

```
char str[10]=" China";
printf(" %d",strlen(str));
```

- 输出结果是5
- 也可以直接测试字符串常量的长度

```
strlen(" China");
```

6.3.6 善于使用字符串处理函数

明德求新 尚用笃行

7. strlwr函数----转换为小写的函数

- 其一般形式为

  strlwr (字符串)

- 函数的作用是将字符串中大写字母换成小写字母

6.3.6 善于使用字符串处理函数

8. strupr函数----转换为大写的函数

- 其一般形式为

  strupr (字符串)

- 函数的作用是将字符串中小写字母换成大写字母

6.3.7字符数组应用举例

例6.8 输入一行字符，统计其中有多少个单词，单词之间用空格分隔开。

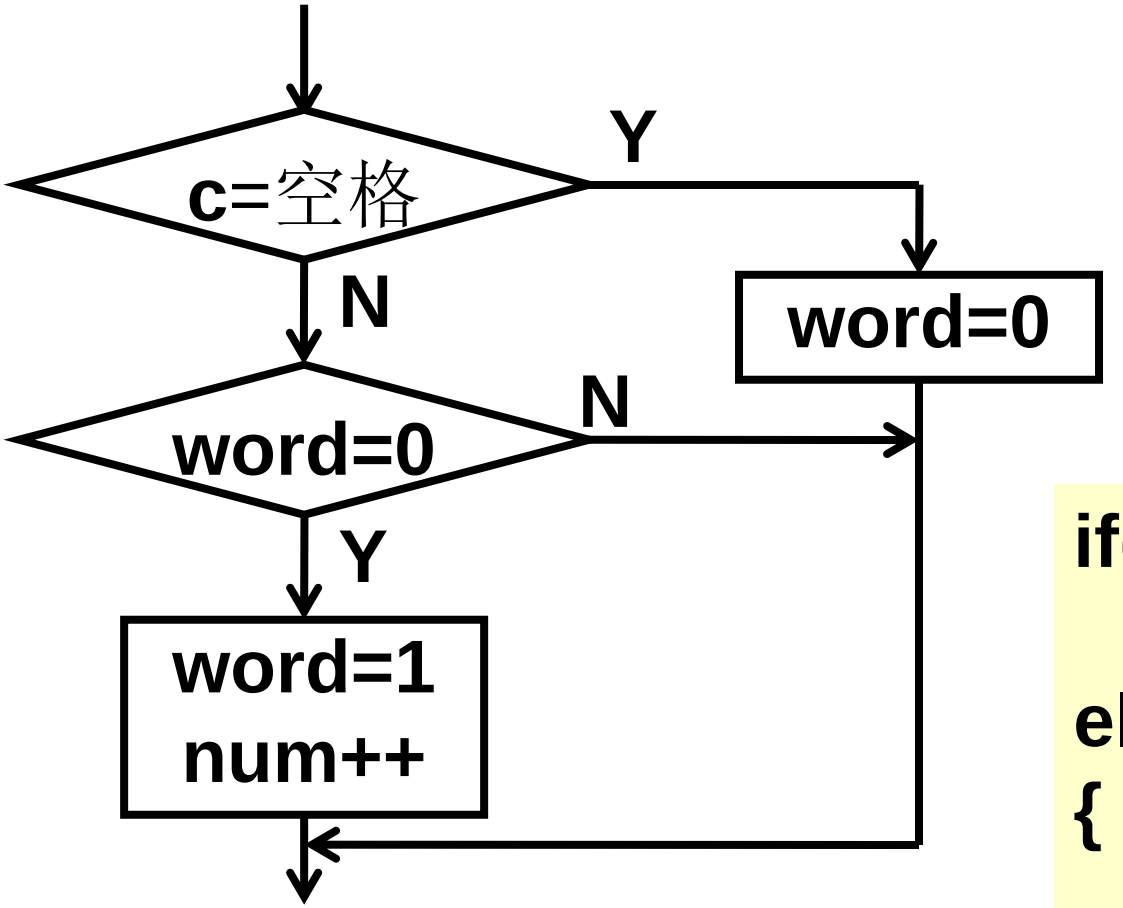
- 解题思路：问题的关键是怎样确定“出现一个新单词了”
  - 从第1个字符开始逐个字符进行检查，判断此字符是否是新单词的开头，如果是，就使变量num的值加1，最后得到的num的值就是单词总数

6.3.7字符数组应用举例

- 判断是否出现新单词，可以由是否有空格出现来决定(连续的若干个空格作为出现一次空格；一行开头的空格不统计在内)
- 如果测出某一个字符为非空格，而它的前面的字符是空格，则表示“新的单词开始了”，此时使**num**累加**1**
- 如果当前字符为非空格而其前面的字符也是非空格，则**num**不应再累加**1**

6.3.7字符数组应用举例

- 用变量word作为判别当前是否开始了一个新单词的标志，若word=0表示未出现新单词，如出现了新单词，就把word置成1
- 前面一个字符是否空格可以从word的值看出来，若word等于0，则表示前一个字符是空格；如果word等于1，意味着前一个字符为非空格




```
if(c==' ')
    word=0;
else if(word==0)
{   word=1;
     num++;
}
```

当前字符	I		a	m		a		b	o	y	.
是否空格	否	是	否	否	是	否	是	否	否	否	否
word原值	0	1	0	1	1	0	1	0	1	1	1
新单词开始否	是	否	是	否	否	是	否	是	否	否	否
word新值	1	0	1	1	0	1	0	1	1	1	1
num值	1	1	2	2	2	3	3	4	4	4	4

……

```
char string[81],c;  int i,num=0,word=0;
gets(string);
for (i=0;(c=string[i])!= '\0' ;i++)
    if(c== ' ' ) word=0;
    else if(word==0)
     { word=1;
       num++;
     }
 printf( "%d words\n",num);
```

一定要设初始值

……

```
……
char string[81],c;  int i,num=0,word=0;
gets(string);
for (i=0;(c=string[i])!= '\0' ;i++)
    if(c== ' ') word=0;
    else if(word==0)
     { word=1;
       num++;
     }
 printf( "%d words\n" ,num);
……
```

相当于
c=string[i];
c!='\0'

```
I am a boy.
4 words
```

例6.9 有3个字符串,要求找出其中最大者。

- 解题思路：设一个二维的字符数组str,大小为3×10。每一行存放一个字符串

```
char str[3][10];
```

- 可以把str[0],str[1],str[2]看作3个一维字符数组，可以把它们如同一维数组那样进行处理

```
for (i=0;i<3;i++)
    gets (str[i]);
```

```
China
Japan
India
```

str[0]	C	h	i	n	a	\0	\0	\0	\0	\0
str[1]	J	a	p	a	n	\0	\0	\0	\0	\0
str[2]	I	n	d	i	a	\0	\0	\0	\0	\0

- 经过三次两两比较,就可得到值最大者,把它放在一维字符数组string中

```
if (strcmp(str[0],str[1])>0)
    strcpy(string,str[0]);
else
    strcpy(string,str[1]);
if (strcmp(str[2],string)>0)
    strcpy(string,str[2]);
```

```
#include<stdio.h>
#include<string.h>
int main ( )
{char str[3][10]; char string[10]; int i;
  for (i=0;i<3;i++)    gets (str[i]);
  if (strcmp(str[0],str[1])>0)
     strcpy(string,str[0]);
  else
     strcpy(string,str[1]);
  if (strcmp(str[2],string)>0)
     strcpy(string,str[2]);
  printf("\nthe largest:\n%s\n",string);
  return 0;
}
```

```
China
Japan
India

the largest:
Japan
```

第7章 用函数实现模块化程序设计

7.1为什么要用函数

- 问题：
  - 如果程序的功能比较多，规模比较大，把所有代码都写在**main**函数中，就会使主函数变得庞杂、头绪不清，阅读和维护变得困难
  - 有时程序中要多次实现某一功能，就需要多次重复编写实现此功能的程序代码，这使程序冗长，不精炼

7.1为什么要用函数

- 解决的方法：用模块化程序设计的思路
  - 采用“组装”的办法简化程序设计的过程
  - 事先编好一批实现各种不同功能的函数
  - 把它们保存在函数库中，需要时直接用

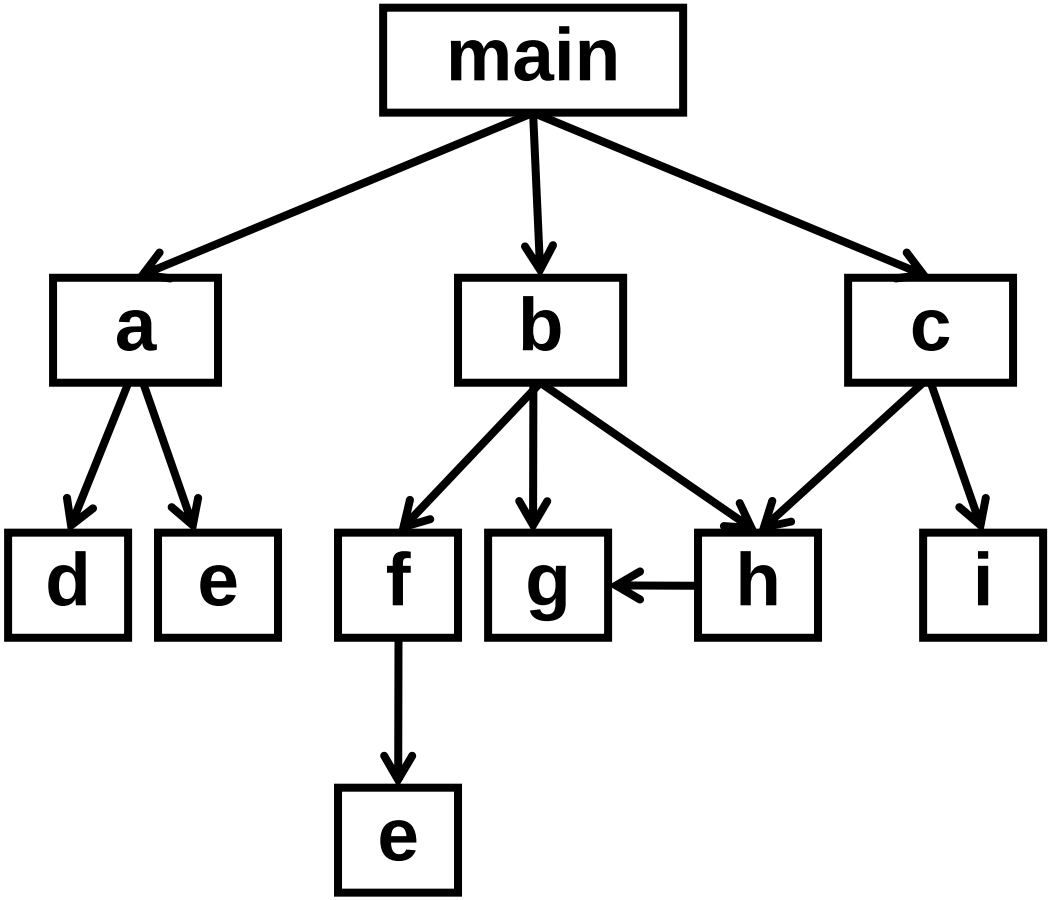
7.1为什么要用函数

- 解决的方法：用模块化程序设计的思路
  - 函数就是功能
  - 每一个函数用来实现一个特定的功能
  - 函数的名字应反映其代表的功能

7.1为什么要用函数

- 在设计一个较大的程序时，往往把它分为若干个程序模块，每一个模块包括一个或多个函数，每个函数实现一个特定的功能
- C程序可由一个主函数和若干个其他函数构成
- 主函数调用其他函数，其他函数也可以互相调用
- 同一个函数可以被一个或多个函数调用任意多次

7.1为什么要用函数



7.1为什么要用函数

- 可以使用库函数
- 可以使用自己编写的函数
- 在程序设计中要善于利用函数，可以减少重复编写程序段的工作量，同时可以方便地实现模块化的程序设计

7.1为什么要用函数

例7.1 输出以下的结果，用函数调用实现。

****

How do you do!

****

7.1为什么要用函数

- 解题思路：
  - 在输出的文字上下分别有一行“*”号，显然不必重复写这段代码，用一个函数**print_star**来实现输出一行“*”号的功能。
  - 再写一个**print_message**函数来输出中间一行文字信息
  - 用主函数分别调用这两个函数

```
#include <stdio.h>
int main()
{ void print_star();
   void print_message();
   print_star();   print_message();
   print_star();
   return 0;
}
void print_star()
{  printf( "**********\n" ); }
void print_message()
{  printf( "  How do you do!\n" );  }
```

输出16个*

输出一行文字

```
#include <stdio.h>
int main()
{ void print_star();
  void print_message();
  print_star();  print_message();
  print_star();
  return 0;
}
void print_star()
{  printf( "********************\n" ); }
void print_message()
{  printf( "  How do you do!\n" );  }
```

声明函数

定义函数

```
#include <stdio.h>
int main()
{ void print_star();
  void print_message();
  print_star();  print_message();
  print_star();
  return 0;
}
void print_star()
{  printf( "********************\n" ); }
void print_message()
{  printf( "  How do you do!\n" );  }
```

明德求新 尚用笃行

School of Software

- 说明：

  (1) 一个C程序由一个或多个程序模块组成，每一个程序模块作为一个源程序文件。对较大的程序，一般不希望把所有内容全放在一个文件中，而是将它们分别放在若干个源文件中，由若干个源程序文件组成一个**C**程序。这样便于分别编写、分别编译，提高调试效率。一个源程序文件可以为多个**C**程序共用。

- 说明：

  (2) 一个源程序文件由一个或多个函数以及其他有关内容（如预处理指令、数据声明与定义等）组成。一个源程序文件是一个编译单位，在程序编译时是以源程序文件为单位进行编译的，而不是以函数为单位进行编译的。

- 说明：

  (3) C程序的执行是从**main**函数开始的，如果在**main**函数中调用其他函数，在调用后流程返回到**main**函数，在**main**函数中结束整个程序的运行。

- 说明：

(4) 所有函数都是平行的，即在定义函数时是分别进行的，是互相独立的。一个函数并不从属于另一个函数，即函数不能嵌套定义。函数间可以互相调用，但不能调用**main**函数。**main**函数是被操作系统调用的。

- 说明：

  (5) 从用户使用的角度看，函数有两种。

  - 库函数，它是由系统提供的，用户不必自己定义而直接使用它们。应该说明，不同的C语言编译系统提供的库函数的数量和功能会有一些不同，当然许多基本的函数是共同的。
  - 用户自己定义的函数。它是用以解决用户专门需要的函数。

- 说明：

(6) 从函数的形式看，函数分两类。

① 无参函数。无参函数一般用来执行指定的一组操作。无参函数可以带回或不带回函数值，但一般以不带回函数值的居多。

② 有参函数。在调用函数时，主调函数在调用被调用函数时，通过参数向被调用函数传递数据，一般情况下，执行被调用函数时会得到一个函数值，供主调函数使用。

明德求新 尚用笃行

7.2 怎样定义函数

7.2.1 为什么要定义函数

7.2.2 定义函数的方法

School of Software

7.2.1 为什么要定义函数

- C语言要求，在程序中用到的所有函数，必须“先定义，后使用”
- 指定函数名字、函数返回值类型、函数实现的功能以及参数的个数与类型，将这些信息通知编译系统。

7.2.1 为什么要定义函数

- 指定函数的名字，以便以后按名调用
- 指定函数类型，即函数返回值的类型
- 指定函数参数的名字和类型，以便在调用函数时向它们传递数据
- 指定函数的功能。这是最重要的，这是在函数体中解决的

7.2.1 为什么要定义函数

- 对于库函数，程序设计者只需用**#include**指令把有关的头文件包含到本文件模块中即可
- 程序设计者需要在程序中自己定义想用的而库函数并没有提供的函数

7.2.2 定义函数的方法

1.定义无参函数

定义无参函数的一般形式为:

```
类型名  函数名()
 {
     函数体
 }
```

```
类型名  函数名(void)
 {
     函数体
 }
```

包括声明部分和语句部分

7.2.2 定义函数的方法

1.定义无

定义无参函数的一般形式为:

指定函数值的类型

```
类型名  函数名()
 {
     函数体
 }
```

```
类型名  函数名(void)
 {
     函数体
 }
```

7.2.2 定义函数的方法

2.定义有参函数

定义有参函数的一般形式为:

```
类型名 函数名（形式参数表列）
{
    函数体
}
```

7.2.2 定义函数的方法

3. 定义空函数

定义空函数的一般形式为:

类型名 函数名（ ）

{ }

- 先用空函数占一个位置，以后逐步扩充
- 好处：程序结构清楚，可读性好，以后扩充新功能方便，对程序结构影响不大

7.3 调用函数

7.3.1函数调用的形式

- 函数调用的一般形式为：

  函数名（实参表列）

- 如果是调用无参函数，则“实参表列”可以没有，但括号不能省略
- 如果实参表列包含多个实参，则各参数间用逗号隔开

7.3.1函数调用的形式

- 按函数调用在程序中出现的形式和位置来分，可以有以下3种函数调用方式：

1．函数调用语句

- 把函数调用单独作为一个语句

  如printf_star()；

- 这时不要求函数带回值，只要求函数完成一定的操作

7.3.1函数调用的形式

- 按函数调用在程序中出现的形式和位置来分，可以有以下3种函数调用方式：

2．函数表达式

- 函数调用出现在另一个表达式中

  如c=max(a,b);

- 这时要求函数带回一个确定的值以参加表达式的运算

7.3.1函数调用的形式

- 按函数调用在程序中出现的形式和位置来分，可以有以下3种函数调用方式：

3．函数参数

- 函数调用作为另一函数调用时的实参

  如m＝max(a,max(b,c));

- 其中max(b,c)是一次函数调用，它的值作为max另一次调用的实参

7.3.2 函数调用时的数据传递

1.形式参数和实际参数

- 在调用有参函数时，主调函数和被调用函数之间有数据传递关系
- 定义函数时函数名后面的变量名称为“形式参数”（简称“形参”）
- 主调函数中调用一个函数时，函数名后面参数称为“实际参数”（简称“实参”）
- 实际参数可以是常量、变量或表达式

7.3.2 函数调用时的数据传递

2. 实参和形参间的数据传递

- 在调用函数过程中，系统会把实参的值传递给被调用函数的形参
- 或者说，形参从实参得到一个值
- 该值在函数调用期间有效，可以参加被调函数中的运算

7.3.2 函数调用时的数据传递

例7.2 输入两个整数，要求输出其中值较大者。要求用函数来找到大数。

- 解题思路：

  (1)函数名应是见名知意，今定名为max

  (2) 由于给定的两个数是整数，返回主调函数的值（即较大数）应该是整型

  (3)max函数应当有两个参数，以便从主函数接收两个整数，因此参数的类型应当是整型

7.3.2 函数调用时的数据传递

先编写max函数：

```
int max(int x,int y)
{
    int z;
    z=x>y?x:y;
    return(z);
}
```

7.3.2 函数调用时的数据传递

在max函数上面，再编写主函数

```
#include <stdio.h>
int main()
{ int max(int x,int y);   int a,b,c;
   printf( “two integer numbers: ");
   scanf( “%d,%d” ,&a,&b);
   c=max(a,b);
   printf( “max is %d\n” ,c);
}
```

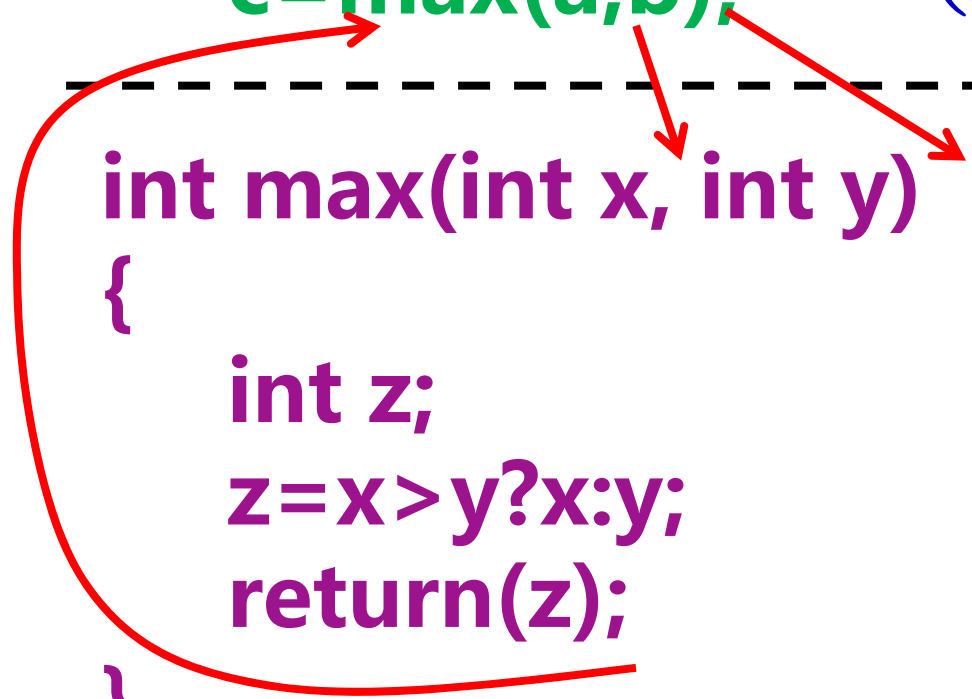
实参可以是常量、变量或表达式

```
two integer numbers:12,-34
max is 12
```

7.3.2 函数调用时的数据传递

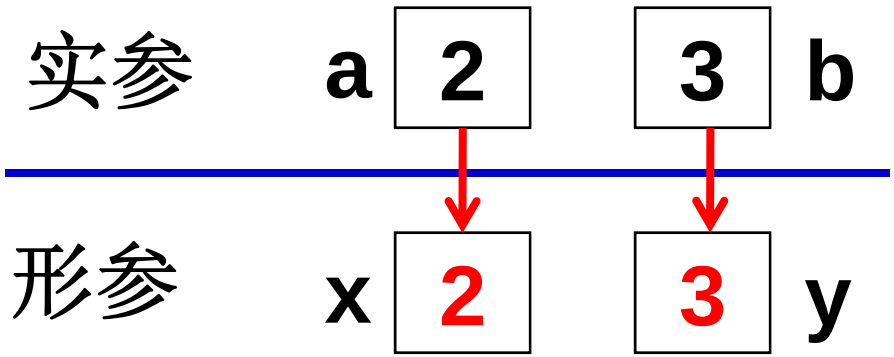
```
c=max(a,b);          （main函数）
```

```
int max(int x, int y)   （max函数）
{
    int z;
    z=x>y?x:y;
    return(z);
}
```



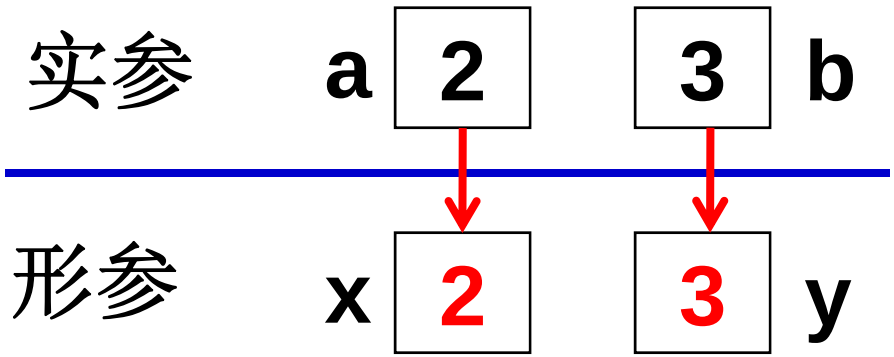
7.3.3 函数调用的过程

在定义函数中指定的形参，在未出现函数调用时，它们并不占内存中的存储单元。在发生函数调用时，函数max的形参被临时分配内存单元。



7.3.3 函数调用的过程

- 调用结束，形参单元被释放
- 实参单元仍保留并维持原值，没有改变
- 如果在执行一个被调用函数时，形参的值发生改变，不会改变主调函数的实参的值



7.3.4. 函数的返回值

□通常，希望通过函数调用使主调函数能得到一个确定的值，这就是函数值(函数的返回值)

(1)函数的返回值是通过函数中的**return**语句获得的。

- 一个函数中可以有一个以上的**return**语句，执行到哪一个**return**语句，哪一个就起作用
- **return**语句后面的括号可以不要

7.3.4. 函数的返回值

□通常，希望通过函数调用使主调函数能得到一个确定的值，这就是函数值(函数的返回值)

(2) 函数值的类型。应当在定义函数时指定函数值的类型

7.3.4. 函数的返回值

通常，希望通过函数调用使主调函数能得到一个确定的值，这就是函数值(函数的返回值)

(3)在定义函数时指定的函数类型一般应该和**return**语句中的表达式类型一致

- 如果函数值的类型和**return**语句中表达式的值不一致，则以函数类型为准

7.3.4. 函数的返回值

例7.3将例7.2稍作改动，将在max函数中定义的变量z改为float型。函数返回值的类型与指定的函数类型不同，分析其处理方法。

解题思路：如果函数返回值的类型与指定的函数类型不同，按照赋值规则处理。

```
#include <stdio.h>
int main()
 { int max(float x,float y);
    float a,b;  int c;
    scanf("%f,%f,",&a,&b);
    c=max(a,b);
    printf("max is %d\n",c);
    return 0;
 }
int max(float x,float y)
 {  float z;
     z=x>y?x:y;
     return( z ) ;
}
```

变为2

```
1.5,2.6
max is 2
```

1.5 2.6

2.6 ⇒ 2

7.4对被调用函数的声明和函数原型

- 在一个函数中调用另一个函数需要具备如下条件：

  (1) 被调用函数必须是已经定义的函数（是库函数或用户自己定义的函数）

  (2) 如果使用库函数，应该在本文件开头加相应的**#include**指令

  (3) 如果使用自己定义的函数，而该函数的位置在调用它的函数后面，应该声明

7.4对被调用函数的声明和函数原型

例7.4 输入两个实数，用一个函数求出它们之和。

- 解题思路：用add函数实现。首先要定义add函数，它为float型，它应有两个参数，也应为float型。特别要注意的是：要对add函数进行声明。

7.4对被调用函数的声明和函数原型

- 分别编写**add**函数和**main**函数，它们组成一个源程序文件
- **main**函数的位置在**add**函数之前
- 在**main**函数中对**add**函数进行声明

```
#include <stdio.h>
int main()
{ float add(float x, float y);
  float a,b,c;
  printf("Please enter a and b:");
  scanf("%f,%f",&a,&b);
  c=add(a,b);
  printf("sum is %f\n",c);
  return 0;
}
```

对add函数声明

```
float add(float x,float y)
{ float z;
  z=x+y;
  return(z);
}
```

求两个实数之和，
函数值也是实型

```
#include <stdio.h>
int main()
{ float add(float x, float y);
   float a,b,c;
   printf("Please enter a and b:");
   scanf("%f,%f",&a,&b);
   c=add(a,b);
   printf("sum is %f\n",c);
   return 0;
}
```

只差一个分号

```
float add(float x,float y)
{ float z;
   z=x+y;
   return(z);
}
```

```
Please enter a and b:3.6,6.5
sum is 10.100000
```

```
#include <
int main()
{ float add(float x, float y);
   float a,b,c;
   printf("Please enter a and b:");
   scanf("%f,%f",&a,&b);
   c=add(a,b);
   printf("sum is %f\n",c);
   return 0;
}
```

调用add函数

```
float add(float x,float y)
{ float z;
   z=x+y;
   return(z);
}
```

定义add函数

- 函数原型的一般形式有两种：

  如 **float add(float x, float y);**

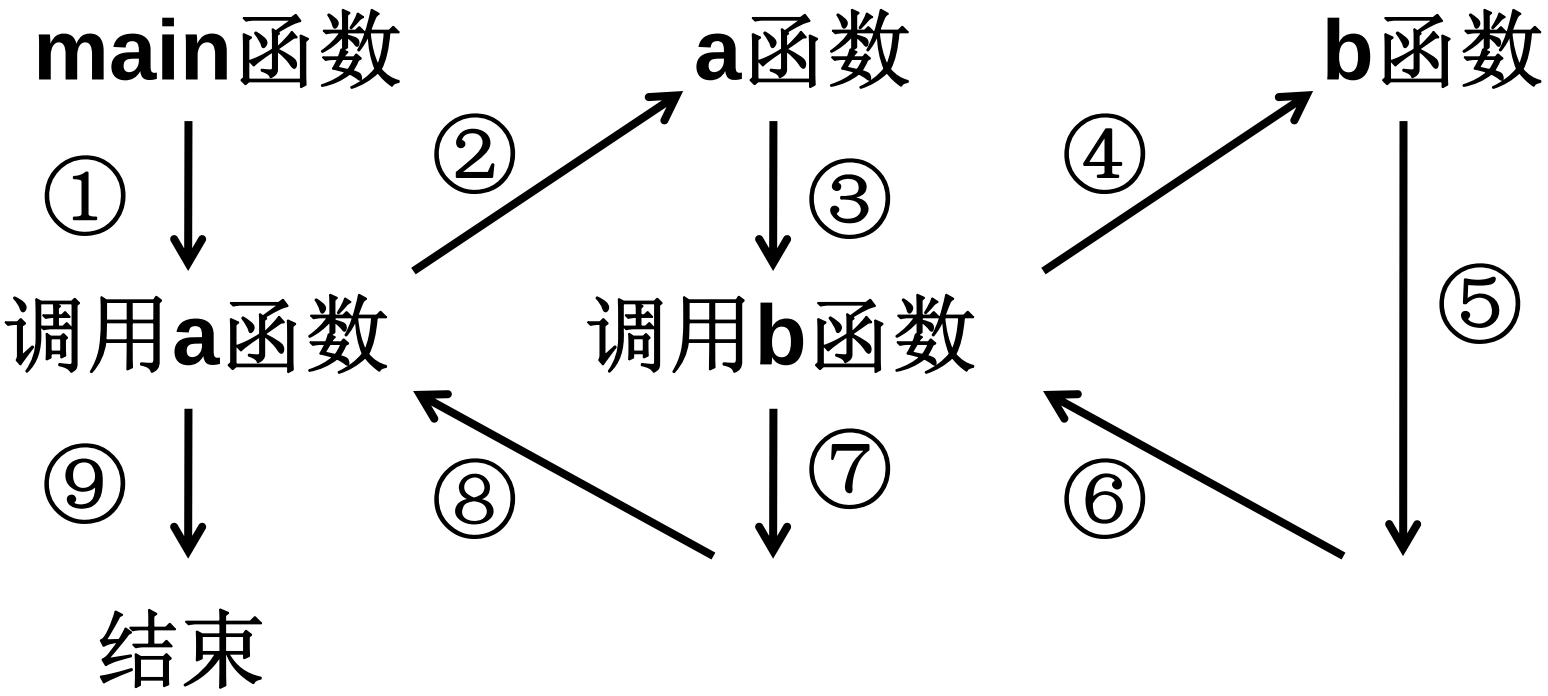
  float add(float, float);

- 原型说明可以放在文件的开头，这时所有函数都可以使用此函数

7.5 函数的嵌套调用

- C语言的函数定义是互相平行、独立的
- 即函数不能嵌套定义
- 但可以嵌套调用函数
- 即调用一个函数的过程中，又可以调用另一个函数

7.5 函数的嵌套调用



7.5 函数的嵌套调用

例7.5 输入4个整数，找出其中最大的数。用函数的嵌套调用来处理。

- 解题思路：
  - main中调用max4函数，找4个数中最大者
  - max4中再调用max2，找两个数中的大者
  - max4中多次调用max2，可找4个数中的大者，然后把它作为函数值返回main函数
  - main函数中输出结果

主函数

对max4 函数声明

```
#include <stdio.h>
int main()
{ int max4(int a,int b,int c,int d);
   int a,b,c,d,max;
   printf( “4 interger numbers:");
   scanf("%d%d%d%d",&a,&b,&c,&d);
   max=max4(a,b,c,d);
   printf("max=%d \n",max);
   return 0;
}
```

主函数

```
#include <stdio.h>
int main()
{ int max4(int a,int b,int c,int d);
   int a,b,c,d,max;
   printf( “4 interger numbers:
   scanf("%d%d%d%d",&a,&b,&c,&d);
   max=max4(a,b,c,d);
   printf("max=%d \n",max);
   return 0;
}
```

输入4个整数

主函数

```
#include <stdio.h>
int main()
{ int max4(int a,int b,int c,int d);
   int a,b,c,d,
   printf( “4 
   scanf("%d%d
   max=max4(a,b,c,d);
   printf("max=%d \n",max);
   return 0;
}
```

调用后肯定是4个数中最大者

输出最大者

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

对max2 函数声明

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

a,b中较大者

a,b,c中较大者

a,b,c,d中最大者

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
    int m;
    m=max2(a,b);
    m=max2(m,c);
    m=max2(m,d);
    return(m);
}
```

找a,b中较大者

max2函数

```
int max2(int a,int b)
{  if(a>=b)
       return a;
   else
       return b;
}
```

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
    int m;
    m=max2(a,b);
    m=max2(m,c);
    m=max2(m,d);
    return(m);
}
```

return(a>b?a:b);

max2函数

```
int max2(int a,int b)
{  if(a>=b)
       return a;
   else
       return b;
}
```

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
    int m;
    m=max2(a,b);
    m=max2(m,c);
    m=max2(m,d);
    return(m);
}
```

```
int max2(int a,int b)
{    return(a>b?a:b);   }
```

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

m=max2(max2(a,b),c);

```
int max2(int a,int b)
{    return(a>b?a:b);   }
```

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(in
    int m;
    m=max2(a,b);
    m=max2(m,c);
    m=max2(m,d);
    return(m);
}
```

m=max2(max2(max2(a,b),c),d);

```
int max2(int a,int b)
{    return(a>b?a:b);   }
```

max4函数

ruturn max2(max2(max2(a,b),c),d);

```
int max4
{  int max2(int a
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

```
int max2(int a,int b)
{    return(a>b?a:b);   }
```

```
4 interger numbers:12 45 -6 89
max=89
```

```
#include
int main()
{  ……
    max=max4(a,b,c,d);
   ……
}
```

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
    ruturn max2(max2(max2(a,b),c),d);
}
```

```
int max2(int a,int b)
{    return(a>b?a:b);   }
```

明德求新 尚用笃行

School of Software

软件学院

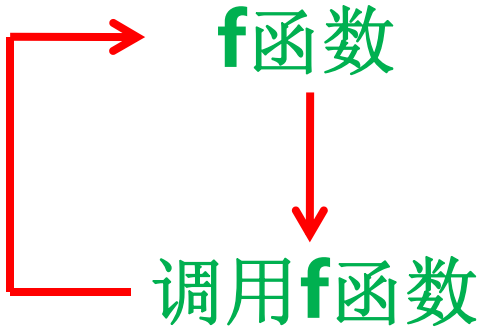
7.6 函数的递归调用

- 在调用一个函数的过程中又出现直接或间接地调用该函数本身，称为函数的递归调用。
- C语言的特点之一就在于允许函数的递归调用。

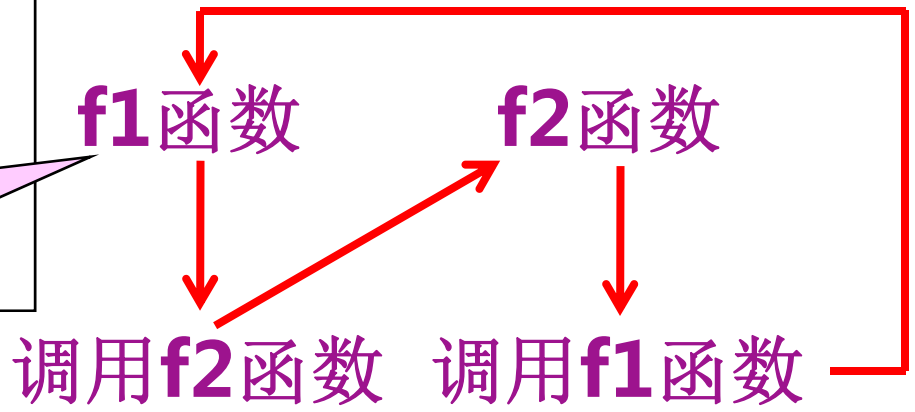
7.6 函数的递归调用

```
int f(int x)
{
    int y,z;
    z=f(y);
    return (2*z);
}
```

直接调用本函数



间接调用本函数



应使用if语句控制结束调用

7.6 函数的递归调用

例7.6 有5个学生坐在一起

- 问第5个学生多少岁？他说比第4个学生大2岁
- 问第4个学生岁数，他说比第3个学生大2岁
- 问第3个学生，又说比第2个学生大2岁
- 问第2个学生，说比第1个学生大2岁
- 最后问第1个学生，他说是10岁
- 请问第5个学生多大

7.6 函数的递归调用

- 解题思路：
  - 要求第 5 个年龄，就必须先知道第 4 个年龄
  - 要求第 4 个年龄必须先知道第 3 个年龄
  - 第 3 个年龄又取决于第 2 个年龄
  - 第 2 个年龄取决于第 1 个年龄
  - 每个学生年龄都比其前 1 个学生的年龄大 2

7.6 函数的递归调用

- 解题思路：

age(5)=age(4)+2

age(4)=age(3)+2

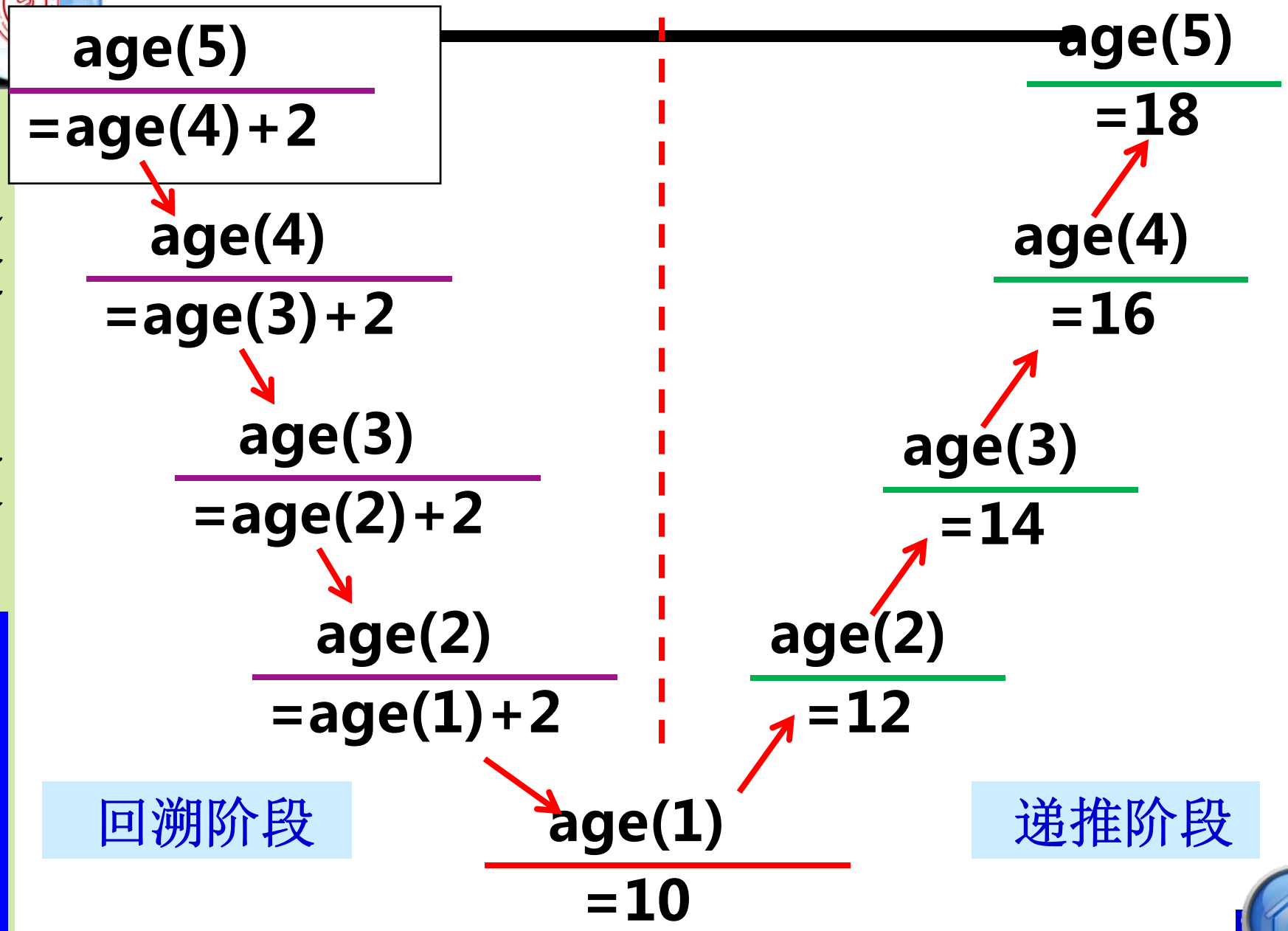
age(3)=age(2)+2

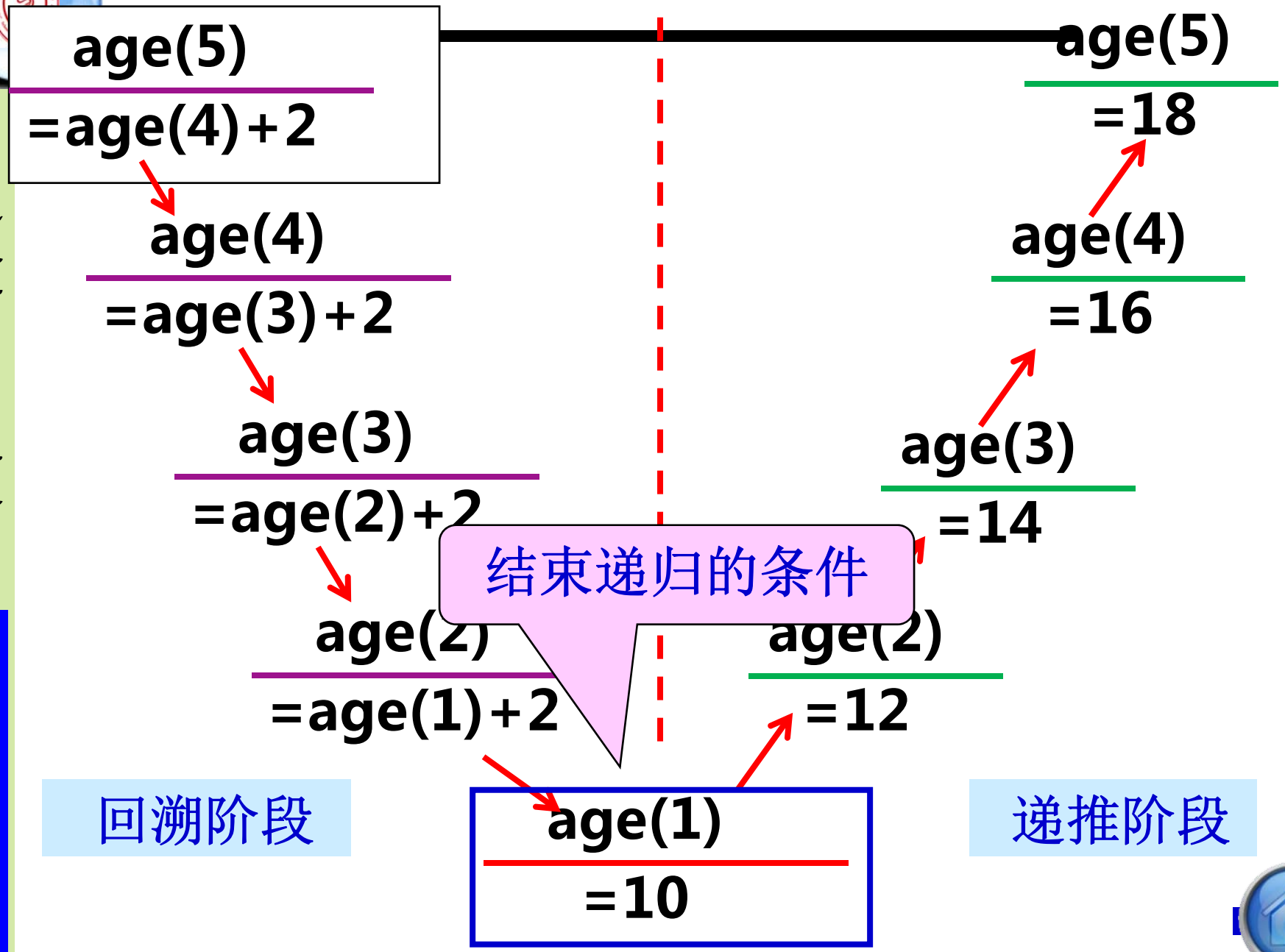
age(2)=age(1)+2

age(1)=10

$$age(n)=10 \qquad (n=1)$$

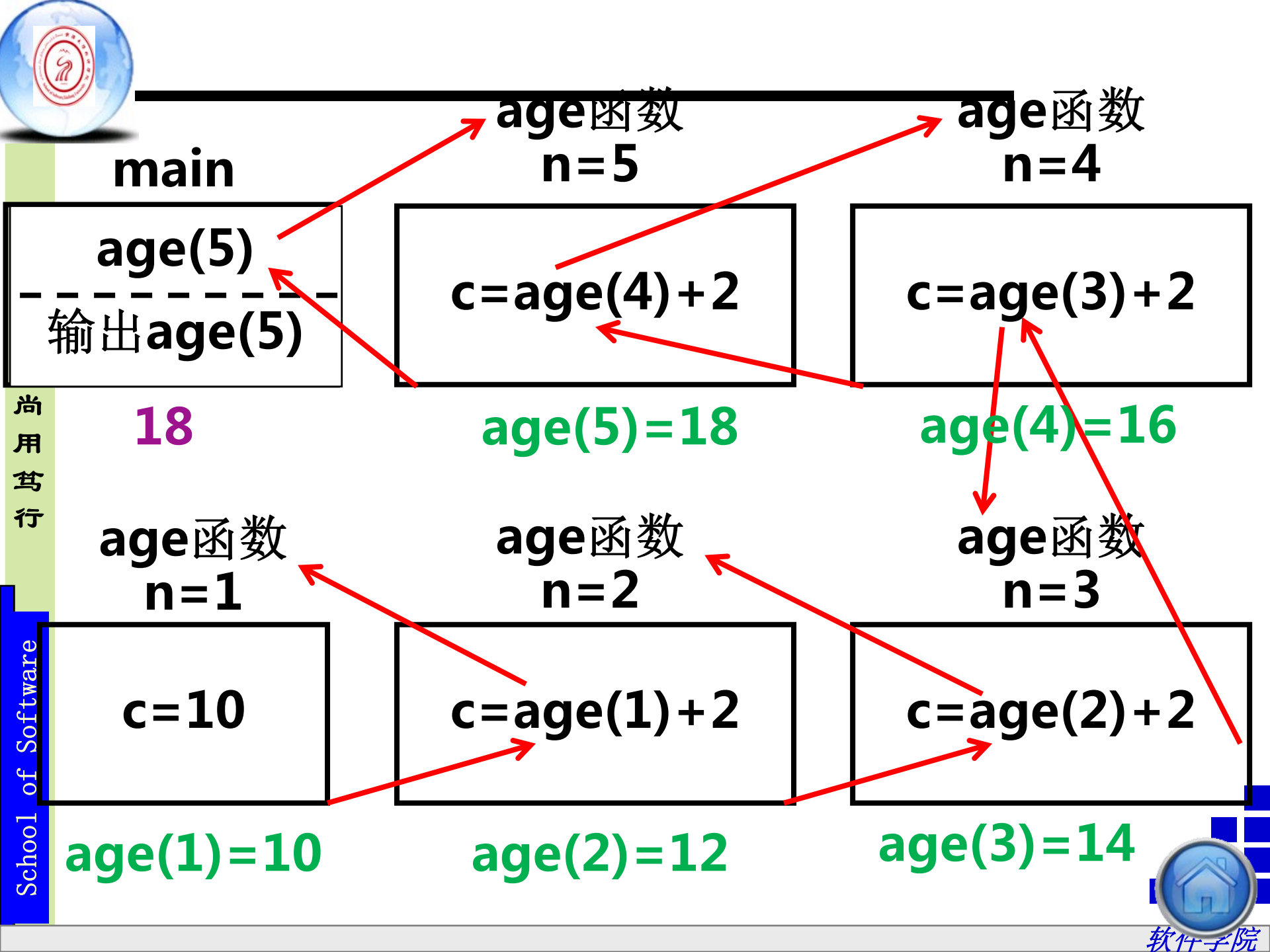
$$age(n)=age(n-1)+2 \qquad (n>1)$$


age(5)
=age(4)+2
age(4)
=age(3)+2
age(3)
=age(2)+2
age(2)
=age(1)+2
age(1)
=10
age(2)
=12
age(3)
=14
age(4)
=16
age(5)
=18
回溯阶段
递推阶段


age(5)
=age(4)+2
age(4)
=age(3)+2
age(3)
=age(2)+2
age(2)
=age(1)+2
age(1)
=10
age(2)
=12
age(3)
=14
age(4)
=16
age(5)
=18
结束递归的条件
回溯阶段
递推阶段

```
#include <stdio.h>
int main()
{ int age(int n);
   printf("NO.5,age:%d\n",age(5));
   return 0;
}
int age(int n)
{ int c;
   if(n==1)   c=10;
   else    c=age(n-1)+2;
   return(c);
}
```

```
NO.5,age:18
```



明德求新 尚用笃行

School of Software

例7.7 用递归方法求n！。

- 解题思路：
  - 求n！可以用递推方法：即从1开始，乘2，再乘3.....一直乘到n。
  - 递推法的特点是从一个已知的事实(如1!=1)出发，按一定规律推出下一个事实(如2!=1!*2)，再从这个新的已知的事实出发，再向下推出一个新的事实(3!=3*2!)。n!=n*(n-1)!。

例7.7 用递归方法求n！。

- 解题思路：
  - 求n！也可以用递归方法，即5！等于4！×5，而4！＝3！×4...，1！＝1
  - 可用下面的递归公式表示：

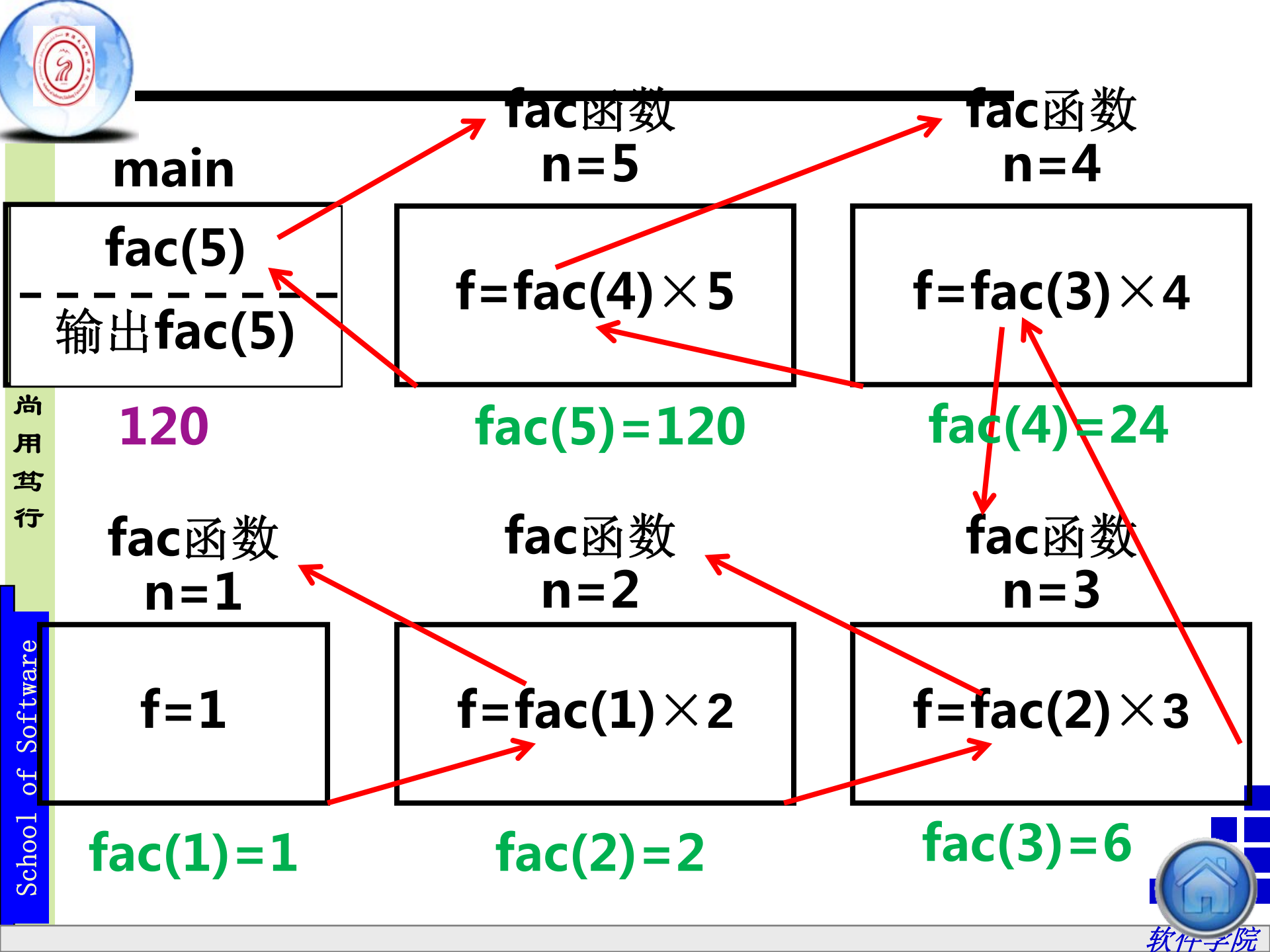
$$n! = \begin{cases} n! = 1 & (n = 0,1) \\ n \bullet (n-1) & (n > 1) \end{cases}$$

```
#include <stdio.h>
int main()
 {int fac(int n);
  int n;  int y;
  printf("input an integer number:");
  scanf("%d",&n);
  y=fac(n);
  printf("%d!=%d\n",n,y);
  return 0;
}
```

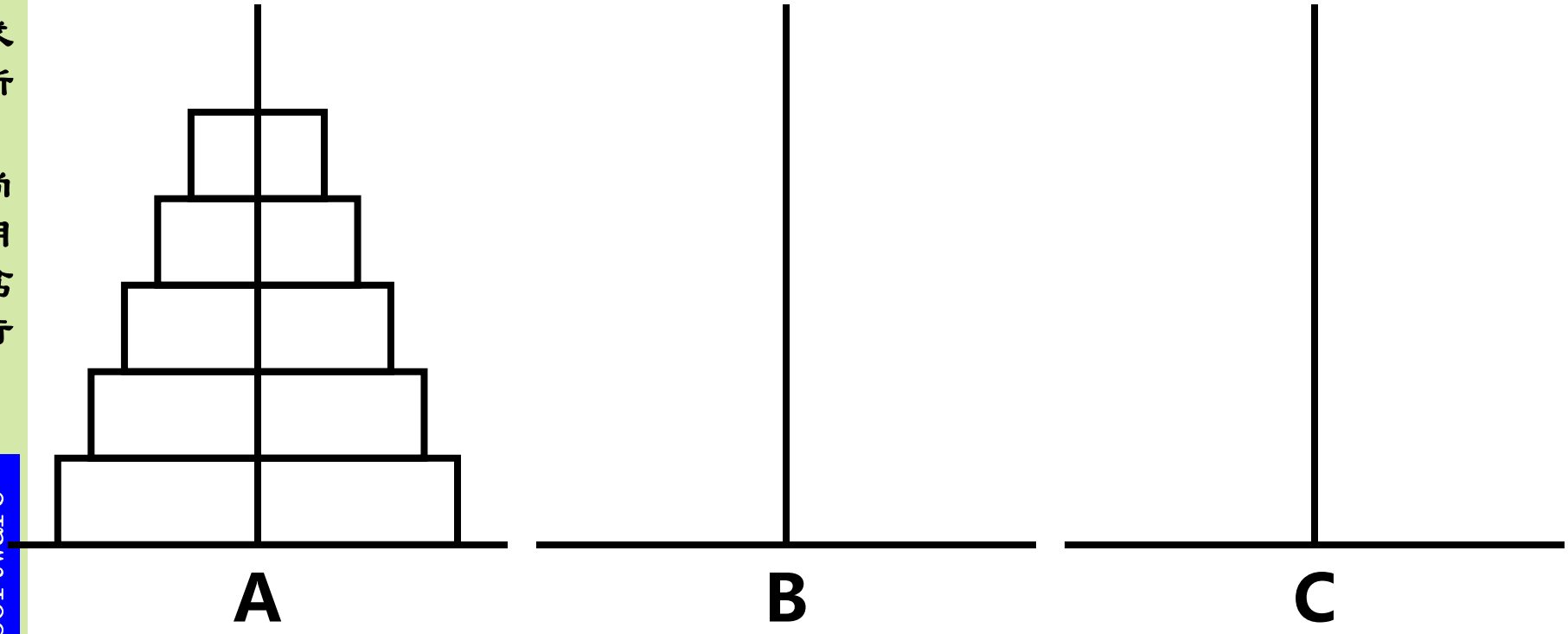


```
int fac(int n)
 {
    int f;
    if(n<0)
       printf("n<0,data error!");
    else if(n==0 || n==1)
       f=1;
    else  f=fac(n-1)*n;
    return(f);
 }
```


```
input an integer number:31
31!=738197504
```

例7.8 Hanoi（汉诺）塔问题。古代有一个梵塔，塔内有3个座A、B、C，开始时A座上有64个盘子，盘子大小不等，大的在下，小的在上。有一个老和尚想把这64个盘子从A座移到C座，但规定每次只允许移动一个盘，且在移动过程中在3个座上都始终保持大盘在下，小盘在上。在移动过程中可以利用B座。要求编程序输出移动一盘子的步骤。


A
B
C

明德求新 尚用笃行

School of Software

- 解题思路：
  - 要把64个盘子从A座移动到C座，需要移动大约2^{64} 次盘子。一般人是不可能直接确定移动盘子的每一个具体步骤的
  - 老和尚会这样想：假如有另外一个和尚能有办法将上面63个盘子从一个座移到另一座。那么，问题就解决了。此时老和尚只需这样做：

- 解题思路：

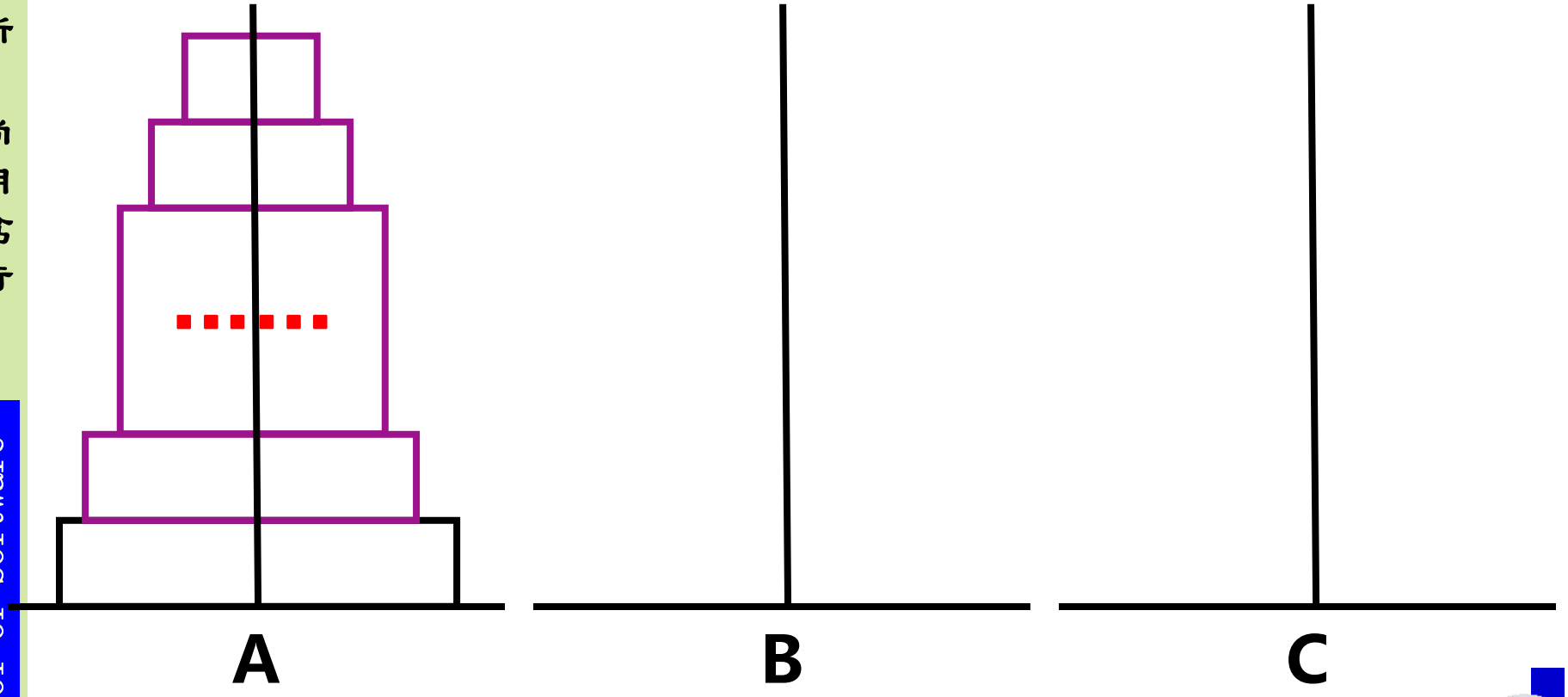
  (1) 命令第2个和尚将63个盘子从A座移到B座

  (2) 自己将1个盘子（最底下的、最大的盘子）从A座移到C座

  (3) 再命令第2个和尚将63个盘子从B座移到C座

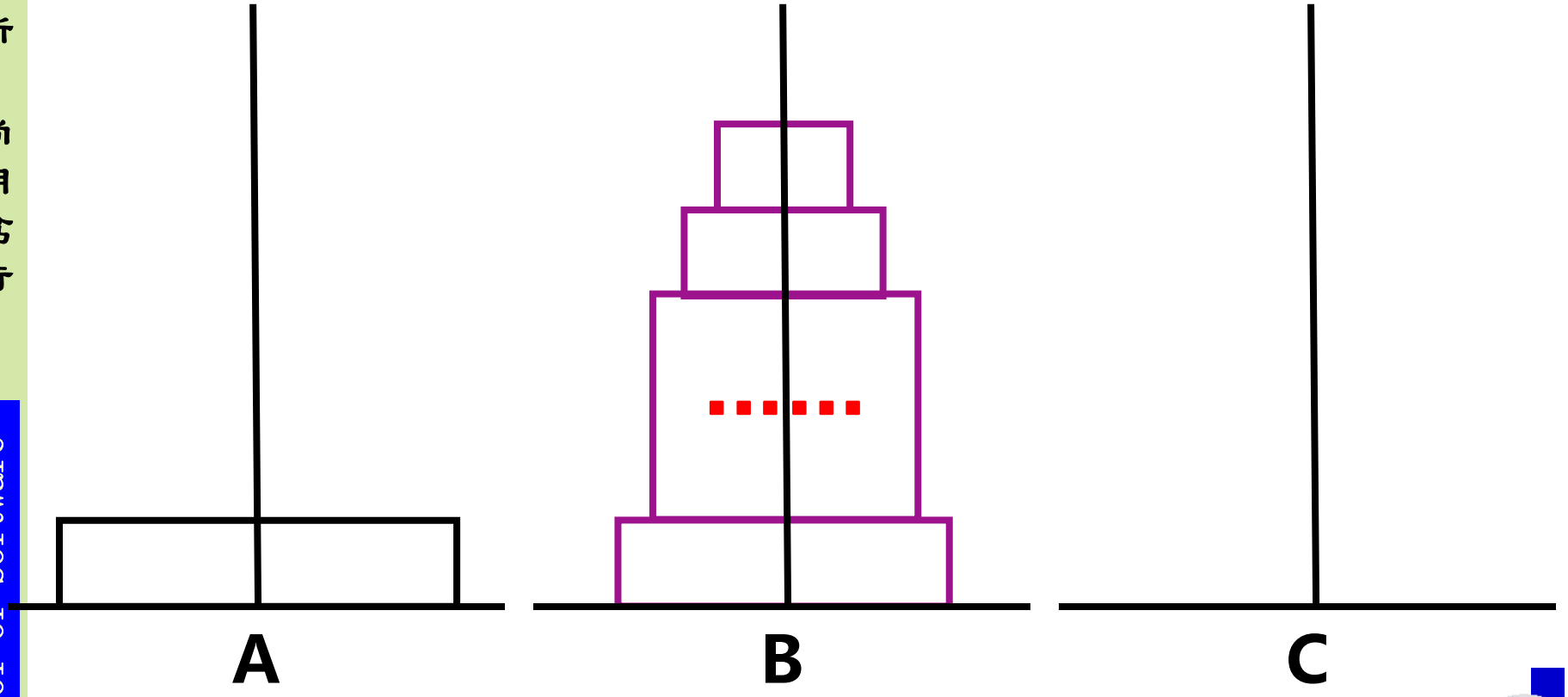
第1个和尚的做法

将63个从A到B



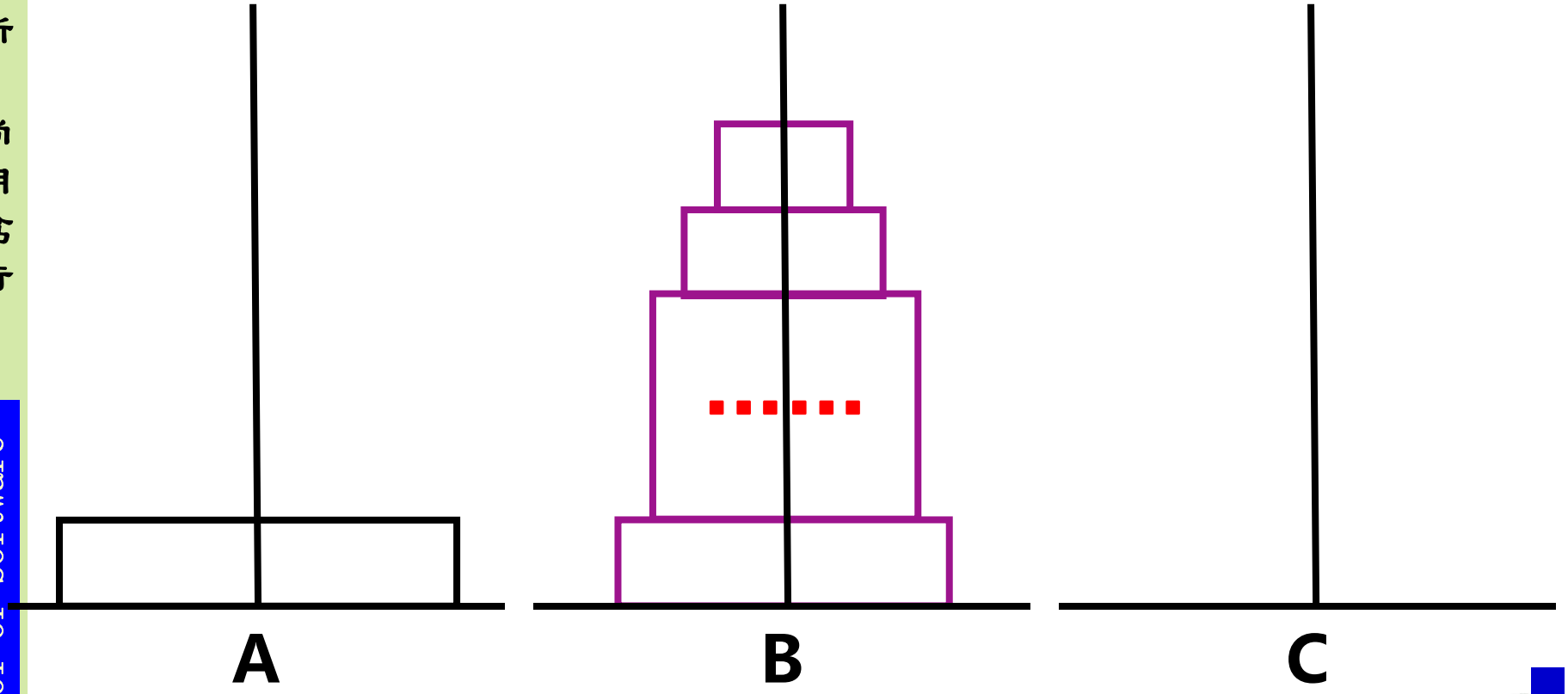
第1个和尚的做法

将63个从A到B



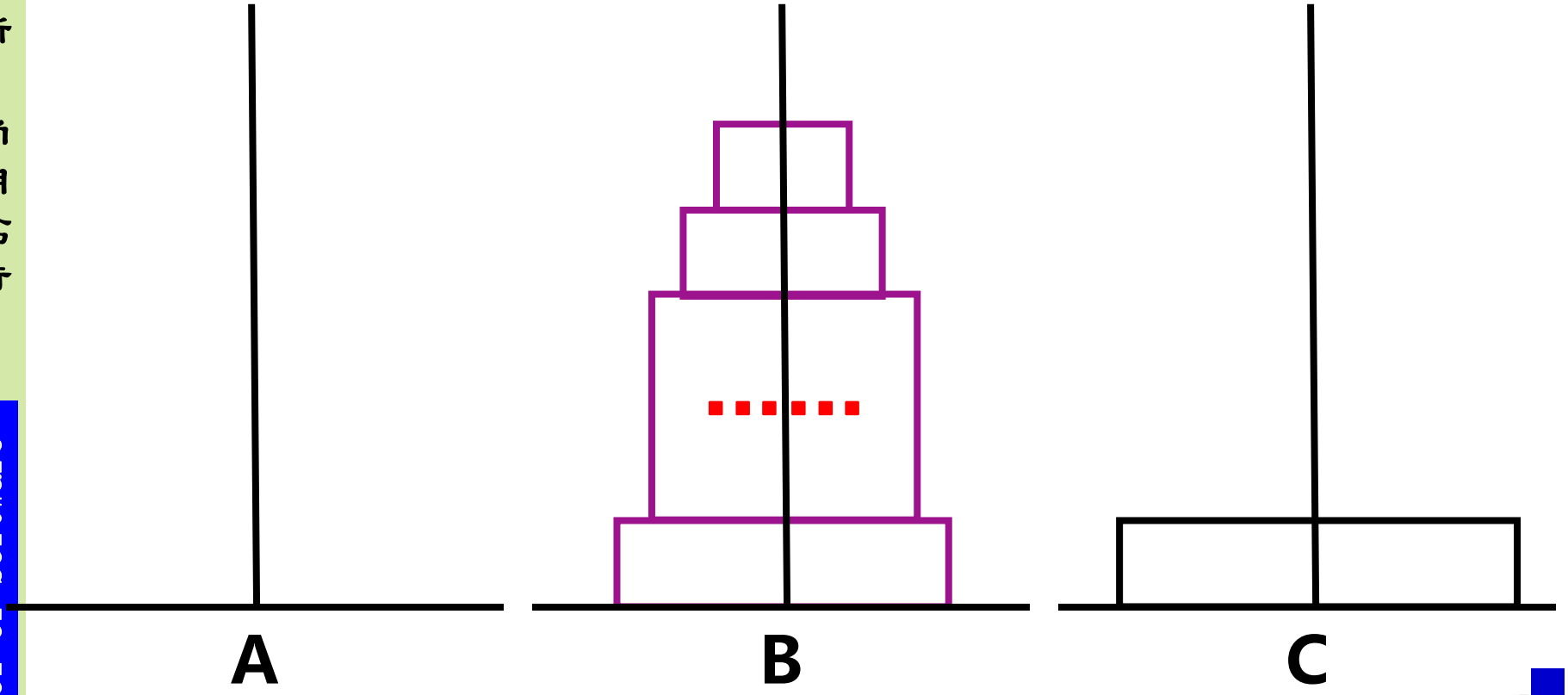
第1个和尚的做法

将1个从A到C



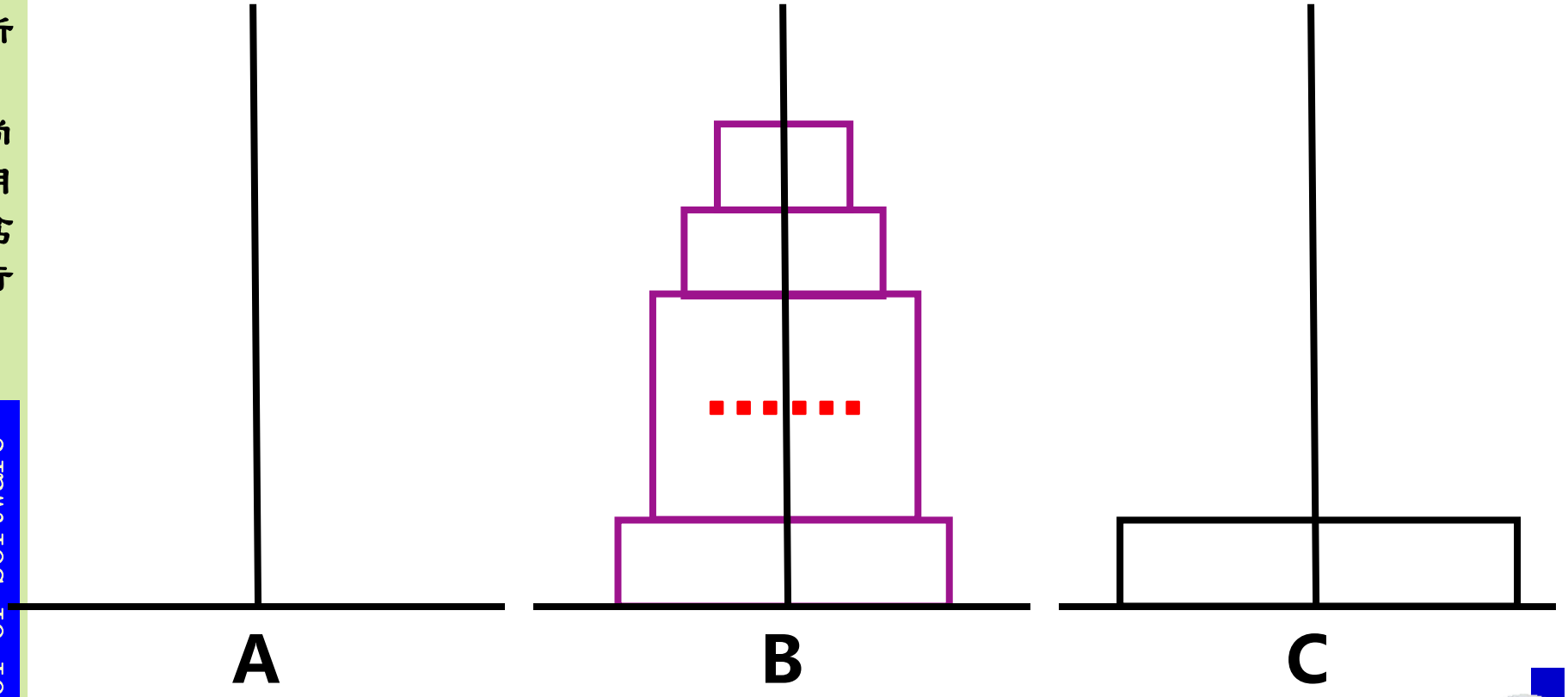
第1个和尚的做法

将1个从A到C



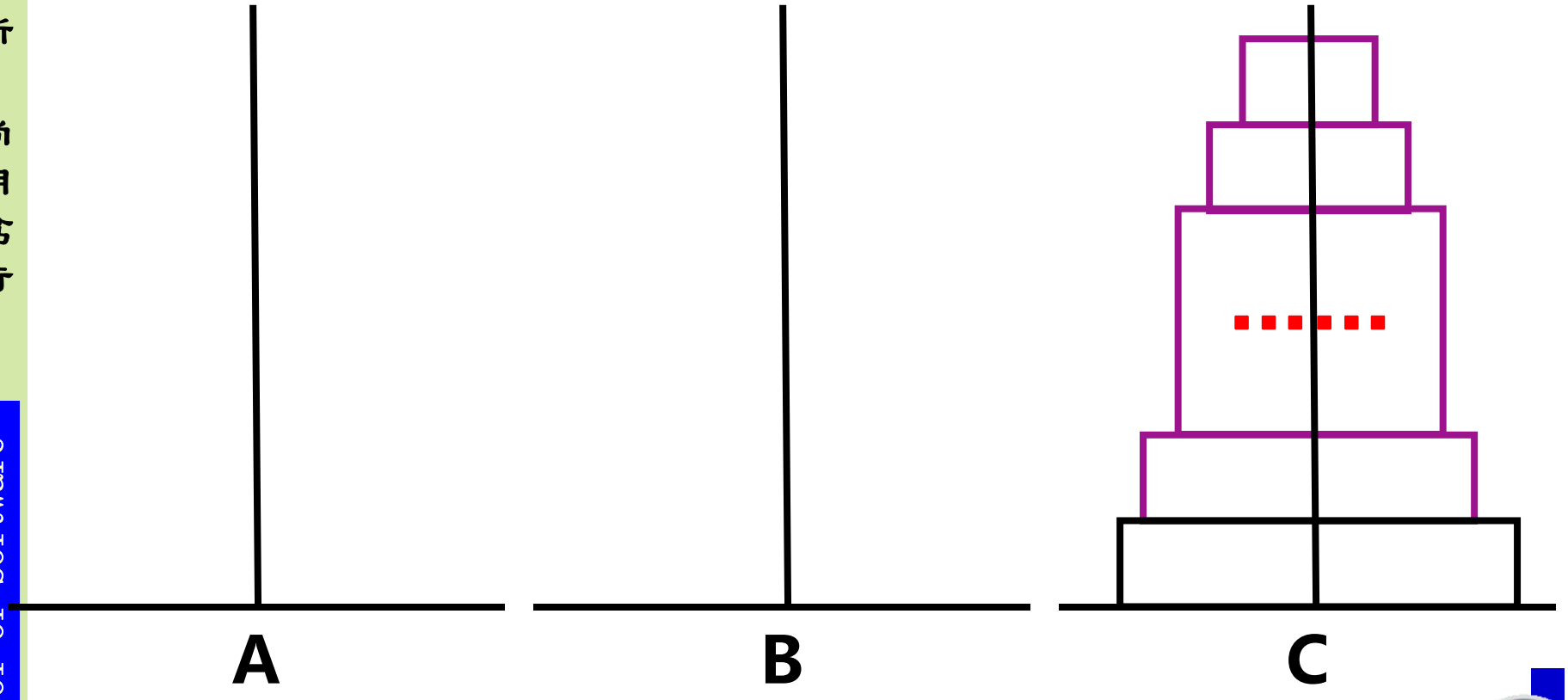
第1个和尚的做法

将63个从B到C



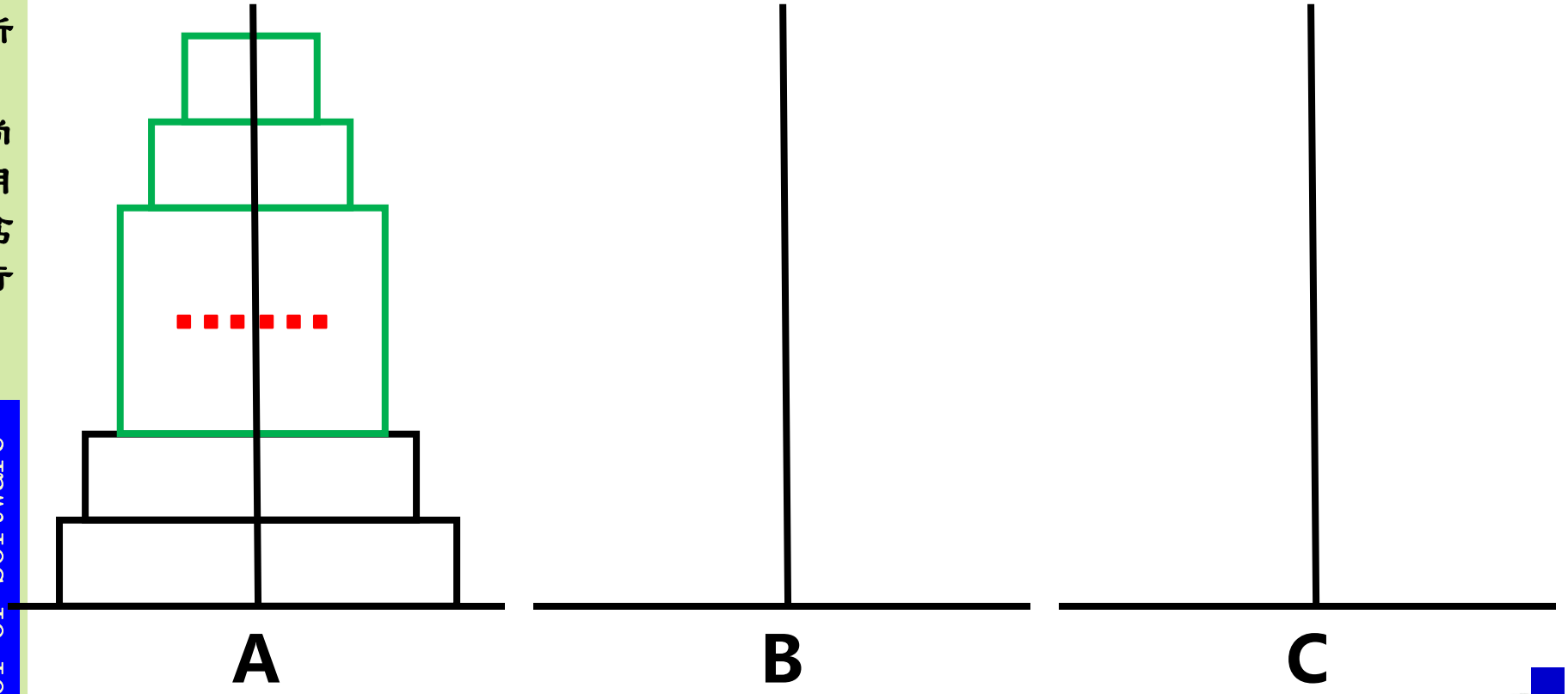
第1个和尚的做法

将63个从B到C



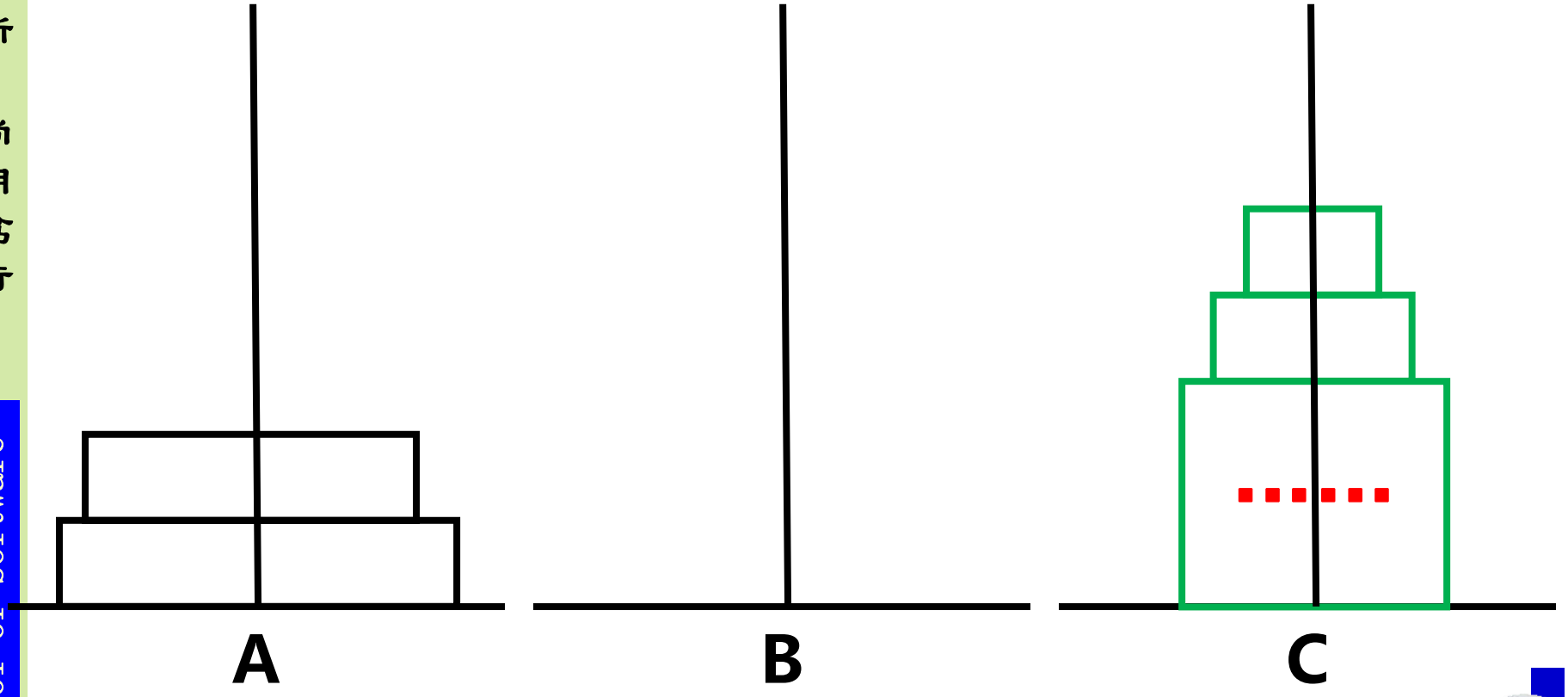
第2个和尚的做法

将62个从A到C



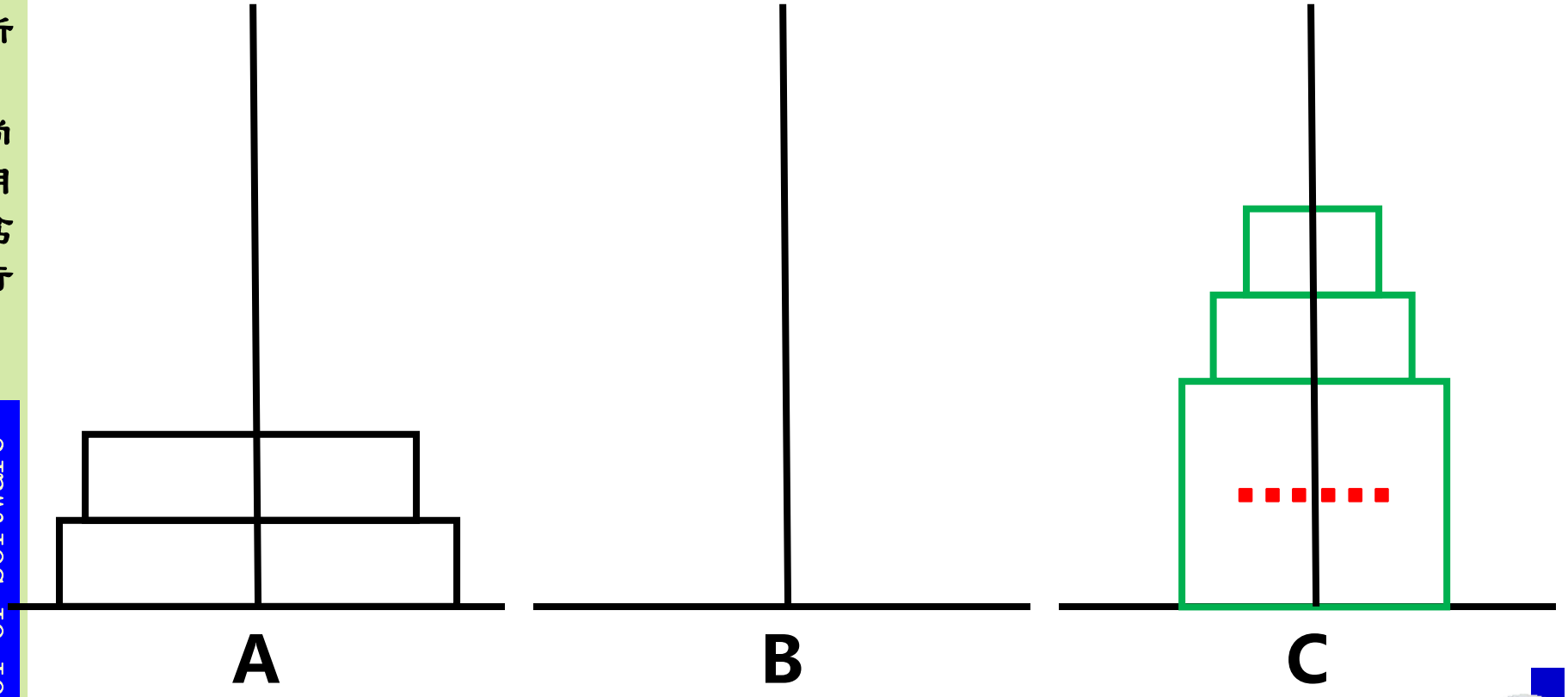
第2个和尚的做法

将62个从A到C



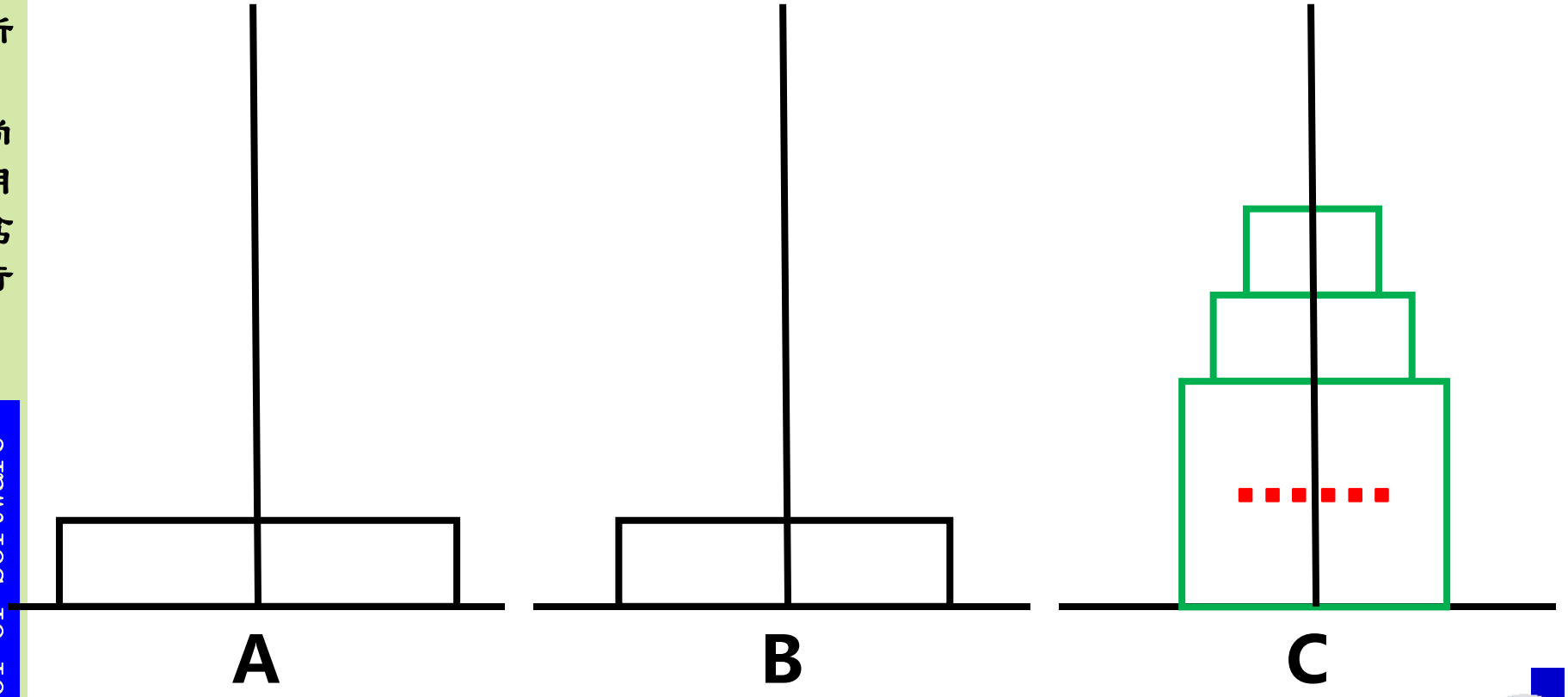
第2个和尚的做法

将1个从A到B



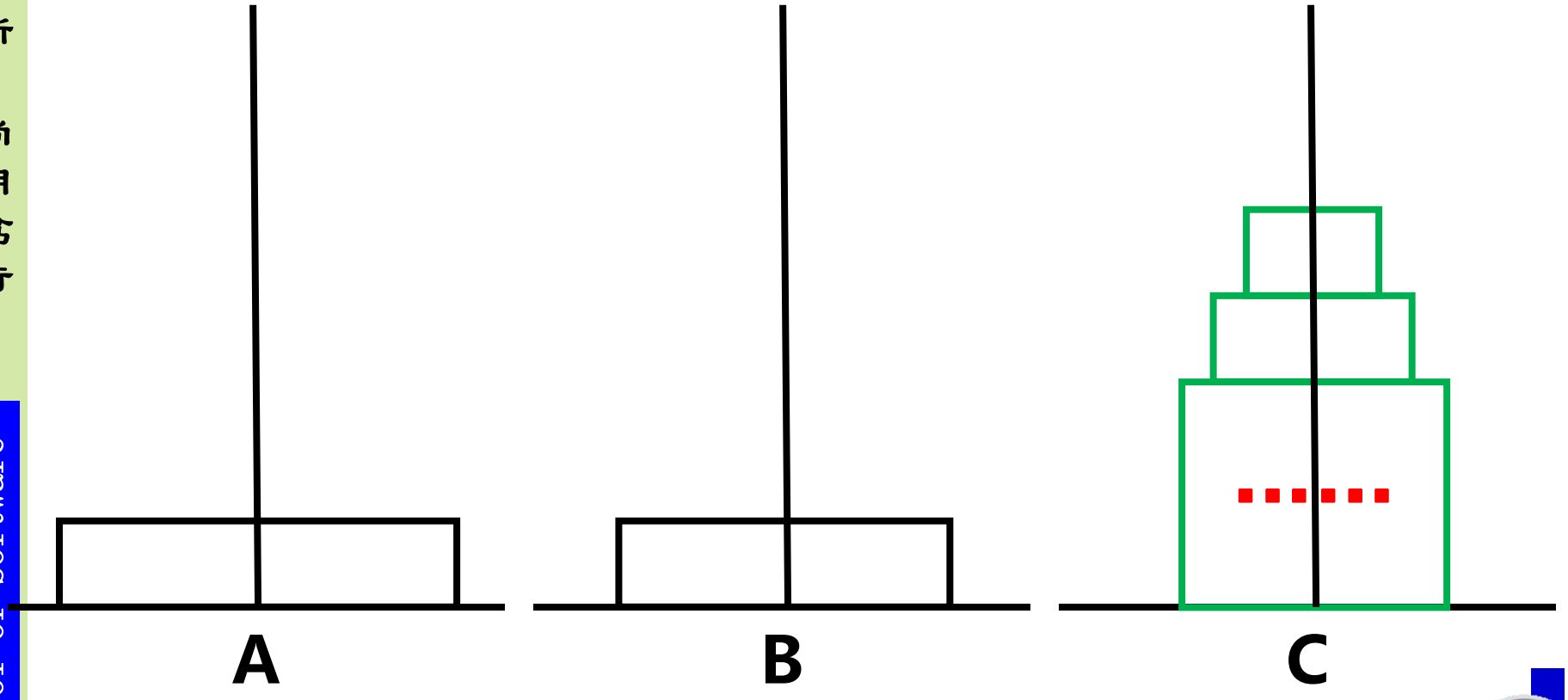
第2个和尚的做法

将1个从A到B



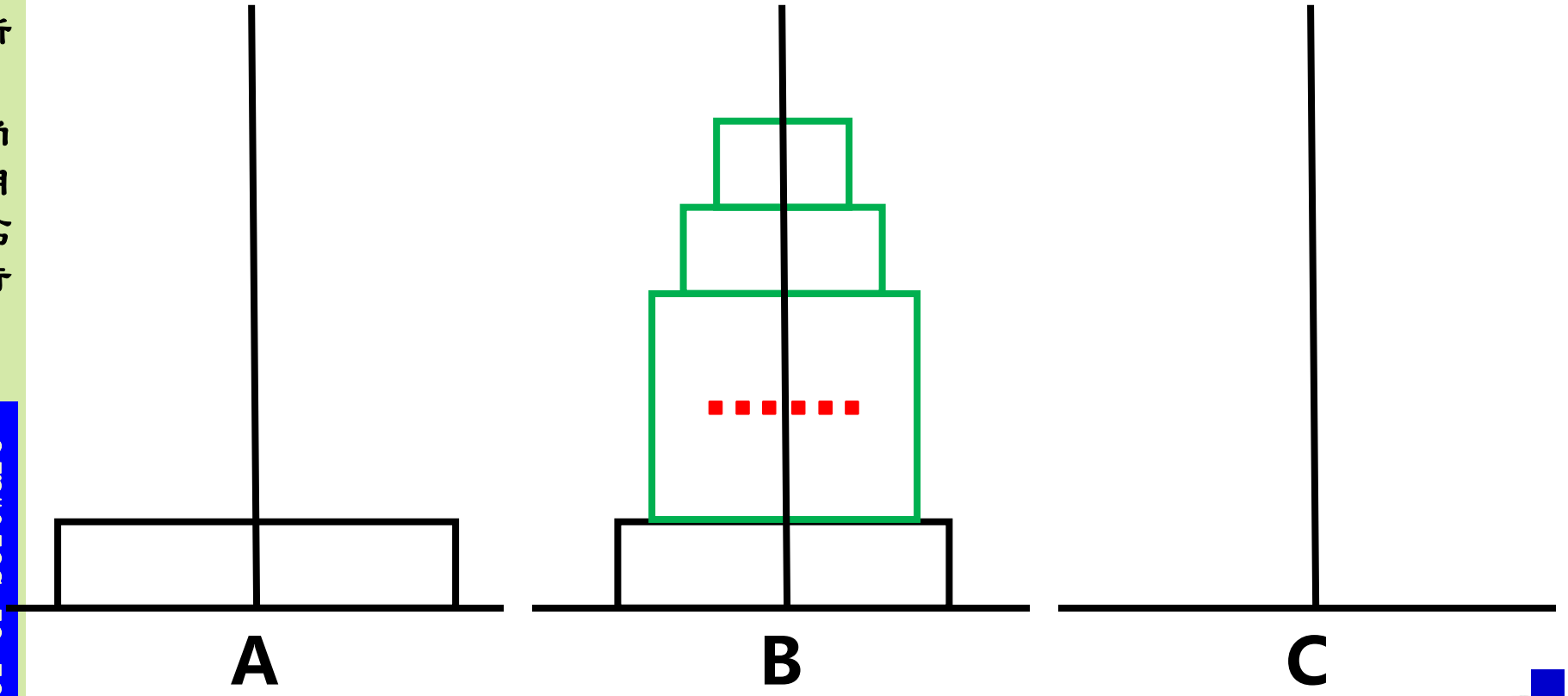
第2个和尚的做法

将62个从C到B



第2个和尚的做法

将62个从C到B



第3个和尚的做法

第4个和尚的做法

第5个和尚的做法

第6个和尚的做法

第7个和尚的做法

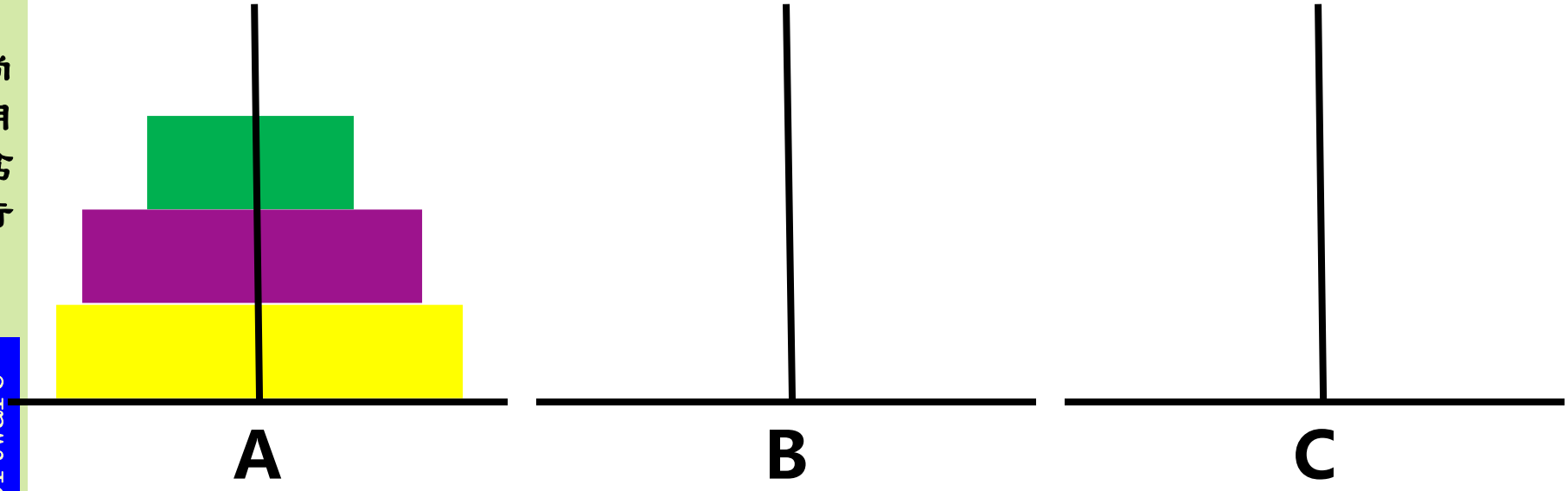
……

第63个和尚的做法

第64个和尚仅做：将1个从A移到C

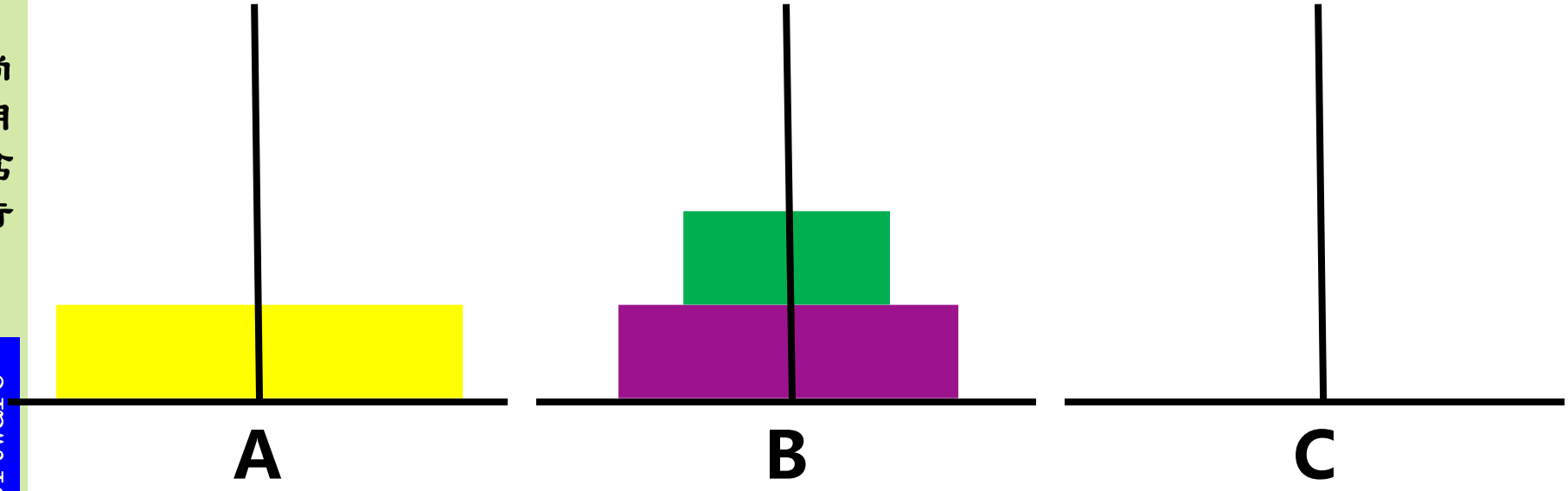
将3个盘子从A移到C的全过程

将2个盘子从A移到B



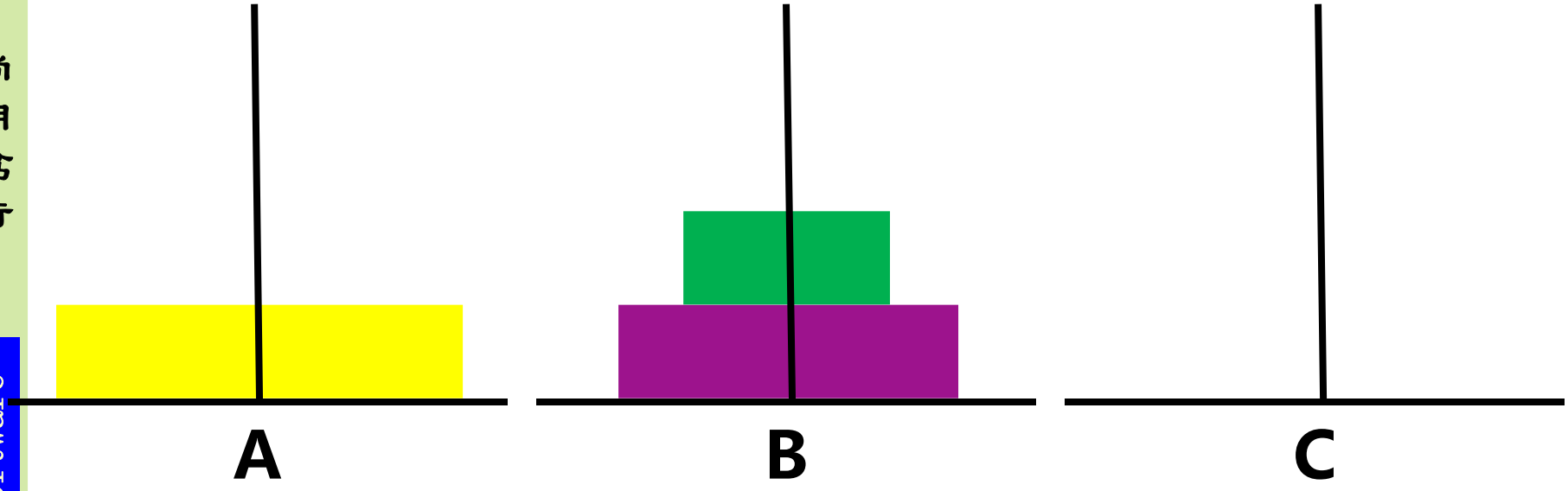
将3个盘子从A移到C的全过程

将2个盘子从A移到B



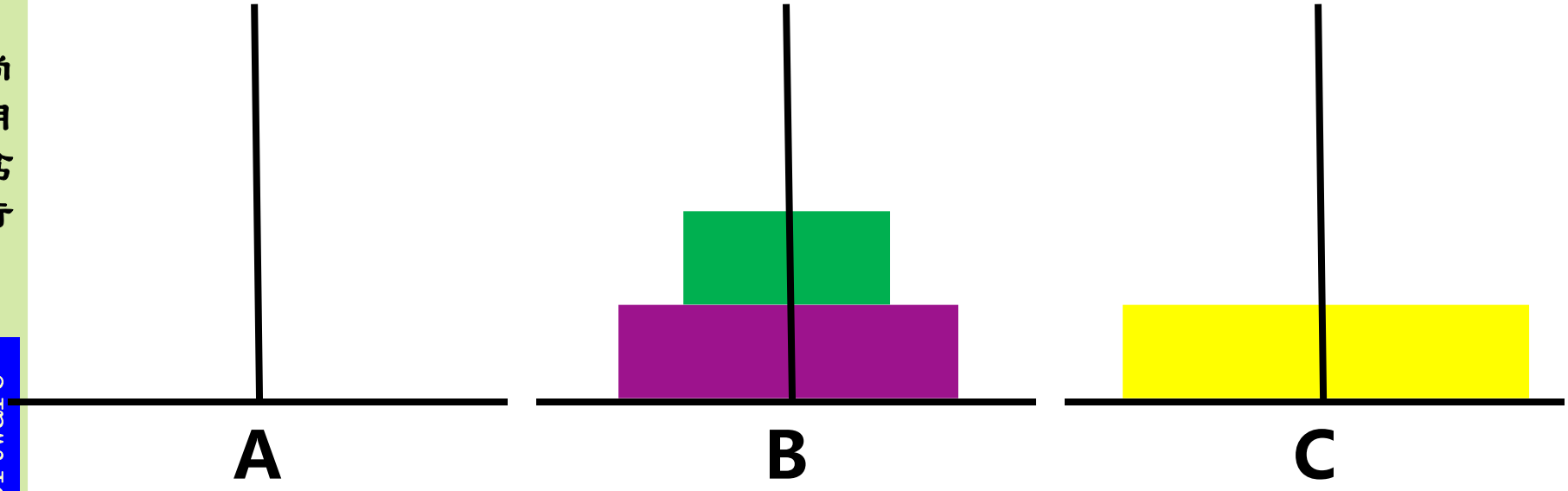
将3个盘子从A移到C的全过程

将1个盘子从A移到C



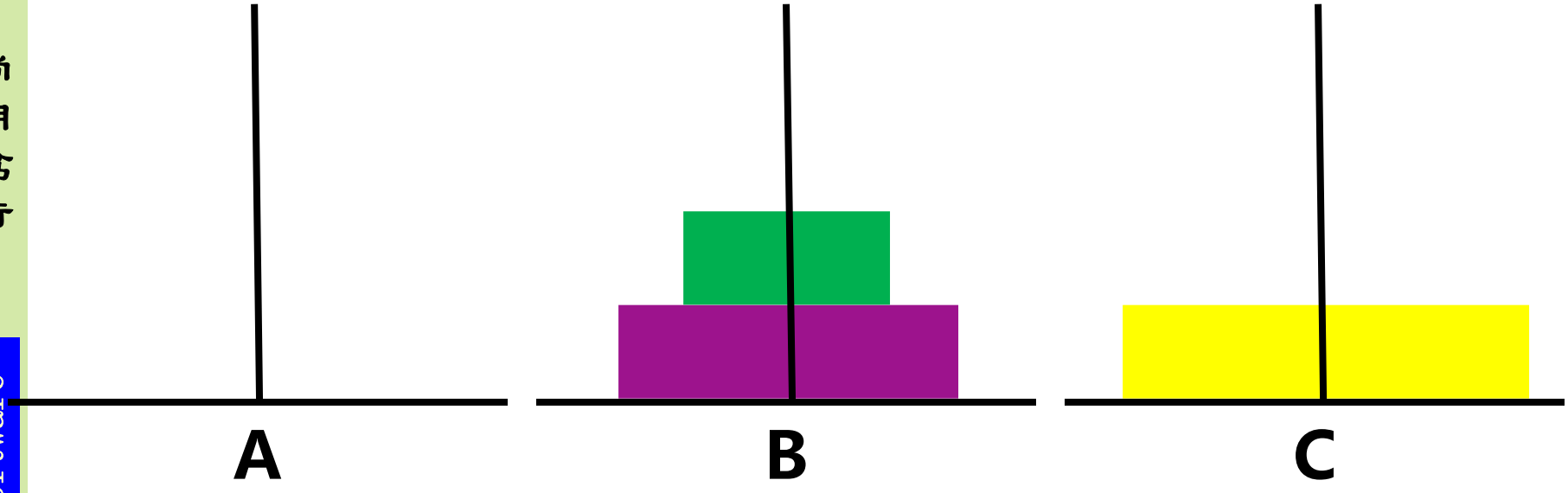
将3个盘子从A移到C的全过程

将1个盘子从A移到C



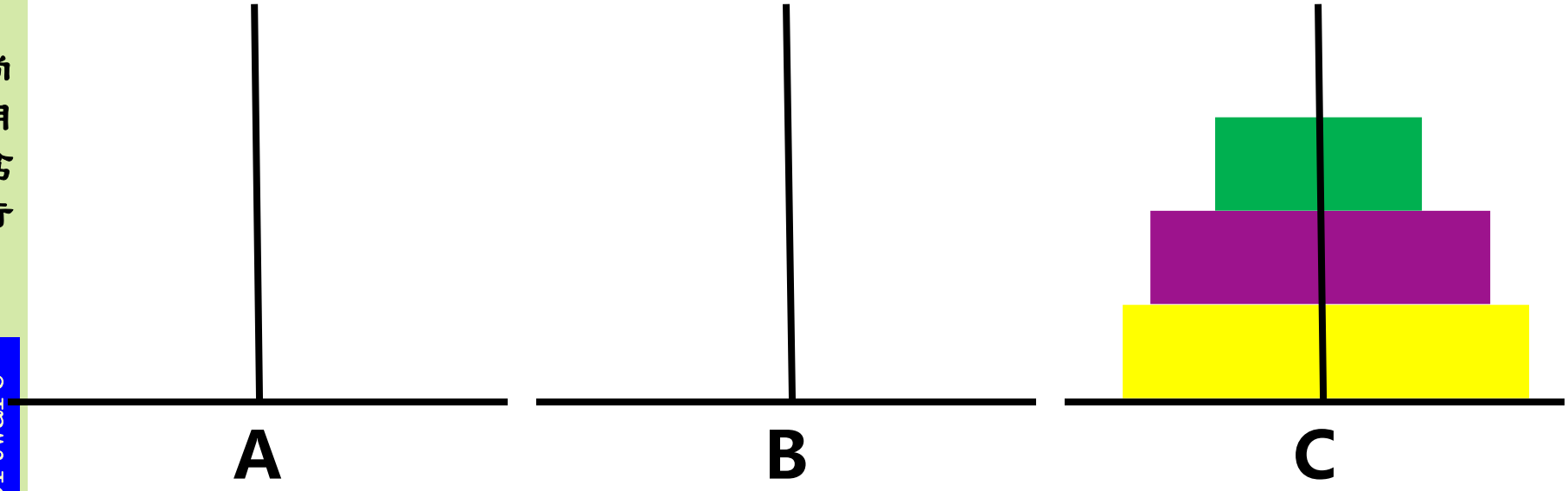
将3个盘子从A移到C的全过程

将2个盘子从B移到C



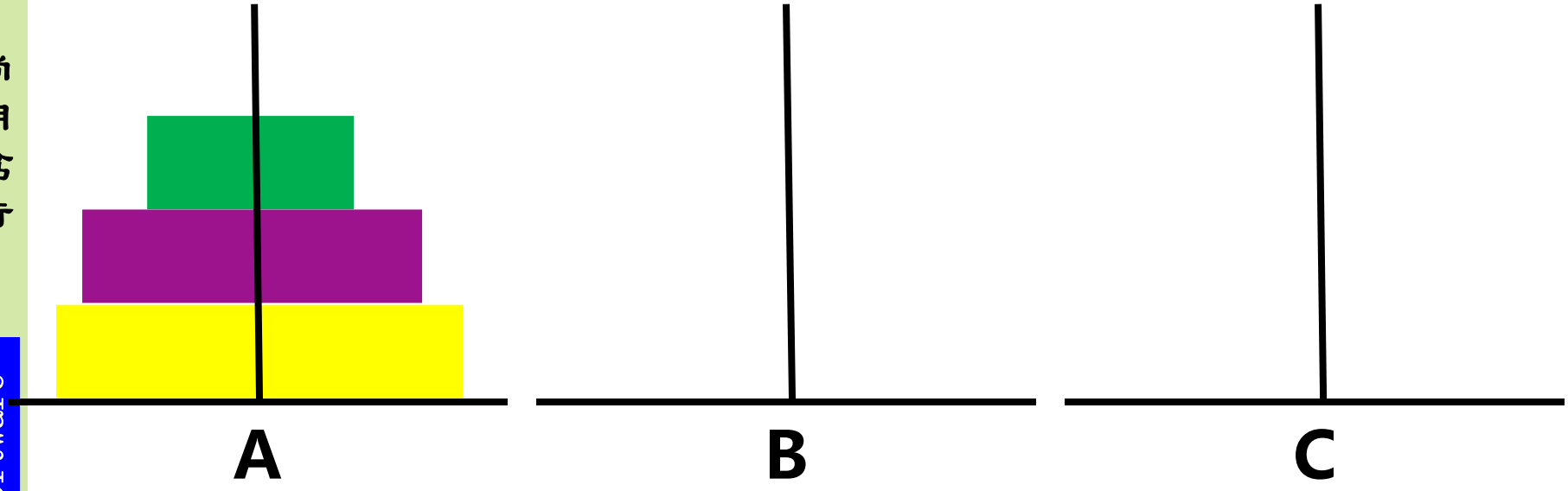
将3个盘子从A移到C的全过程

将2个盘子从B移到C



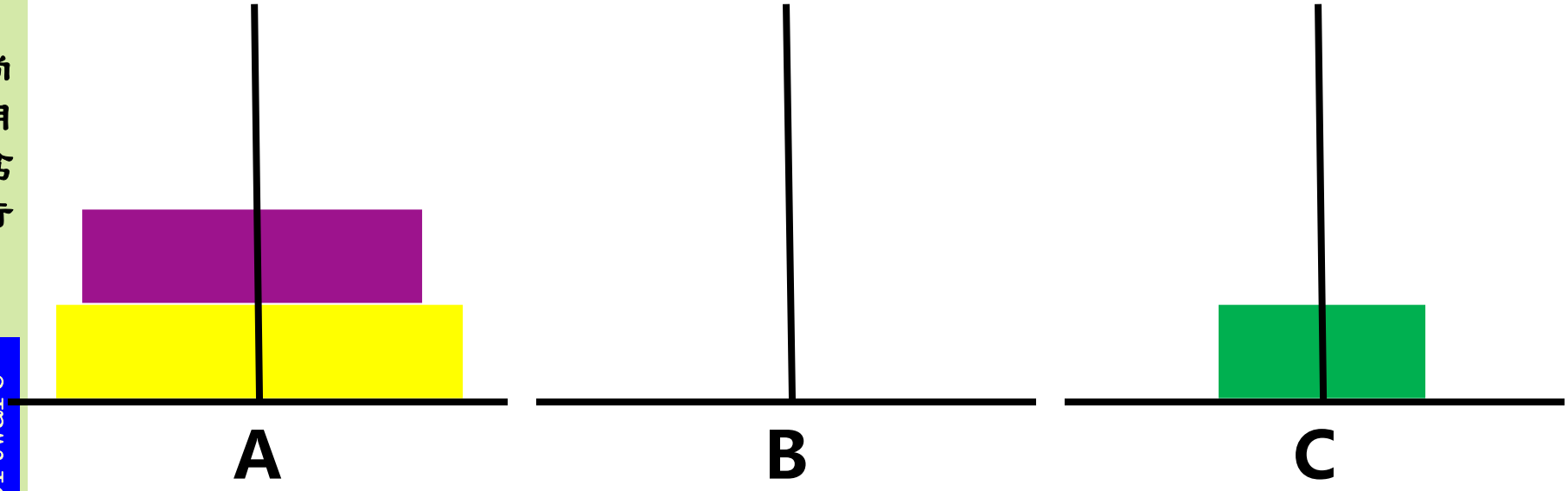
将2个盘子从A移到B的过程

将1个盘子从A移到C



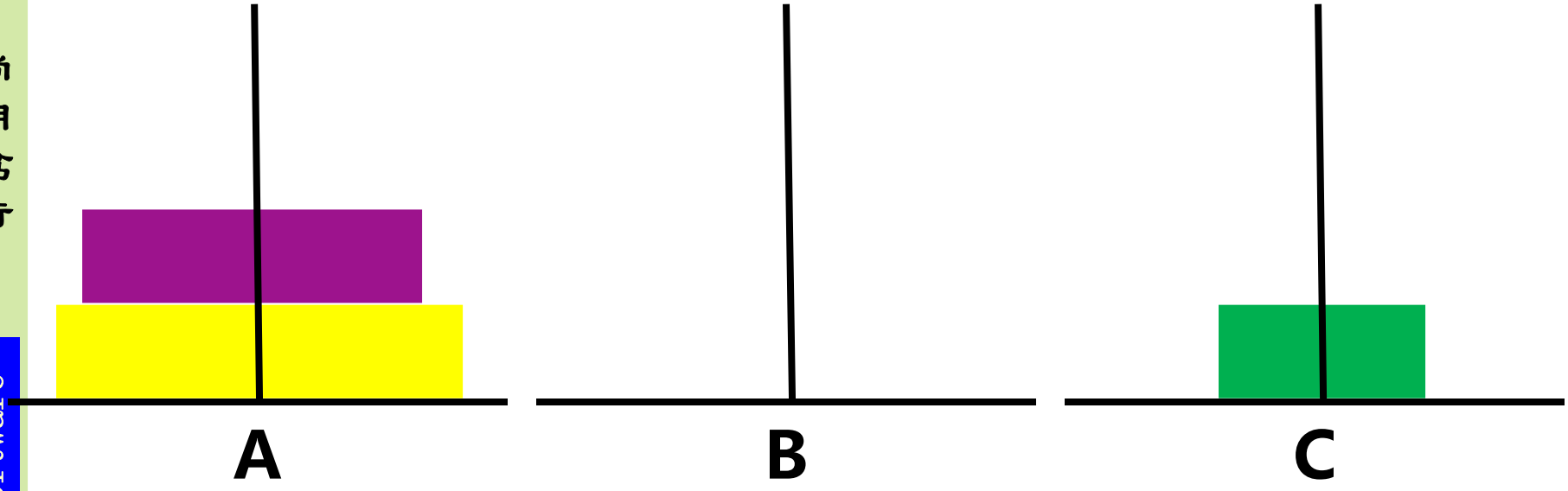
将2个盘子从A移到B的过程

将1个盘子从A移到C



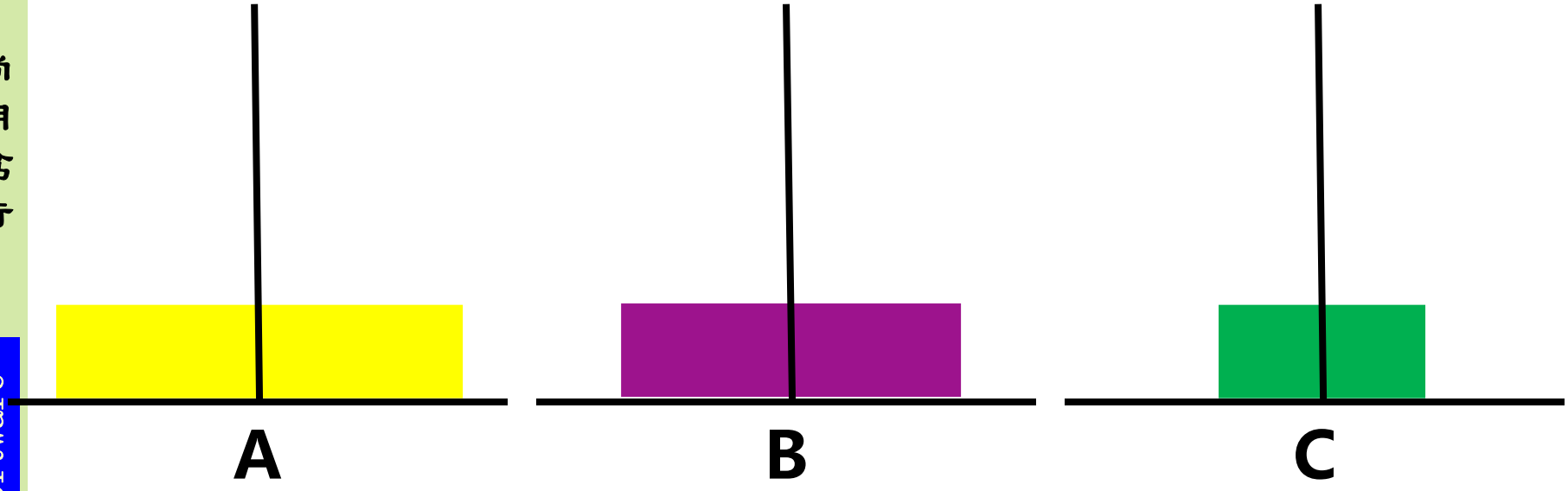
将2个盘子从A移到B的过程

将1个盘子从A移到B



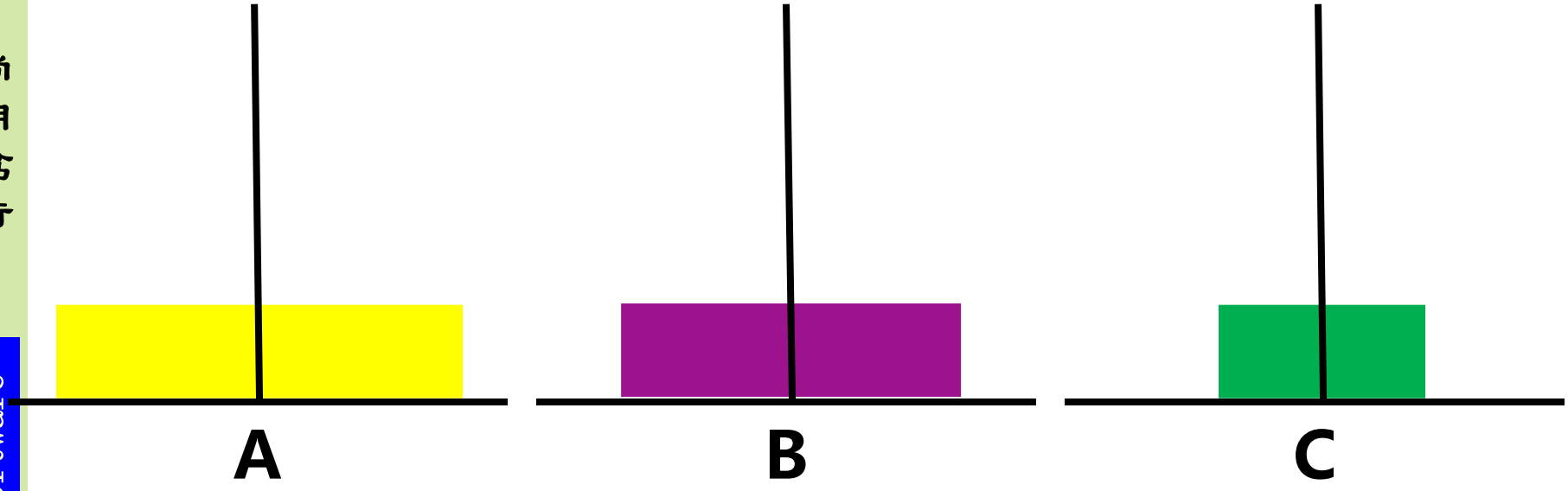
将2个盘子从A移到B的过程

将1个盘子从A移到B



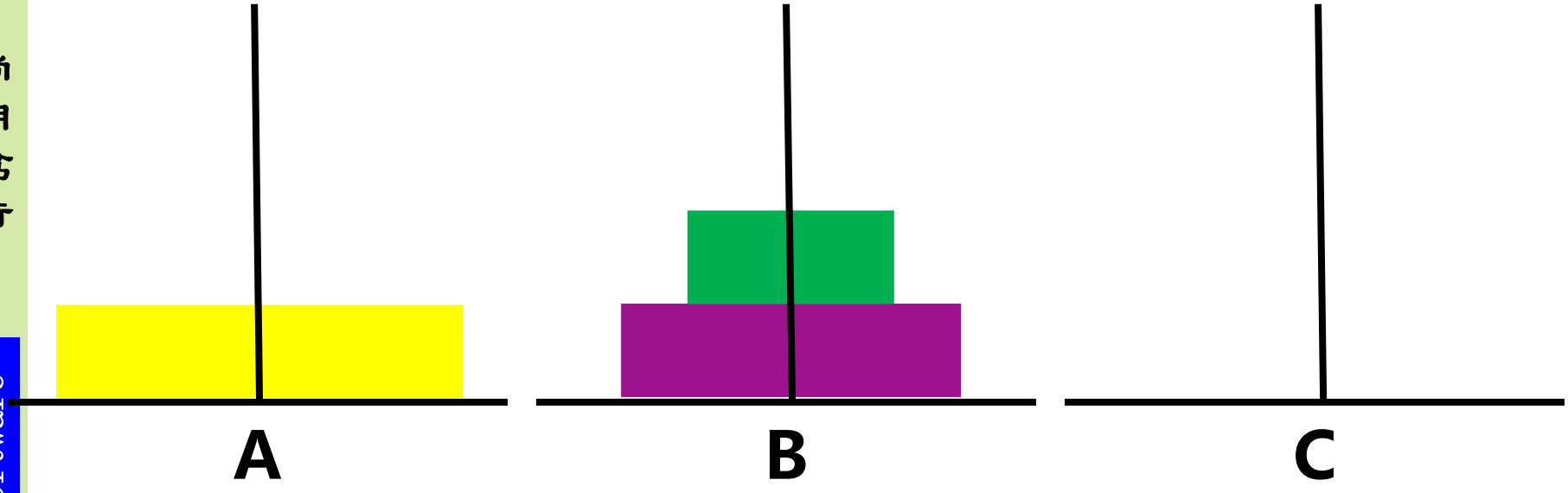
将2个盘子从A移到B的过程

将1个盘子从C移到B

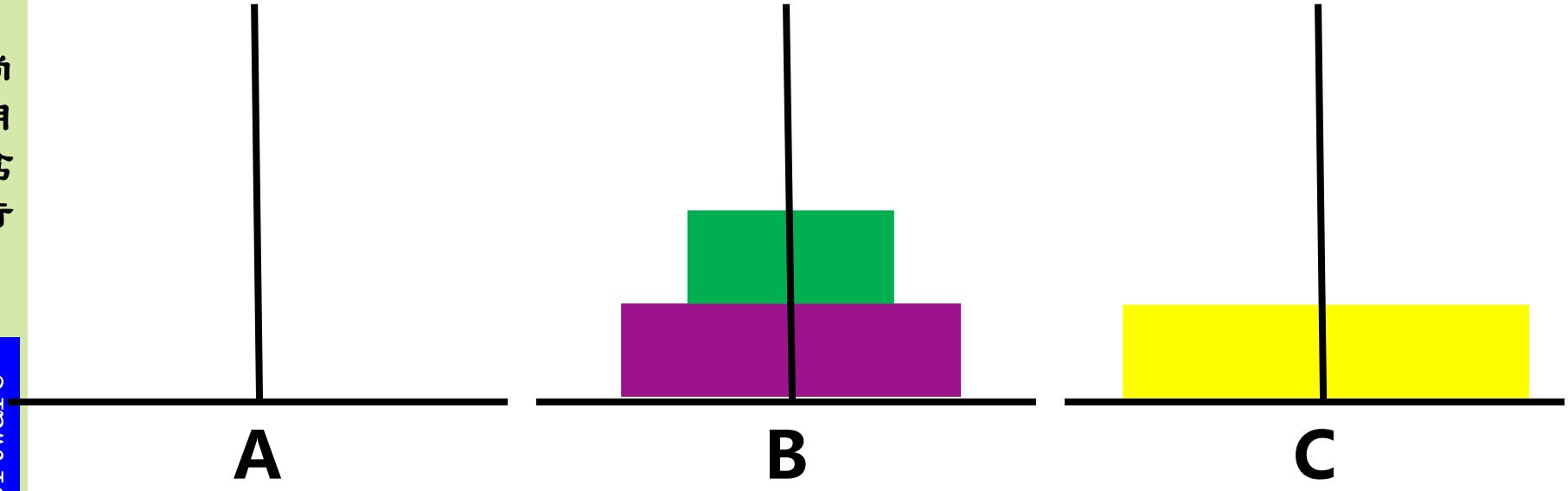


将2个盘子从A移到B的过程

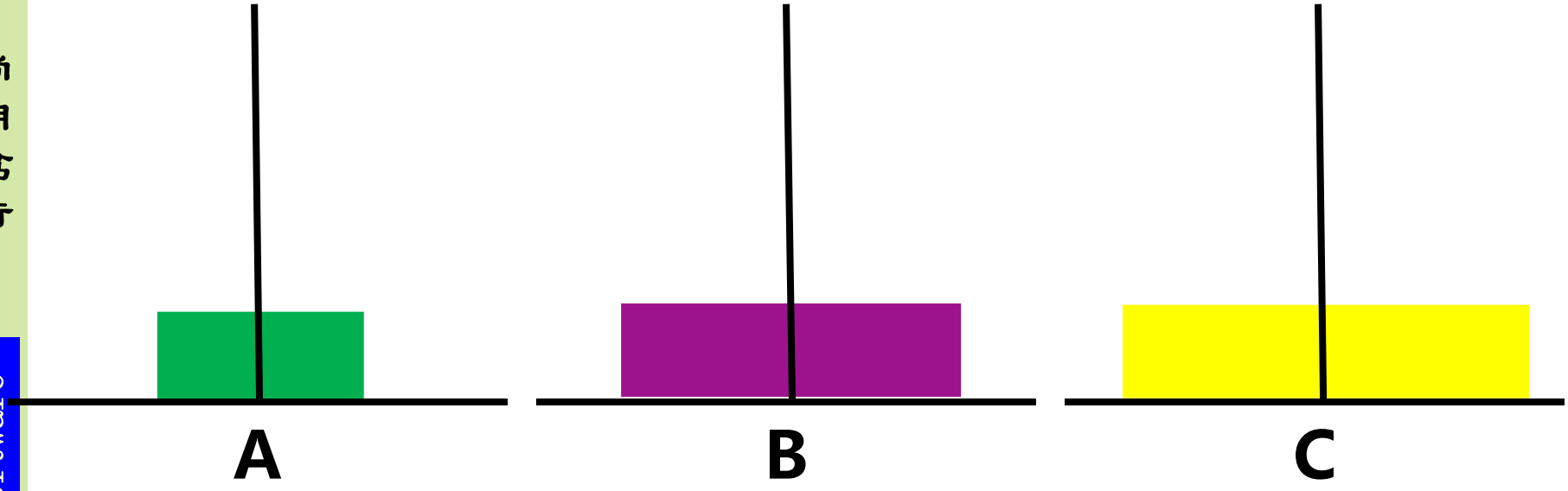
将1个盘子从C移到B



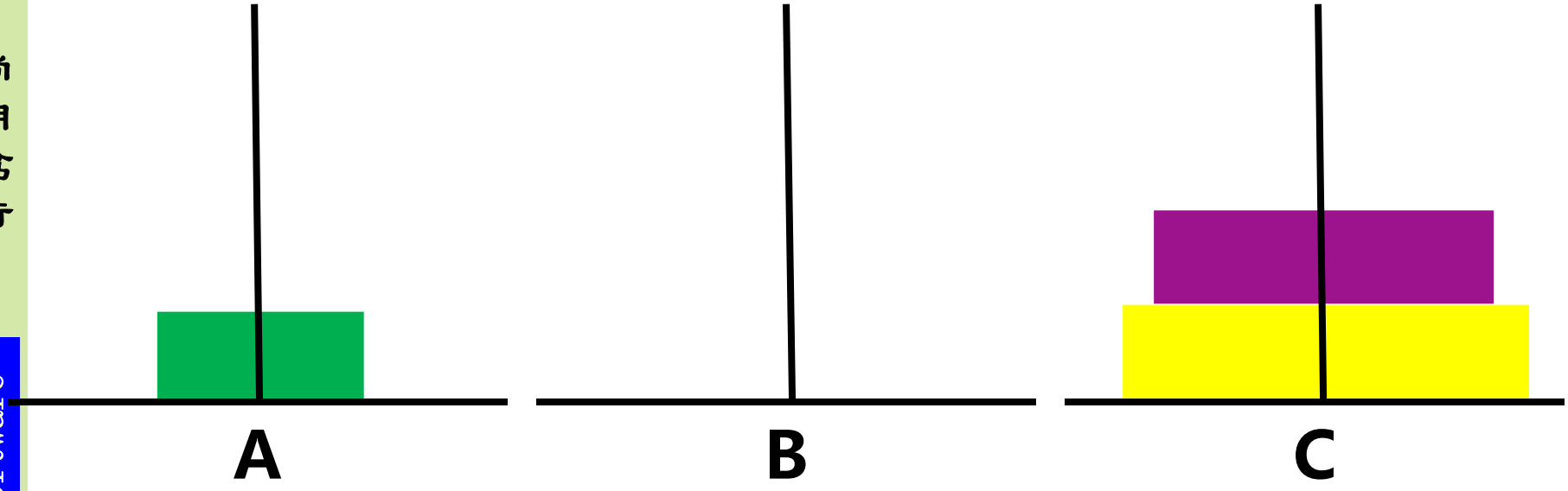
将2个盘子从B移到C的过程



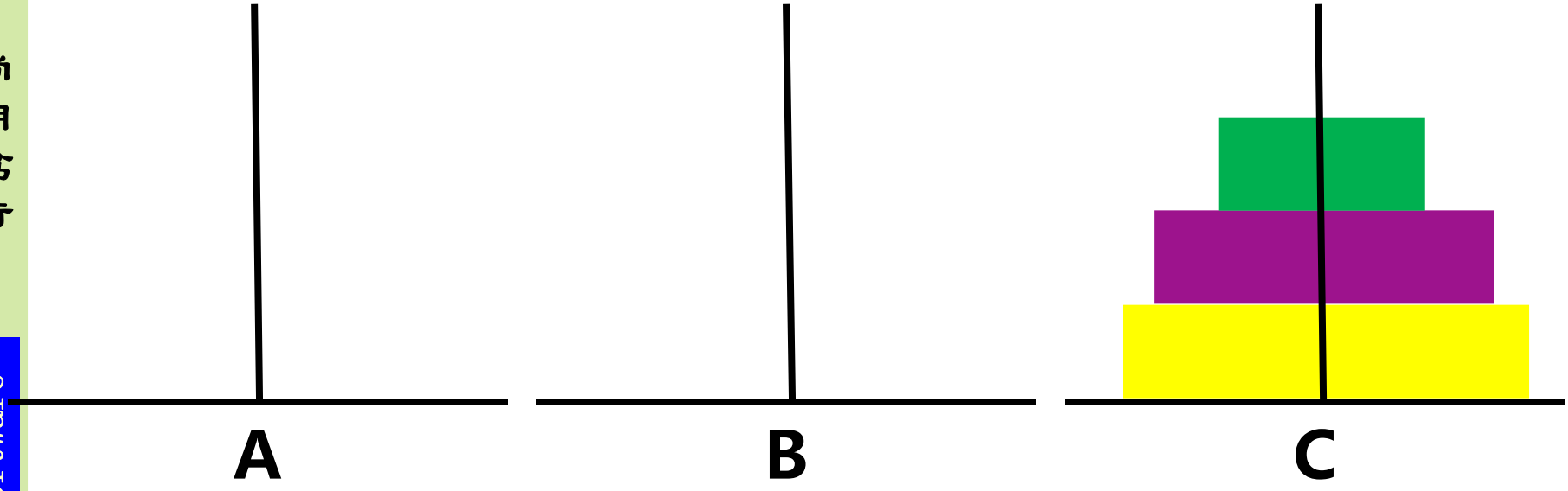
将2个盘子从B移到C的过程



将2个盘子从B移到C的过程



将2个盘子从B移到C的过程



- 由上面的分析可知：将n个盘子从A座移到C座可以分解为以下3个步骤：
  - (1) 将A上n-1个盘借助C座先移到B座上
  - (2) 把A座上剩下的一个盘移到C座上
  - (3) 将n-1个盘从B座借助于A座移到C座上

- 可以将第(1)步和第(3)步表示为：
  - 将“one”座上n-1个盘移到“two”座(借助“three”座)。
  - 在第(1)步和第(3)步中，one 、two、three和A、B、C的对应关系不同。
  - 对第(1)步，对应关系是one对应A，two对应B，three对应C。
  - 对第(3)步，对应关系是one对应B，two对应C，three对应A。

- 把上面3个步骤分成两类操作：

  (1) 将n-1个盘从一个座移到另一个座上（n>1）。这就是大和尚让小和尚做的工作，它是一个递归的过程，即和尚将任务层层下放，直到第64个和尚为止。

  (2) 将1个盘子从一个座上移到另一座上。这是大和尚自己做的工作。

- 编写程序。
  - 用**hanoi**函数实现第**1**类操作（即模拟小和尚的任务）
  - 用**move**函数实现第**2**类操作（模拟大和尚自己移盘）
  - 函数调用**hanoi(n,one,two.three)**表示将**n**个盘子从“**one**”座移到“**three**”座的过程**(**借助“**two**”座**)**
  - 函数调用**move(x,y)**表示将**1**个盘子从**x** 座移到**y** 座的过程。**x**和**y**是代表**A**、**B**、**C**座之一，根据每次不同情况分别取**A**、**B**、**C**代入


```
#include <stdio.h>
int main()
{  void hanoi(int n,char one,
                    char two,char three);
    int m;
    printf( “the number of diskes:");
    scanf("%d",&m);
    printf("move %d diskes:\n",m);
    hanoi(m,'A','B','C');
}
```


```
void hanoi(int n,char one,char two,
                                char three)
{  void move(char x,char y);
   if(n==1)
      move(one,three);
   else
   {  hanoi(n-1,one,three,two);
       move(one,three);
       hanoi(n-1,two,one,three);
    }
}
```

```
void move(char x,char y)
 {
     printf("%c-->%c\n",x,y);
 }
```

```
the number of diskes:3
move 3 diskes:
A-->C
A-->B
C-->B
A-->C
B-->A
B-->C
A-->C
```

7.7数组作为函数参数

7.7.1数组元素作函数实参

例7.9 输入10个数，要求输出其中值最大的元素和该数是第几个数。

7.7.1数组元素作函数实参

- 解题思路：
  - 定义数组a，用来存放10个数
  - 设计函数max，用来求两个数中的大者
  - 在主函数中定义变量m，初值为a[0]，每次调用max函数后的返回值存放在m中
  - 用“打擂台”算法，依次将数组元素a[1]到a[9]与m比较，最后得到的m值就是10个数中的最大者


```
#include <stdio.h>
int main()
{  int max(int x,int y);
    int a[10],m,n,i;
    printf( “10 integer numbers:\n");
    for(i=0;i<10;i++)
       scanf("%d",&a[i]);
    printf("\n");
```

```
10 integer numbers:
4 7 0 -3 4 34 67 -42 31 -76
```

```
for(i=1,m=a[0],n=0;i<10;i++)
{ if (max(m,a[i])>m)
   {  m=max(m,a[i]);
      n=i;
   }
 }
 printf( "largest number is %d\n",m);
 printf( "%dth number.\n ",n+1);
}
```

```
largest number is 67
7th number.
```

```
int max(int x,int y)
{  return(x>y?x:y);  }
```

7.7.2数组名作函数参数

- 除了可以用数组元素作为函数参数外，还可以用数组名作函数参数(包括实参和形参)
- 用数组元素作实参时，向形参变量传递的是数组元素的值
- 用数组名作函数实参时，向形参 传递的是数组首元素的地址

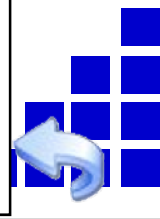
7.7.2数组名作函数参数

例7.10 有一个一维数组score，内放10个学生成绩，求平均成绩。

- 解题思路：
  - 用函数average求平均成绩，用数组名作为函数实参，形参也用数组名
  - 在average函数中引用各数组元素，求平均成绩并返回main函数


```
#include <stdio.h>
int main()
{ float average(float array[10]);
   float score[10],aver;  int i;
   printf("input 10 scores:\n");
   for(i=0;i<10;i++)
      scanf("%f",&score[i]);
   printf("\n");
   aver=average(score);
   printf("%5.2f\n",aver);
   return 0;
}
```

定义实参数组



定义形参数组

```
float average(float array[10])
{  int i;
   float aver,sum=array[0];
   for(i=1;i<10;i++)
      sum=sum+array[i];
   aver=sum/10;
   return(aver);
}
```

相当于score[0]

相当于score[i]

```
input 10 scores:
100 56 78 98 67.5 99 54 88.5 76 58

77.50
```

例7.11 有两个班级，分别有35名和30名学生，调用一个average函数，分别求这两个班的学生的平均成绩。

- 解题思路：
  - 需要解决怎样用同一个函数求两个不同长度的数组的平均值的问题
  - 定义**average**函数时不指定数组的长度，在形参表中增加一个整型变量**i**
  - 从主函数把数组实际长度从实参传递给形参**i**
  - 这个**i**用来在**average**函数中控制循环的次数
  - 为简化，设两个班的学生数分别为**5**和**10**

```
#include <stdio.h>
int main()
{ float average(float array[ ],int n);
   float score1[5]={98.5,97,91.5,60,55};
   float score2[10]={67.5,89.5,99,69.5,
                     77,89.5,76.5,54,60,99.5};
   printf( "%6.2f\n" ,average(score1,5));
   printf( "%6.2f\n" ,average(score2,10));
   return 0;
}
```

调用形式为average(score1,5)时

```
float average(float array[ ],int n)
 { int i;
    float aver,sum=array[0];
    for(i=1;i<n;i++)
       sum=sum+array[i];
    aver=sum/n;
    return(aver);
}
```

相当于5

相当于score1[0]

相当于score1[i]

调用形式为average(score2,10)时

```
float average(float array[ ],int n)
 { int i;
    float aver,sum=array[0];
    for(i=1;i<n;i++)
       sum=sum+array[i];
    aver=sum/n;
    return(aver);
}
```

相当于10

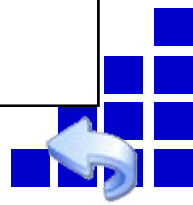
相当于score2[0]

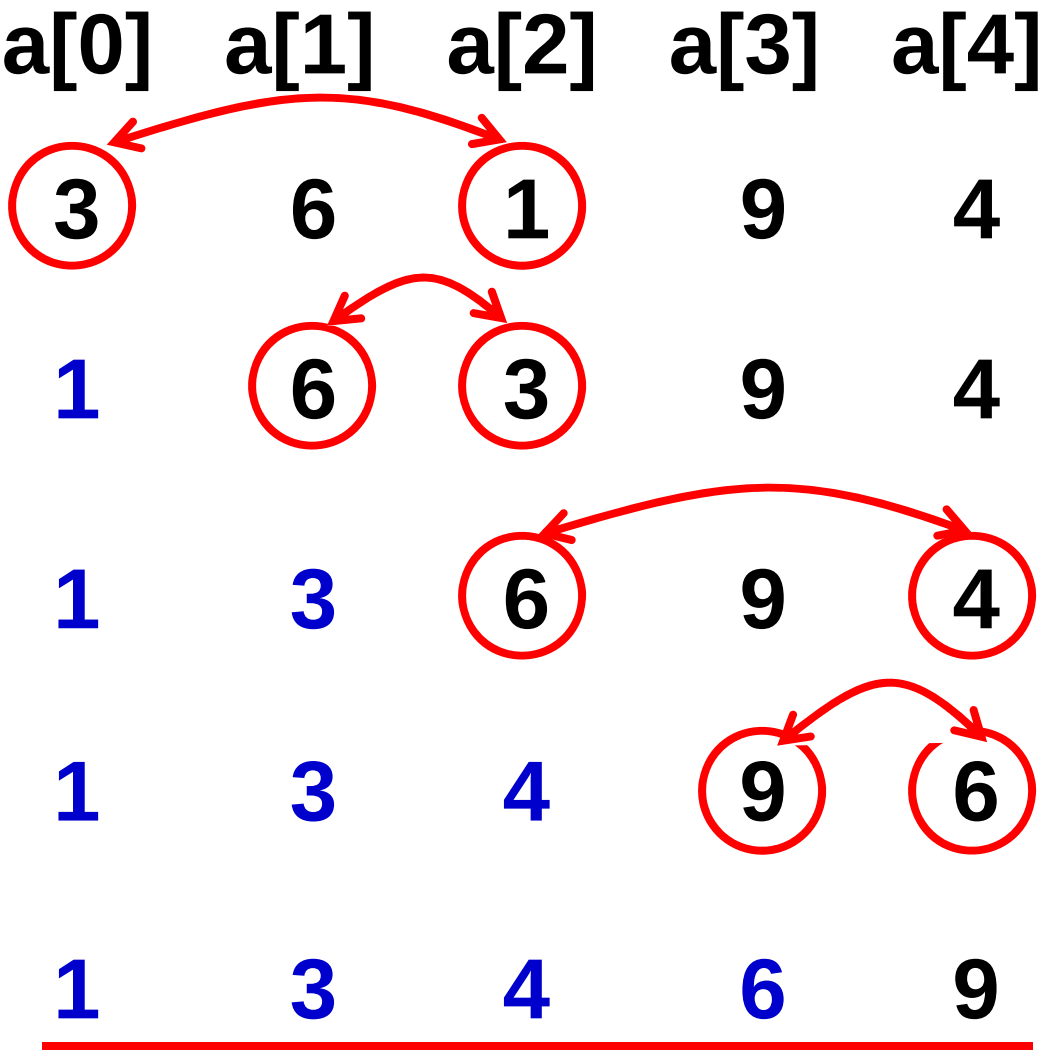
相当于score2[i]

```
80.40
78.20
```

例7.12用选择法对数组中10个整数按由小到大排序。

- 解题思路：
  - 所谓选择法就是先将10个数中最小的数与a[0]对换；再将a[1]到a[9]中最小的数与a[1]对换……每比较一轮，找出一个未经排序的数中最小的一个
  - 共比较9轮





小到大排序

```
#include <stdio.h>
int main()
{ void sort(int array[],int n);
   int a[10],i;
   printf("enter array:\n");
   for(i=0;i<10;i++)  scanf("%d",&a[i]);
   sort(a,10);
   printf("The sorted array:\n");
   for(i=0;i<10;i++)   printf("%d ",a[i]);
   printf("\n");
   return 0;
}
```

```
void sort(int array[],int n)
 { int i,j,k,t;
   for(i=0;i<n-1;i++)
   { k=i;
     for(j=i+1;j<n;j++)
      if(array[j]<array[k])    k=j;
     t=array[k];
     array[k]=array[i];
     array[i]=
   }
}
```

在sort[i]~sort[9]中，最小数与sort[i]对换

```
enter array:
45 2 9 0 -3 54 12 5 66 33
The sorted array:
-3 0 2 5 9 12 33 45 54 66
```

7.7.3多维数组名作函数参数

例7.13 有一个３×４的矩阵，求所有元素中的最大值。

- 解题思路：先使变量max的初值等于矩阵中第一个元素的值，然后将矩阵中各个元素的值与max相比，每次比较后都把“大者”存放在max中，全部元素比较完后，max 的值就是所有元素的最大值。

不能省略
要与形参数组第二维大小相同

```
#include <stdio.h>
int main()
{ int max_value(int array[][4]);
   int a[3][4]={{1,3,5,7},{2,4,6,8},
                          {15,17,34,12}};
   printf( “Max value is %d\n”,
                          max_value(a));
   return 0;
}
```

要与实参数组第二维大小相同

```
int max_value(int array[][4])
{  int i,j,max;
    max = array[0][0];
    for (i=0;i<3;i++)
        for(j=0;j<4;j++)
            if (array[i][j]>max)
                max = array[i][j];
    return (max);
}
```

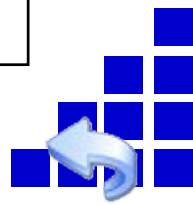
7.8局部变量和全局变量

7.8.1 局部变量

- 定义变量可能有三种情况：
  - 在函数的开头定义
  - 在函数内的复合语句内定义
  - 在函数的外部定义

7.8.1 局部变量

- 在一个函数内部定义的变量只在本函数范围内有效
- 在复合语句内定义的变量只在本复合语句范围内有效
- 在函数内部或复合语句内部定义的变量称为“局部变量”



```
float f1( int a)
{ int b,c;
    ……
}
```

a、b、c仅在此函数内有效

```
char f2(int x,int y) 玒
{ int i,j;
    ……
}
```

x、y、i、j仅在此函数内有效

```
int main( ) 玒
{ int m,n;
    ……
    return 0;
}
```

m、n仅在此函数内有效

```
float f1( int a)
{ int b,c;
   ……
}
char f2(int x,int
{ int i,j;
   ……
}
int main( ) 玒
{ int a,b;
   ……
   return 0;
}
```

类似于不同班同名学生

a、b也仅在此函数内有效


```
int main ( )
{ int a,b;
    ......
  { int c;
    c=a+b;
    ......
  }
  ......
}
```

a、b仅在此复合语句内有效

c仅在此复合语句内有效

7.8.2全局变量

- 在函数内定义的变量是局部变量，而在函数之外定义的变量称为外部变量
- 外部变量是全局变量(也称全程变量)
- 全局变量可以为本文件中其他函数所共用
- 有效范围为从定义变量的位置开始到本源文件结束

```
int p=1,q=5
float f1(int a)
{  int b,c;   ……  }
char c1,c2;
char f2 (int x, int y)
{  int i,j;   ……  }
int main ( )
{  int m,n;
    ……
    return 0;
}
```

p、q、c1、c2
为全局变量

```
int p=1,q=5
float f1(int a)
{  int b,c;   ……  }
char c1,c2;
char f2 (int x, int y)
{  int i,j;   ……  }
int main ( )
{  int m,n;
    ……
    return 0;
}
```

p、q的有效范围

c1、c2的有效范围

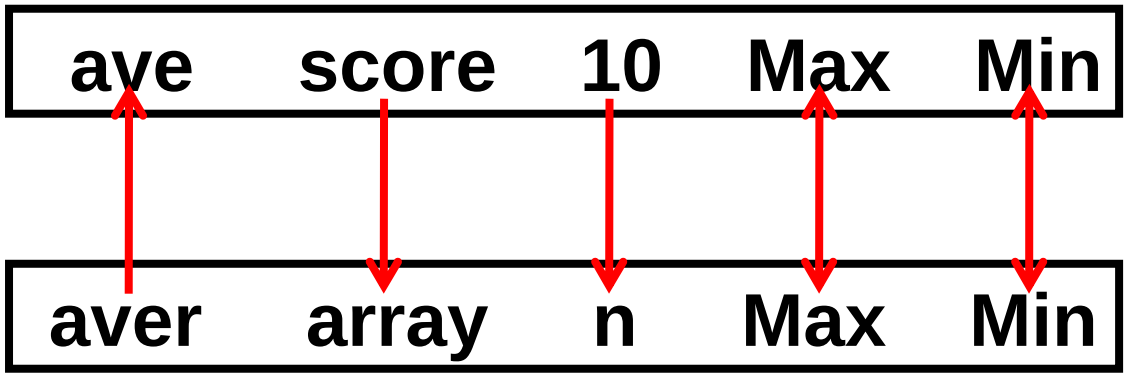
例7.14 有一个一维数组，内放10个学生成绩，写一个函数，当主函数调用此函数后，能求出平均分、最高分和最低分。

- 解题思路：调用一个函数可以得到一个函数返回值，现在希望通过函数调用能得到3个结果。可以利用全局变量来达到此目的。

```
#include <stdio.h>
float Max=0,Min=0;
int main()
{ float average(float array[ ],int n);
   float ave,score[10];  int i;
   printf("Please enter 10 scores:\n");
   for(i=0;i<10;i++)
       scanf("%f",&score[i]);
   ave=average(score,10);
   printf("max=%6.2f\nmin=%6.2f\n
         average=%6.2f\n",Max,Min,ave);
   return 0;
 }
```

```
float average(float array[ ],int n)
{ int i;  float aver,sum=array[0];
   Max=Min=array[0];
   for(i=1;i<n;i++)
   { if(array[i]>Max) Max=array[i];
      else if(array[i]<Min) Min=array[i];
      sum=sum+array[i];
    }
    aver=sum/n
    return(aver)
}
```

```
Please enter 10 scores:
89 95 87.5 100 67.5 97 59 84 73 90
max=100.00
min= 59.00
average= 84.20
```

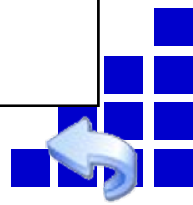


main
函数

average
函数

建议不在必要时不要使用全局变量

例7.15 若外部变量与局部变量同名，分析结果。



```
#include <stdio.h>
int a=3,b=5;
int main()
{ int max(int a,int b);
   int a=8;
   printf( “max=%d\n” ,max(a,b));
   return 0;
}
int max(int a,int b)
{ int c;
   c=a>b?a:b;
   return(c);
}
```

b为全部变量

a为局部变量，仅在此函数内有效

```
#include <stdio.h>
int a=3,b=5;
int main()
{ int max(int a,int b);
   int a=8;
   printf( "max=%d\n" ,max(a,b));
   return 0;
}
int max(int a,int b)
{ int c;
   c=a>b?a:b;
   return(c);
}
```

```
max=8
```

a、b为局部变量，仅在此函数内有效

7.9变量的存储方式和生存期

7.9.1动态存储方式与静态存储方式

- 从变量的作用域的角度来观察，变量可以分为全局变量和局部变量
- 从变量值存在的时间(即生存期)观察，变量的存储有两种不同的方式：静态存储方式和动态存储方式
  - 静态存储方式是指在程序运行期间由系统分配固定的存储空间的方式
  - 动态存储方式是在程序运行期间根据需要进行动态的分配存储空间的方式

程序开始执行时给全局变量分配存储区，程序执行完毕就释放。在程

函数调用开始时分配，函数结束时释放。在程序执行过程中，这种分配和释放是动态的

- 每一个变量和函数都有两个属性：数据类型和数据的存储类别
  - 数据类型，如整型、浮点型等
  - 存储类别指的是数据在内存中存储的方式(如静态存储和动态存储)
  - 存储类别包括：

    自动的、静态的、寄存器的、外部的
  - 根据变量的存储类别，可以知道变量的作用域和生存期

7.9.2 局部变量的存储类别

1.自动变量(auto变量)

- 局部变量，如果不专门声明存储类别，都是动态地分配存储空间的
- 调用函数时，系统会给局部变量分配存储空间，调用结束时就自动释放空间。因此这类局部变量称为自动变量
- 自动变量用关键字**auto**作存储类别的声明

7.9.2 局部变量的存储类别

```
int f(int a)
{
    auto int b,c=3;
}
```

可以省略

7.9.2 局部变量的存储类别

2.静态局部变量(static局部变量)

- 希望函数中的局部变量在函数调用结束后不消失而继续保留原值，即其占用的存储单元不释放，在下一次再调用该函数时，该变量已有值(就是上一次函数调用结束时的值)，这时就应该指定该局部变量为“静态局部变量”，用关键字**static**进行声明

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

调用三次

每调用一次，开辟新a和b，但c不是

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

b	c
0	3

第一次调用开始

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

b	c
1	4

第一次调用期间

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

7

c

4

第一次调用结束

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

b	c
0	4

第二次调用开始

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

b	c
1	5

第二次调用期间

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

8

c

5

第二次调用结束

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

b	c
0	5

第三次调用开始

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

b	c
1	6

第三次调用期间

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

9

c

6

第三次调用结束

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
   return(a+b+c);
}
```

```
7
8
9
```

整个程序结束

例7.16 考察静态局部变

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1;
   c=c+1;
}
```

在函数调用时赋初值

在编译时赋初值

例7.16 考察静态局部变

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1
   c=
}
```

若不赋初值，不确定

若不赋初值，是0

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
   int a=2,i;
   for(i=0;i<3;i++)
     printf( "%d\n" ,f(a));
   return 0;
}
```

```
int f(int a)
{ auto  int b=0;
   static c=3;
   b=b+1
   c=
}
```

仅在本函数内有效

明德求新 尚用笃行

例7.17 输出1到5的阶乘值。

- 解题思路：可以编一个函数用来进行连乘，如第1次调用时进行1乘1，第2次调用时再乘以2，第3次调用时再乘以3，依此规律进行下去。

```
#include <stdio.h>
int main()
{ int fac(int n);
   int i;
   for(i=1;i<=5;i++)
       printf( "%d!=%d\n" ,i,fac(i));
   return 0;
}
int fac(int n)
{ static int f=1;
   f=f*n;
   return(f);
}
```

若非必要，不要多用静态局部变量

```
1!=1
2!=2
3!=6
4!=24
5!=120
```

3. 寄存器变量(register变量)

- 一般情况下，变量（包括静态存储方式和动态存储方式）的值是存放在内存中的
- 寄存器变量允许将局部变量的值放在CPU中的寄存器中
- 现在的计算机能够识别使用频繁的变量，从而自动地将这些变量放在寄存器中，而不需要程序设计者指定

7.9.3 全局变量的存储类别

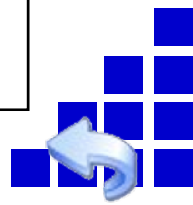
- 全局变量都是存放在静态存储区中的。因此它们的生存期是固定的，存在于程序的整个运行过程
- 一般来说，外部变量是在函数的外部定义的全局变量，它的作用域是从变量的定义处开始，到本程序文件的末尾。在此作用域内，全局变量可以为程序中各个函数所引用。

明德求新 尚用笃行

School of Software

1. 在一个文件内扩展外部变量的作用域

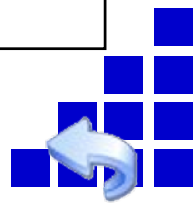
- 外部变量有效的作用范围只限于定义处到本文件结束。
- 如果用关键字**extern**对某变量作“外部变量声明”，则可以从“声明”处起，合法地使用该外部变量



明德求新 尚用笃行

例7.18 调用函数，求3个整数中的大者。

- 解题思路：用extern声明外部变量，扩展外部变量在程序文件中的作用域。



```
#include <stdio.h>
int main()
{ int max( );
   extern int A,B,C;
   scanf( “%d %d %d” ,&A,&B,&C);
   printf("max is %d\n",max());
   return 0;
}
int A ,B ,C;
int max( )
{ int m;
   m=A>B?A:B;
   if (C>m) m=C;
   return(m);
}
```

```
34 67 12
max is 67
```

2. 将外部变量的作用域扩展到其他文件

- 如果一个程序包含两个文件，在两个文件中都要用到同一个外部变量**Num**，不能分别在两个文件中各自定义一个外部变量**Num**
- 应在任一个文件中定义外部变量**Num**，而在另一文件中用**extern**对**Num**作“外部变量声明”
- 在编译和连接时，系统会由此知道**Num**有“外部链接”，可以从别处找到已定义的外部变量**Num**，并将在另一文件中定义的外部变量**num**的作用域扩展到本文件

例7.19 给定b的值，输入a和m，求a*b和a^m的值。

- 解题思路：
  - 分别编写两个文件模块，其中文件file1包含主函数，另一个文件file2包含求a^m的函数。
  - 在file1文件中定义外部变量A，在file2中用extern声明外部变量A，把A的作用域扩展到file2文件。

文件file1.c：

```
#include <stdio.h>
int A;
int main()
{ int power(int);
   int b=3,c,d,m;  scanf("%d,%d",&A,&m);
   c=A*b;
   printf("%d*%d=%d\n",A,b,c);
   d=power(m);
   printf("%d**%d=%d\n",A,m,d);
   return 0;
}
```

文件file2.c：

```
extern  A;
int power(int n)
{ int i,y=1;
   for(i=1;i<=n;i++)
      y*=A;
   return(y);
}
```

编译和运行包括多个文件的程序，可参考《C程序设计学习辅导》一书的“C语言上机指南”部分

```
13,3
13*3=39
13**3=2197
```

3.将外部变量的作用域限制在本文件中

- 有时在程序设计中希望某些外部变量只限于被本文件引用。这时可以在定义外部变

只能用于本文件

本文件仍然不能用

file1.c

```
static int A;
int main ( )
{
    ……
}
```

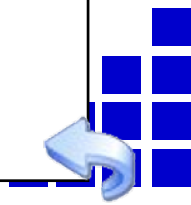
file2.c

```
extern   A;
void fun (int n)
{  ……
    A=A*n;
    ……
}
```

- 说明:
  - 不要误认为对外部变量加**static**声明后才采取静态存储方式，而不加**static**的是采取动态存储
  - 声明局部变量的存储类型和声明全局变量的存储类型的含义是不同的
  - 对于局部变量来说，声明存储类型的作用是指定变量存储的区域以及由此产生的生存期的问题，而对于全局变量来说，声明存储类型的作用是变量作用域的扩展问题

明德求新 尚用笃行

- 用static 声明一个变量的作用是：
  (1) 对局部变量用static声明，把它分配在静态存储区，该变量在整个程序执行期间不释放，其所分配的空间始终存在。
  (2) 对全局变量用static声明，则该变量的作用域只限于本文件模块(即被声明的文件中)。



- 注意：用**auto**、**register**、**static**声明变量时，是在定义变量的基础上加上这些关键字，而不能单独使用。
- 下面用法不对：

int a; 灌

static a;

编译时会被认为“重新定义”。菖

7.9.4 存储类别小结

- 对一个数[illegible]种属性：
  - 数据类型[illegible]两个关键字

例如：

static int a; 灌

auto char c; 灌

register int d;

- 可以用**extern**声明[illegible]已定义的外部变量

例如： **extern b;**

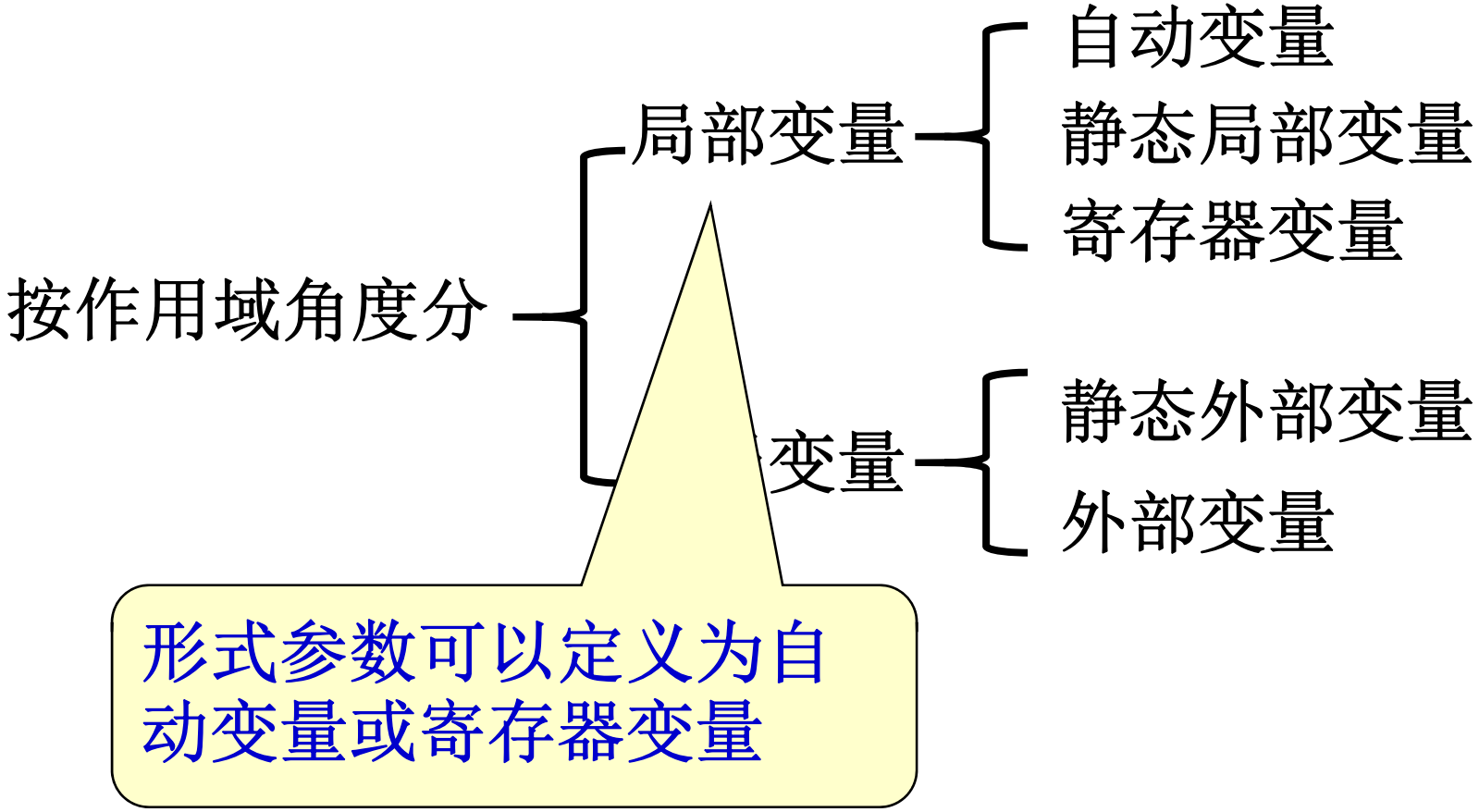
静态局部整型变量或
静态外部整型变量

自动变量，在

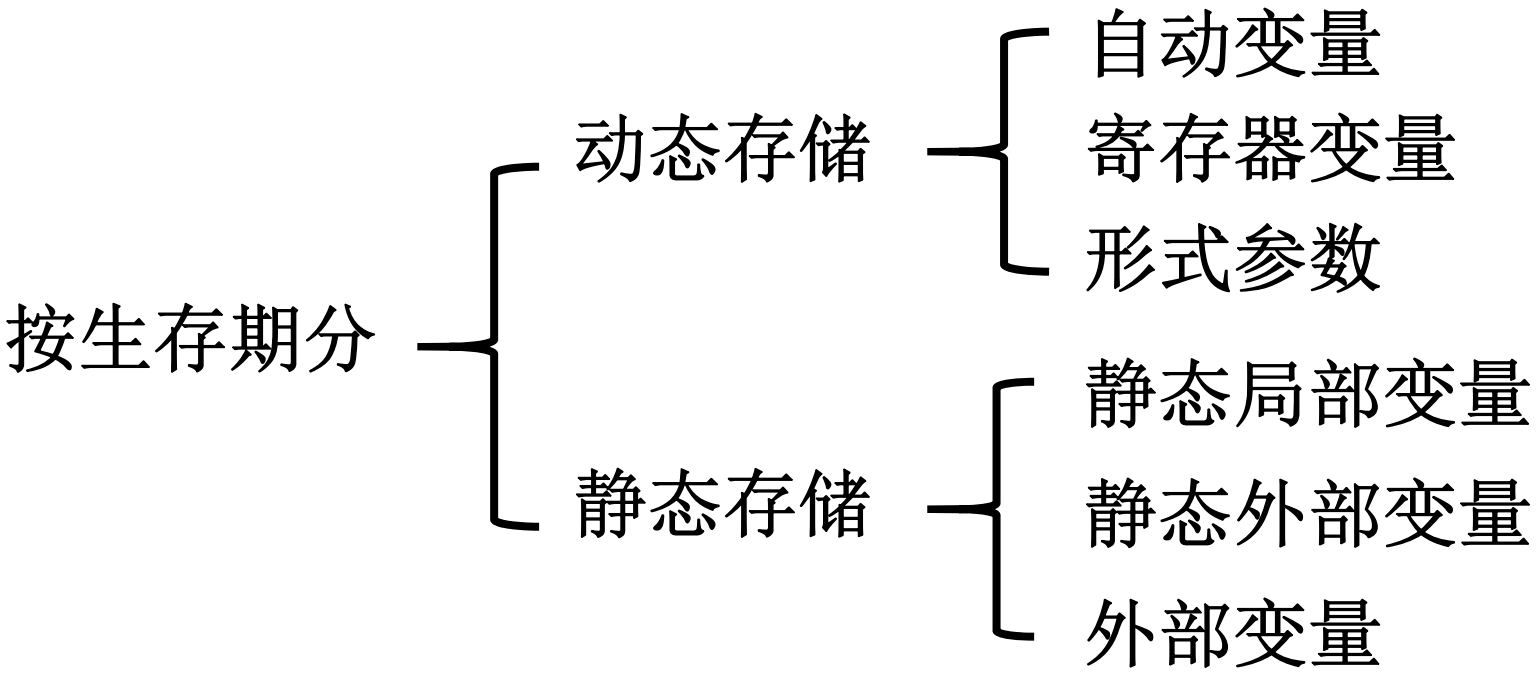
寄存器变量，
在函数内定义

将已定义的外部变量
b的作用域扩展至此

（1）从作用域角度分，有局部变量和全局变量。它们采用的存储类别如下：



（2）从变量存在的时间区分,有动态存储和静态存储两种类型。静态存储是程序整个运行时间都存在,而动态存储则是在调用函数时临时分配单元



（3）从变量值存放的位置来区分,可分为:

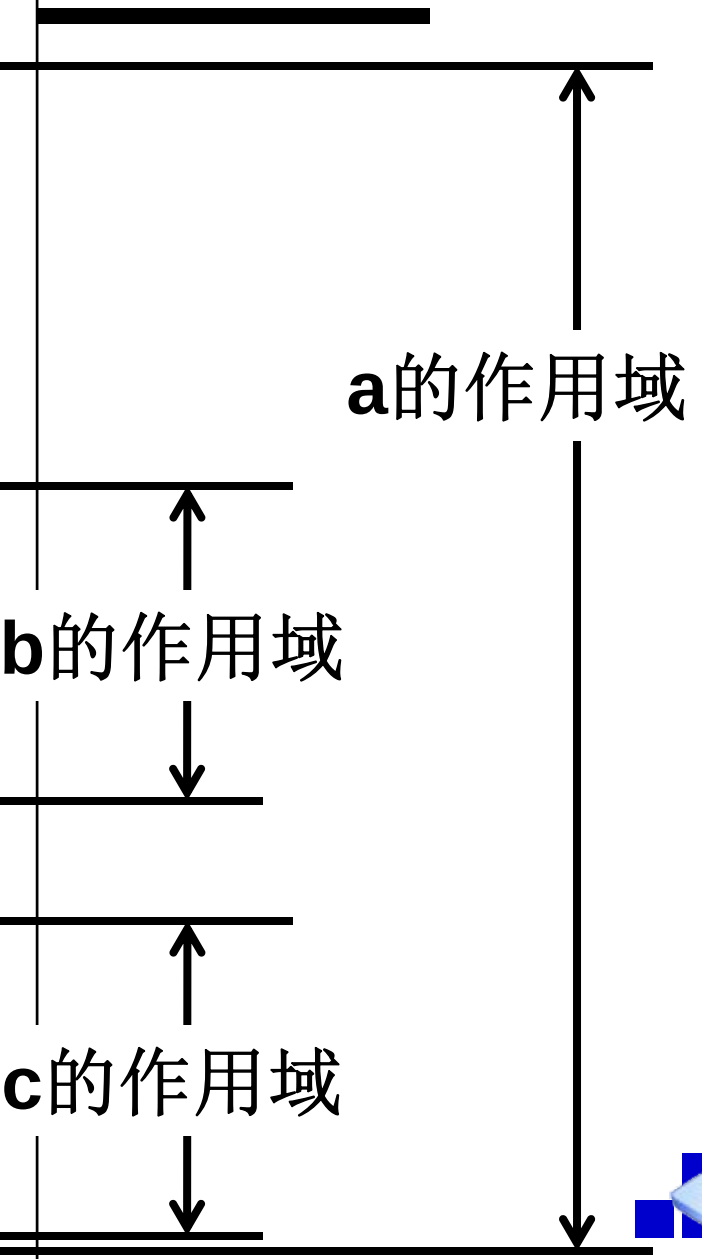
- 按变量值存放的位置分
  - 内存中静态存储区
    - 静态局部变量
    - 静态外部变量
    - 外部变量
  - 内存中动态存储区——自动变量和形式参数
  - CPU中的寄存器 —— 寄存器变量

(4)关于作用域和生存期的概念

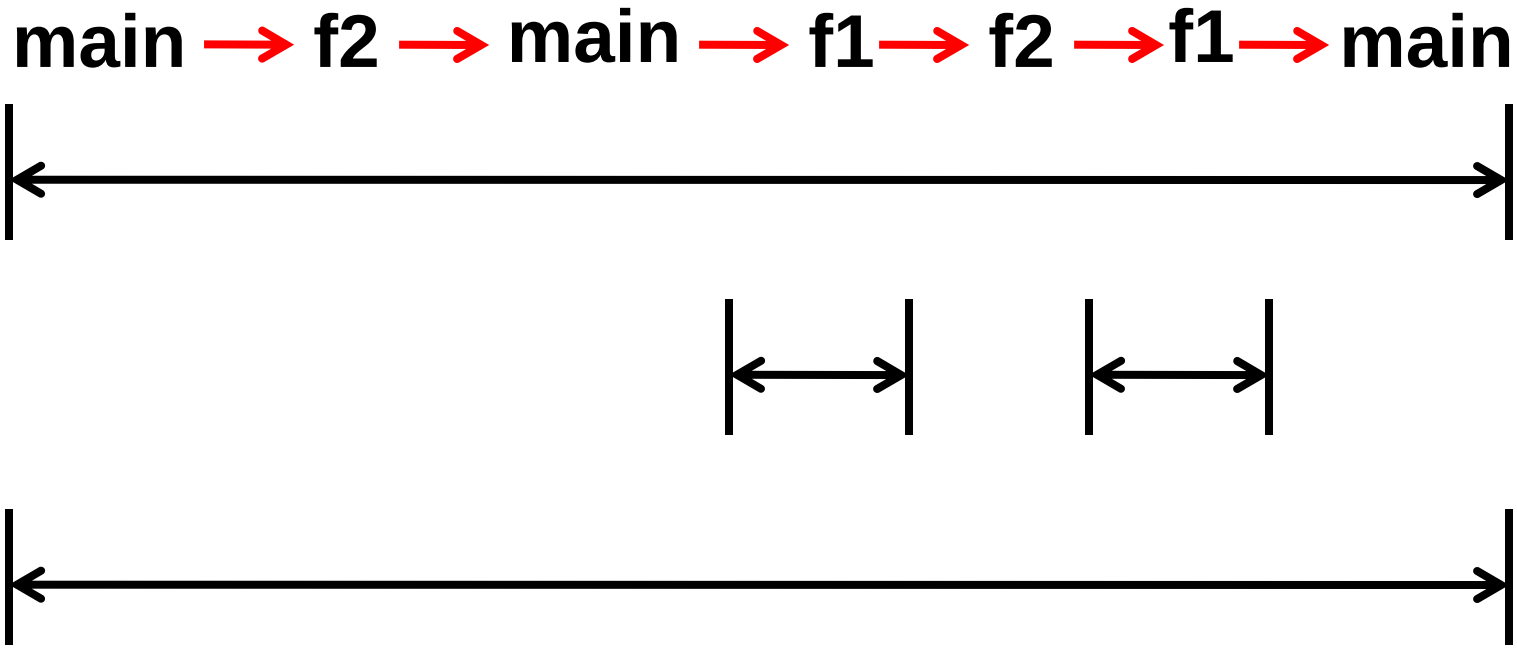
- 对一个变量的属性可以从两个方面分析：
  - 作用域：如果一个变量在某个文件或函数范围内是有效的，就称该范围为该变量的作用域
  - 生存期：如果一个变量值在某一时刻是存在的，则认为这一时刻属于该变量的生存期
- 作用域是从空间的角度，生存期是从时间的角度
- 二者有联系但不是同一回事

文件file1.c

```
int a;
int main( )
{ …f2( );…f1( );… }
void f1( )
{ auto int b;
   …   f2( );
   …
}
void f2( )
{ static int c;
……
}
```



程序执行过程



各种类型变量的作用域和存在性的情况

变量存储类别	函 数 内		函 数 外	
	作用域	存在性	作用域	存在性
自动变量和寄存器变量	√	√	×	×
静态局部变量	√	√	×	√
静态外部变量	√	√	√(只限本文件)	√
外部变量	√	√	√	√

(5) static对局部变量和全局变量的作用不同

- 局部变量使变量由动态存储方式改变为静态存储方式
- 全局变量使变量局部化(局部于本文件)，但仍为静态存储方式
- 从作用域角度看，凡有**static**声明的，其作用域都是局限的，或者是局限于本函数内(静态局部变量)，或者局限于本文件内(静态外部变量)

7.10 关于变量的声明和定义

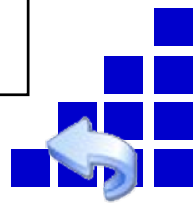
- 一般为了叙述方便，把建立存储空间的变量声明称定义，而把不需要建立存储空间的声明称为声明
- 在函数中出现的对变量的声明(除了用**extern**声明的以外)都是定义
- 在函数中对其他函数的声明不是函数的定义

7.11 内部函数和外部函数

7.11.1 内部函数

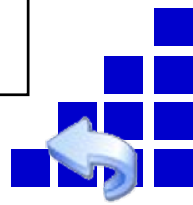
- 如果一个函数只能被本文件中其他函数所调用，它称为内部函数。
- 在定义内部函数时，在函数名和函数类型的前面加**static**，即:

  static 类型名 函数名(形参表)



7.11.1 内部函数

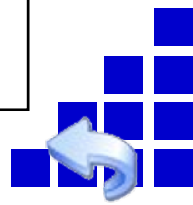
- 内部函数又称静态函数，因为它是用**static**声明的
- 通常把只能由本文件使用的函数和外部变量放在文件的开头，前面都冠以**static**使之局部化，其他文件不能引用
- 提高了程序的可靠性



7.11.2 外部函数

- 如果在定义函数时，在函数首部的最左端加关键字**extern**，则此函数是外部函数，可供其他文件调用。囍
- 如函数首部可以为

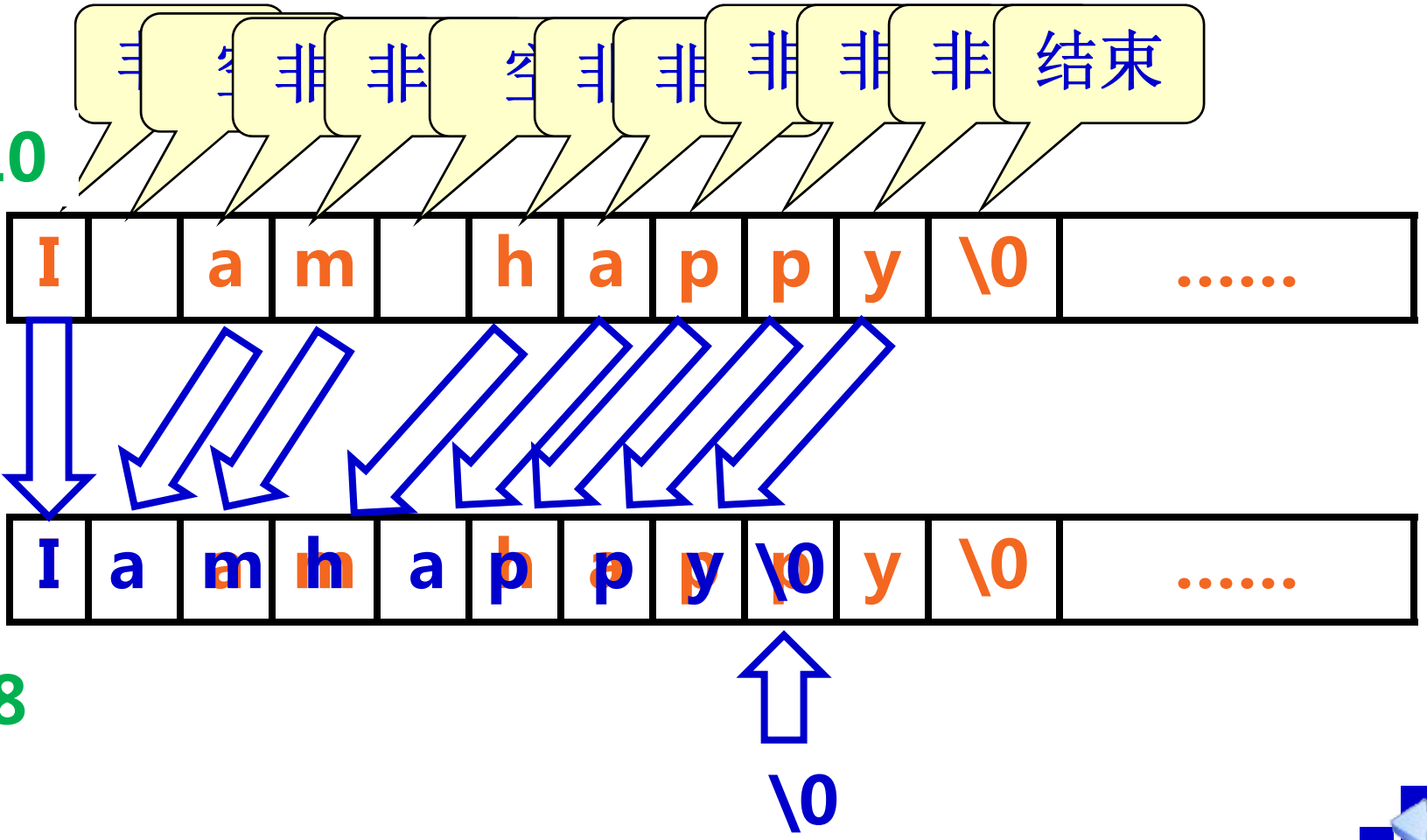
  extern int fun (int a, int b)
- 如果在定义函数时省略**extern**，则默认为外部函数



例7.20 有一个字符串，内有若干个字符，今输入一个字符，要求程序将字符串中该字符删去。用外部函数实现。

- 解题思路：
  - 分别定义3个函数用来输入字符串、删除字符、输出字符串
  - 按题目要求把以上3个函数分别放在3个文件中。main函数在另一文件中，main函数调用以上3个函数，实现题目的要求

删除空格的思路



file1（文件1）

```
#include <stdio.h>
int main()
{ extern void enter_string(char str[]);
  extern void delete_string(char str[],
                            char ch);
  extern void print_string(char str[]);
  char c,str[80];
  enter_string(str);
  scanf( “%c” ,&c);
  delete_string(str,c);
  print_string(str);
  return 0;
}
```

声明在本函数中将要调用的已在其他文件中定义的3个函数

```
void enter_string(char str[80])
{  gets(str);   }
void delete_string(char str[],char ch)
{ int i,j;
   for(i=j=0;str[i]!='\0';i++)
      if(str[i]!=ch)   str[j++]=str[i];
   str[j]='\0';
}
void print_string(char str[])
{ printf("%s\n",str); }
```

file2（文件2）

file3（文件3）

file4（文件4）

第8章 善于利用指针

8.1 指针是什么

- 如果在程序中定义了一个变量，在对程序进行编译时，系统就会给该变量分配内存单元
- 编译系统根据程序中定义的变量类型，分配一定长度的空间
- 例如，**VC++**为整型变量分配**4**个字节，对单精度浮点型变量分配 4 个字节，对字符型变量分配 1 个字节

8.1 指针是什么

- 内存区的每一个字节有一个编号，这就是“地址”，它相当于旅馆中的房间号。
- 在地址所标识的内存单元中存放数据，这相当于旅馆房间中居住的旅客一样。
- 由于通过地址能找到所需的变量单元，我们可以说，地址指向该变量单元。
- 将地址形象化地称为“指针”

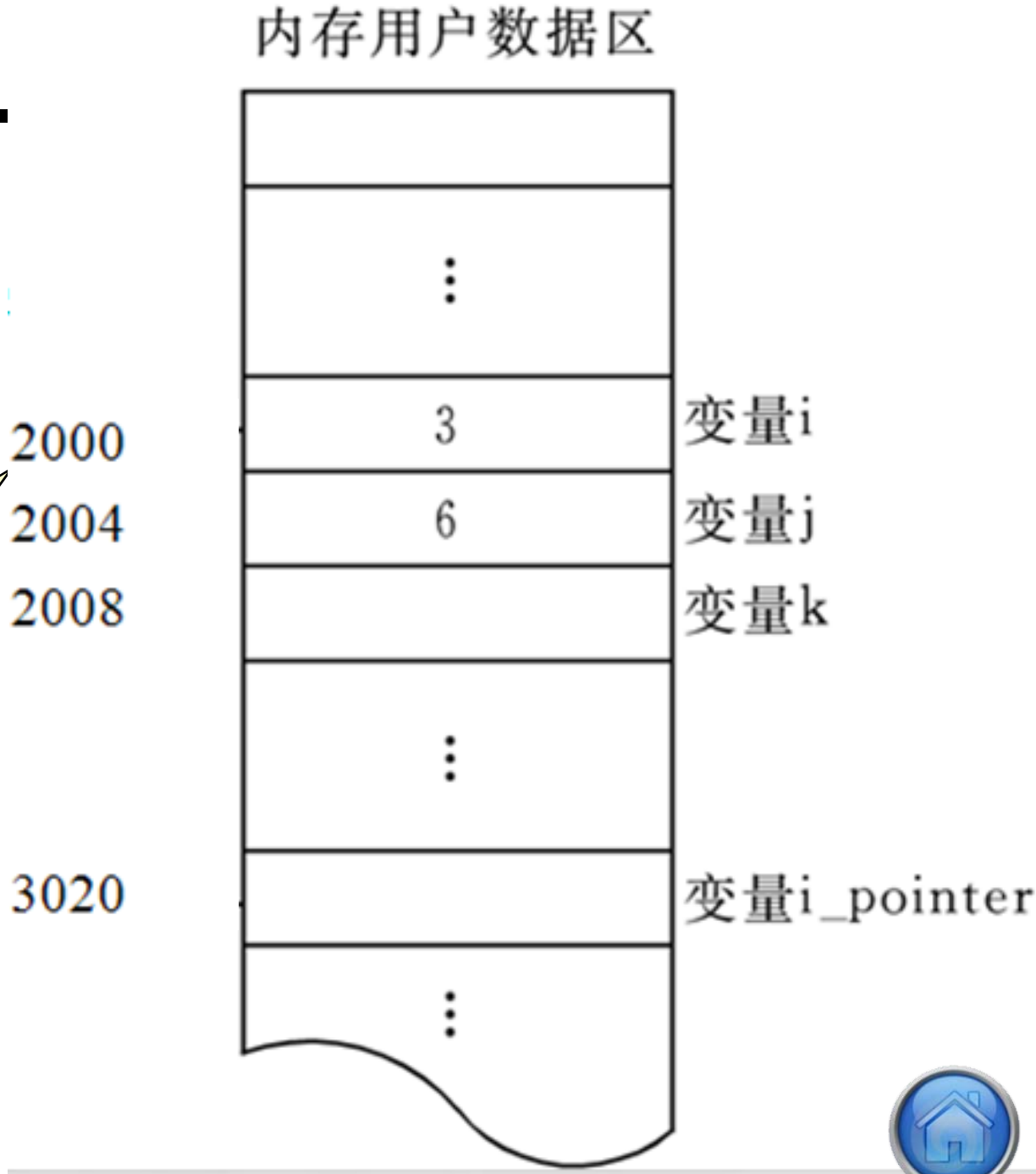
- 务必弄清楚存储单元的地址和存储单元的内容这两个概念的区别

例如：

```
int i=3,j=6,k;

printf( “%d” ,i);
```

通过变量名i

找到i的地址2000，从而从存储单元读取3



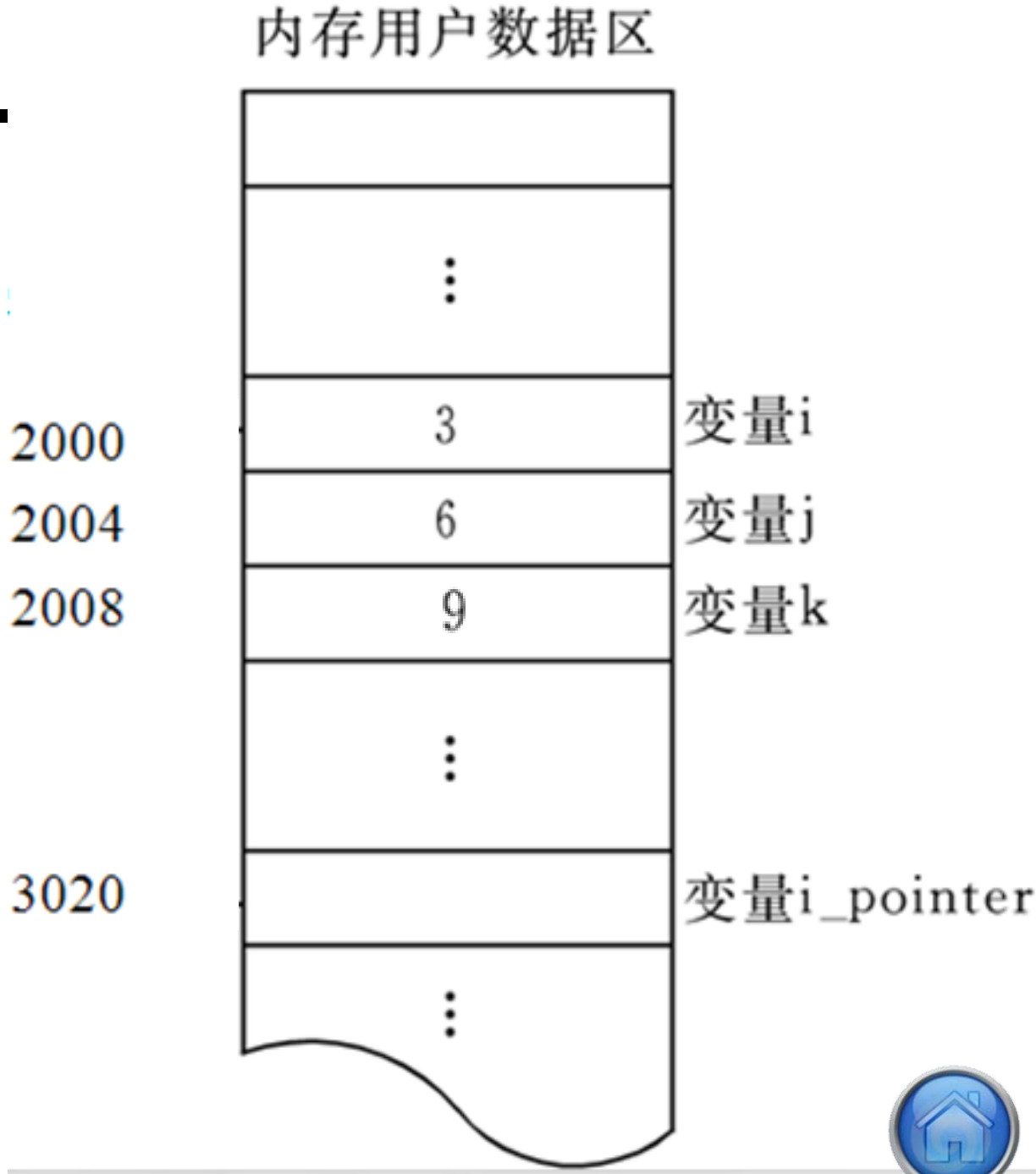
```
int i=3,j=6,k;
k=i+j;
```

从这里取3

从这里取6

将9送到这里

直接存取



明德求新 尚用笃行

School of Software

int i=3,j=6,k;

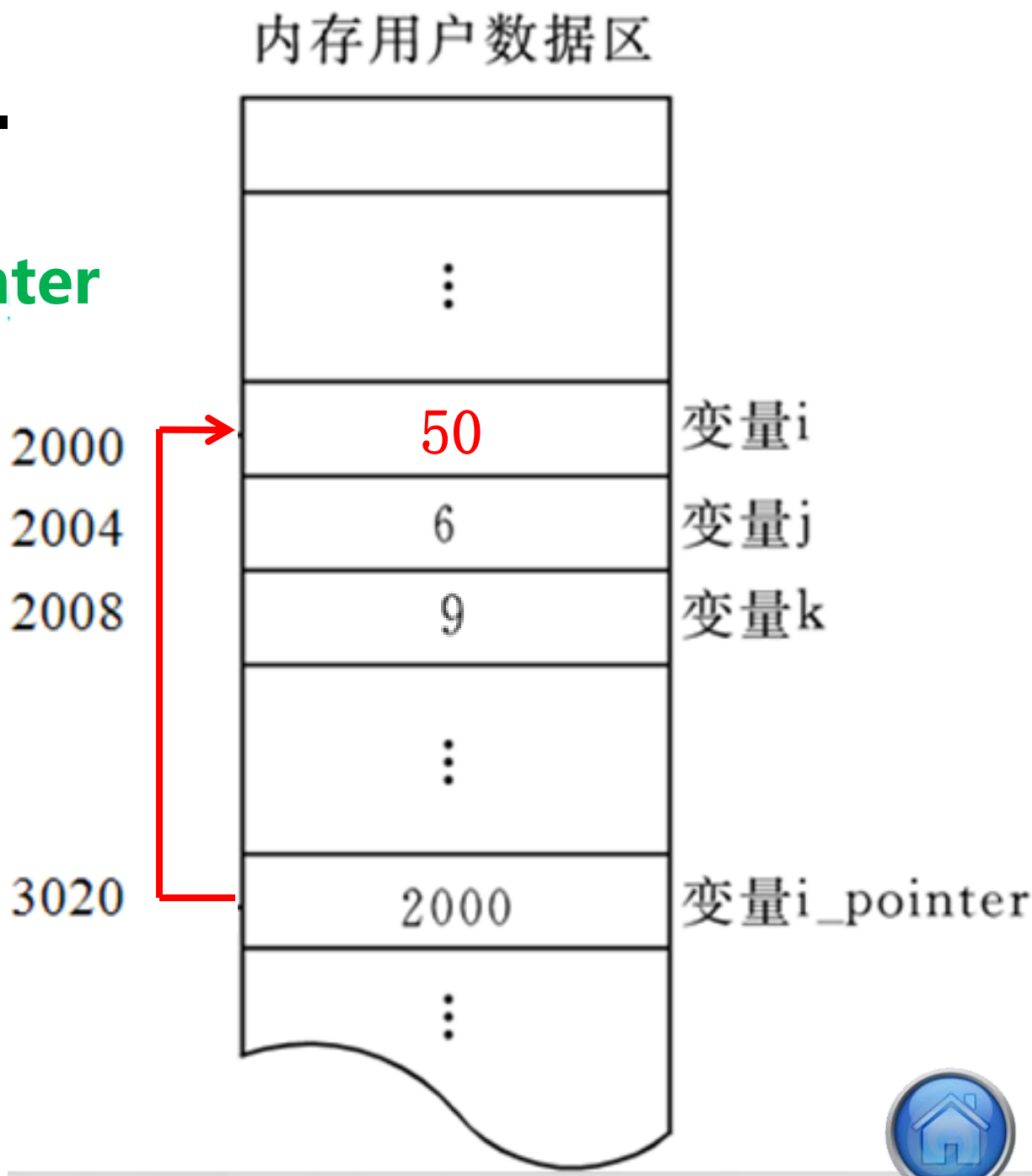
定义特殊变量i_pointer

i_pointer=&i;

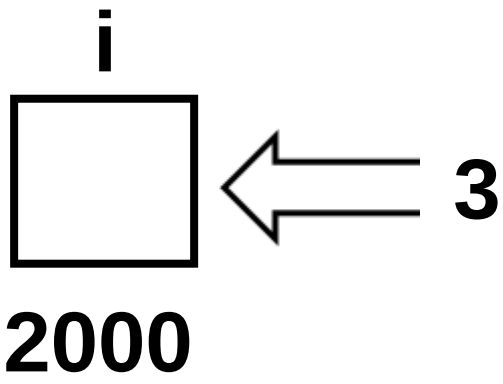
*i_pointer=50;

将i的地址存到这里

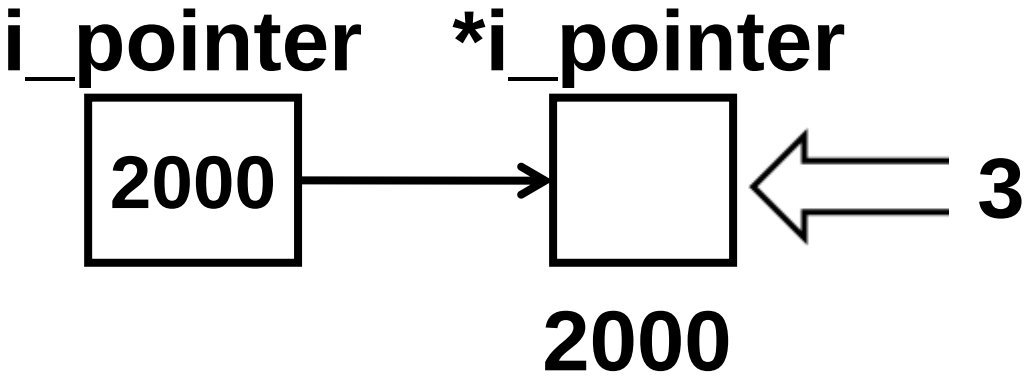
间接存取



直接存取



间接存取



明德求新 尚用笃行

School of Software

- 为了表示将数值 3 送到变量中，可以有两种表达方法：

(1) 将**3**直接送到变量**i**所标识的单元中，例如：**i=3;**

(2) 将**3**送到变量**i_pointer**所指向的单元（即变量**i**的存储单元），例如：***i_pointer=3;** 其中***i_pointer**表示**i_pointer**指向的对象

明德求新 尚用笃行

- 指向就是通过地址来体现的
  - 假设**i_pointer**中的值是变量 i 的地址**(2000)**，这样就在**i_pointer**和变量 i 之间建立起一种联系，即通过**i_pointer**能知道**i**的地址，从而找到变量**i**的内存单元

- 由于通过地址能找到所需的变量单元，因此说，地址指向该变量单元
- 将地址形象化地称为“指针”。意思是通过它能找到以它为地址的内存单元

明德求新 尚用笃行

- 一个变量的地址称为该变量的“指针”

例如，地址2000是变量 i 的指针

- 如果有一个变量专门用来存放另一变量的地址（即指针），则它称为“指针变量”
- **i_pointer**就是一个指针变量。指针变量就是地址变量，用来存放地址的变量，指针变量的值是地址（即指针）

- “指针”和“指针变量”是不同的概念
- 可以说变量i的指针是2000，而不能说i的指针变量是2000
- 指针是一个地址，而指针变量是存放地址的变量

8.2 指针变量

8.2.1使用指针变量的例子

例8.1 通过指针变量访问整型变量。

- 解题思路：先定义2个整型变量，再定义2个指针变量，分别指向这两个整型变量，通过访问指针变量，可以找到它们所指向的变量，从而得到这些变量的值。

```
a=100,b=10
*pointer_1=100,*pointer_2=10
```

```
#include <stdio.h>
int main()
{ int a=100,b=10;
   int *pointer_1, *pointer_2;
   pointer_1=&a;
   pointer_2=&b;
   printf( “a=%d,b=%d\n” ,a,b);
   printf( “*pointer_1=%d,*pointer_2=
            %d\n” ,*pointer_1,*pointer_2);
   return 0;
}
```

定义两个指针变量

直接输出变量a和b的值

间接输出变量a和b的值

```
#include <stdio.h>
int main()
{ int a=100,b=10;
   int *pointer_1, *pointer_2;
   pointer_1=&a;
   pointer_2=&b;
   printf( “a=%d,b=%d\n” ,a,b);
   printf( “*pointer_1=%d,*pointer_2=
            %d\n” ,*pointer_1,*pointer_2);
   return 0;
}
```

此处*与类型名在一起。此时共同定义指针变量

此处*与指针变量一起使用。此时代表指针变量所指向的变量

8.2.2 怎样定义指针变量

- 定义指针变量的一般形式为：

  类型 * 指针变量名;

  如：**int *pointer_1, *pointer_2;**

  - **int**是为指针变量指定的“基类型”
  - 基类型指定指针变量可指向的变量类型
  - 如**pointer_1**可以指向整型变量，但不能指向浮点型变量

8.2.2 怎样定义指针变量

- 下面都是合法的定义和初始化：

```
float  *pointer_3;
char  *pointer_4;
int  a,b;
int  *pointer_1=&a,*pointer_2=&b;
```

pointer_3=&a; 错误

8.2.3 怎样引用指针变量

使p指向a

- 在引用指针变量时，可能有三种情况：
  - 给指针变量赋值：p=&a;
  - 引用指针变量指向的变量。如有

    p=&a; *p=1;

    则执行printf(“%d” ,*p); 将输出1
  - 引用指针变量的值。如：printf(“%o” ,p);

*p相当于a

以八进制输出a的地址

8.2.3 怎样引用指针变量

- 要熟练掌握两个有关的运算符：囍

(1) & 取地址运算符。

&a是变量a的地址

(2) * 指针运算符（“间接访问”运算符）

如果： p指向变量a，则*p就代表a。

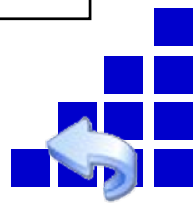
k=*p; (把a的值赋给k)

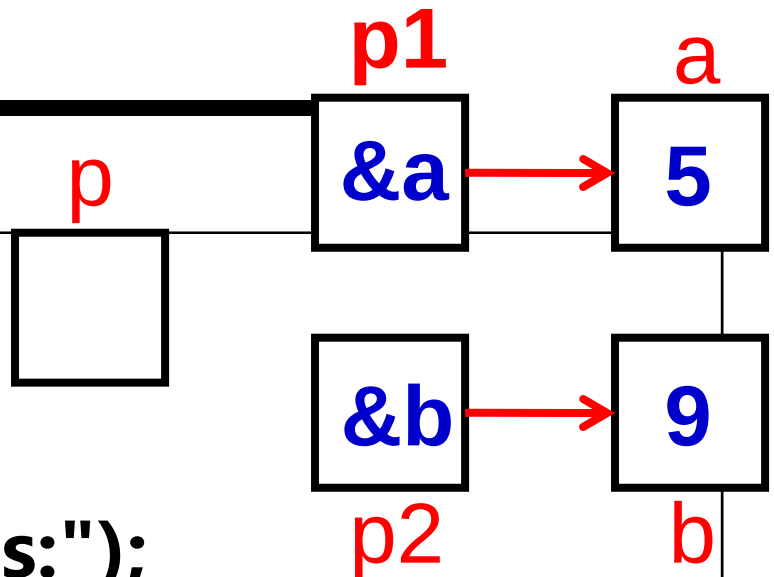
*p=1; (把1赋给a)

明德求新 尚用笃行

例8.2 输入a和b两个整数，按先大后小的顺序输出a和b。

- 解题思路：用指针方法来处理这个问题。不交换整型变量的值，而是交换两个指针变量的值。

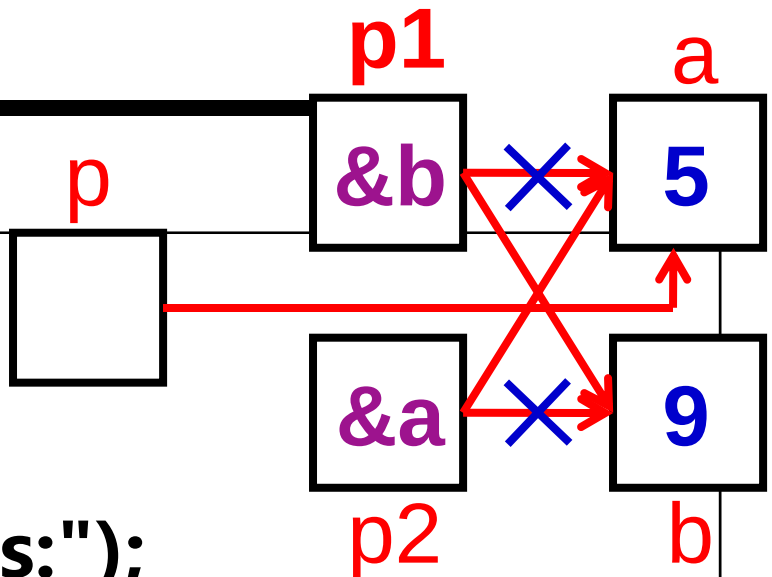




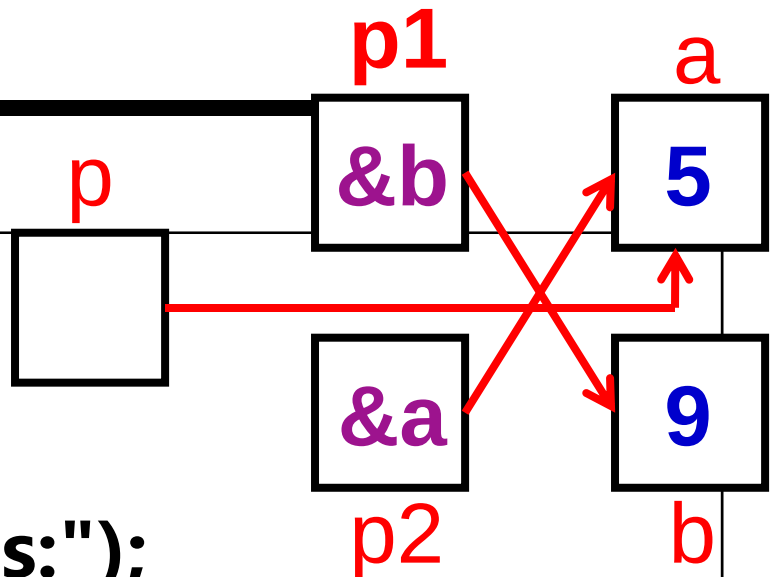

```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b;
   printf( "integer numbers:");
   scanf( "%d,%d" ,&a,&b);
   p1=&a;   p2=&b;
   if(a<b)
   {  p=p1; p1=p2; p2=p; }
   printf( "a=%d,b=%d\n" ,a,b);
   printf( "%d,%d\n" ,*p1,*p2);
   return 0;
}
```

5,9

成立

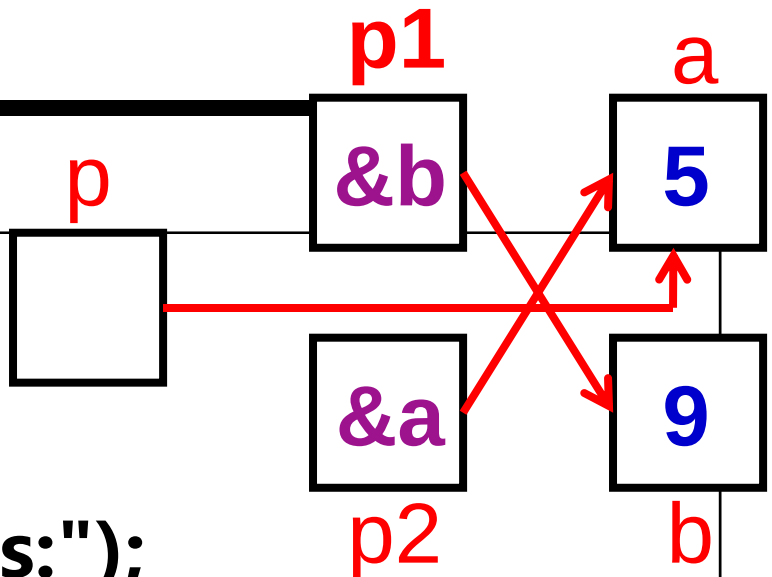



```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b;
   printf( “integer numbers:");
   scanf( “%d,%d” ,&a,&b);
   p1=&a;    p2=&b;
   if(a<b)
   {  p=p1; p1=p2; p2=p; }
   printf( “a=%d,b=%d\n” ,a,b);
   printf( “%d,%d\n” ,*p1,*p2);
   return 0;
}
```




```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b;
   printf( “integer numbers:");
   scanf( “%d,%d” ,&a,&b);
   p1=&a;    p2=&b;
   if(a<b)
   {  p=p1; p1=p2; p2=p; }
   printf( “a=%d,b=%d\n” ,a,b);
   printf( “%d,%d\n” ,*p1,*p2);
   return 0;
}
```

```
a=5,b=9
9,5
```




```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b;
   printf( "integer numbers:");
   scanf( "%d,%d" ,&a,&b);
   p1=&a;    p2=&b;
   if(a<b)
   {  p=p1; p1=p2; p2=p; }
   printf( "a=%d,b=%d\n" ,a,b);
   printf( "%d,%d\n" ,*p1,*p2);
   return 0;
}
```

可否改为p1=&b; p2=&a;?

- 注意:
  - a和b的值并未交换，它们仍保持原值
  - 但p1和p2的值改变了。p1的值原为&a，后来变成&b，p2原值为&b，后来变成&a
  - 这样在输出*p1和*p2时，实际上是输出变量b和a的值，所以先输出9，然后输出5

8.2.4 指针变量作为函数参数

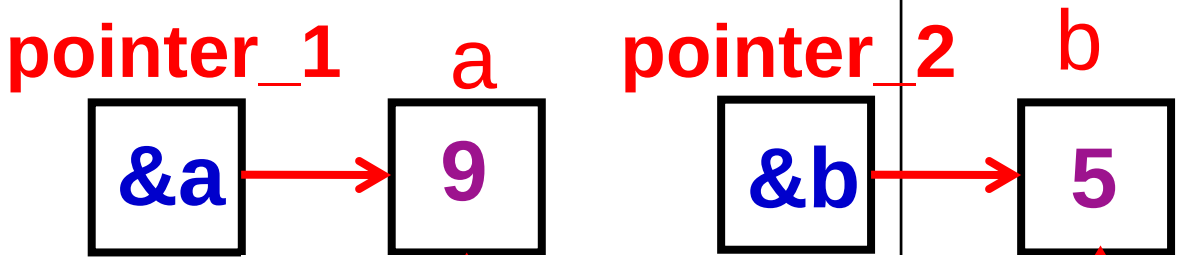
例8.3 题目要求同例8.2，即对输入的两个整数按大小顺序输出。现用函数处理，而且用指针类型的数据作函数参数。

- 解题思路：定义一个函数swap，将指向两个整型变量的指针变量作为实参传递给swap函数的形参指针变量，在函数中通过指针实现交换两个变量的值。

```
#include <stdio.h>
int main()
{void swap(int *p1,int *p2);
  int a,b;  int*pointer_1,*pointer_2;
  printf("please enter a and b:");
  scanf( “%d,%d” ,&a,&b);
  pointer_1=&a;
  pointer_2=&b;
  if (a<b)  swap(pointer_1,pointer_2);
  printf( “max=%d,min=%d\n” ,a,b);
  return 0;
 }
```

5,9

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




```
please enter a and b:5,9
max=9,min=5
```

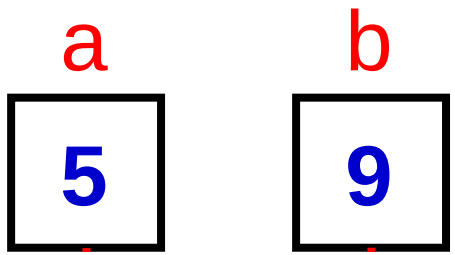
```
void swap(int *p1,int *p2)
{ int temp;
   temp=*p1;
   *p1=*p2;
   *p2=temp;
}
```

错！！！
无确定的指向

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

```
#include <stdio.h>
int main()
{……
  if (a<b)  swap(a,b);
  printf( “max=%d,min=%d\n” ,a,b);
  return 0;
 }
void swap(int x,int y)
{ int temp;
   temp=x;  x=y;   y=temp;
}
```

错！！！
无法交换a,b



- 如果想通过函数调用得到 n 个要改变的值：
  - ① 在主调函数中设 n 个变量，用 n 个指针变量指向它们
  - ② 设计一个函数，有**n**个指针形参。在这个函数中改变这 n 个形参的值
  - ③ 在主调函数中调用这个函数，在调用时将这**n**个指针变量作实参，将它们的地址传给该函数的形参
  - ④ 在执行该函数的过程中，通过形参指针变量，改变它们所指向的 n 个变量的值
  - ⑤主调函数中就可以使用这些改变了值的变量

例8.4 对输入的两个整数按大小顺序输出。

- 解题思路：尝试调用swap函数来实现题目要求。在函数中改变形参(指针变量)的值，希望能由此改变实参(指针变量)的值

```
#include <stdio.h>
int main()
 {void swap(int *p1,int *p2);
  int a,b;  int*pointer_1,*pointer_2;
  scanf("%d,%d",&a,&b);
  pointer_1=&a;  pointer_2=&b;
  if (a<b)  swap(pointer_1,pointer_2);
  printf("max=%d,min=%d\n",a,b);
  return 0;
 }
```

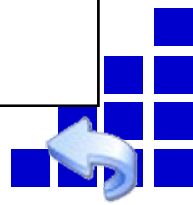
```
5,9
```

```
max=5,min=9
```

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

错！！！
只交换形参指向

- 注意：函数的调用可以（而且只可以）得到一个返回值（即函数值），而使用指针变量作参数，可以得到多个变化了的值。如果不用指针变量是难以做到这一点的。
- 要善于利用指针法。



例8.5 输入3个整数a,b,c，要求按由大到小的顺序将它们输出。用函数实现。

- 解题思路：采用例8.3的方法在函数中改变这3个变量的值。用swap函数交换两个变量的值，用exchange函数改变这3个变量的值。

```
#include <stdio.h>
int main()
{ void exchange(int *q1, int *q2, int *q3);
   int a,b,c,*p1,*p2,*p3;
   scanf("%d,%d,%d",&a,&b,&c);
   p1=&a;p2=&b;p3=&c;
   exchange(p1,p2,p3);
   printf( “%d,%d,%d\n",a,b,c);
   return 0;
}
```

```
20,-54,67
```

调用结束后不会改变指针的指向

```
void exchange(int *q1, int *q2, int *q3)
{ void swap(int *pt1, int *pt2);
   if(*q1<*q2) swap(q1,q2);
   if(*q1<*q3) swap(q1,q3);
   if(*q2<*q3) swap(q2,q3);
}
void swap(int *pt1, int *pt2)
{ int temp;
   temp=*pt1;  *pt1=*pt2;  *pt2=temp;
 }
```

```
20,-54,67
67,20,-54
```

交换指针指向的变量值

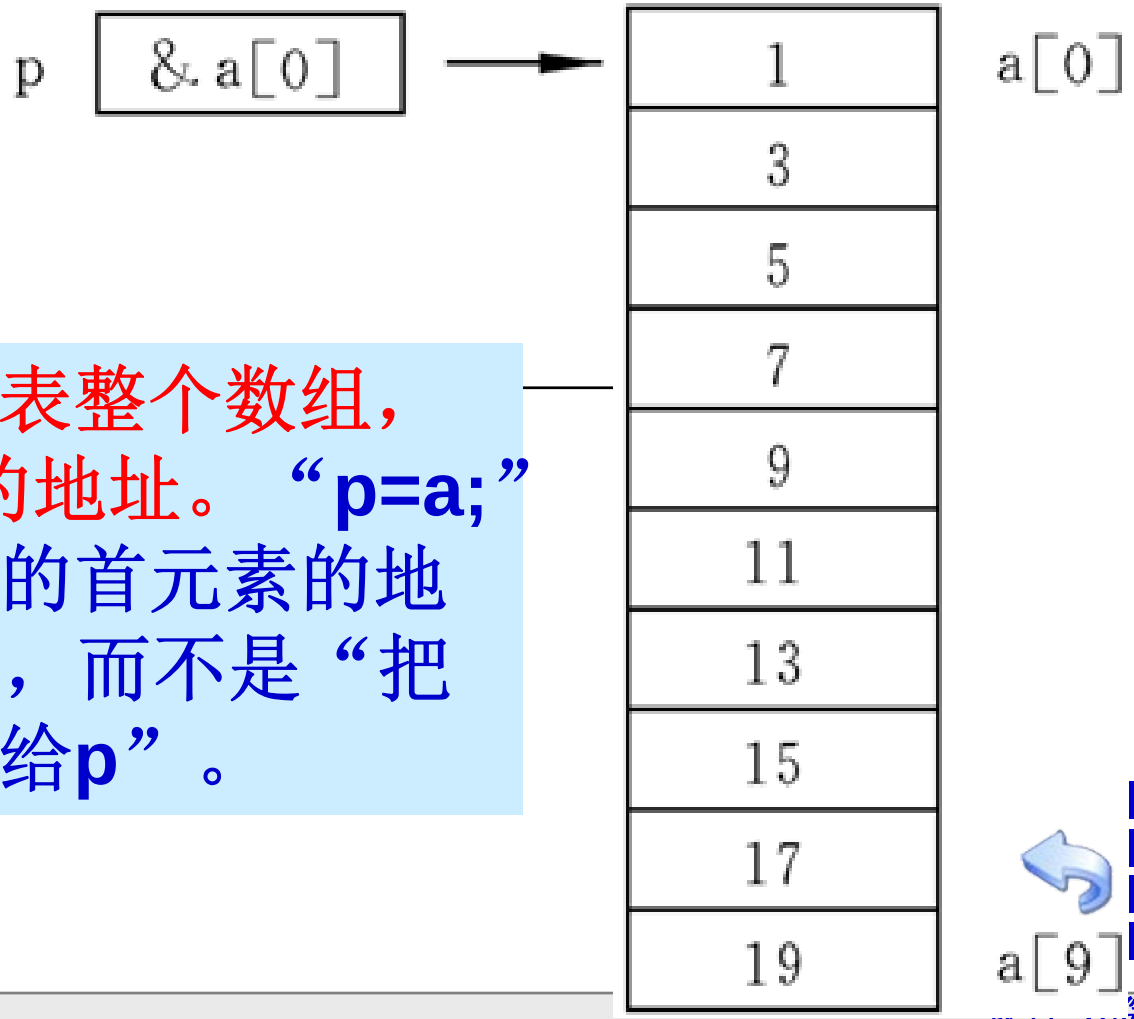
8.3通过指针引用数组

8.3.1 数组元素的指针

- 一个变量有地址，一个数组包含若干元素，每个数组元素都有相应的地址
- 指针变量可以指向数组元素（把某一元素的地址放到一个指针变量中）
- 所谓数组元素的指针就是数组元素的地址

- 可以用一个指针变量指向一个数组元素

```
int a[10]={1,3,5,7,9,11,13,15,17,19};
int  *p;
p=&a[0];
```



注意：数组名a不代表整个数组，只代表数组首元素的地址。“p=a;”的作用是“把a数组的首元素的地址赋给指针变量p”，而不是“把数组a各元素的值赋给p”。

8.3.2 在引用数组元素时指针的运算

- 在指针指向数组元素时，允许以下运算：
  - 加一个整数(用+或+=)，如p+1
  - 减一个整数(用-或-=)，如p-1
  - 自加运算，如p++，++p
  - 自减运算，如p--，--p
  - 两个指针相减，如p1-p2 (只有p1和p2都指向同一数组中的元素时才有意义)

(1) 如果指针变量p已指向数组中的一个元素，则p+1指向同一数组中的下一个元素，p-1指向同一数组中的上一个元素。

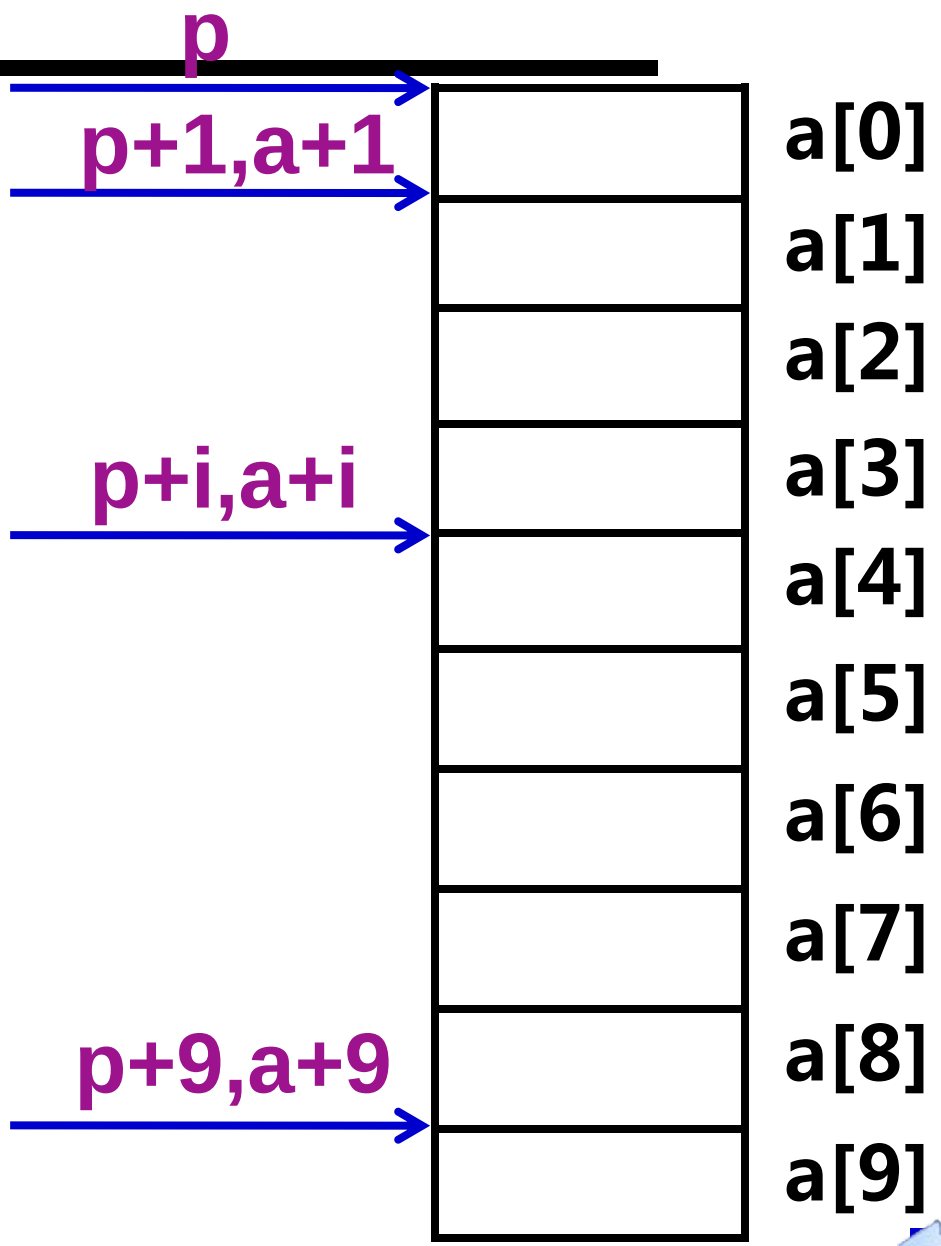
```
float a[10],*p=a;
```

假设a[0]的地址为2000，则

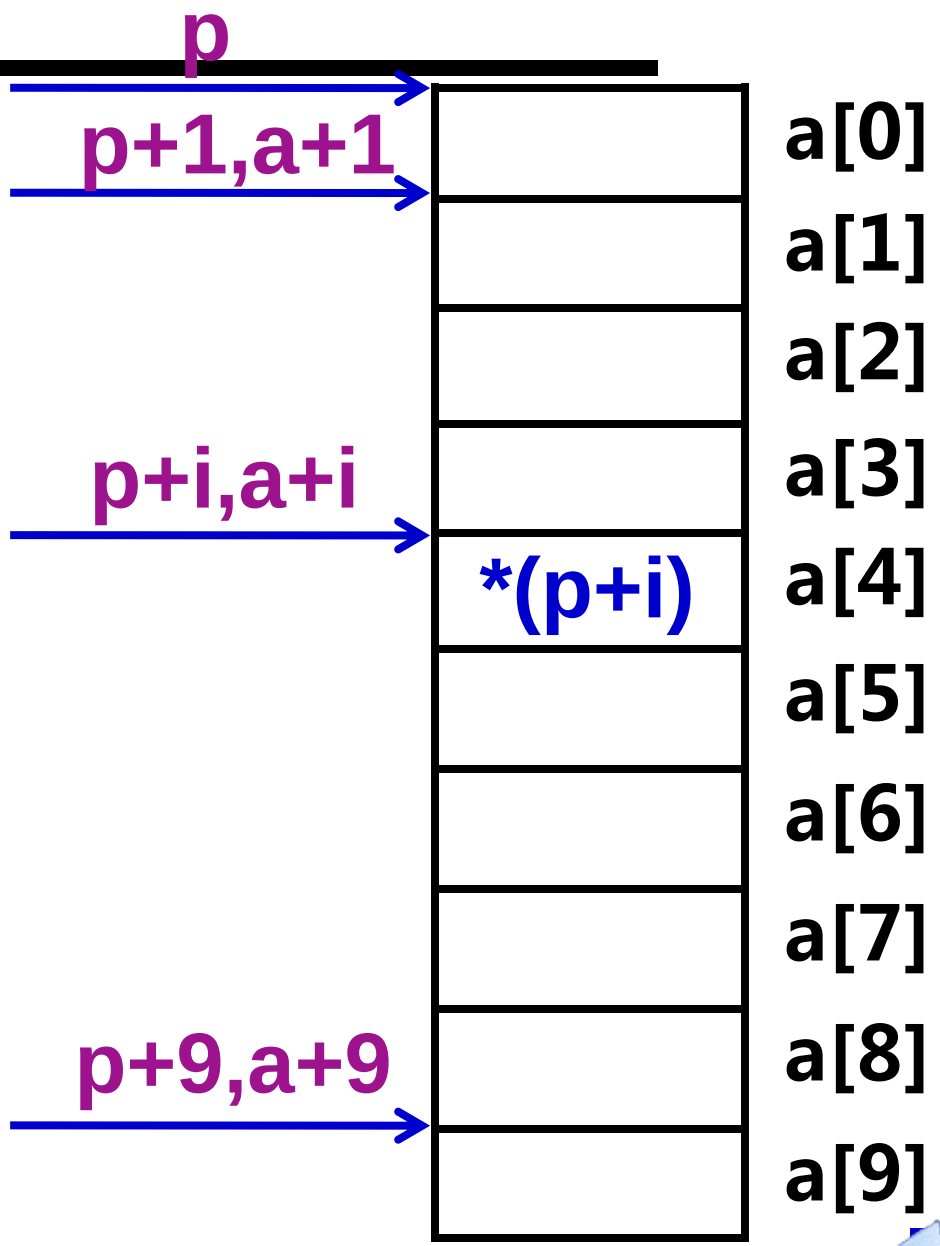
- p的值为2000
- p+1的值为2004
- P-1的值为1996

越界

(2) 如果 p 的初值为&a[0]，则p+i和a+i就是数组元素a[i]的地址，或者说，它们指向a数组序号为i的元素



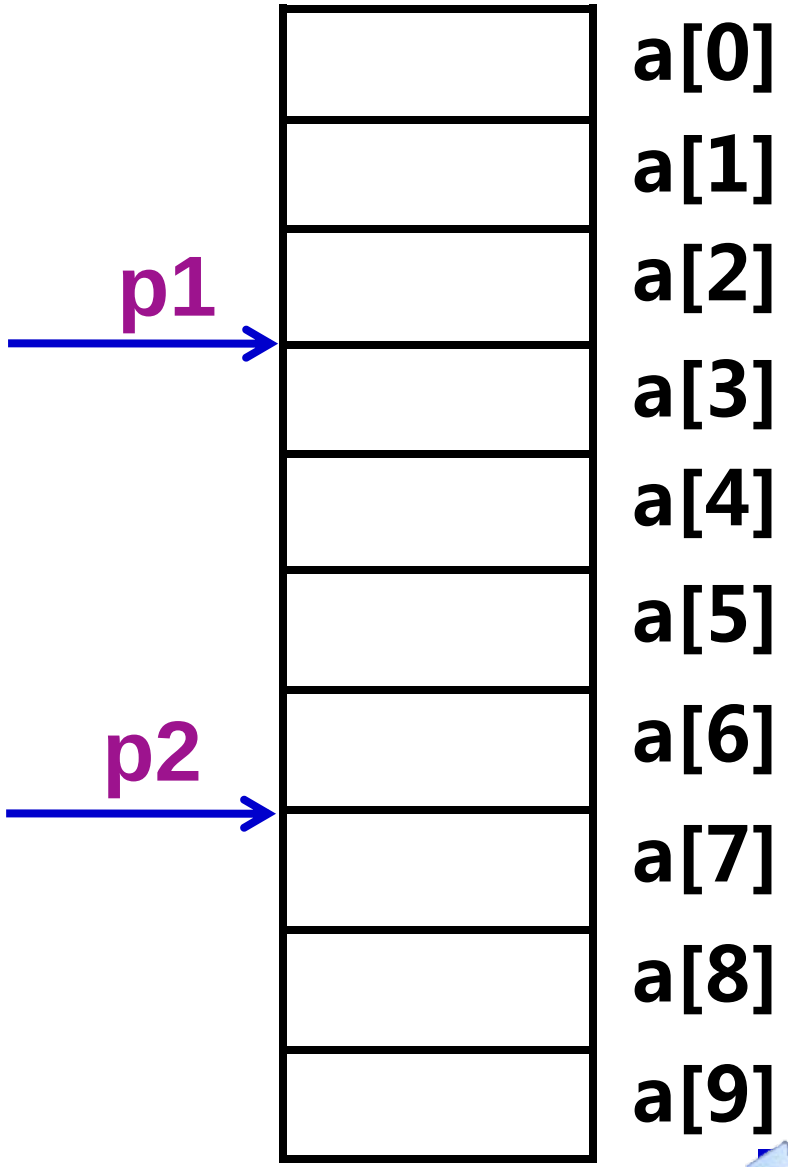
(3) *(p+i)或*(a+i)是p+i或a+i所指向的数组元素，即a[i]。



(4) 如果指针p1和p2都指向同一数组

p2-p1的值是4

不能p1+p2



8.3.3 通过指针引用数组元素

- 引用一个数组元素，可用下面两种方法：
  （1） 下标法，如a[i]形式
  （2） 指针法，如*(a+i)或*(p+i)

其中a是数组名，p是指向数组元素的指针变量，其初值p=a

8.3.3 通过指针引用数组元素

例8.6 有一个整型数组a，有10个元素，要求输出数组中的全部元素。

- 解题思路：引用数组中各元素的值有3种方法：(1)下标法；(2)通过数组名计算数组元素地址，找出元素的值；(3) 用指针变量指向数组元素
- 分别写出程序，以资比较分析。

(1) 下标法。

```
#include <stdio.h>
int main()
{ int a[10];  int i;
   printf( “enter 10 integer numbers:\n");
   for(i=0;i<10;i++) scanf("%d",&a[i]);
   for(i=0;i<10;i++)  printf( “%d ” ,a[i]);
   printf("%\n");
   return 0;
 }
```

```
enter 10 integer numbers:
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

(2) 通过数组名计算数组元素地址，找出元素的值

```
#include <stdio.h>
int main()
{ int a[10];  int i;
   printf( “enter 10 integer numbers:\n");
   for(i=0;i<10;i++)  scanf("%d",&a[i]);
   for(i=0;i<10;i++)
         printf( “%d ” ,*(a+i));
   printf("\n");
   return 0;
 }
```

scanf("%d",a+i);

(3) 用指针变量指向数组元素

```
#include <stdio.h>
int main()
{ int a[10];  int *p,i;
   printf( “enter 10 integer numbers:\n");
   for(i=0;i<10;i++)  scanf("%d",&a[i]);
   for(p=a;p<(a+10);p++)
        printf( “%d ” ,*p);
   printf("\n");
   return 0;
}
```

```
for(p=a;p<(a+10);p++)
```

```
for(p=a;p<(a+10);a++)
    printf( “%d ” ,*a);
```

错！

- 3种方法的比较：

① 第(1)和第(2)种方法执行效率相同

- C编译系统是将a[i]转换为*(a+i)处理的，即先计算元素地址。
- 因此用第(1)和第(2)种方法找数组元素费时较多。

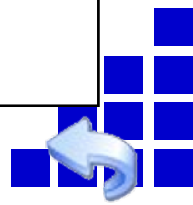
明德求新 尚用笃行

School of Software

- 3种方法的比较：

② 第(3)种方法比第(1)、第(2)种方法快

- 用指针变量直接指向元素，不必每次都重新计算地址，像p++这样的自加操作是比较快的
- 这种有规律地改变地址值(p++)能大大提高执行效率



- 3种方法的比较：

③ 用下标法比较直观，能直接知道是第几个元素。

用地址法或指针变量的方法不直观，难以很快地判断出当前处理的是哪一个元素。

例8.7 通过指针变量输出整型数组a的10个元素。

- 解题思路：

  用指针变量p指向数组元素，通过改变指针变量的值，使p先后指向a[0]到a[9]各元素。

```
#include <stdio.h>
int  main()
{ int *p,i,a[10];
   p=a;
   printf( “enter 10 integer numbers:\n");
   for(i=0;i<10;i++) scanf( “%d” ,p++);
   for(i=0;i<10;i++,p++)
        printf( “%d ” ,*p);
   printf("\n");
   return 0;
}
```

重新执行
p=a;

退出循环时**p**指向**a[9]**后面的存储单元

因此执行此循环出问题

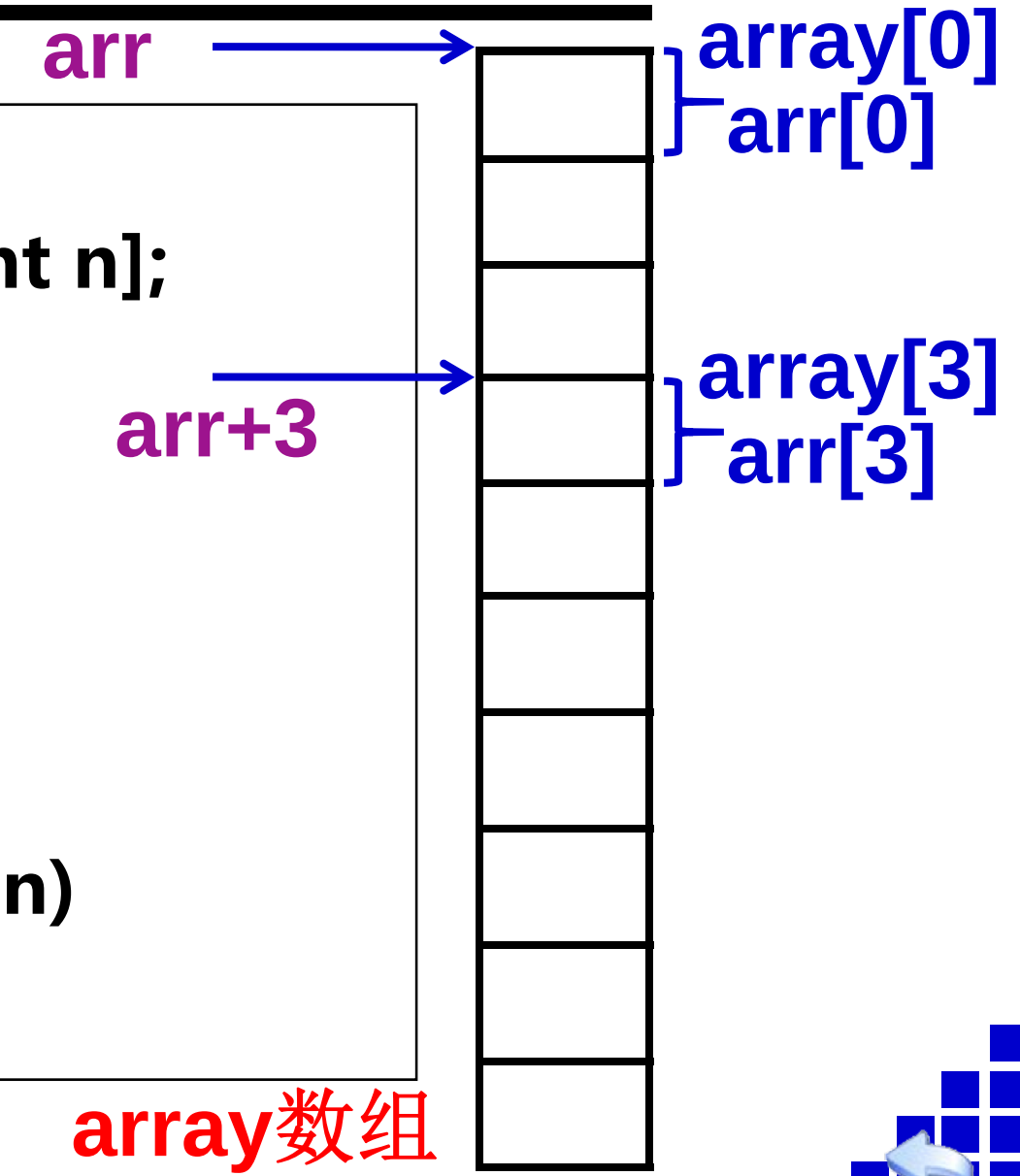
8.3.4 用数组名作函数参数

- 用数组名作函数参数时，因为实参数组名代表该数组首元素的地址，形参应该是一个指针变量
- C编译都是将形参数组名作为指针变量来处理的

```
int main()
{ void fun(int arr[],int n];
   int array[10];     ⋮
    ┆
   fun (array,10);
   return 0;
}
void fun(int arr[ ],int n)
{ ┆  }
```

fun(int *arr,int n)

```
int main()
{ void fun(int arr[],int n];
   int array[10];
    ┆
   fun (array,10);
   return 0;
}
void fun(int *arr,int n)
{ ┆ }
```

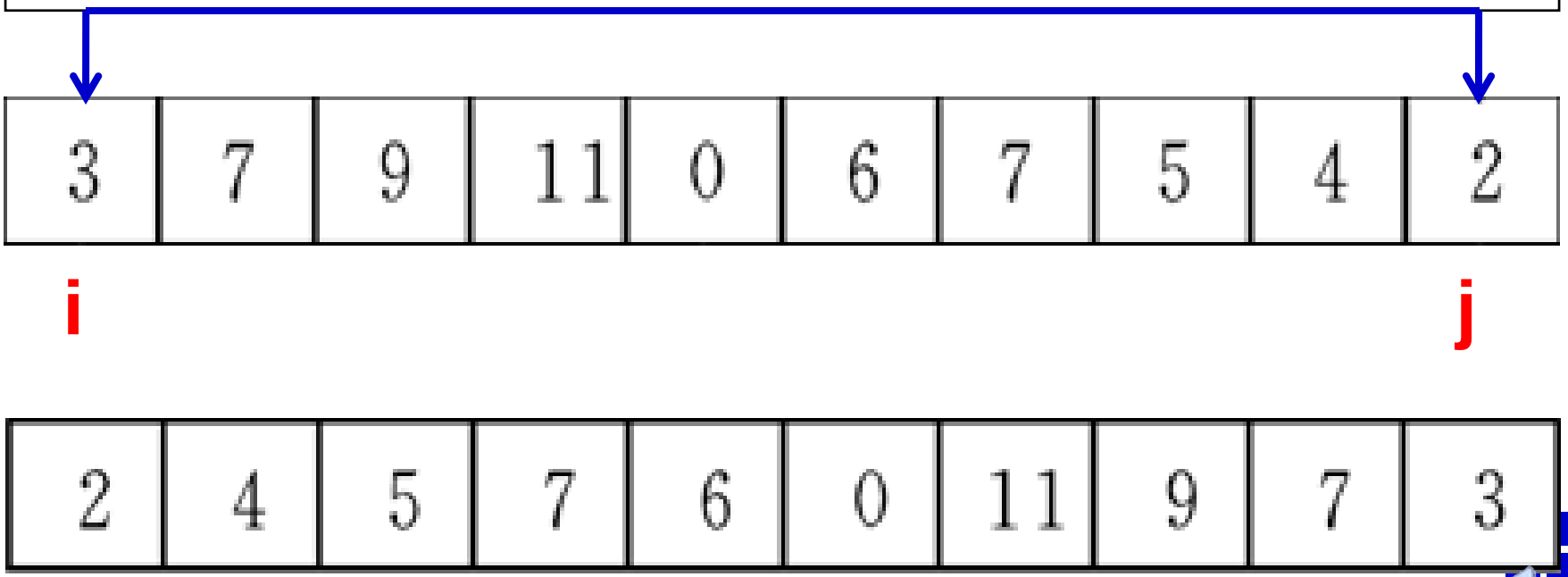


- 实参数组名是指针常量，但形参数组名是按指针变量处理
- 在函数调用进行虚实结合后，它的值就是实参数组首元素的地址
- 在函数执行期间，形参数组可以再被赋值

```
void fun (arr[ ],int n)
{ printf("%d\n", *arr);
   arr=arr+3;
   printf("%d\n", *arr);
}
```

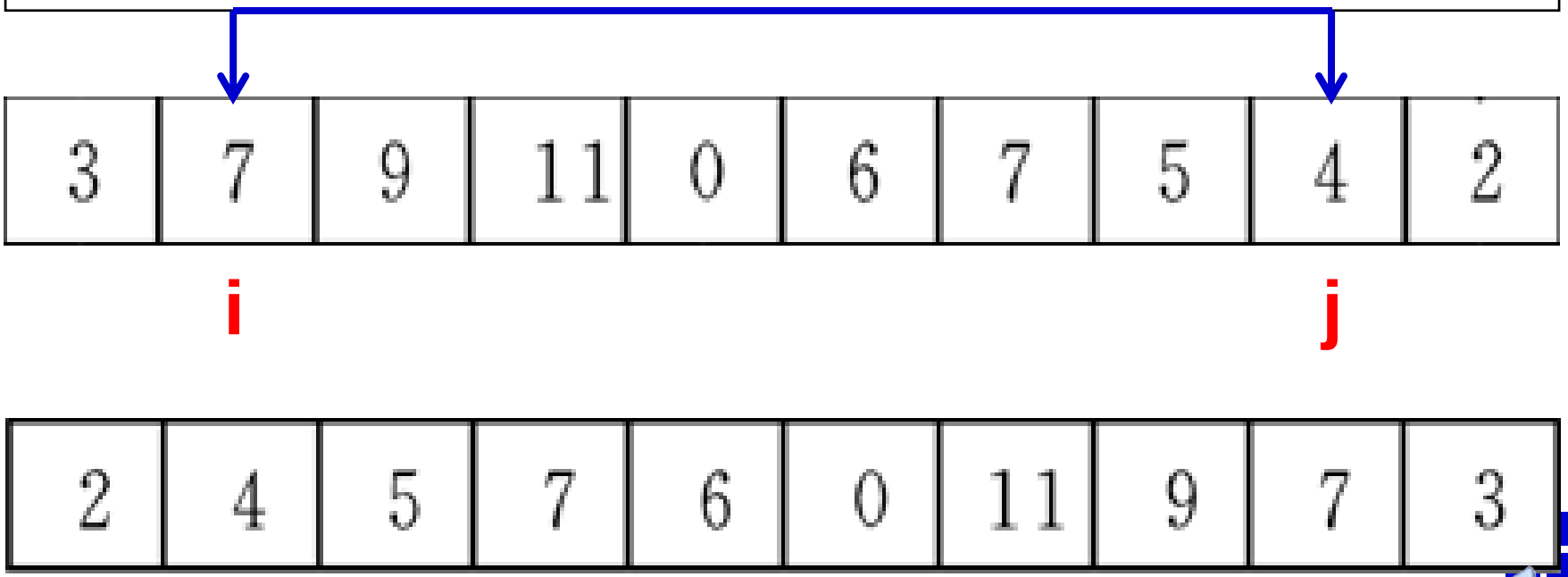
例8.8 将数组a中n个整数按相反顺序存放

- 解题思路：将a[0]与a[n-1]对换，……将a[4]与a[5]对换。



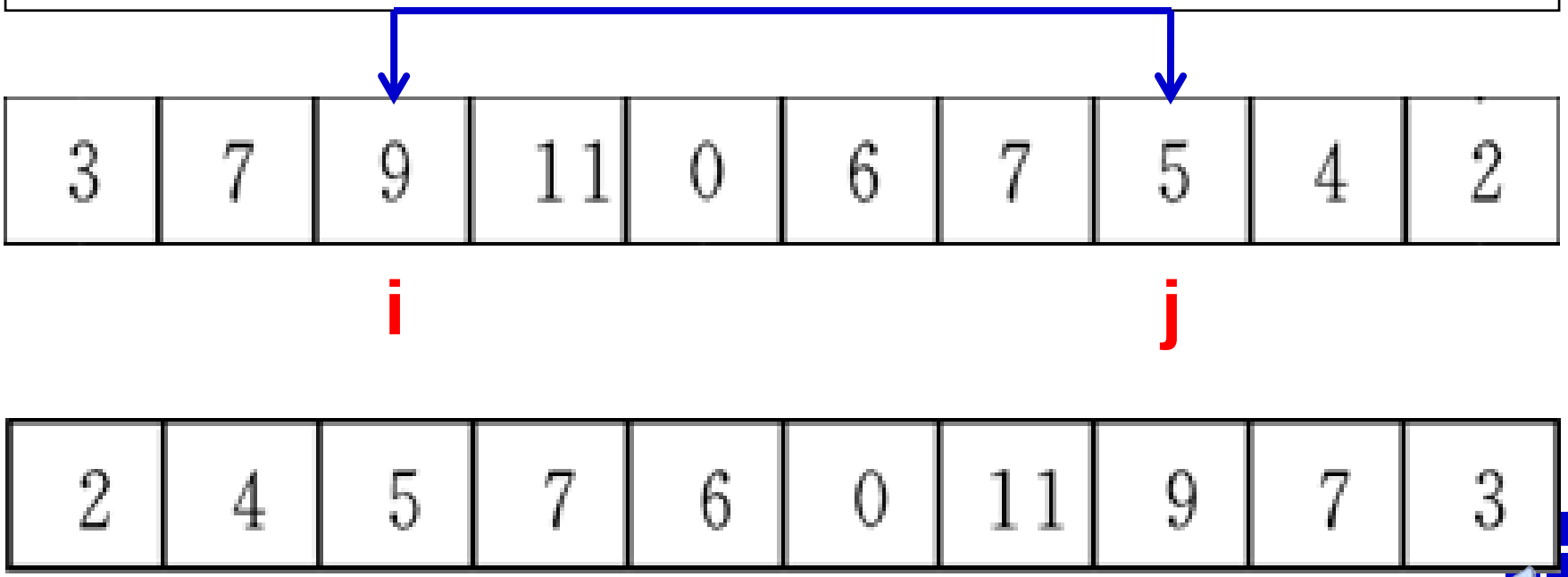
例8.8 将数组a中n个整数按相反顺序存放

- 解题思路：将a[0]与a[n-1]对换，……将a[4]与a[5]对换。



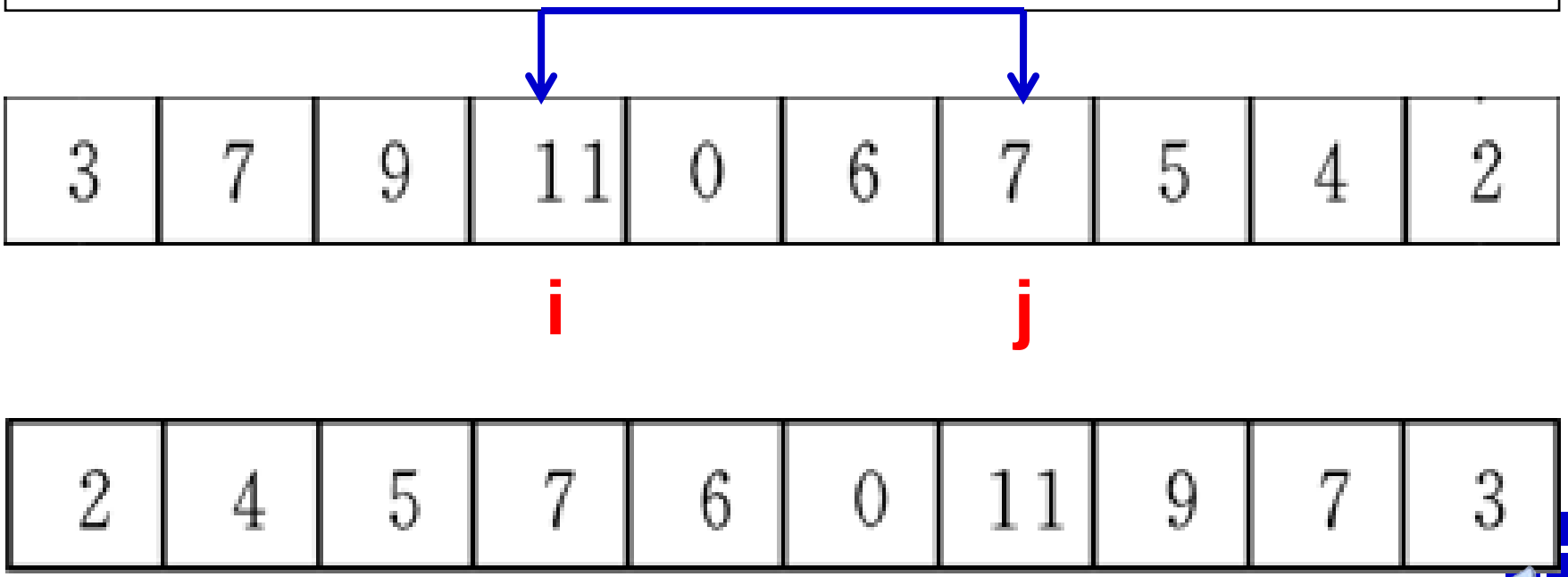
例8.8 将数组a中n个整数按相反顺序存放

- 解题思路：将a[0]与a[n-1]对换，……将a[4]与a[5]对换。



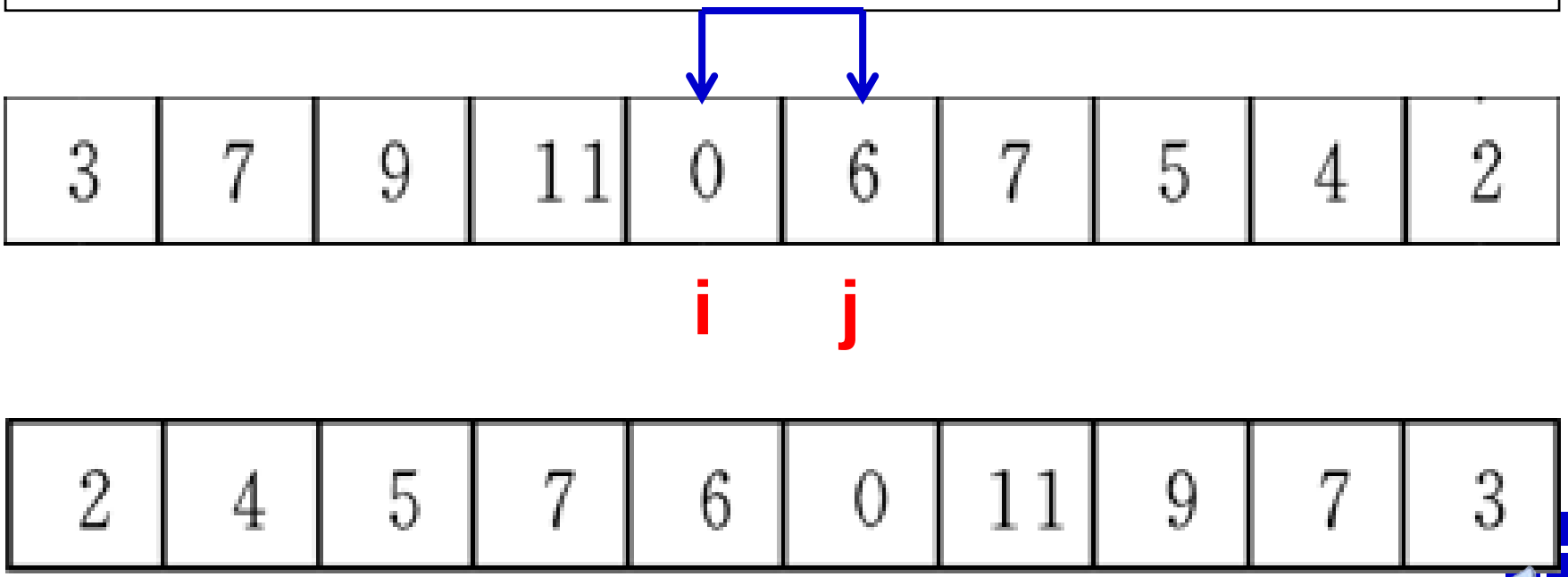
例8.8 将数组a中n个整数按相反顺序存放

- 解题思路：将a[0]与a[n-1]对换，……将a[4]与a[5]对换。



例8.8 将数组a中n个整数按相反顺序存放

- 解题思路：将a[0]与a[n-1]对换，……将a[4]与a[5]对换。



```
#include <stdio.h>
int main()
{ void inv(int x[ ],int n);
   int i, a[10]={3,7,9,11,0,6,7,5,4,2};
   for(i=0;i<10;i++) printf( “%d ” ,a[i]);
   printf("\n");
   inv(a,10);
   for(i=0;i<10;i++) printf( “%d ” ,a[i]);
   printf("\n");
   return 0;
}
```

```
void inv(int x[ ],int n)
{ int temp,i,j,m=(n-1)/2;
   for(i=0;i<=m;i++)
   { j=n-1-i;
      temp=x[i];x[i]=x[j];x[j]=temp;
   }
}
```

优化

```
void inv(int x[ ],int n)
{ int temp,*i,*j;
   i=x;  j=x+n-1;
   for(   ; i<j; i++,j--)
   { temp=*i; *i=*j; *j=temp; }
}
```

例8.9 改写例8.8，用指针变量作实参。

```
#include <stdio.h>
int main()
{ void inv(int *x,int n);
   int i, arr[10],*p=arr;
   for(i=0;i<10;i++,p++)
       scanf( “%d” ,p);
   inv(p,10);
   for(p=arr;p<arr+10;p++)
       printf( “%d ” ,*p);
   printf("\n");
   return 0;
}
```

不可少！！！

例8.10 用指针方法对10个整数按由大到小顺序排序。

- 解题思路：
  - 在主函数中定义数组a存放10个整数，定义int *型指针变量p指向a[0]
  - 定义函数sort使数组a中的元素按由大到小的顺序排列
  - 在主函数中调用sort函数，用指针p作实参
  - 用选择法进行排序

```
#include <stdio.h>
int main()
{ void sort(int x[ ],int n);
   int i,*p,a[10];
   p=a;
   for(i=0;i<10;i++)  scanf( “%d” ,p++);
   p=a;
   sort(p,10);
   for(p=a,i=0;i<10;i++)
   { printf( “%d ” ,*p);   p++;   }
   printf("\n");
   return 0;
}
```

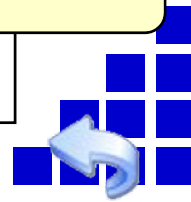
void sort(int *x,int n)

```
void sort(int x[],int n)
{ int i,j,k,t;
   for(i=0;i<n-1;i++)
   { k=i;
      for(j=i+1;j<n;j++)
        if(x[j]>x[k]) k=j;
          if(k!=i)
            { t=x[i];x[i]=x[k];x[k]=t; }
   }
}
```

if (*(x+j)>*(x+k)) k=j;

{t=*(x+i);*(x+i)=*(x+k);*(x+k)=t;}

```
12 34 5 689 -43 56 -21 0 24 65
689 65 56 34 24 12 5 0 -21 -43
```



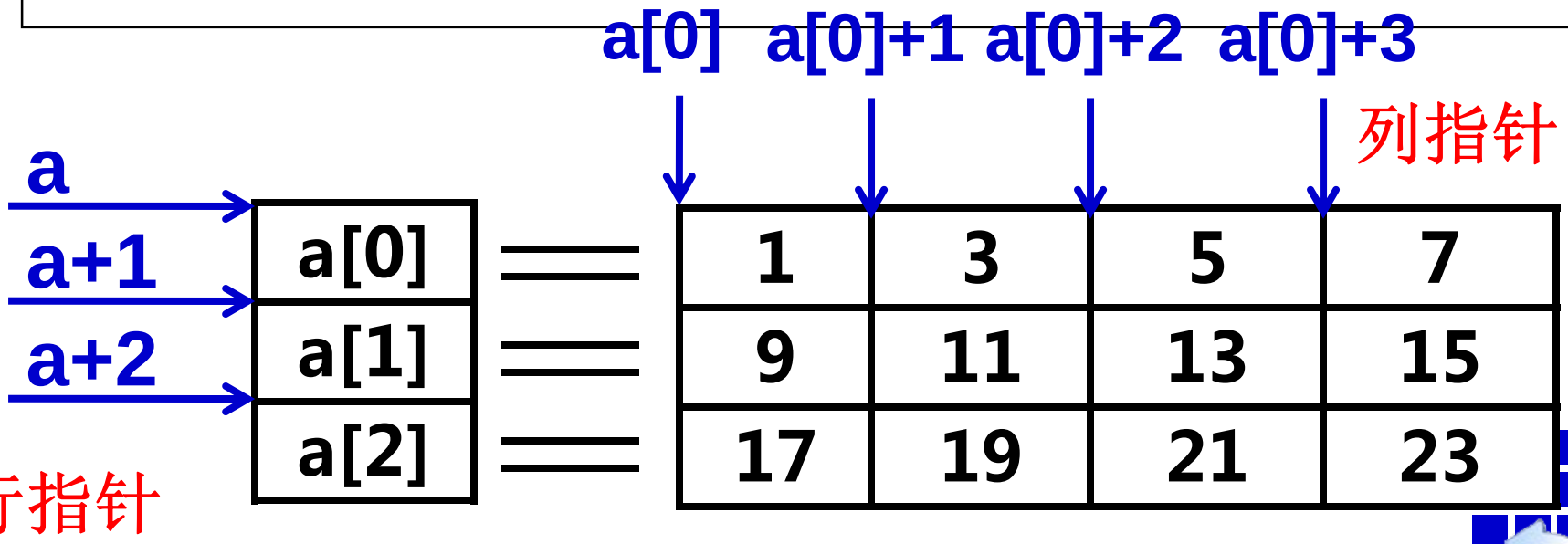
8.3.5 通过指针引用多维数组

- 指针变量可以指向一维数组中的元素，也可以指向多维数组中的元素。但在概念上和使用方法上，多维数组的指针比一维数组的指针要复杂一些。

8.3.5 通过指针引用多维数组

1. 多维数组元素的地址

```
int a[3][4]={{1,3,5,7},
     {9,11,13,15},{17,19,21,23}};
```

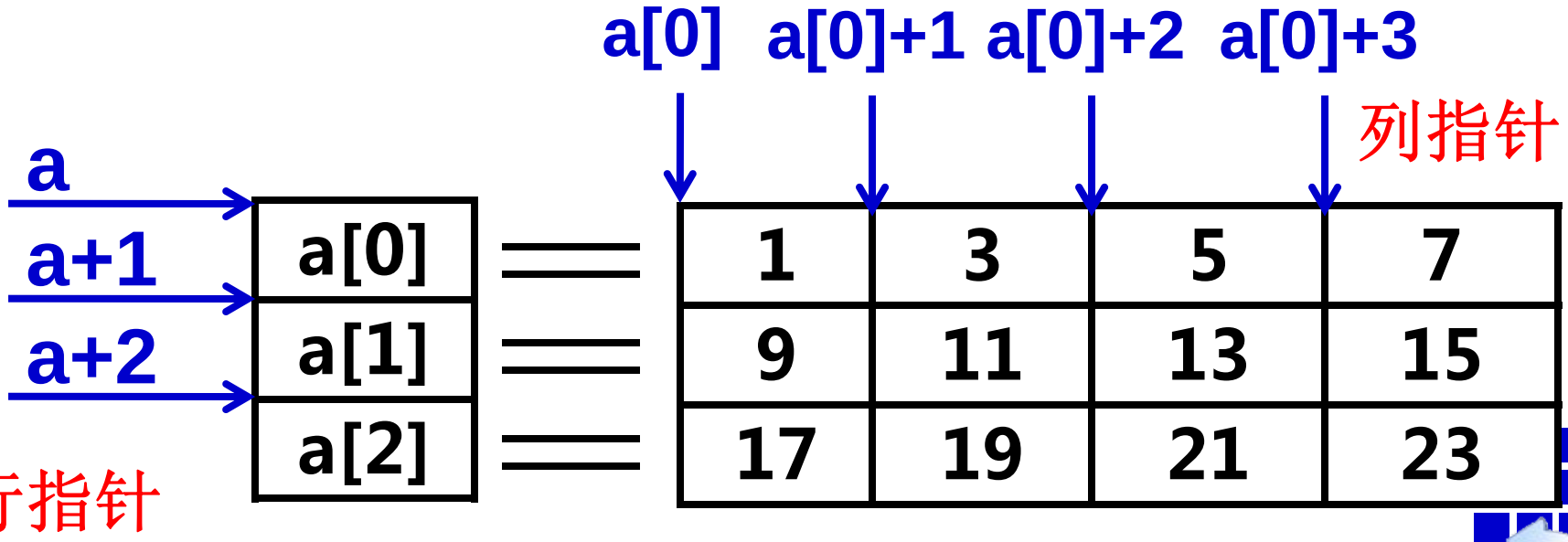


行指针每加1，走一行

a代表第0行首地址

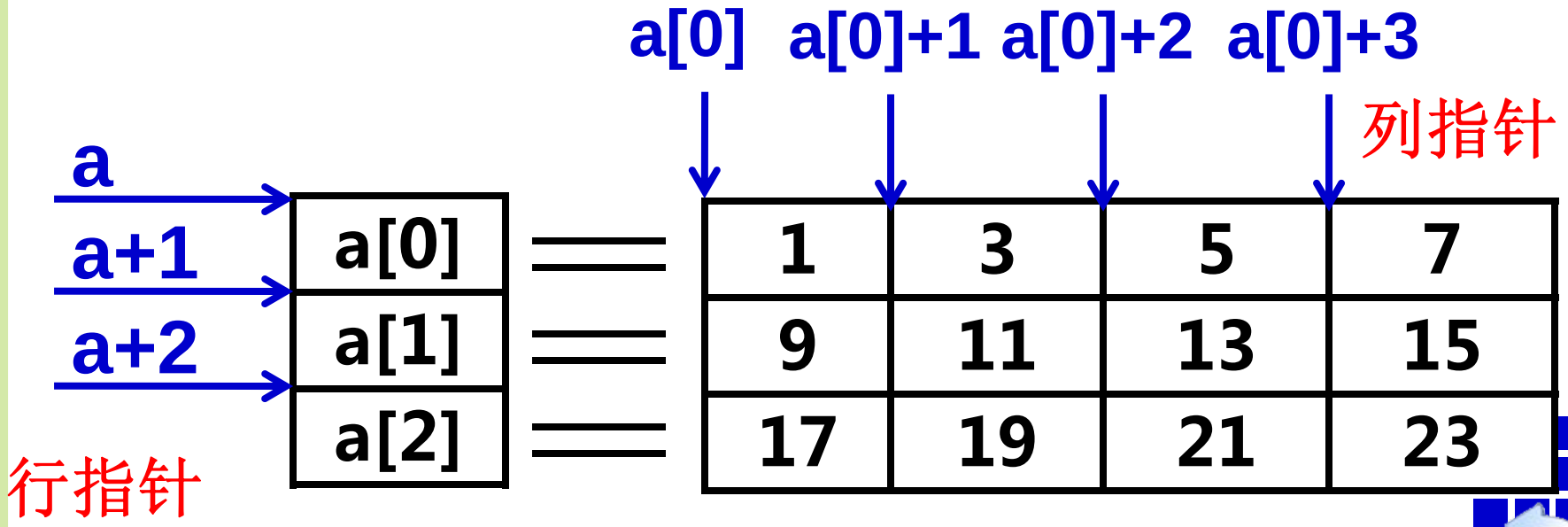
a+1代表第1行首地址

a+2代表第2行首地址



a+i代表行号为i的行首地址（按行变化）

*(a+i)代表什么？　　相当于a[i]



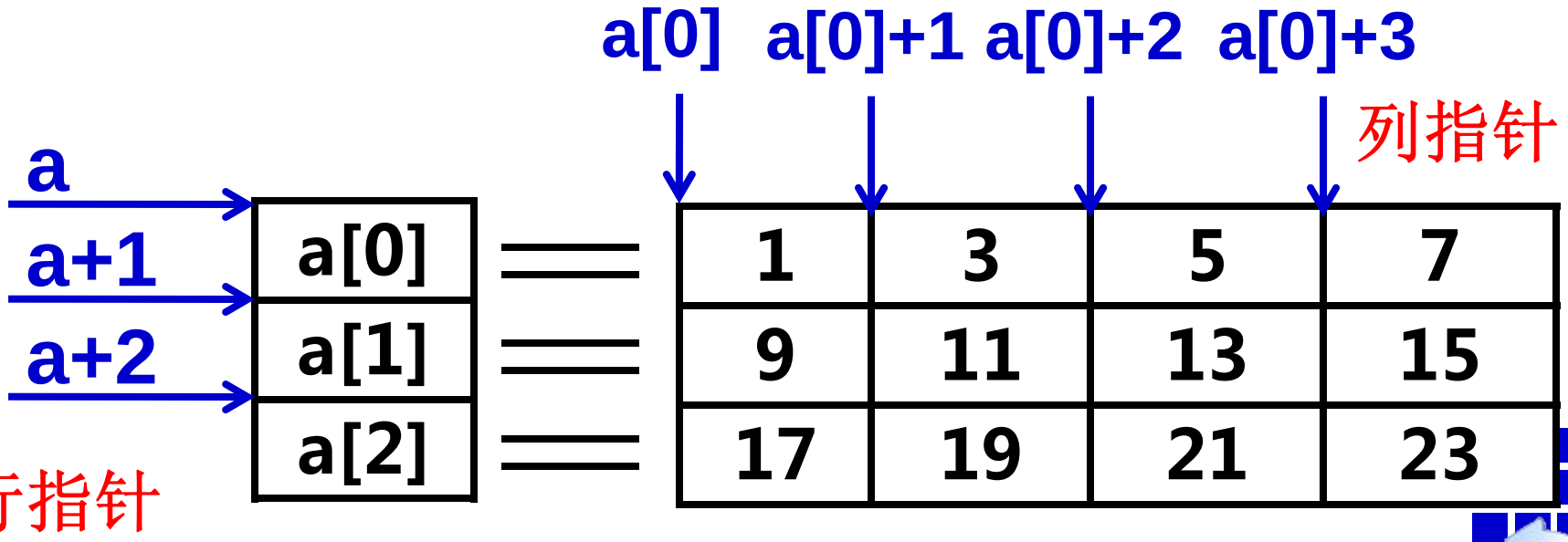
列指针每加1，走一列

a[0]代表a[0][0]的地址

a[0]+1代表a[0][1]的地址

a[0]+2代表a[0][2]的地址

a[0]+3代表a[0][3]的地址

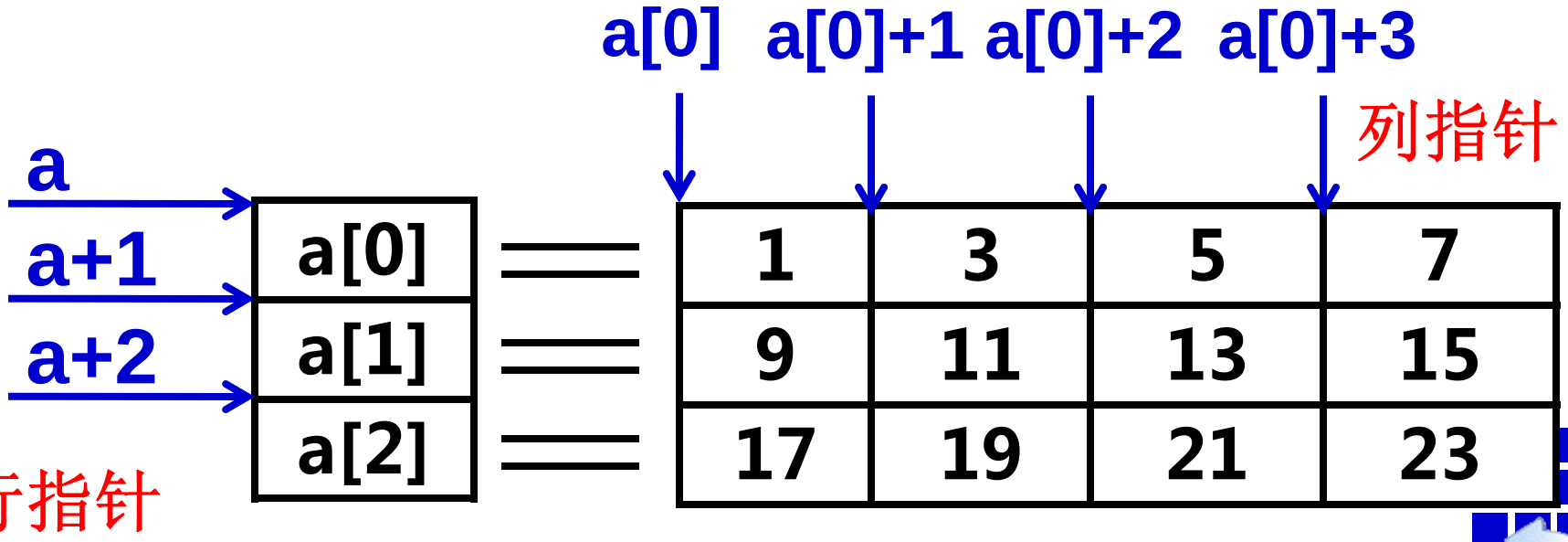


a[1]代表谁的地址？

a[1]+1代表谁的地址？

a[1]+2代表谁的地址？

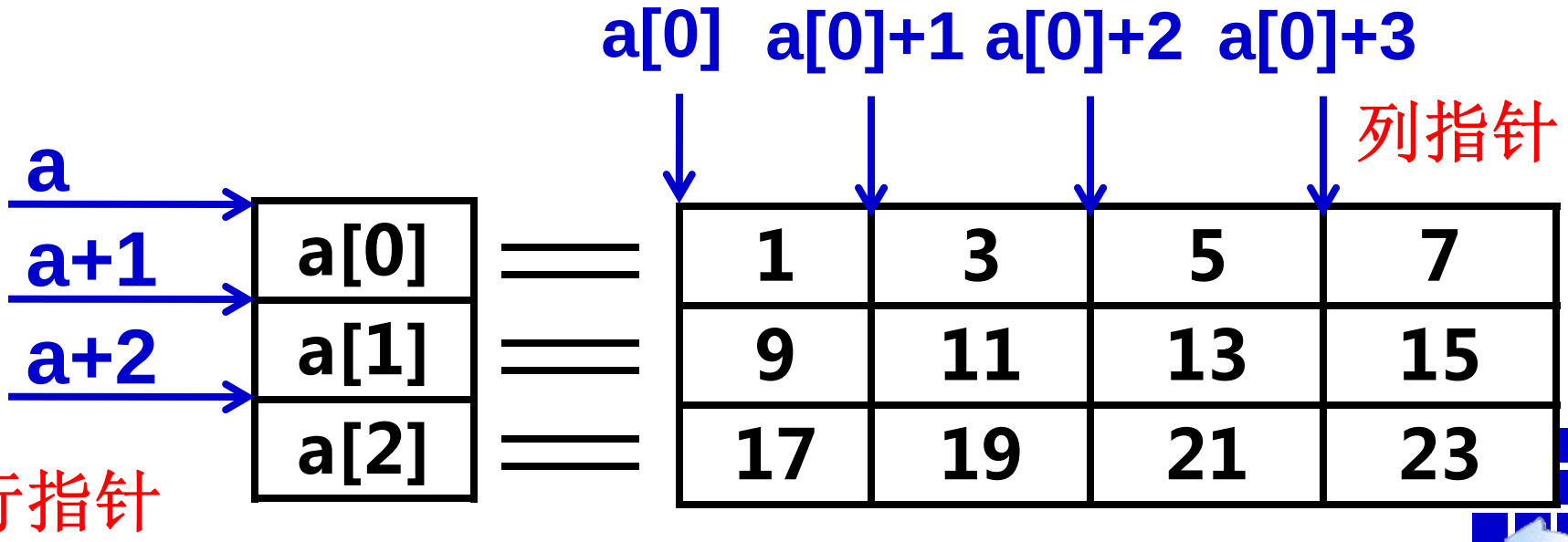
a[1]+3代表谁的地址？



a[i]+j代表谁的地址？ 代表a[i][j]的地址

*(a[i]+j)代表什么？ 代表元素a[i][j]

((a+i)+j)代表什么？ 与*(a[i]+j)等价



明德求新 尚用笃行

例8.11 二维数组的有关数据(地址和值)

```
#include <stdio.h>
int main()
{ int a[3][4]={1,3,5,7,9,11,13,15,
               17,19,21,23};
```

```
printf( "%d,%d\n" ,a,*a);
printf( "%d,%d\n" ,a[0],*(a+0));
printf( "%d,%d\n" ,&a[0],&a[0][0]);
printf( "%d,%d\n" ,a[1],a+1);
printf( "%d,
printf( "%d,
printf( "%d,
printf( "%d,%d\n" ,a[1][0],*(*(a+1)+0));
printf( "%d,%d\n" ,*a[2],*(*(a+2)+0));
return 0;
}
```

```
1245008,1245008
1245008,1245008
1245008,1245008
```

```
printf( "%d,%d\n" ,a,*a);
printf( "%d,%d\n" ,a[0],*(a+0));
printf( "%d,%d\n" ,&a[0],&a[0][0]);
printf( "%d,%d\n" ,a[1],a+1);
printf( "%d,%d\n" ,&a[1][0],*(a+1)+0);
printf( "%d,%d\n" ,a[2],*(a+2));
printf( "%d,%d\n" ,&a[2],a+2);
printf( "%d,
printf( "%d,
return 0;
}
```

```
1245024,1245024
1245024,1245024
1245040,1245040
1245040,1245040
```

```
printf( "%d,%d\n" ,a,*a);
printf( "%d,%d\n" ,a[0],*(a+0));
printf( "%d,%d\n" ,&a[0],&a[0][0]);
printf( "%d,%d\n" ,a[1],a+1);
printf( "%d,%d\n" ,&a[1][0],*(a+1)+0);
printf( "%d,%d\n" ,a[2],*(a+2));
printf( "%d,%d\n" ,&a[2],a+2);
printf( "%d,%d\n" ,a[1][0],*(*(a+1)+0));
printf( "%d,%d\n" ,*a[2],*(*(a+2)+0));
return 0;
}
```

```
9,9
17,17
```

2. 指向多维数组元素的指针变量

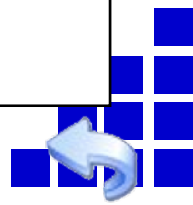
(1) 指向数组元素的指针变量

例8.12 有一个3×4的二维数组，要求用指向元素的指针变量输出二维数组各元素的值。

明德求新 尚用笃行

School of Software

- 解题思路：
  - 二维数组的元素是整型的，它相当于整型变量，可以用**int***型指针变量指向它
  - 二维数组的元素在内存中是按行顺序存放的，即存放完序号为**0**的行中的全部元素后，接着存放序号为**1**的行中的全部元素，依此类推
  - 因此可以用一个指向整型元素的指针变量，依次指向各个元素



```
#include <stdio.h>
int ma
{ int a[3][4]          ,7,9,11,13,15,
                  17,19,21,23};
   int *p;
   for(p=a[0];p<a[0]+12;p++)
   { if((p-a[0])%4==0) printf( "\n" );
      printf( "%4d",*p);
   }
   printf("\n");
   return 0;
}
```

逐个访问各元素时常用此类指针

控制换行

```
   1    3    5    7
   9   11   13   15
  17   19   21   23
```

(2) 指向由m个元素组成的一维数组的指针变量

例8.13 输出二维数组任一行任一列元素的值。

- 解题思路：假设仍然用例8.12程序中的二维数组，例8.12中定义的指针变量是指向变量或数组元素的，现在改用指向一维数组的指针变量。

```
#include <stdio.h>
int main()
{int a[3][4]={1,3,5,7,9,11,13,15,
                                  19,21,23};
  int (*p)[4],i,j;
  p=a;
  printf( "enter row and colum:");
  scanf( "%d,%d" ,&i,&j);
  printf( "a[%d,%d]=%d\n" ,
                        i,j,*(*(p+i)+j));
  return 0;
}
```

行指针

a[i][j]

```
enter row and colum:1,2
a[1,2]=13
```

3. 用指向数组的指针作函数参数

- 一维数组名可以作为函数参数，多维数组名也可作函数参数。
- 用指针变量作形参，以接受实参数组名传递来的地址。
- 可以有两种方法：

  ①用指向变量的指针变量

  ②用指向一维数组的指针变量

例8.14 有一个班，3个学生，各学4门课，计算总平均分数以及第n个学生的成绩。

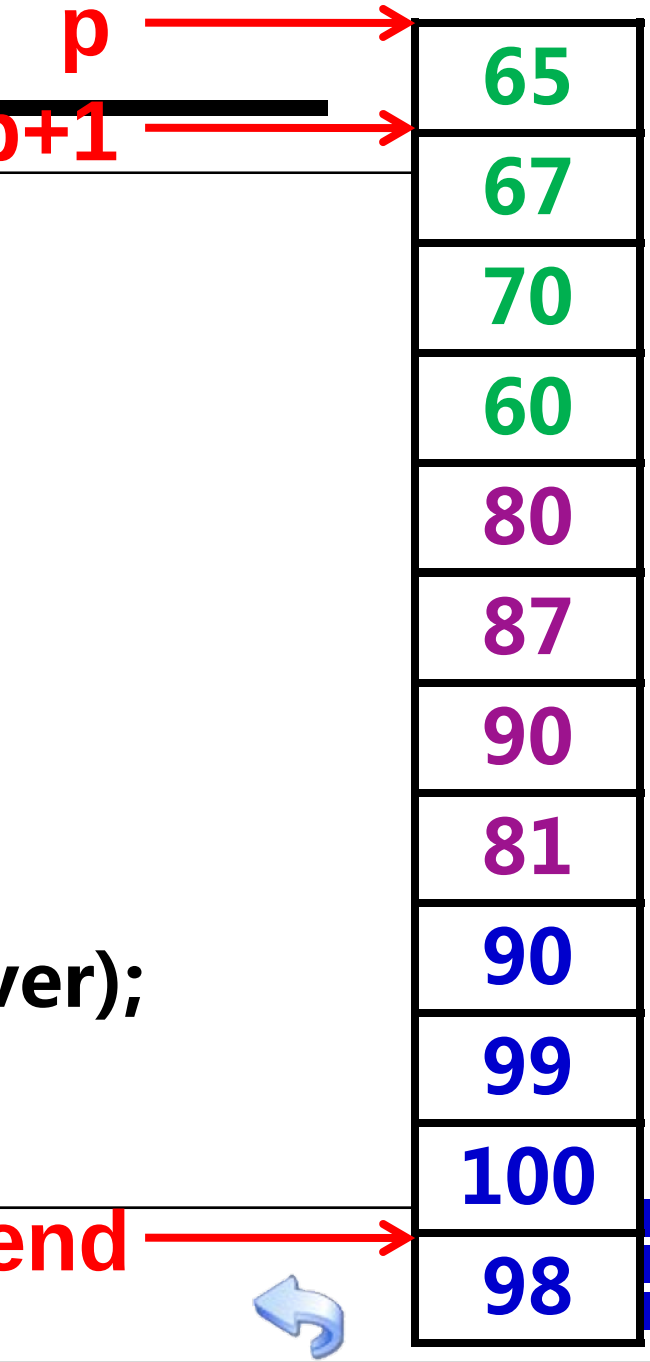
- 解题思路：这个题目是很简单的。本例用指向数组的指针作函数参数。用函数average求总平均成绩，用函数search找出并输出第i个学生的成绩。

```
#include <stdio.h>
int main()
{ void average(float *p,int n);
  void search(float (*p)[4],int n);
  float score[3][4]={{65,67,70,60},
        {80,87,90,81},{90,99,100,98}};
  average(*score,12);
  search(score,2);
  return 0;
}
```

score[0][0]的地址

```
void average(float *p,int n)
{ float *p_end;
   float sum=0,aver;
   p_end=p+n-1;
   for(   ;p<=p_end; p++)
        sum=sum+(*p);
   aver=sum/n;
   printf("average=%5.2f\n",aver);
}
```

```
average=82.25
```

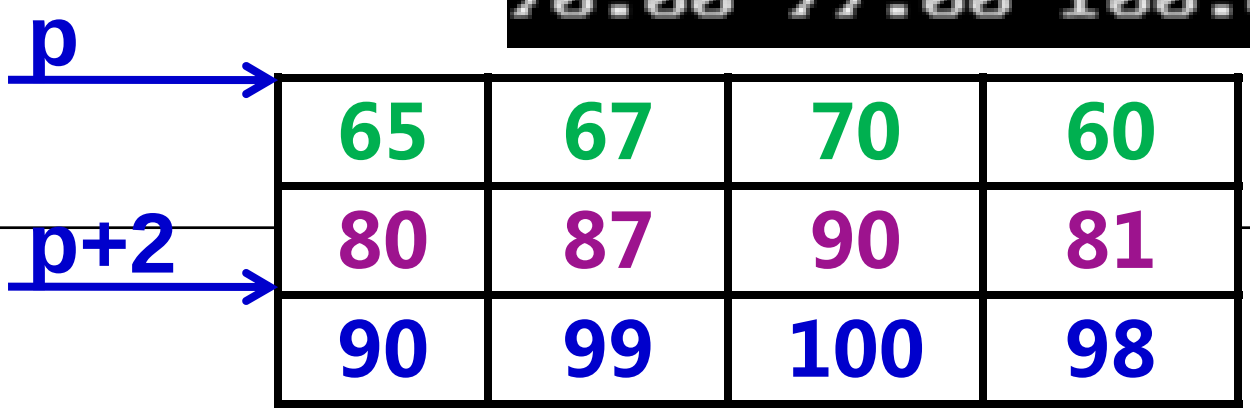


```
#include <stdio.h>
int main()
{ void average(float *p,int n);
   void search(float (*p)[4],int n);
   float score[3][4]={{65,67,70,60},
          {80,87,90,81},{90,99,100,98}};
   average(*score,12);
   search(score,2);
   return 0;
}
```

二维数组首行地址

```
void search(float (*p)[4],int n)
{ int i;
   printf("The score of No.%d are:\n",n);
   for(i=0;i<4;i++)
      printf("%5.2f ",*(*(p+n)+i));
   printf("\n");
}
```

```
The score of No.2 are:
90.00 99.00 100.00 98.00
```



例8.15 在上题基础上，查找有一门以上课程不及格的学生，输出他们的全部课程的成绩。

- 解题思路：在主函数中定义二维数组score，定义search函数实现输出有一门以上课程不及格的学生的全部课程的成绩，形参p的类型是float(*)[4]。在调用search函数时，用score作为实参，把score[0]的地址传给形参p。

```
#include <stdio.h>
int main()
{ void search(float (*p)[4],int n);
   float score[3][4]={{65,57,70,60},
         {58,87,90,81},{90,99,100,98}};
   search(score,3);
   return 0;
}
```

```
void search(float (*p)[4],int n)
{ int i,j,flag;
   for(j=0;j<n;j++)
   { flag=0;
      for(i=0;i<4;i++)
         if(*(*(p+j)+i)<60) flag=1;
      if(flag==1)
      { printf("No.%d fails\n",j+1);
         for(i=0;i<4;i++)
            printf( "%5.1f " ,*(*(p+j)+i));
         printf("\n");
      }
   }
}
```

```
No.1 fails
 65.0  57.0  70.0  60.0
No.2 fails
 58.0  87.0  90.0  81.0
```

p →

65	57	70	60
58	87	90	81
90	99	100	98

```
void search(float (*p)[4],int n)
{ int i,j,flag;
   for(j=0;j<n;j++)
   { flag=0;
      for(i=0;i<4;i
         if(*(*(p+j)+i)
      if(flag==1)
      { printf("No.%d fails\n",j+1);
         for(i=0;i<4;i++)
            printf( "%5.1f " ,*(*(p+j)+i));
         printf("\n");
      }
   }
}
```

格，赋1

若有不及格，则输出

不用flag，而用break语句如何改程序？

8.4 通过指针引用字符串

8.4.1 字符串的引用方式

8.4.2 字符指针作函数参数

8.4.3 使用字符指针变量和字符数组的比较

8.4.1 字符串的引用方式

- 字符串是存放在字符数组中的。引用一个字符串，可以用以下两种方法。

  (1) 用字符数组存放一个字符串，可以通过数组名和格式声明“**%s**”输出该字符串，也可以通过数组名和下标引用字符串中一个字符。

  (2) 用字符指针变量指向一个字符串常量，通过字符指针变量引用字符串常量。

例8.16 定义一个字符数组，在其中存放字符串“I love China!”，输出该字符串和第8个字符。

- 解题思路：定义字符数组string，对它初始化，由于在初始化时字符的个数是确定的，因此可不必指定数组的长度。用数组名string和输出格式%s可以输出整个字符串。用数组名和下标可以引用任一数组元素。

```
#include <stdio.h>
int main()
{ char string[]= "I love China!" ;
   printf( "%s\n" ,string);
   printf( "%c\n" ,string[7]);
   return 0;
}
```

string ↓ ↓ string+7

```
I love China!
C
```

例8.17 通过字符指针变量输出一个字符串。

- 解题思路：可以不定义字符数组，只定义一个字符指针变量，用它指向字符串常量中的字符。通过字符指针变量输出该字符串。

```
#include <stdio.h>
int main()
{ char *string= "I love China!" ;
   printf( "%s\n" ,string);
   return 0;
}
```

string

char *string;
string=" I love China!" ;

```
I love China!
```

```
#include <stdio.h>
int main()
{ char *string= "I love China!" ;
   printf( "%s\n" ,string);
   string=" I am a student." ;
   printf( "%s\n" ,string);
   return 0;
}
```

string

```
#include <stdio.h>
int main()
{ char *string= "I love China!";
   pri(  s\n" ,string);
   string=" I am a student.";
   printf( "%s\n" ,string);
   return 0;
}
```

string

```
I love China!
I am a student.
```

例8.18 将字符串a复制为字符串b，然后输出字符串b。

- 解题思路：定义两个字符数组a和b，用“I am a student.”对a数组初始化。将a数组中的字符逐个复制到b数组中。可以用不同的方法引用并输出字符数组元素，今用地址法算出各元素的值。

```
#include <stdio.h>
int main()
{ char a[ ]= “I am a student.” ,b[20];
   int i;
   for(i=0;*(a+i)!='\0';i++)
        *(b+i)=*(a+i);
   *(b+i)= ‘\0’;
   printf( “string a is:%s\n” ,a);
   printf("string b is:");
   for(i=0;b[i]!='\0';i++)
        printf( “%c” ,b[i]);
   printf("\n");
   return 0;
}
```

printf("string b is:%s\n “,b);

```
string a is:I am a student.
string b is:I am a student.
```

例8.19 用指针变量来处理例8.18问题。

- 解题思路：定义两个指针变量p1和p2，分别指向字符数组a和b。改变指针变量p1和p2的值，使它们顺序指向数组中的各元素，进行对应元素的复制。

```
#include <stdio.h>
int main()
{char a[]="I am a boy.",b[20],*p1,*p2;
  p1=a;  p2=b;
  for(  ; *p1!= '\0' ; p1++,p2++)
      *p2=*p1;
  *p2= '\0' ;
  printf( "string a is:%s\n" ,a);
  printf( "string b is:%s\n" ,b);
  return 0;
}
```

```
string a is:I am a student.
string b is:I am a student.
```

8.4.2 字符指针作函数参数

- 如果想把一个字符串从一个函数“传递”到另一个函数，可以用地址传递的办法，即用字符数组名作参数，也可以用字符指针变量作参数。
- 在被调用的函数中可以改变字符串的内容
- 在主调函数中可以引用改变后的字符串。

8.4.2 字符指针作函数参数

例8.20 用函数调用实现字符串的复制。

- 解题思路：定义一个函数copy_string用来实现字符串复制的功能，在主函数中调用此函数，函数的形参和实参可以分别用字符数组名或字符指针变量。分别编程，以供分析比较。

(1) 用字符数组名作为函数参数

```
#include <stdio.h>
int main()
{void copy_string(char from[],char to[]);
  char a[]="I am a teacher.";
  char b[]="you are a student.";
  printf( “a=%s\nb=%s\n",a,b);
  printf("copy string a to string b:\n");
  copy_string(a,b);
  printf( “a=%s\nb=%s\n",a,b);
  return 0;
}
```

```
void copy_string(char from[], char to[])
{ int i=0;
   while(from[i]!='\0')
   {   to[i]=from[i];
       i++;
    }
    to[i]='\0';
}
```

```
a=I am a teacher.
b=You are a student.
copy string a to string b:
a=I am a teacher.
b=I am a teacher.
```

明德求新 尚用笃行

School of Software

(2)用字符型指针变量作实参

- **copy_string**不变，在**main**函数中定义字符指针变量**from**和**to**，分别指向两个字符数组**a,b**。
- 仅需要修改主函数代码

```
#include <stdio.h>
int main()
{void copy_string(char from[], char to[]);
  char a[]= “I am a teacher.”;
  char b[]= “you are a student.”;
  char *from=a,*to=b;
  printf( “a=%s\nb=%s\n",a,b);
  printf("\ncopy string a to string b:\n");
  copy_string(from,to);
  printf( “a=%s\nb=%s\n",a,b);
  return 0;
}
```

明德求新 尚用笃行

School of Software

(3)用字符指针变量作形参和实参

```
#include <stdio.h>
int main()
{void copy_string(char *from, char *to);
  char *a= “I am a teacher.” ;
  char b[]= “You are a student.” ;
  char *p=b;
  printf( “a=%s\nb=%s\n” ,a,b);
  printf("\ncopy string a to string b:\n");
  copy_string(a,p);
  printf( “a=%s\nb=%s\n” ,a,b);
  return 0;
}
```

```
void copy_string(char *from, char *to)
{  for(  ;*from!='\0'; from++,to++)
      {  *to=*from;   }
   *to='\0';
}
```

函数体有多种简化写法，请见主教材

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

  (1) 字符数组由若干个元素组成，每个元素中放一个字符，而字符指针变量中存放的是地址（字符串第**1**个字符的地址），决不是将字符串放到字符指针变量中。

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(2) 赋值方式。可以对字符指针变量赋值，但不能对数组名赋值。

char *a; a=”I love China!”； 对

char str[14];str[0]=’I’； 对

char str[14]; str=”I love China!”；错

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

（3）初始化的含义

char *a=”I love China！”;与

char *a; a=”I love China！”;等价

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

（3）初始化的含义

char str[14]=”I love China！”;与

char str[14];

str[]=”I love China！”； 不等价

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(4) 存储单元的内容

编译时为字符数组分配若干存储单元，以存放各元素的值，而对字符指针变量，只分配一个存储单元

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(4) 存储单元的内容

```
char *a; scnaf( “%s” ,a);   错
char *a,str[10];
a=str;
scanf ( “%s” ,a);      对
```

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(5) 指针变量的值是可以改变的，而数组名代表一个固定的值(数组首元素的地址)，不能改变。

例8.21 改变

```
#include <st
int main()
{ char *a="I love China!";
   a=a+7;
   printf( "%s\n" ,a);
   return 0;
}
```

不能改为
char a[]= "I love China!" ;

```
China!
```

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(6) 字符数组中各元素的值是可以改变的，但字符指针变量指向的字符串常量中的内容是不可以被取代的。

char a[]=”House”,*b=” House”;

a[2]=’r’; 对

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(6) 字符数组中各元素的值是可以改变的，但字符指针变量指向的字符串常量中的内容是不可以被取代的。

```
char a[]=" House",*b=" House";
b[2]='r';
```

错

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(7) 引用数组元数

对字符数组可以用下标法和地址法引用数组元素（**a[5],*(a+5)**）。如果字符指针变量**p=a**，则也可以用指针变量带下标的形式和地址法引用（**p[5],*(p+5)**）。

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

  char *a="I love China!";

  则**a[5]**的值是第**6**个字符，即字母’ **e**’

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(8) 用指针变量指向一个格式字符串，可以用它代替**printf**函数中的格式字符串。

8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

```
char *format;
format=”a=%d,b=%f\n”;
printf(format,a,b);
```

相当于

```
printf(“a=%d,b=%f\n”,a,b);
```

8.5 指向函数的指针

8.5.1什么是函数指针

- 如果在程序中定义了一个函数，在编译时，编译系统为函数代码分配一段存储空间，这段存储空间的起始地址，称为这个函数的指针。

8.5.1什么是函数指针

- 可以定义一个指向函数的指针变量，用来存放某一函数的起始地址，这就意味着此指针变量指向该函数。例如：

  int (*p)(int,int);

  定义**p**是指向函数的指针变量，它可以指向类型为整型且有两个整型参数的函数。**p**的类型用**int (*)(int,int)**表示

8.5.2 用函数指针变量调用函数

例8.22 用函数求整数a和b中的大者。

- 解题思路：定义一个函数max，实现求两个整数中的大者。在主函数调用max函数，除了可以通过函数名调用外，还可以通过指向函数的指针变量来实现。分别编程并作比较。

（1）通过函数名调用函数

```
#include <stdio.h>
int main()
{ int max(int,int);
   int a,b,c;
   printf("please enter a and b:");
   scanf("%d,%d",&a,&b);
   c=max(a,b);
   printf( “%d,%d,max=%d\n",a,b,c);
   return 0;
}
```


```
int max(int x,int y)
{  int z;
    if(x>y)  z=x;
    else     z=y;
    return(z);
}
```

(2)通过指针变量访问它所指向的函数

```
#include <stdio.h>
int main()
{ int max(int,int);
  int (*p)(int,int);  int a,b,c;
  p=max;
  printf("please en
  scanf("%d,%d",&a,&b);
  c=(*p)(a,b);
  printf( "%d,%d,max=%d\n",a,b,c);
  return 0;
}
```

只能指向函数返回值为整型且有两个整型参数的函数

必须先指向，若写成 p=max(a,b); 错

8.5.3 怎样定义和使用指向函数的指针变量

- 定义指向函数的指针变量的一般形式为

数据类型 (*指针变量名)(函数参数表列);

如 int (*p)(int,int);

p=max; 对

p=max(a,b); 错

p+n,p++,p--等运算无意义

例8.23 输入两个整数，然后让用户选择1或2，选1时调用max函数，输出二者中的大数，选2时调用min函数，输出二者中的小数。

- 解题思路：定义两个函数max和min，分别用来求大数和小数。在主函数中根据用户输入的数字1或2，使指针变量指向max函数或min函数。

```
#include <stdio.h>
int main()
{int max(int,int);   int min(int x,int y);
  int (*p)(int,int);   int a,b,c,n;
  scanf("%d,%d",&a,&b);
  scanf( “%d” ,&n);
  if (n==1) p=max;
  else if (n==2) p=min;
  c=(*p)(a,b);
  printf("a=%d,b=%d\n",a,b);
  if (n==1) printf("max=%d\n",c);
  else  printf("min=%d\n",c);
  return 0;
}
```

只看此行看不出调用哪函数


```
int max(int x,int y)
{ int z;
   if(x>y)  z=x;
   else     z=y;
   return(z);
}
int min(int x,int y)
{ int z;
   if(x<y)  z=x;
   else     z=y;
   return(z);
}
```

8.5.4 用指向函数的指针作函数参数

- 指向函数的指针变量的一个重要用途是把函数的地址作为参数传递到其他函数
- 指向函数的指针可以作为函数参数，把函数的入口地址传递给形参，这样就能够在被调用的函数中使用实参函数

8.5.4 用指向函数的指针作函数参数

```
……
int main()
{  ……   fun(f1,f2)  ……   }

void  fun(int (*x1)(int),int (*x2)(int,int))
{  int a,b,i=3,j=5;
    a=(*x1)(i);        相当于a=f1(i);
    b=(*x2)(i,j);      相当于b=f2(i,j);
}
```

例8.24 有两个整数a和b，由用户输入1,2或3。如输入1，程序就给出a和b中大者，输入2，就给出a和b中小者，输入3，则求a与b之和。

- 解题思路：与例8.23相似，但现在用一个函数fun来实现以上功能。


```
#include <stdio.h>
int main()
{void fun(int x,int y, int (*p)(int,int));
  int max(int,int);  int min(int,int);
  int add(int,int);   int a=34,b=-21,n;
  printf("please choose 1,2 or 3:");
  scanf( “%d” ,&n);
  if (n==1)    fun(a,b,max);
  else if (n==2) fun(a,b,min);
  else if (n==3) fun(a,b,add);
  return 0;
 }
```

```
int fun(int x,int y,int (*p)(int,int))
{ int resout;
   resout=(*p)(x,y);
   printf( "%d\n" ,resout);
}
int max(int x,int y)
{ int z;
   if(x>y)  z=x;
   else   z=y;
   printf("max=" );
   return(z);
  }
```

输入的选项为1时

相当于max(x,y)

```
int fun(int x,int y,int (*p)(int,int))
{ int resout;
   resout=(*p)(x,y);
   printf( "%d\n" ,resout);
}
int max(int x,int y)
{ int z;
   if(x>y)  z=x;
   else   z=y;
   printf("max=" );
   return(z);
  }
```

输入的选项为2时

相当于min(x,y)

```
int fun(int x,int y,int (*p)(int,int))
{ int result;
   result=(*p)(x,y);
   printf( "%d\n" ,result);
}
int max(int x,int y)
{ int z;
   if(x>y)  z=x;
   else   z=y;
   printf("max=" );
   return(z);
  }
```

输入的选项为3时

相当于add(x,y)

```
int min(int x,int y)
{ int z;
   if(x<y) z=x;
   else z=y;
   printf("min=");
   return(z);
}
int add(int x,int y)
{ int z;
   z=x+y;
   printf("sum=");
   return(z);
}
```


8.6 返回指针值的函数

- 一个函数可以返回一个整型值、字符值、实型值等，也可以返回指针型的数据，即地址。其概念与以前类似，只是返回的值的类型是指针类型而已
- 定义返回指针值的函数的一般形式为

  类型名 *函数名(参数表列);

例8.25有a个学生，每个学生有b门课程的成绩。要求在用户输入学生序号以后，能输出该学生的全部成绩。用指针函数实现。

解题思路：

- 定义二维数组score存放成绩
- 定义输出某学生全部成绩的函数search，它是返回指针的函数，形参是行指针和整型
- 主函数将score和要找的学号k传递给形参
- 函数的返回值是&score[k][0](k号学生的序号为0的课程地址)
- 在主函数中输出该生的全部成绩

```
#include <stdio.h>
int main()
{float score[ ][4]={{60,70,80,90},
        {56,89,67,88},{34,78,90,66}};
  float  *search(float (*pointer)[4],int n);
  float  *p;  int i,k;
  scanf( “%d” ,&k);
  printf("The scores of No.%d are:\n",k);
  p=search(score,k);
  for(i=0;i<4;i++)
     printf( “%5.2f\t” ,*(p+i));
  printf("\n");
  return 0;
}
```

返回k号学生课程首地址


```
float *search(float (*pointer)[4],int n)
{ float *pt;
   pt=*(pointer+n);
   return(pt);
}
```


例8.26对例8.25中的学生，找出其中有不及格的课程的学生及其学生号。

- 解题思路：
  - 在例8.25程序基础上修改。
  - main函数不是只调用一次search函数，而是先后调用3次search函数，其中检查3个学生有无不及格的课程，如果有，就返回该学生的0号课程的地址&score[i][0]，否则返回NULL
  - 在main函数中检查返回值，输出有不及格学生4门课的成绩

```
    ……
float  *search(float (*pointer)[4]);
float  *p;  int i,j;
for(i=0;i<3;i++)
{  p=search(score+i);
   if(p==*(score+i))        相当于if(p!=NULL)
   { printf("No.%d score:",i);
       for(j=0;j<4;j++)
         printf( “%5.2f  ” ,*(p+j));
       printf("\n");
   }
 }
   ……
```

```
 float *search(float (*pointer)[4])
{ int i=0;
   float *pt;
   pt=NULL;
   for(   ;i<4;i++)
      if(*(*pointer+i)<60)
         pt=*pointer;
   return(pt);
}
```

8.7 指针数组和多重指针

8.7.1 什么是指针数组

8.7.2 指向指针数据的指针

8.7.3 指针数组作main函数的形参

8.7.1 什么是指针数组

- 一个数组，若其元素均为指针类型数据，称为指针数组，也就是说，指针数组中的每一个元素都存放一个地址，相当于一个指针变量。
- 定义一维指针数组的一般形式为

  类型名*数组名[数组长度];

  int *p[4];

8.7.1 什么是指针数组

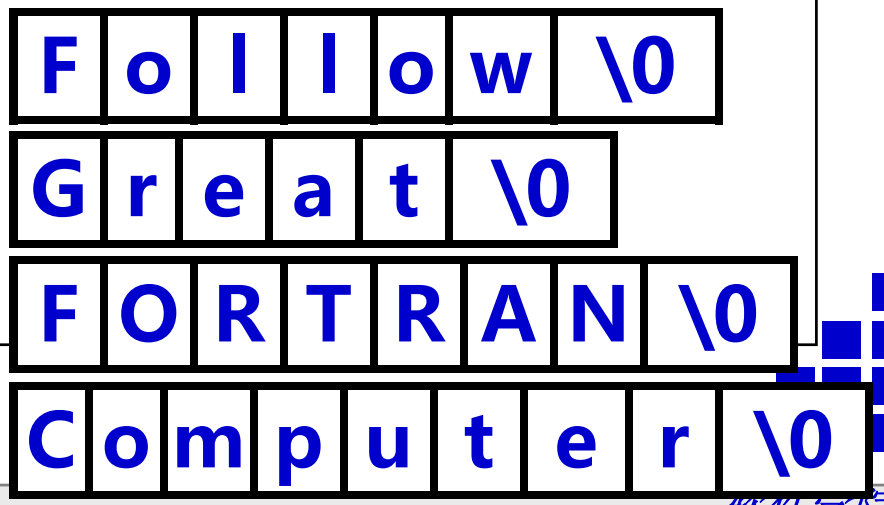
- 指针数组比较适合用来指向若干个字符串，使字符串处理更加方便灵活
- 可以分别定义一些字符串，然后用指针数组中的元素分别指向各字符串
- 由于各字符串长度一般是不相等的，所以比用二维数组节省内存单元

8.7.1 什么是指针数组

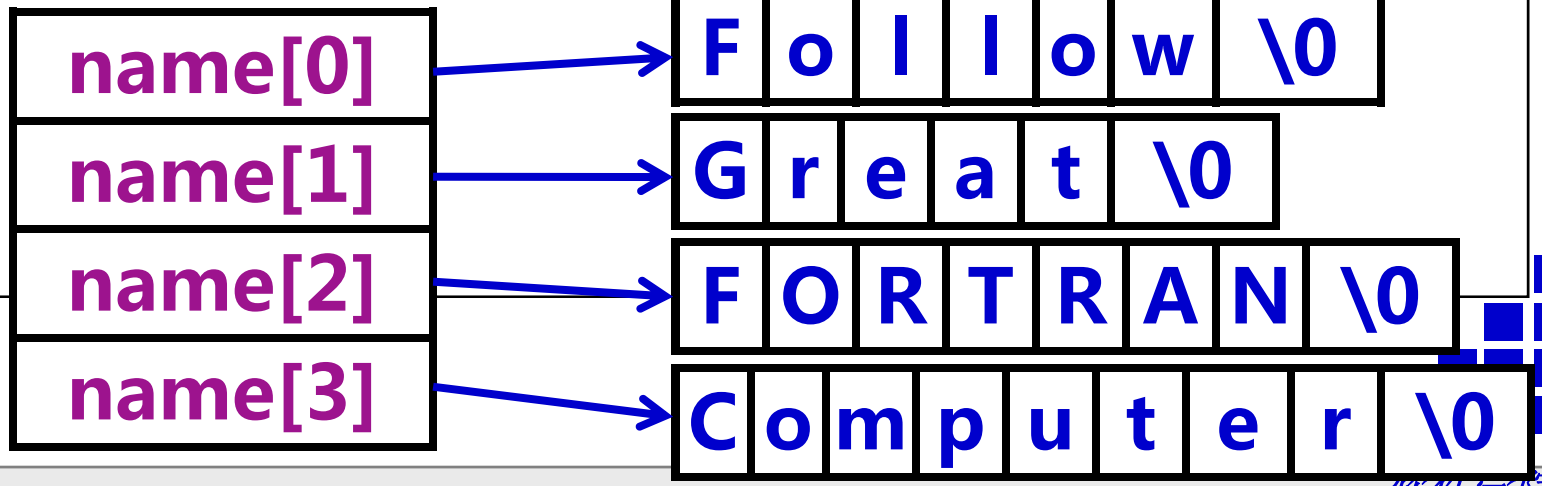
例8.27 将若干字符串按字母顺序（由小到大）输出。

- 解题思路：定义一个指针数组，用各字符串对它进行初始化，然后用选择法排序，但不是移动字符串，而是改变指针数组的各元素的指向。

```
#include <stdio.h>
#include <string.h>
int main()
{void sort(char *name[ ],int n);
  void print(char *name[ ],int n);
  char *name[ ]={ "Follow" , "Great" ,
                  "FORTRAN" , "Computer" };
  int n=4;
  sort(name,n);
  print(name,n);
  return 0;
}
```



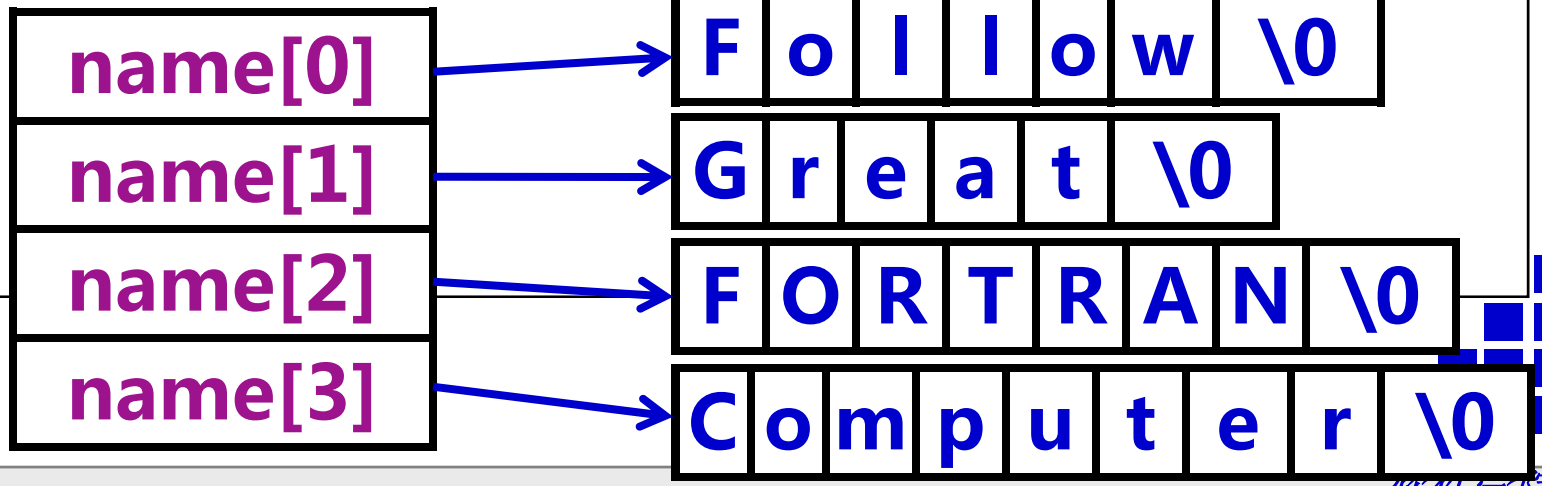
```
void sort(char *name[ ],int n)
{char *temp;  int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
    }
  }
}
```



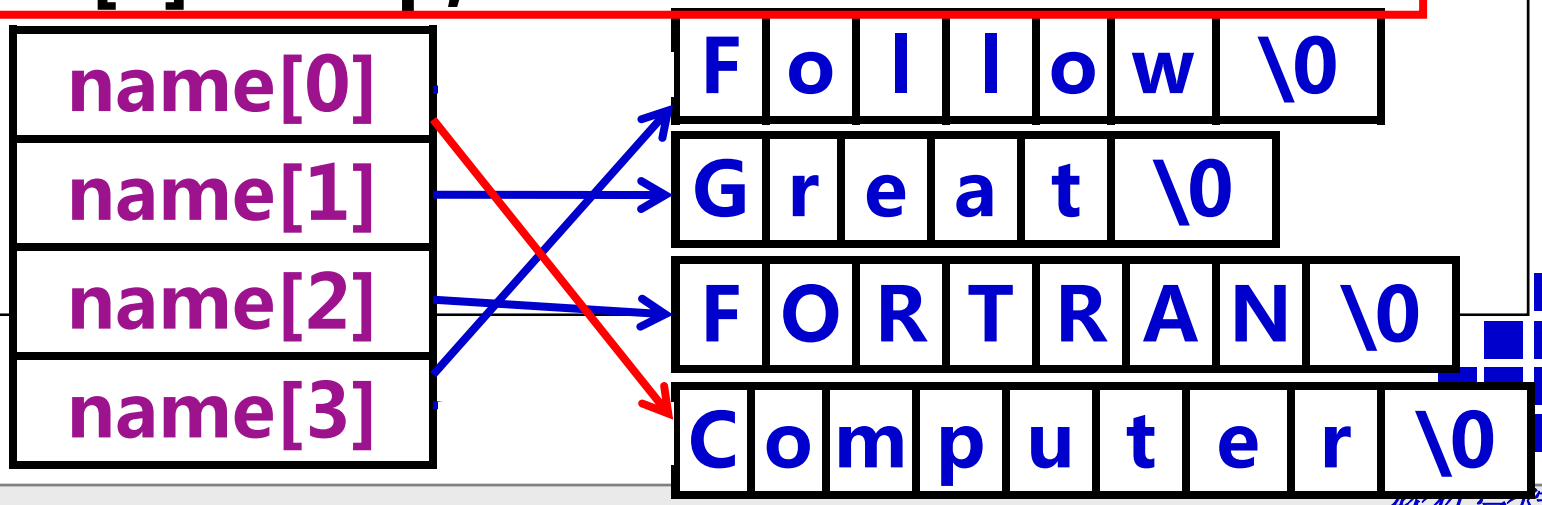
```
void sort(char *name[ ],int n)
{char *temp;  int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
    }
  }
}
```

i=0时

执行后k变为3



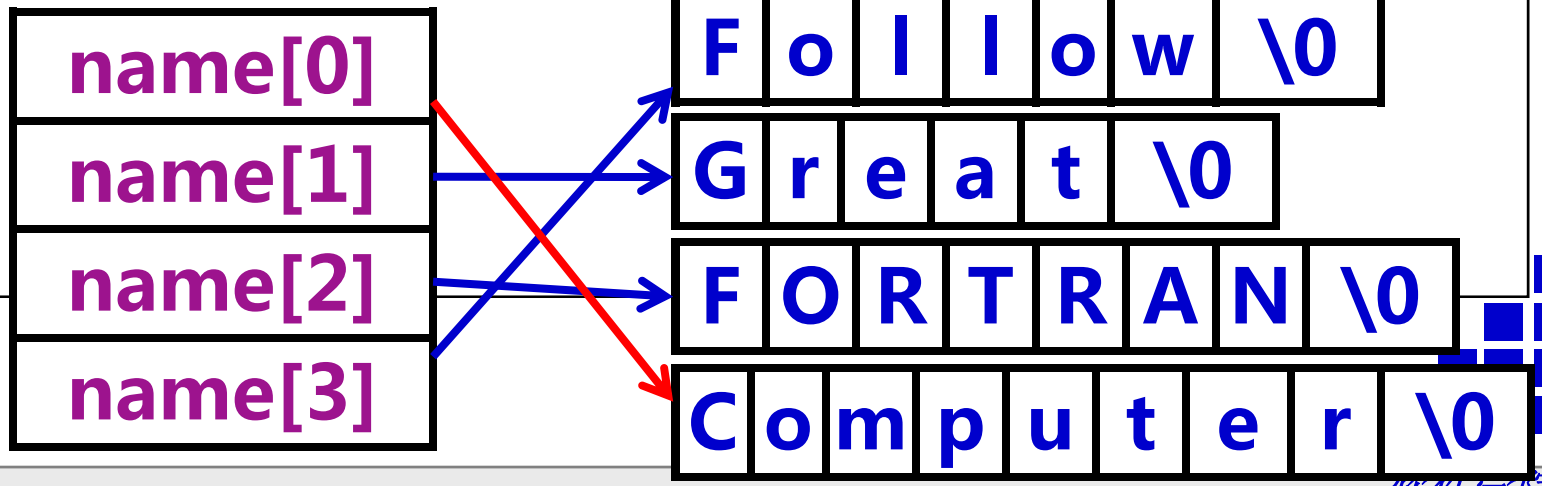
```
void sort(char *name[ ],int n)
{char *temp;  int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
     for (j=i+1;j<n;j++)
       if(strcmp(name[k],name[j])>0) k=j;
     if (k!=i)
     { temp=name[i]; name[i]=name[k];
       name[k]=temp;
     }
  }
}
```



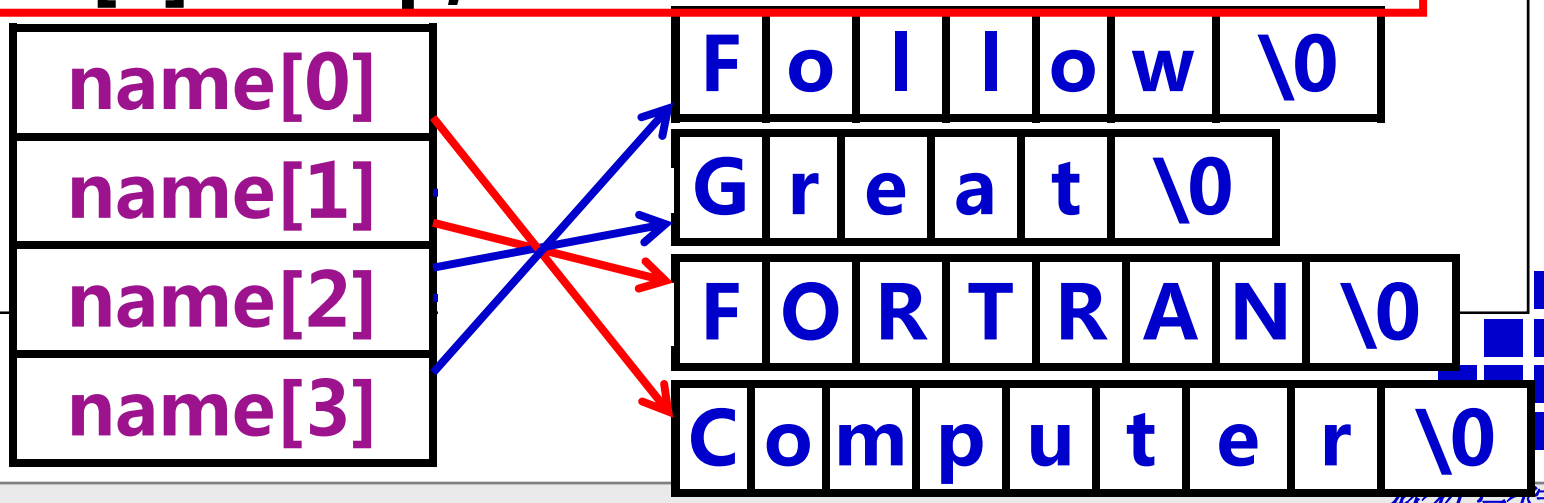
```
void sort(char *name[ ],int n)
{char *temp;  int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
    }
  }
}
```

i=1时

执行后k变为2



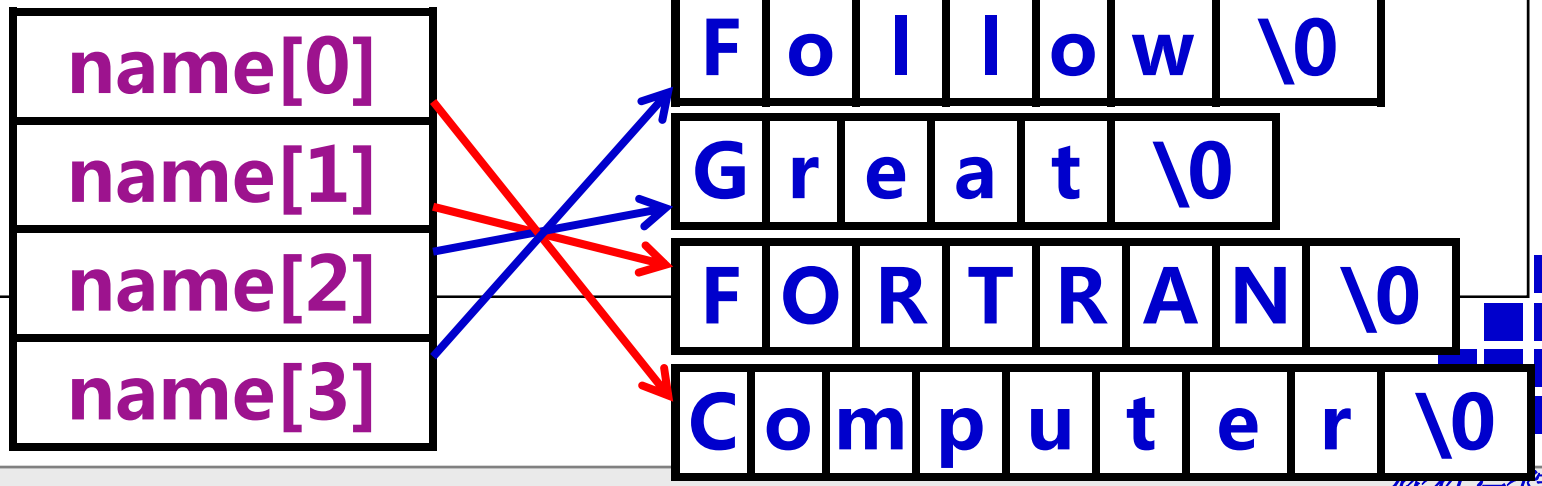
```
void sort(char *name[ ],int n)
{char *temp;  int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
    }
  }
}
```



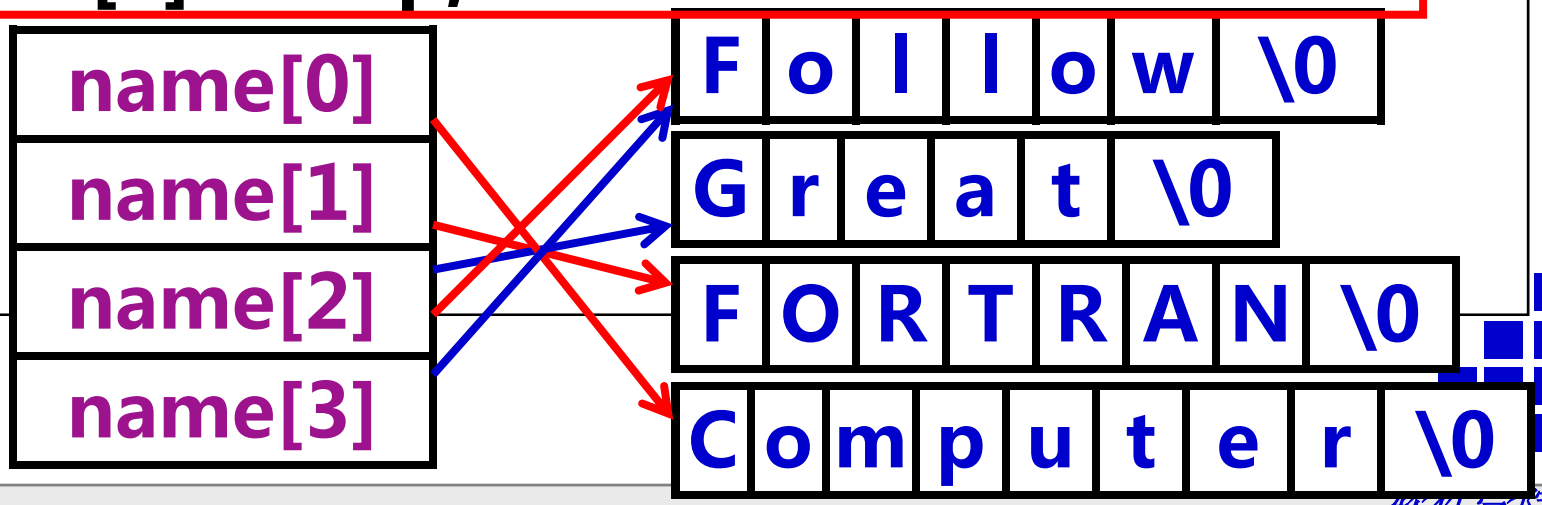
```
void sort(char *name[ ],int n)
{char *temp;  int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
    }
  }
}
```

i=2时

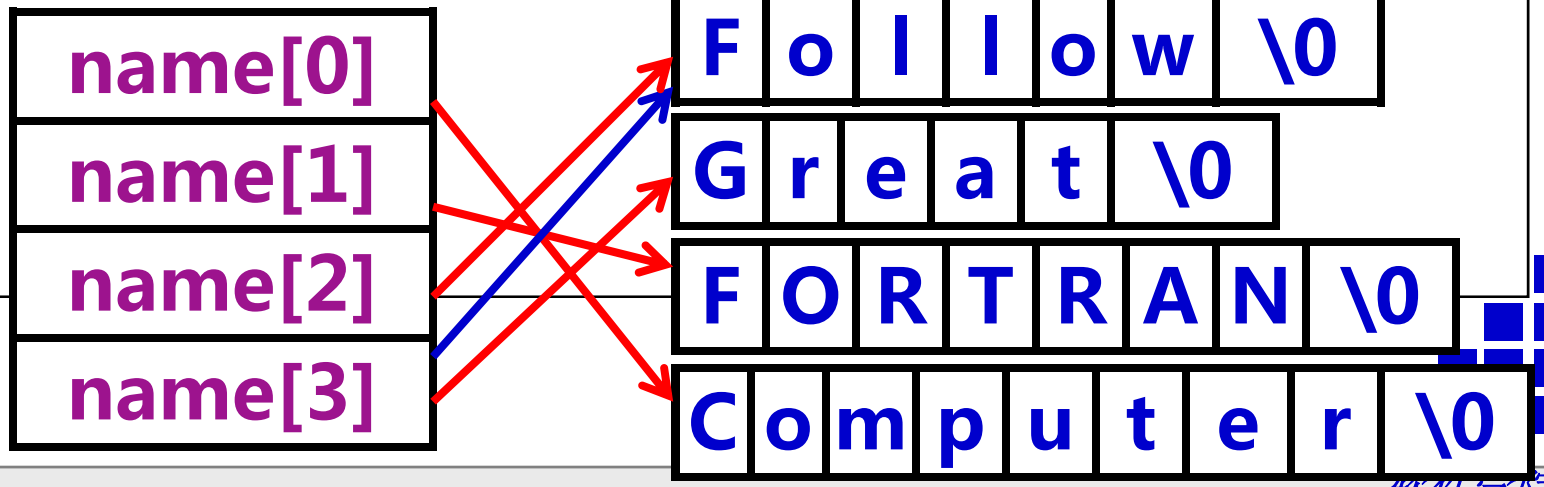
执行后k变为3



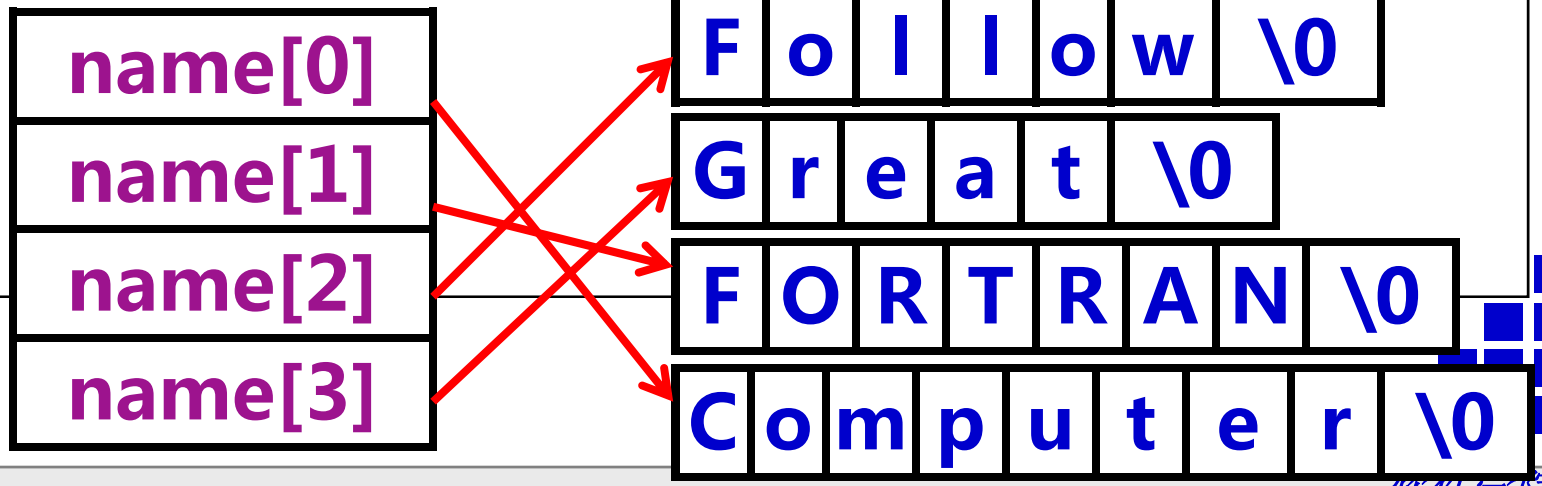
```
void sort(char *name[ ],int n)
{char *temp;  int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
     for (j=i+1;j<n;j++)
       if(strcmp(name[k],name[j])>0) k=j;
     if (k!=i)
     { temp=name[i]; name[i]=name[k];
       name[k]=temp;
     }
  }
}
```



```
void sort(char *name[ ],int n)
{char *temp;  int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
     for (j=i+1;j<n;j++)
       if(strcmp(name[k],name[j])>0) k=j;
     if (k!=i)
     { temp=name[i]; name[i]=name[k];
        name[k]=temp;
     }
  }
}
```

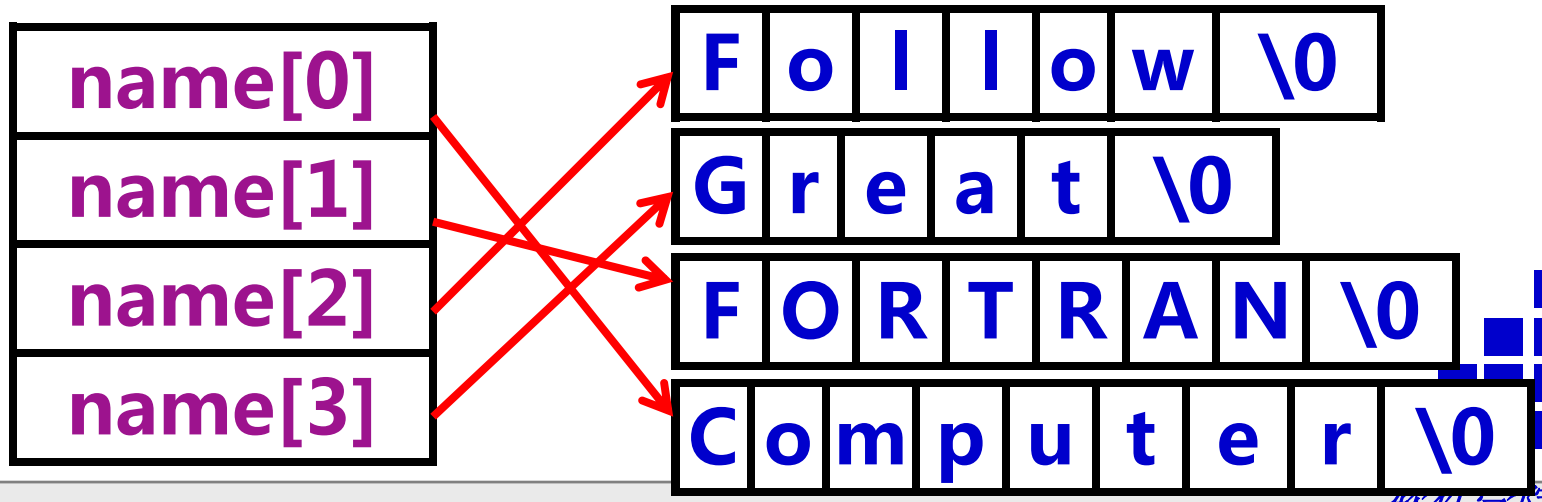


```
void sort(char *name[ ],int n)
{char *temp;  int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
     for (j=i+1;j<n;j++)
       if(strcmp(name[k],name[j])>0) k=j;
     if (k!=i)
     { temp=name[i]; name[i]=name[k];
        name[k]=temp;
     }
  }
}
```



```
void print(char *name[ ],int n)
{  int i;
    for(i=0;i<n;i++)
        printf( "%s\n" ,name[i]);
}
```

```
Computer
FORTRAN
Follow
Great
```

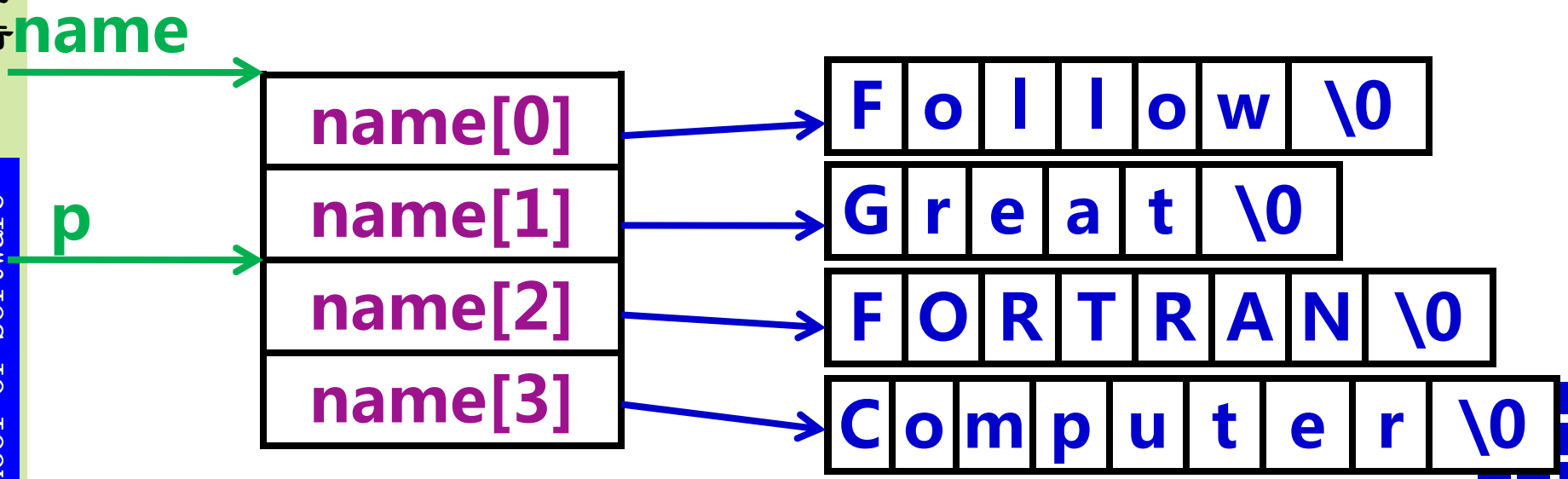


```
void print(char *name[ ],int n)
{  int i;
    for(i=0;i<n;i++)
        printf( “%s\n” ,name[i]);
}
```

```
void print(char *name[ ],int n)
{ int i=0;  char *p;
   p=name[0];
   while(i<n)
   { p=*(name+i++);
      printf("%s\n",p);
   }
}
```

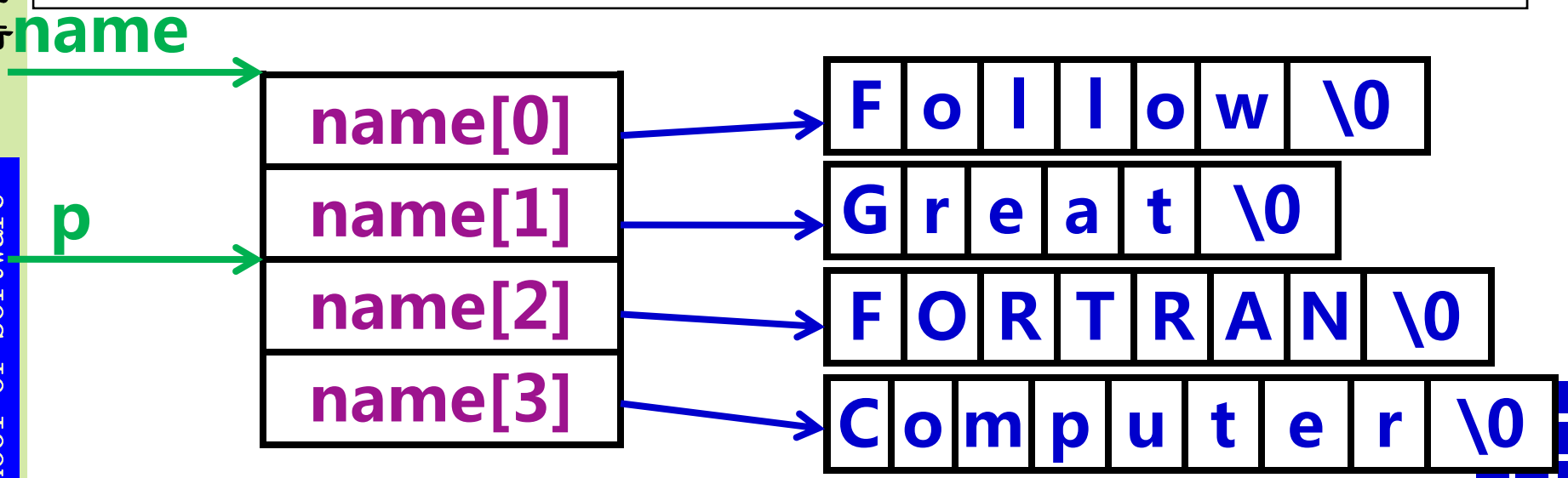
8.7.2 指向指针数据的指针

- 在了解了指针数组的基础上，需要了解指向指针数据的指针变量，简称为指向指针的指针。



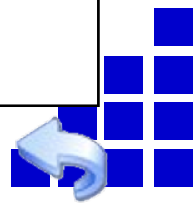
例8.28 使用指向指针数据的指针变量。

```
char *name[]={ “Follow”,, “Great”,
              “FORTRAN”, “Computer” };
char **p;   int i;
for(i=0;i<5;i++)
{  p=name+i;   printf("%s\n",*p);  }
```

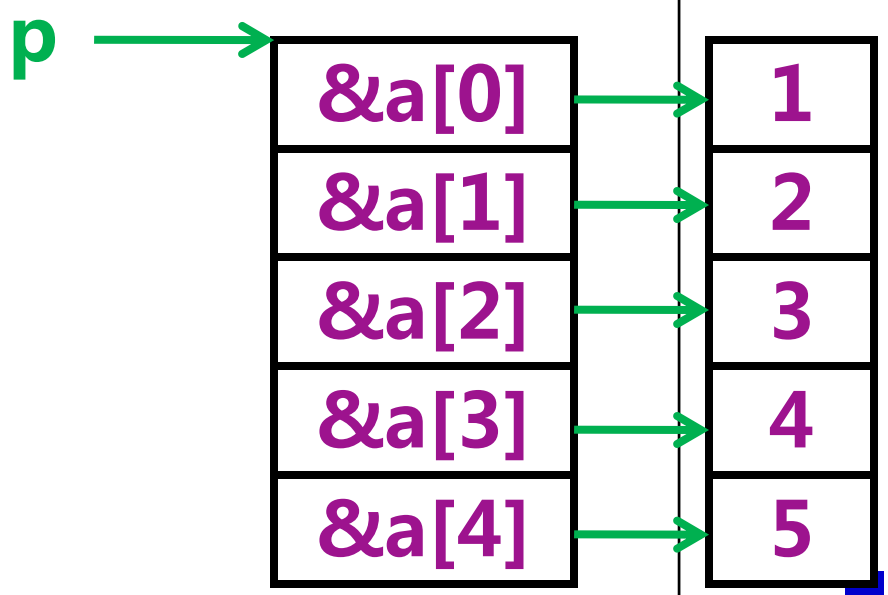


明德求新 尚用笃行

例8.29 有一个指针数组，其元素分别指向一个整型数组的元素，用指向指针数据的指针变量，输出整型数组各元素的值。



```
#include <stdio.h>
int main()
{int a[5]={1,3,5,7,9};
  int *num[5]={&a[0],&a[1],&a[2],
                   &a[3],&a[4]};
  int **p,i;
  p=num;
  for(i=0;i<5;i++)
  {  printf("%d ",**p);
      p++;
  }
  printf("\n");  return 0;
}
```



8.7.3 指针数组作main函数的形参

- 指针数组的一个重要应用是作为main函数的形参。在以往的程序中，main函数的第一行一般写成以下形式：

  int main() 或 int main(void)

- 表示main函数没有参数，调用main函数时不必给出实参。
- 这是一般程序常采用的形式。

8.7.3 指针数组作main函数的形参

- 实际上，在某些情况下，**main**函数可以有参数，例如：

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

  其中，**argc**和**argv**就是**main**函数的形参，它们是程序的“命令行参数”。

- **argv**是***char**指针数组，数组中每一个元素(其值为指针)指向命令行中的一个字符串。

8.7.3 指针数组作main函数的形参

- 通常main函数和其他函数组成一个文件模块，有一个文件名。
- 对这个文件进行编译和连接，得到可执行文件（后缀为.exe）。用户执行这个可执行文件，操作系统就调用main函数，然后由main函数调用其他函数，从而完成程序的功能。

8.7.3 指针数组作main函数的形参

- main函数的形参是从哪里传递给它们的呢？
- 显然形参的值不可能在程序中得到。
- main函数是操作系统调用的，实参只能由操作系统给出。

```
#include <stdio.h>
int main(int argc,char *argv[])
{ while(argc>1)
   { ++argv;
     printf( "%s\n" , *argv);
     --argc;
   }
   return 0;
}
```

```
China
Beijing
```

在VC++环境下编译、连接后，“工程”—“设置”—“调试”—“程序变量”中输入“China Beijing”，再运行就可得到结果

8.8 动态内存分配与指向它的指针变量

8.8.1 什么是内存的动态分配

- 非静态的局部变量是分配在内存中的动态存储区的，这个存储区是一个称为栈的区域
- C语言还允许建立内存动态分配区域，以存放一些临时用的数据，这些数据需要时随时开辟，不需要时随时释放。这些数据是临时存放在一个特别的自由存储区，称为堆区

8.8.2 怎样建立内存的动态分配

- 对内存的动态分配是通过系统提供的库函数来实现的，主要有**malloc，calloc，free，realloc**这4个函数。

8.8.2 怎样建立内存的动态分配

1．malloc函数

- 其函数原型为

void *malloc(unsigned int size);

  - 其作用是在内存的动态存储区中分配一个长度为**size**的连续空间
  - 函数的值是所分配区域的第一个字节的地址，或者说，此函数是一个指针型函数，返回的指针指向该分配域的开头位置

8.8.2 怎样建立内存的动态分配

malloc(100);

- 开辟**100**字节的临时分配域，函数值为其第**1**个字节的地址

- 注意指针的基类型为**void**，即不指向任何类型的数据，只提供一个地址
- 如果此函数未能成功地执行（例如内存空间不足），则返回空指针**(NULL)**

8.8.2 怎样建立内存的动态分配

2．calloc函数

- 其函数原型为

  void *calloc(unsigned n,unsigned size);

- 其作用是在内存的动态存储区中分配**n**个长度为**size**的连续空间，这个空间一般比较大，足以保存一个数组。

8.8.2 怎样建立内存的动态分配

- 用calloc函数可以为一维数组开辟动态存储空间，n为数组元素个数，每个元素长度为size。这就是动态数组。函数返回指向所分配域的起始位置的指针；如果分配不成功，返回NULL。如：

  p=calloc(50,4);

  开辟50×4个字节的临时分配域，把起始地址赋给指针变量p

8.8.2 怎样建立内存的动态分配

3．free函数

- 其函数原型为

  void free(void *p);

- 其作用是释放指针变量 p 所指向的动态空间，使这部分空间能重新被其他变量使用。**p**应是最近一次调用**calloc**或**malloc**函数时得到的函数返回值。

8.8.2 怎样建立内存的动态分配

free(p);

- 释放指针变量 p 所指向的已分配的动态空间
- **free**函数无返回值

8.8.2 怎样建立内存的动态分配

4. realloc函数

- 其函数原型为

void *realloc(void *p,unsigned int size);

- 如果已经通过**malloc**函数或**calloc**函数获得了动态空间，想改变其大小，可以用**recalloc**函数重新分配。

8.8.2 怎样建立内存的动态分配

- 用realloc函数将p所指向的动态空间的大小改变为size。p的值不变。如果重分配不成功，返回NULL。如

  realloc(p,50);

将p所指向的已分配的动态空间改为50字节

8.8.2 怎样建立内存的动态分配

- 以上4个函数的声明在stdlib.h头文件中，在用到这些函数时应当用“#include <stdlib.h>”指令把stdlib.h头文件包含到程序文件中。

8.8.3 void指针类型

例8.30 建立动态数组，输入5个学生的成绩，另外用一个函放数检查其中有无低于60分的，输出不合格的成绩。

8.8.3 void指针类型

- 解题思路：用malloc函数开辟一个动态自由区域，用来存5个学生的成绩，会得到这个动态域第一个字节的地址，它的基类型是void型。用一个基类型为int的指针变量p来指向动态数组的各元素，并输出它们的值。但必须先把malloc函数返回的void指针转换为整型指针，然后赋给p1


```
#include <stdio.h>
#include <stdlib.h>
int main()
{ void check(int *);
   int *p1,i;
   p1=(int *)malloc(5*sizeof(int));
   for(i=0;i<5;i++)
      scanf("%d",p1+i);
   check(p1);
   return 0;
}
```

```
void check(int *p)
{ int i;
   printf("They are fail:");
   for(i=0;i<5;i++)
      if (p[i]<60)
         printf("%d ",p[i]);
   printf("\n");
}
```

```
67 98 59 78 57
They are fail:59 57
```

8.9有关指针的小结

1.首先要准确地弄清楚指针的含义。指针就是地址，凡是出现“指针”的地方，都可以用“地址”代替，例如，变量的指针就是变量的地址，指针变量就是地址变量

- 要区别指针和指针变量。指针就是地址本身，而指针变量是用来存放地址的变量。

8.9有关指针的小结

2. 什么叫“指向”？地址就意味着指向，因为通过地址能找到具有该地址的对象。对于指针变量来说，把谁的地址存放在指针变量中，就说此指针变量指向谁。但应注意：只有与指针变量的基类型相同的数据的地址才能存放在相应的指针变量中。

8.9有关指针的小结

void *指针是一种特殊的指针，不指向任何类型的数据，如果需要用此地址指向某类型的数据，应先对地址进行类型转换。可以在程序中进行显式的类型转换，也可以由编译系统自动进行隐式转换。无论用哪种转换，读者必须了解要进行类型转换

8.9有关指针的小结

3. 要深入掌握在对数组的操作中怎样正确地使用指针，搞清楚指针的指向。一维数组名代表数组首元素的地址

8.9有关指针的小结

int *p,a[10];
p=a;

- p是指向int类型的指针变量，p只能指向数组中的元素，而不是指向整个数组。在进行赋值时一定要先确定赋值号两侧的类型是否相同，是否允许赋值。
- 对“p=a;”，准确地说应该是：p指向a数组的首元素

8.9有关指针的小结

4.有关指针变量的定义形式的归纳比较，见主教材中表8.4。

8.9有关指针的小结

5.指针运算

(1)指针变量加（减）一个整数

例如：p++,p--,p+i,p-i,p+=i, p -=i等均是指针变量加（减）一个整数。

- 将该指针变量的原值(是一个地址)和它指向的变量所占用的存储单元的字节数相加（减）。

8.9有关指针的小结

5.指针运算

(2)指针变量赋值

- 将一个变量地址赋给一个指针变量
- 不应把一个整数赋给指针变量

8.9有关指针的小结

5.指针运算

(3) 两个指针变量可以相减

- 如果两个指针变量都指向同一个数组中的元素，则两个指针变量值之差是两个指针之间的元素个数

8.9有关指针的小结

5.指针运算

(4) 两个指针变量比较

- 若两个指针指向同一个数组的元素，则可以进行比较
- 指向前面的元素的指针变量“小于”指向后面元素的指针变量
- 如果**p1**和**p2**不指向同一数组则比较无意义

8.9有关指针的小结

5.指针运算

(5) 指针变量可以有空值，即该指针变量不指向任何变量，可以这样表示：

```
p=NULL;
```

第9章 用户自己建立数据类型

9.1 定义和使用结构体变量

明德求新 尚用笃行

9.1.1 自己建立结构体类型

9.1.2 定义结构体类型变量

9.1.3 结构体变量的初始化和引用

9.1.1 自己建立结构体类型

- 用户自己建立由不同类型数据组成的组合型的数据结构，它称为结构体
- 例如，一个学生的学号、姓名、性别、年龄、成绩、家庭地址等项，是属于同一个学生的，因此组成一个组合数据，如**student_1**的变量，反映它们之间的内在联系

9.1.1 自己建立结构体类型

```
struct Student
{  int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};
```

- 由程序设计者指定了一个结构体类型 **struct Student**
- 它包括 **num,name,sex,age, score,addr**等不同类型的成员

9.1.1 自己建立结构体类型

- 声明一个结构体类型的一般形式为：

  struct 结构体名

  { 成员表列 **};**

类型名 成员名;

9.1.1 自己建立结构体类型

- 说明：

(1)结构体类型并非只有一种，而是可以设计出许多种结构体类型，例如

struct Teacher

struct Worker

struct Date等结构体类型

- 各自包含不同的成员

9.1.1 自己建立结构体类型

明德求新 尚用笃行

School of Software

- 说明：

(2) 成员可以属于另一个结构体类型。

```
struct Date
{   int month;  int day;  int year; };
 struct Stu
 {   int num;char name[20];
     char sex;int age;
     struct Date birthday;
     char addr[30];
 };
```

9.1.1 自己建立结构体类型

- 说明：

(2) 成员可以属于另一个结构体类型。

<table>
<tr><td rowspan="2">num</td><td rowspan="2">name</td><td rowspan="2">sex</td><td rowspan="2">age</td><td colspan="3">birthday</td><td rowspan="2">addr</td></tr>
<tr><td>month</td><td>day</td><td>year</td></tr>
</table>

9.1.2 定义结构体类型变量

- 前面只是建立了一个结构体类型，它相当于一个模型，并没有定义变量，其中并无具体数据，系统对之也不分配存储单元。
- 相当于设计好了图纸，但并未建成具体的房屋。为了能在程序中使用结构体类型的数据，应当定义结构体类型的变量，并在其中存放具体的数据。

9.1.2 定义结构体类型变量

1. 先声明结构体类型，再定义该类型变量

- 声明结构体类型**struct Student**，可以用它来定义变量

struct Student student1,student2;

结构体类型名

结构体变量名

9.1.2 定义结构体类型变量

1. 先声明结构体类型，再定义该类型变量

- 声明结构体类型**struct Student**，可以用它来定义变量

struct Student student1,student2;

student1

10001	Zhang Xin	M	19	90.5	Shanghai

student2

10002	Wang Li	F	20	98	Beijing

9.1.2 定义结构体类型变量

2.在声明类型的同时定义变量

```
struct Student
{  int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} student1,student2;
```

9.1.2 定义结构体类型变量

3. 不指定类型名而直接定义结构体类型变量

- 其一般形式为:

struct

{ 成员表列 **}**变量名表列**;**

指定了一个无名的结构体类型 。

9.1.2 定义结构体类型变量

(1) 结构体类型与结构体变量是不同的概念，不要混同。只能对变量赋值、存取或运算，而不能对一个类型赋值、存取或运算。在编译时，对类型是不分配空间的，只对变量分配空间。

9.1.2 定义结构体类型变量

(2) 结构体类型中的成员名可以与程序中的变量名相同，但二者不代表同一对象。

(3) 对结构体变量中的成员（即“域”），可以单独使用，它的作用与地位相当于普通变量。

9.1.3 结构体变量的初始化和引用

例9.1 把一个学生的信息(包括学号、姓名、性别、住址)放在一个结构体变量中，然后输出这个学生的信息。

- 解题思路：
  - 自己建立一个结构体类型，包括有关学生信息的各成员
  - 用它定义结构体变量，同时赋以初值
  - 输出该结构体变量的各成员

```
NO.:10101
name:Li Lin
sex:M
address:123 Beijing Road
```

```
#include <stdio.h>
int main()
{struct Student
  { long int num;        char name[20];
    char sex;            char addr[20];
   }a={10101, "Li Lin", 'M' ,
                    "123 Beijing Road" };
  printf("NO.:%ld\nname:%s\n
                  sex:%c\naddress:%s\n",
               a.num,a.name,a.sex,a.addr);
  return 0;
}
```

```
#include <stdio.h>
int main()
{struct Student
  { long int num;       char name[20];
    char sex;           char addr[20];
   }a={{10101, “Li Lin”, ‘M’,
                  “123 Beijing Road”}; }
  printf("NO.:%ld\nname:%s\n
                sex:%c\naddress:%s\n",
              a.num,a.name,a.sex,a.addr);
  return 0;
}
```

```
#include <stdio.h>
int main()
{struct Student
  { long int num;        char name[20];
    char sex;            char addr[20];
   }a={10101, "Li Lin", 'M',
                    "123 Beijing Road" };
    a.num=10010;   对
    printf( "%s\n",a);  不对
     ......
}
```

```
#include <stdio.h>
int main()
{struct Student
  { long int num;        char name[20];
    char sex;            char addr[20];
   }a={10101, "Li Lin", 'M',
                    "123 Beijing Road"};
    struct Student b;
    b=a;   对
    b.num++;   对
     ……
}
```

```
#include <stdio.h>
int main()
{struct Student
  { long int num;        char name[20];
    char sex;            char addr[20];
   }a={10101, "Li Lin", 'M',
                    "123 Beijing Road" };
    scanf("%ld",&a.num);   对
    printf("%o",&a);   对
    scanf( "%ld,%s,%c,%s\n" ,&a); 错
     ……
}
```

```
#include <stdio.h>
int main()
{ struct Date
   {   int month;  int day;  int year; };
   struct Stu
   {   int num;char name[20];
       char sex;int age;
       struct Date birthday;
       char addr[30];
   }a,b;
```

a.birthday.month=12;　对

a.age=10;　b.age=9;　对

sum=a.age+b.age;　对

例9.2 输入两个学生的学号、姓名和成绩，输出成绩较高学生的学号、姓名和成绩

- 解题思路：

(1)定义两个结构相同的结构体变量**student1**和**student2**；

(2)分别输入两个学生的学号、姓名和成绩；

(3)比较两个学生的成绩，如果学生**1**的成绩高于学生**2**，就输出学生**1**的全部信息，如果学生**2**的成绩高于学生**1**，就输出学生**2**的全部信息。如果二者相等，输出**2**个学生的全部信息

```
#include <stdio.h>
int main()
{ struct Student
   { int num;
      char name[20];
      float score;
   }student1,student2;
  scanf("%d%s%f",&student1.num,
        student1.name, &student1.score);
  scanf( “%d%s%f” ,&student2.num,
        student2.name, &student2.score);
```

不能加&

```
  printf("The higher score is:\n");
  if (student1.score>student2.score)
    printf("%d %s %6.2f\n",student1.num,
           student1.name, student1.score);
  else if (student1.score<student2.score)
    printf("%d %s %6.2f\n",student2.num,
           student2.name, student2.score);
  else
  {printf("%d %s %6.2f\n",student1.num,
          student1.name, student1.score);
    printf("%d %s %6.2f\n",student2.num,
          student2.name, student2.score);
  }
  return 0;
}
```

```
10101 Wang 89
10103 Ling 90
The higher score is:
10103  Ling   90.00
```

9.2 使用结构体数组

9.2.1定义结构体数组

例9.3 有3个候选人，每个选民只能投票选一人，要求编一个统计选票的程序，先后输入被选人的名字，最后输出各人得票结果。

9.2.1定义结构体数组

- 解题思路：
  - 设一个结构体数组，数组中包含3个元素
  - 每个元素中的信息应包括候选人的姓名(字符型)和得票数(整型)
  - 输入被选人的姓名，然后与数组元素中的“姓名”成员比较，如果相同，就给这个元素中的“得票数”成员的值加1
  - 输出所有元素的信息

```
#include <string.h>
#include <stdio.h>
struct Person
{  char name[20];
    int count;
}leader[3]={ "Li" ,0, "Zhang" ,0, "Sun" ,0};
```

全局的结构体数组

leader[0]

name	count
Li	0
Zhang	0
Sun	0

```
int main()
{ int i,j;   char leader_name[20];
   for (i=1;i<=10;i++)
   { scanf( "%s" ,leader_name);
      for(j=0;j<3;j++)
         if(strcmp(leader_name,
                  leader[j].name)==0)
            leader[j].count++;
   }
   for(i                    ++)
                           leader[i].count);
   return 0;
}
```

leader[j].count=leader[j].count+1;

```
int main()
{ int i,j;   char leader_name[20];
  for (i=1;i<=10;i++)
  { scanf( “%s” ,leader_name);
    for(j=0;j<3;j++)
      if(strcmp(leader_name,
                leader[j].name)==0)
        leader[j].count++;
  }
  for(i=0;i<3;i++)
    printf("%5s:%d\n “,leader[i].name,
                eader[i].count);
  return
}
```

```
Li
Li
Fun
Zhang
Zhabg
Fun
Li
Fun
Zhang
Li
```

```
Li:4
Zhang:3
Sun:3
```

- 说明：

(1)定义结构体数组一般形式是

① **struct** 结构体名

{成员表列} 数组名[数组长度];

② 先声明一个结构体类型，然后再用此类型定义结构体数组：

结构体类型 数组名[数组长度];

如：

struct Person leader[3];

明德求新 尚用笃行

- 说明：

(2)对结构体数组初始化的形式是在定义数组的后面加上：

＝｛初值表列｝；

如：

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

9.2.2 结构体数组的应用举例

例9.4 有n个学生的信息(包括学号、姓名、成绩)，要求按照成绩的高低顺序输出各学生的信息。

- 解题思路：用结构体数组存放n个学生信息，采用选择法对各元素进行排序(进行比较的是各元素中的成绩)。

```
#include <stdio.h>
struct Student
{ int num; char name[20]; float score;  };
int main()
{ struct Student stu[5]={{10101,"Zhang",78  },
                         {10103,"Wang",98.5},
                         {10106,"Li",     86   },
                         {10108, “Ling” ,  73.5},
                         {10110, “Fun” ,  100  } };
   struct Student tem;
   const int n = 5 ;  int i,j,k;
```

常变量

若人数变为30

30

```
#include <stdio.h>
struct Student
{ int num; char name[20]; float score;  };
int main()
{ struct Student stu[5]={{10101,"Zhang",78  },
                         {10103,"Wang",98.5},
                         {10106,"Li",     86   },
                         {10108, “Ling” , 73.5},
                         {10110, “Fun” , 100  } };
  struct Student temp;
  const int n = 5 ;  int i,j,k;
```

#define N 5

注意temp的类型

```
  printf("The order is:\n");
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++)
      if(stu[j].score>stu[k].score)  k=j;
    temp=stu[k];
    stu[k]=stu[i];   stu[i]=temp;
   }
  for(i=0;i<n;i++)
    printf("%6d %8s %6.2
      stu[i].num,stu[i].na
  printf("\n");
  return 0;
}
```

写法上与普通变量一致

9.3 结构体指针

9.3.1 指向结构体变量的指针

9.3.2 指向结构体数组的指针

9.3.3 用结构体变量和结构体变量的指针作函数参数

9.3.1 指向结构体变量的指针

- 指向结构体对象的指针变量既可以指向结构体变量，也可以用来指向结构体数组中的元素。
- 指针变量的基类型必须与结构体变量的类型相同。例如：

```
struct Student *pt;
```

9.3.1 指向结构体变量的指针

例9.5 通过指向结构体变量的指针变量输出结构体变量中成员的信息。

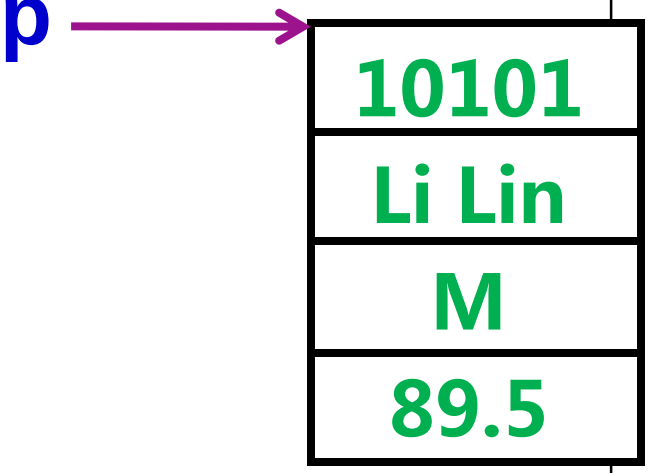
- 解题思路：在已有的基础上，本题要解决两个问题：
  - 怎样对结构体变量成员赋值；
  - 怎样通过指向结构体变量的指针访问结构体变量中成员。


```
#include <stdio.h>
#include <string.h>
int main()
{  struct Student
   { long num;
      char name[20];
      char sex;
      float score;
   };
……
```

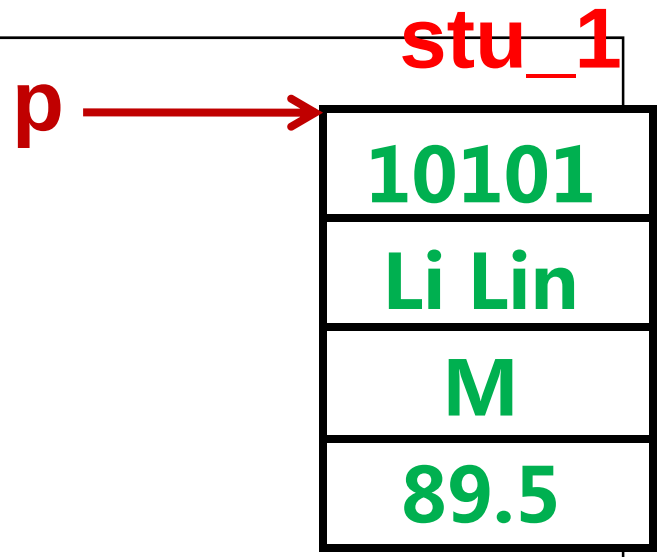
```
struct Student stu_1;
struct Student * p;
p=&stu_1;
stu_1.num
strcpy(stu_
stu_1.sex=
printf("No.:%ld\n" ,stu_1.num);
printf("name:%s\n",stu_1.name);
printf("sex:%c\n" ,stu_1.sex);
printf(" score:%5.1f\n" ,stu_1.score);
return 0;
}
```

```
No.:10101
name:Li Lin
sex:M
score: 89.5
```

```
);
re=89.5;
```



```
struct Student stu_1;
struct Student * p;
p=&stu_1;
stu_1.num=10101;
strcpy(stu_1.name, “Li Lin” );
stu_1.sex='M ‘;  stu_1.score=89.5;
printf("No.:%ld\n” ,stu_1.num);
printf("name:%s\n",stu_1.name);
printf("sex:%c\n",(*p).sex);
printf(” score:%f\n",(*p).score);
return 0;
}
```



(*p).num

(*p).name

(*p).sex

(*p).score

- 说明：
  - 为了使用方便和直观，C语言允许把(*p).num用p->num来代替
  - (*p).name等价于p->name
  - 如果p指向一个结构体变量stu，以下等价：

① stu.成员名(如stu.num)

② (*p).成员名(如(*p).num)

p->成员名(如p->num)

9.3.2 指向结构体数组的指针

例9.6 有3个学生的信息，放在结构体数组中，要求输出全部学生的信息。

- 解题思路：用指向结构体变量的指针处理

(1)声明**struct Student**，并定义结构体数组、初始化

(2)定义指向**struct Student**类型指针**p**

(3)使**p**指向数组首元素，输出元素中各信息

(4)使**p**指向下一个元素，输出元素中各信息

(5)再使**p**指向结构体数组的下一个元素，输出它指向的元素中的有关信息

```
#include <stdio.h>
struct Student
{ int num;    char name[20];
   char sex;   int age;
};
struct Student stu[3]={
            {10101,"Li Lin",'M',18},
            {10102,"Zhang Fun",'M',19},
            {10104,"Wang Min",'F',20} };
```

```
No.  Name                    sex  age
```

```
int main()
{ struct Student *p;
   printf(" No.  Name        sex  age\n");
   for(p=stu;p<stu+3;p++)
      printf( "%5d %-20s %2c %4d\n" ,
                  p->num, p->name,
                     p->sex, p->age);
   return 0;
}
```

10101	Li Lin	M	18	stu[0]
10102	Zhang Fang	M	19	stu[1]
10104	Wang Min	F	20	stu[2]

```
 No.  Name                 sex  age
10101 Li Lin                M    18
```

```
int main()
{ struct Student *p;
  printf(" No.  Name        sex  age\n");
  for(p=stu;p<stu+3;p++)
    printf( "%5d %-20s %2c %4d\n" ,
                p->num, p->name,
                   p->sex, p->age);
  return 0;
}
```

p →

10101	Li Lin	M	18	stu[0]
10102	Zhang Fang	M	19	stu[1]
10104	Wang Min	F	20	stu[2]

```
 No.  Name                 sex  age
10101 Li Lin                M    18
10102 Zhang Fun             M    19
```

```
int main()
{ struct Student *p;
  printf(" No.  Name        sex  age\n");
  for(p=stu;p<stu+3;p++)
    printf( "%5d %-20s %2c %4d\n" ,
                p->num, p->name,
                    p->sex, p->age);
  return 0;
}
```

p →

10101	Li Lin	M	18
10102	Zhang Fang	M	19
10104	Wang Min	F	20

stu[0]
stu[1]
stu[2]

```
 No.  Name                    sex  age
10101 Li Lin                   M    18
10102 Zhang Fun                M    19
10104 Wang Min                 F    20
```

```
int main()
{ struct Student
   printf(" No.  Name          sex  age\n");
   for(p=stu;p<stu+3;p++)
      printf( "%5d %-20s %2c %4d\n" ,
                  p->num, p->name,
                     p->sex, p->age);
   return 0;
}
```

10101	Li Lin	M	18	stu[0]
10102	Zhang Fang	M	19	stu[1]
10104	Wang Min	F	20	stu[2]

p →

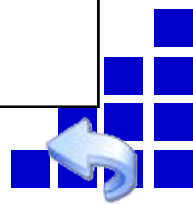
9.3.3 用结构体变量和结构体变量的指针作函数参数

- 将一个结构体变量的值传递给另一个函数，有3个方法。

(1) 用结构体变量的成员作参数。

例如，用**stu[1].num**或**stu[2].name**作函数实参，将实参值传给形参。

- 用法和用普通变量作实参是一样的，属于“值传递”方式。
- 应当注意实参与形参的类型保持一致。



(2) 用结构体变量作实参。

- 用结构体变量作实参时，将结构体变量所占的内存单元的内容全部按顺序传递给形参，形参也必须是同类型的结构体变量
- 在函数调用期间形参也要占用内存单元。这种传递方式在空间和时间上开销较大
- 在被调用函数期间改变形参（也是结构体变量）的值，不能返回主调函数
- 一般较少用这种方法

(3)用指向结构体变量（或数组元素）的指针作实参，将结构体变量（或数组元素）的地址传给形参。

例9.7 有n个结构体变量，内含学生学号、姓名和3门课程的成绩。要求输出平均成绩最高的学生的信息(包括学号、姓名、3门课程成绩和平均成绩)。

- 解题思路：将n个学生的数据表示为结构体数组。按照功能函数化的思想，分别用3个函数来实现不同的功能：
  - 用input函数输入数据和求各学生平均成绩
  - 用max函数找平均成绩最高的学生
  - 用print函数输出成绩最高学生的信息
  - 在主函数中先后调用这3个函数，用指向结构体变量的指针作实参。最后得到结果。
  - 本程序假设n=3

```
#include <stdio.h>
#define N 3
struct Student
{  int num;
   char name[20];
   float score[3];
   float aver;
};
```

4个成员

输入前3个成员值

计算最后成员值

```
int main()
{ void input(struct Student stu[]);
   struct Student max(struct Student stu[]);
   void print(struct Student stu);
   struct Student stu[N],*p=stu;
   input(p);
   print(max(p));
   return 0;
}
```

```
void input(struct Student stu[])
{ int i;
   printf("请输入各学生的信息:
                学号、姓名、三门课成绩:\n");
   for(i=0;i<N;i
   {scanf("%d %s %
              &stu[i].num, stu[i].name,
          &stu[i].score[1],
            &stu[i].score[2]);
      stu[i].aver=(stu[i].score[0]+
          stu[i].score[1]+stu[i].score[2])/3.0;
   }
}
```

i=0

输入第1个成员值

输入第2个成员值

输入第3个成员值

计算第4个成员值

stu

10101	Li	78	89	98	88.33	stu[0]
						stu[1]
						stu[2]

```
void input(struct Student stu[])
{ int i;
   printf("请输入各学生的信息:
                学号、姓名、三门课成绩:\n");
   for(i=0;i<N;i
   {scanf("%d %s %
                &stu[i].num, stu[i].name,
            &stu[i].score[1],
              &stu[i].score[2]);
     stu[i].aver=(stu[i].score[0]+
        stu[i].score[1]+stu[i].score[2])/3.0;
   }
}
```

i=1

输入第1个成员值

输入第2个成员值

输入第3个成员值

计算第4个成员值

stu

10101	Li	78	89	98	88.33	stu[0]
10103	Wang	98.5	87	69	84.83	stu[1]

```
void input(struct Student stu[])
{ int i;
   printf("请输入各学生的信息:
                学号、姓名、三门课成绩:\n");
   for(i=0;i<N;i
   {scanf("%d %s %
                &stu[i].num,stu[i].name,
             &stu[i].score[1],
               &stu[i].score[2]);
     stu[i].aver=(stu[i].score[0]+
        stu[i].score[1]+stu[i].score[2])/3.0;
   }
}
```

i=2

输入第1个成员值

输入第2个成员值

输入第3个成员值

计算第4个成员值

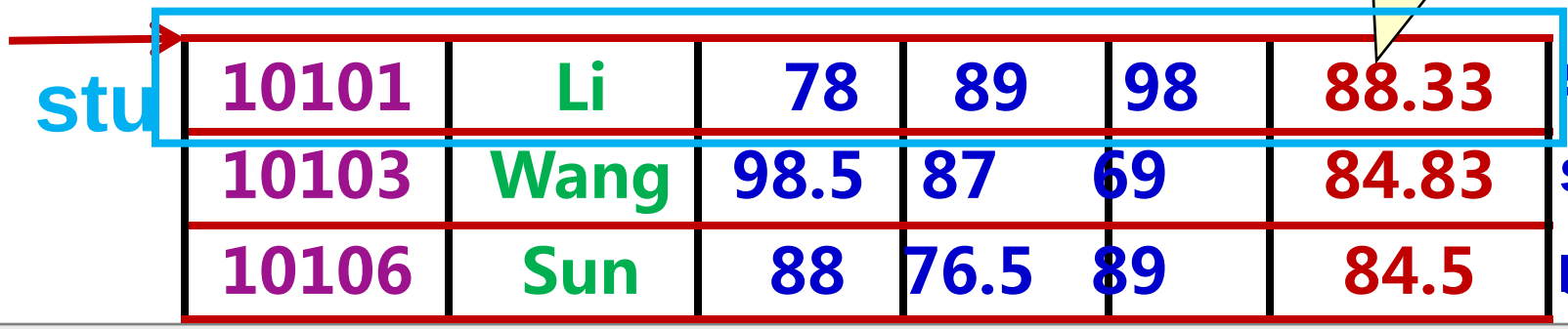
stu

10101	Li	78	89	98	88.33	stu[0]
10103	Wang	98.5	87	69	84.83	stu[1]
10106	Sun	88	76.5	89	84.5	stu[2]

```
struct Student max(struct Student stu[])
{int i,m=0;
  for(i=0;i<N;i++)
     if (stu[i].aver>stu[m].aver) m=i;
  return stu[m];
}
```

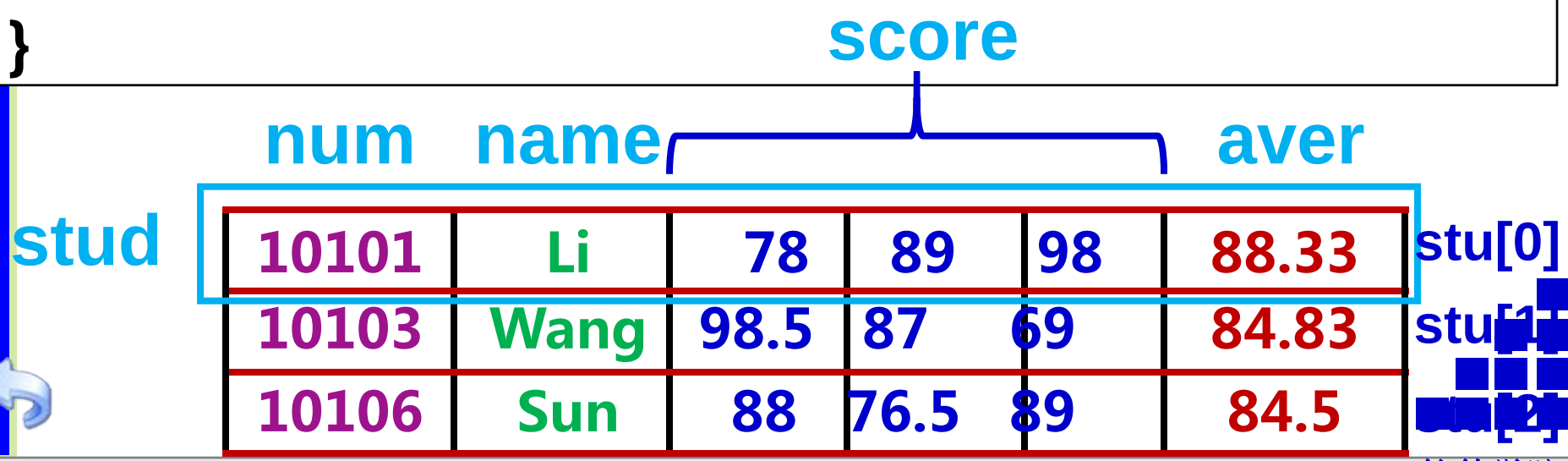
返回

最大



```
void print(struct
{ printf("\n成绩最
   printf("学号:%d
        三门课成绩:%5.1f,%5.1f,%5.1f\n
          平均成绩:%6.2f\n”, stud.num,
           stud.name,stud.score[0],
    stud.score[1],stud.score[2],stud.aver);
}
```

```
成绩最高的学生是:
学号:10101
姓名:Li
三门课成绩: 78.0, 89.0, 98.0
平均成绩: 88.33
```

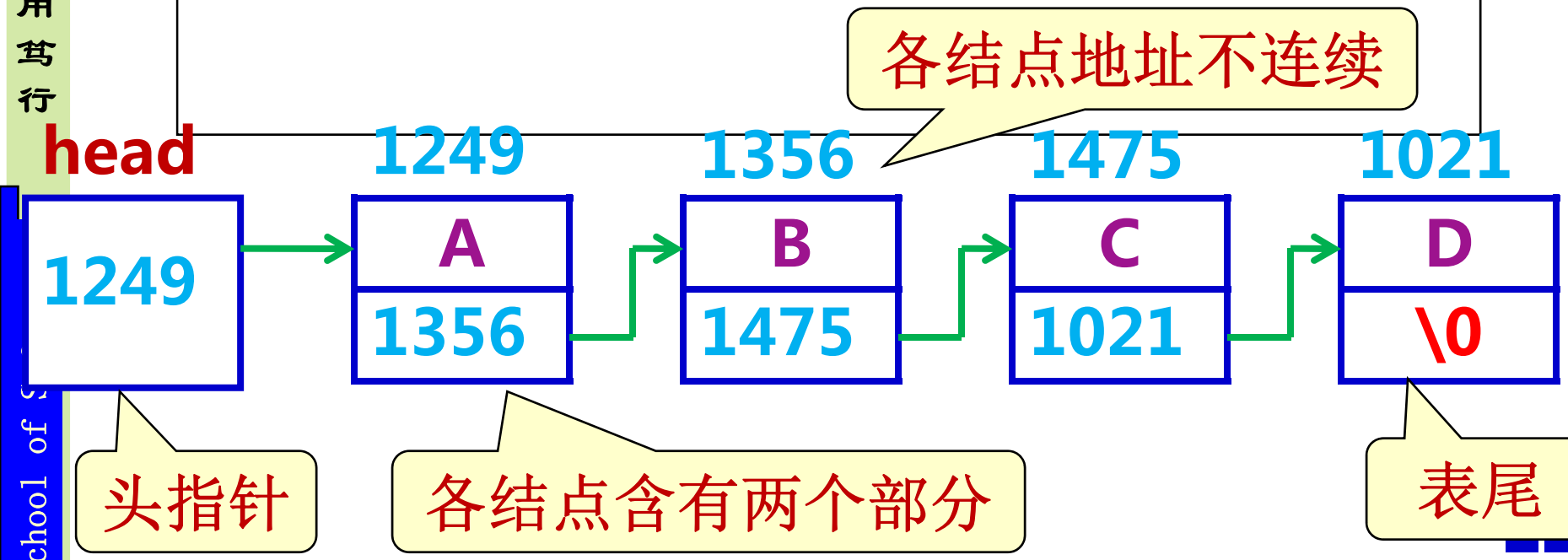


- 以上3个函数的调用，情况各不相同：
  - 调用**input**函数时，实参是指针变量，形参是结构体数组，传递的是结构体元素的地址，函数无返回值。
  - 调用**max**函数时，实参是指针变量，形参是结构体数组，传递的是结构体元素的地址，函数的返回值是结构体类型数据。
  - 调用**print**函数时，实参是结构体变量，形参是结构体变量，传递的是结构体变量中各成员的值，函数无返回值。

9.4 用指针处理链表

9.4.1 什么是链表

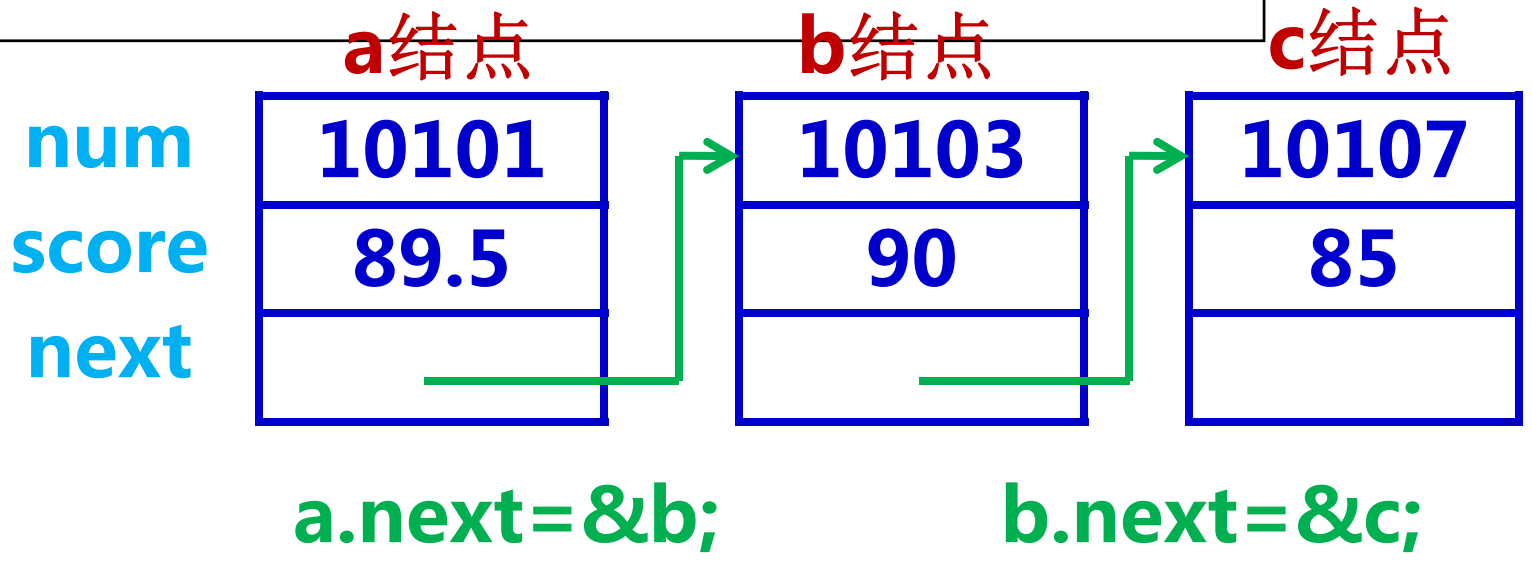
- 链表是一种常见的重要的数据结构
- 它是动态地进行存储分配的一种结构



9.4.1 什么是链表

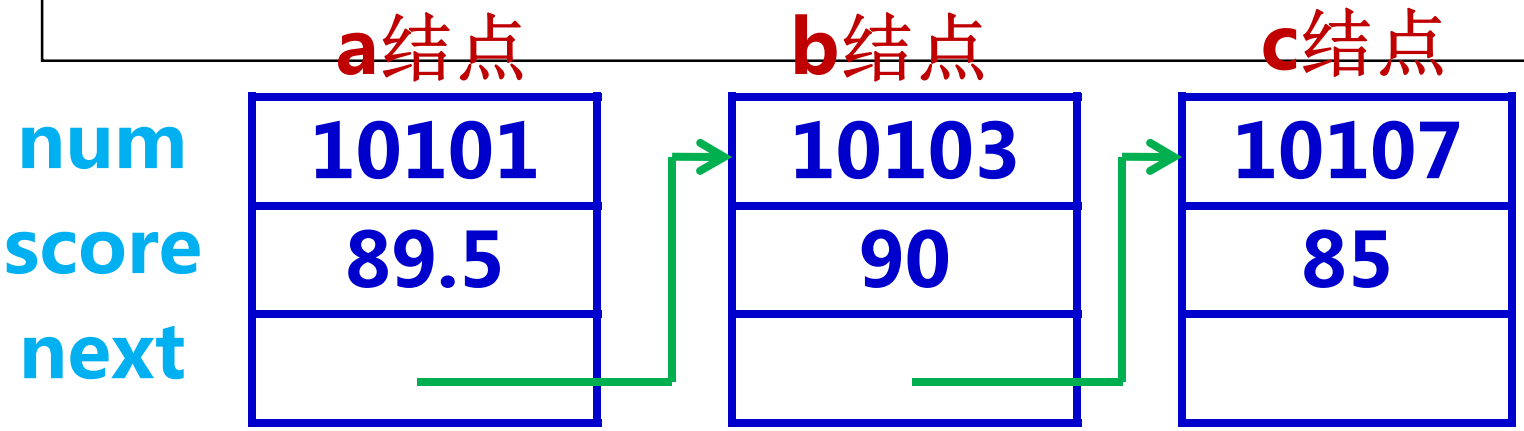
- 链表是一种常见的重要的数据结构
- 它是动态地进行存储分配的一种结构
- 链表必须利用指针变量才能实现

```
struct Student
{  int num;
    float score;
    struct Student *next;
}a,b,c;
```



9.4.2 建立简单的静态链表

例9.8 建立一个如图所示的简单链表，它由3个学生数据的结点组成，要求输出各结点中的数据。

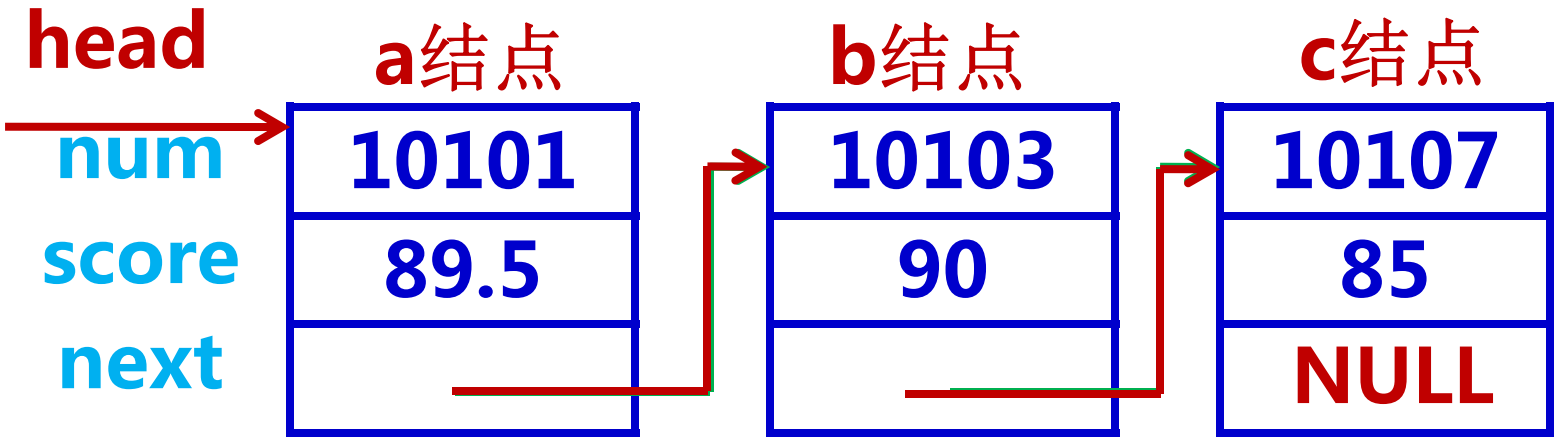


9.4.2 建立简单的静态链表

- 解题思路：

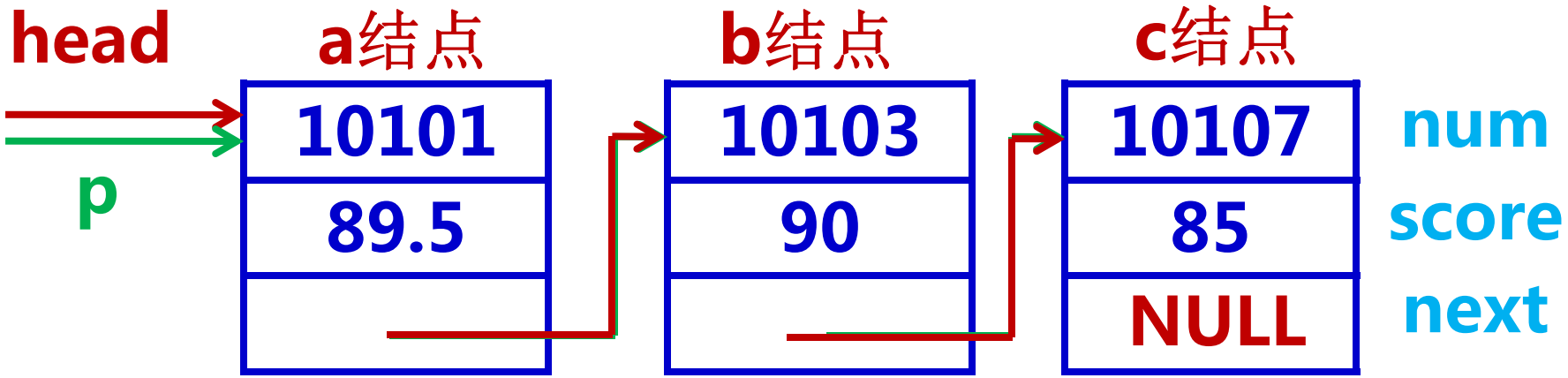
head=&a;　　a.next=&b;

b.next=&c;　　c.next=NULL;



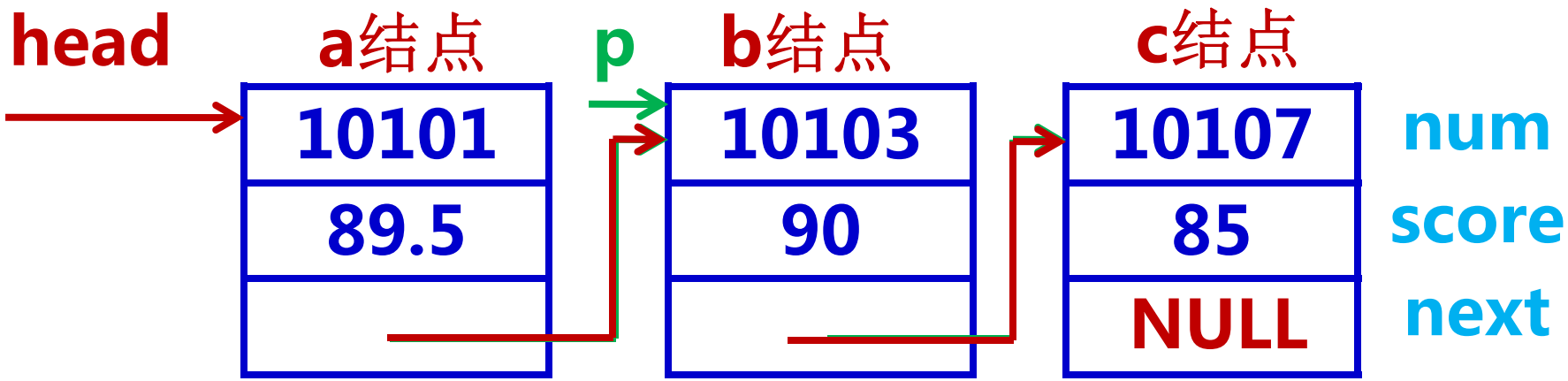
```
#include <stdio.h>
struct Student
{ int num;
   float score;
   struct Student *next;
};
```

```
int main()
{ struct Student a,b,c,*head,*p;
   a. num=10101;  a.score=89.5;
   b. num=10103;  b.score=90;
   c. num=10107;  c.score=85;
   head=&a;          a.next=&b;
   b.next=&c;        c.next=NULL;
   p=head;
   do
   {printf( "%ld%5.1f\n" ,p->num,p->score);
     p=p->next;
    }while(p!=NULL);
   return 0;
}
```



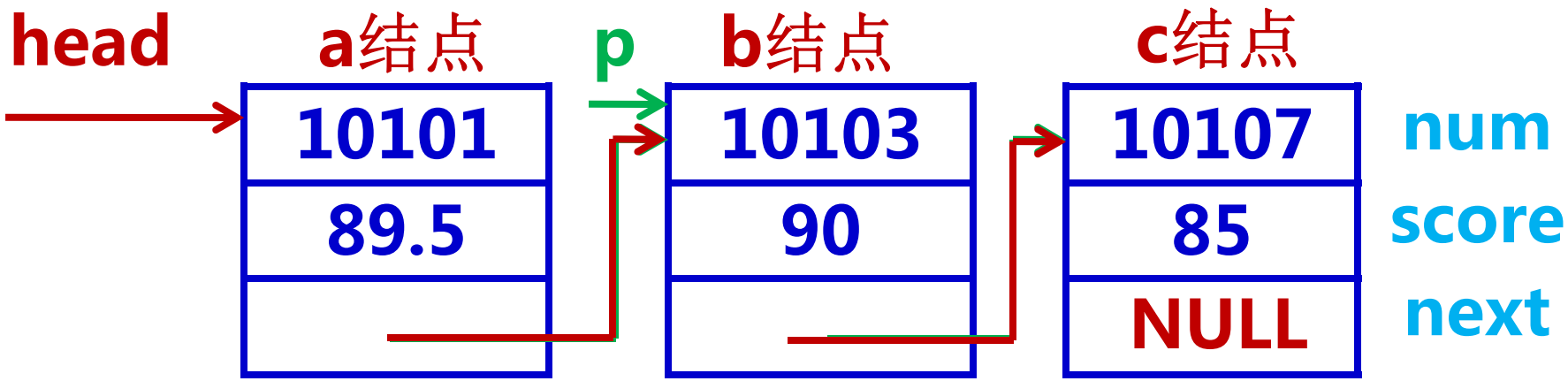

```
p=head;
do
{printf( “%ld%5.1f\n” ,p->num,p->score);
  p=p->next;      相当于p=&b;
 }while(p!=NULL);
return 0;
}
```

```
10101  89.5
```




```
p=head;
do
{printf( “%ld%5.1f\n” ,p->num,p->score);
  p=p->next;      相当于p=&b;
 }while(p!=NULL);
return 0;
}
```

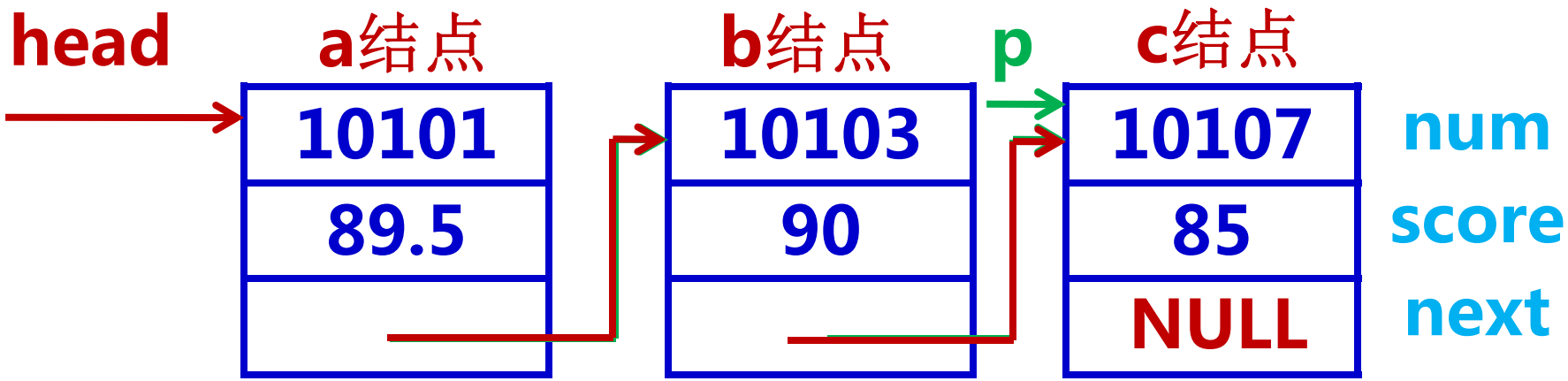
```
10101  89.5
```




```
p=head;
do
{printf( “%ld%5.1f\n” ,p->num,p->score);
  p=p->next;
 }while(p!=NULL);
return 0;
}
```

相当于p=&c;

```
10101  89.5
10103  90.0
```

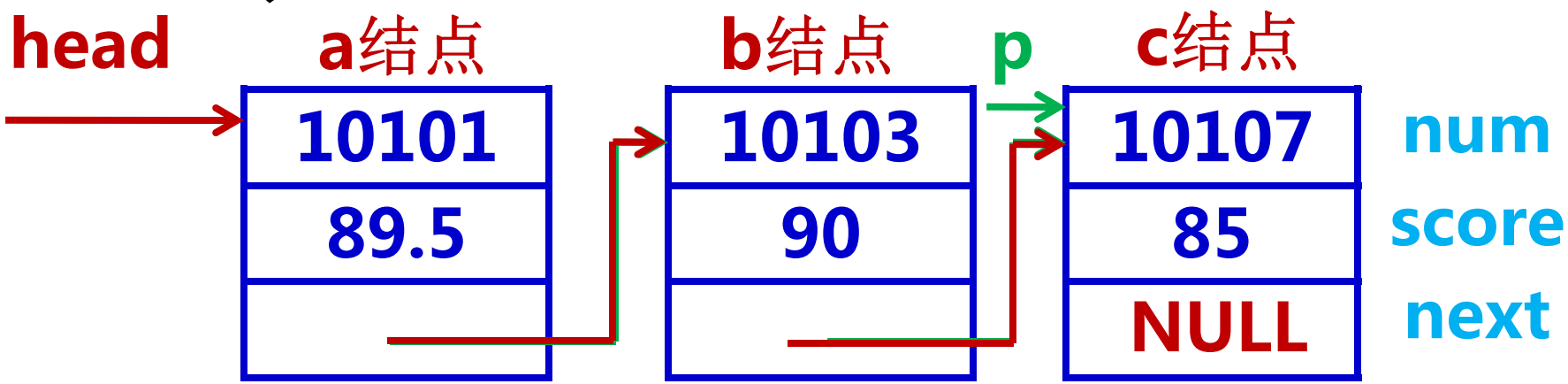



```
p=head;
do
{printf( “%ld%5.1f\n” ,p->num,p->score);
  p=p->next;
 }while(p!=NULL);
return 0;
}
```

相当于p=&c;

```
10101  89.5
10103  90.0
```

静态链表




```
p=head;
do
{printf( “%ld%5.1f\n” ,p->num,p->score);
  p=p->next;
 }while(p!=NULL);
return 0;
}
```

相当于p=NULL;

```
10101   89.5
10103   90.0
10107   85.0
```

9.4.3 建立动态链表

- 所谓建立动态链表是指在程序执行过程中从无到有地建立起一个链表，即一个一个地开辟结点和输入各结点数据，并建立起前后相链的关系。

9.4.3 建立动态链表

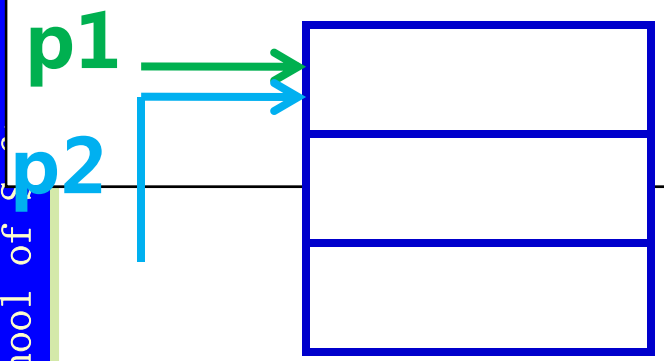
例9.9 写一函数建立一个有3名学生数据的单向动态链表。

- 解题思路：
  - 定义3个指针变量：**head,p1**和**p2**，它们都是用来指向**struct Student**类型数据

```
struct Student *head,*p1,*p2;
```

- 解题思路：
  - 用**malloc**函数开辟第一个结点，并使**p1**和**p2**指向它

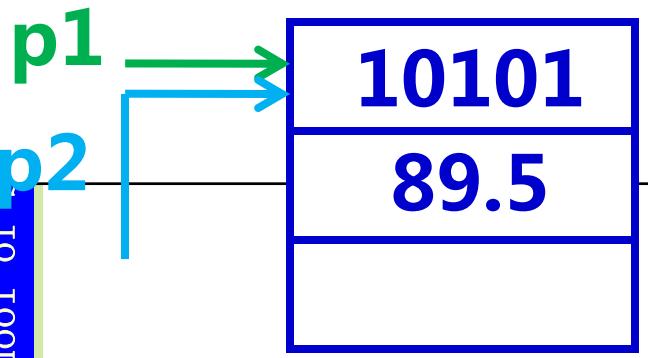
```
p1=p2=(struct Student*)malloc(LEN);
```



解题思路：

- 读入一个学生的数据给**p1**所指的第一个结点

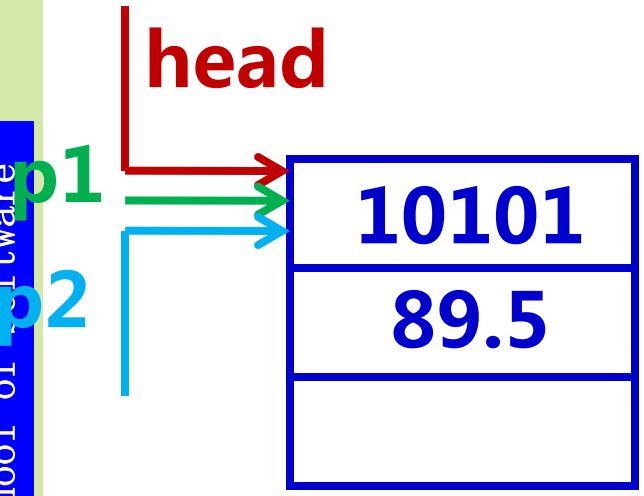
```
scanf("%ld,%f",&p1->num,&p1->score);
```



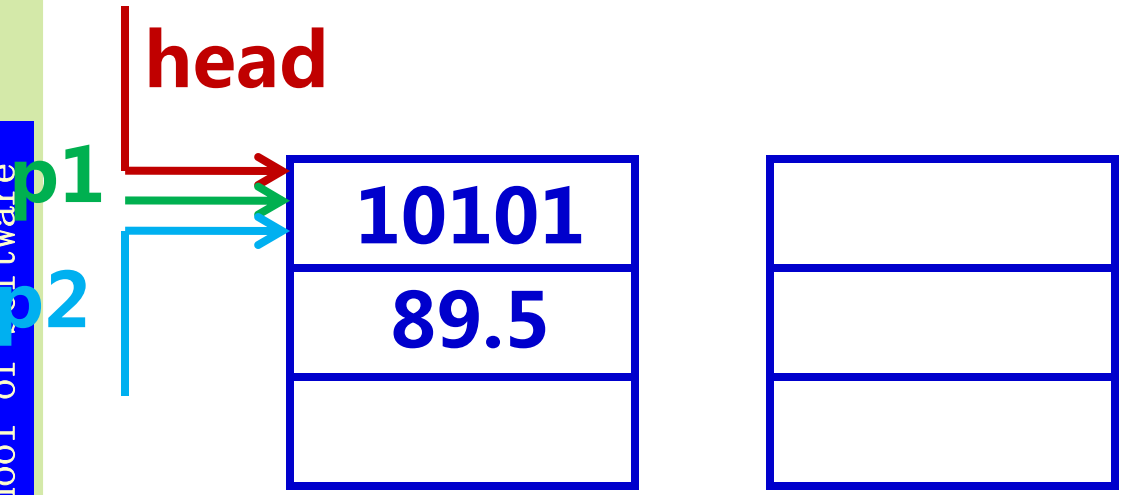
- 解题思路：
  - 读入一个学生的数据给**p1**所指的第一个结点

```
scanf("%ld,%f",&p1->num,&p1->score);
```

  - 使**head**也指向新开辟的结点

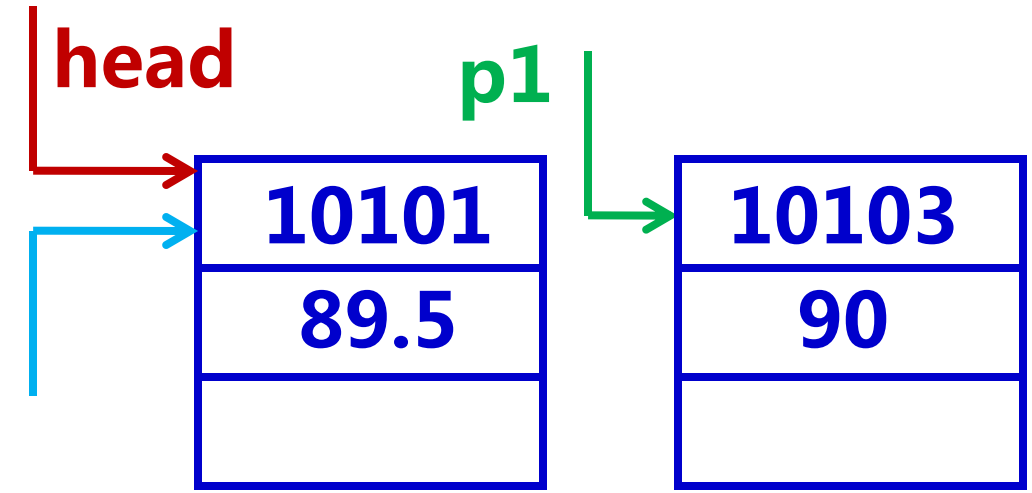


- 解题思路：
  - 再开辟另一个结点并使**p1**指向它，接着输入该结点的数据



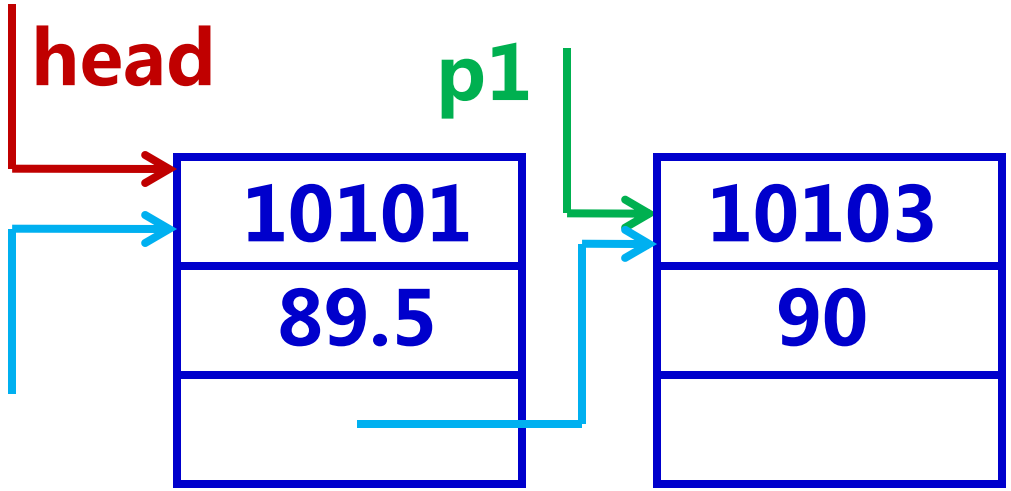
- 解题思路：
  - 再开辟另一个结点并使**p1**指向它，接着输入该结点的数据

```
p1=(struct Student*)malloc(LEN);
scanf("%ld,%f",&p1->num,&p1->score);
```



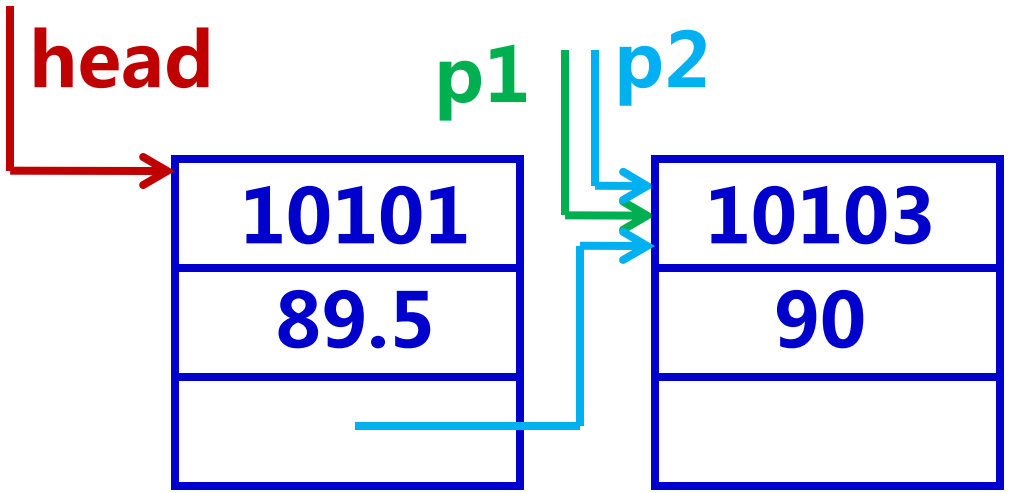
- 解题思路：
  - 使第一个结点的**next**成员指向第二个结点，即连接第一个结点与第二个结点

    p2->next=p1;
  - 使**p2**指向刚才建立的结点

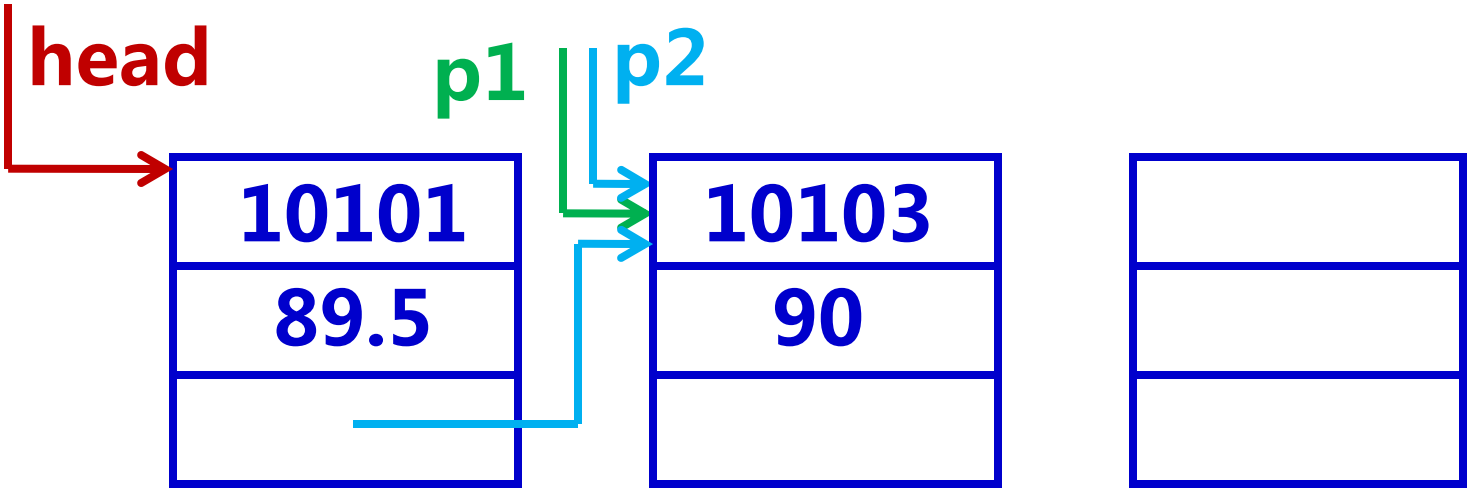


- 解题思路：
  - 使第一个结点的**next**成员指向第二个结点，即连接第一个结点与第二个结点

    p2->next=p1;
  - 使**p2**指向刚才建立的结点　　**p2=p1;**

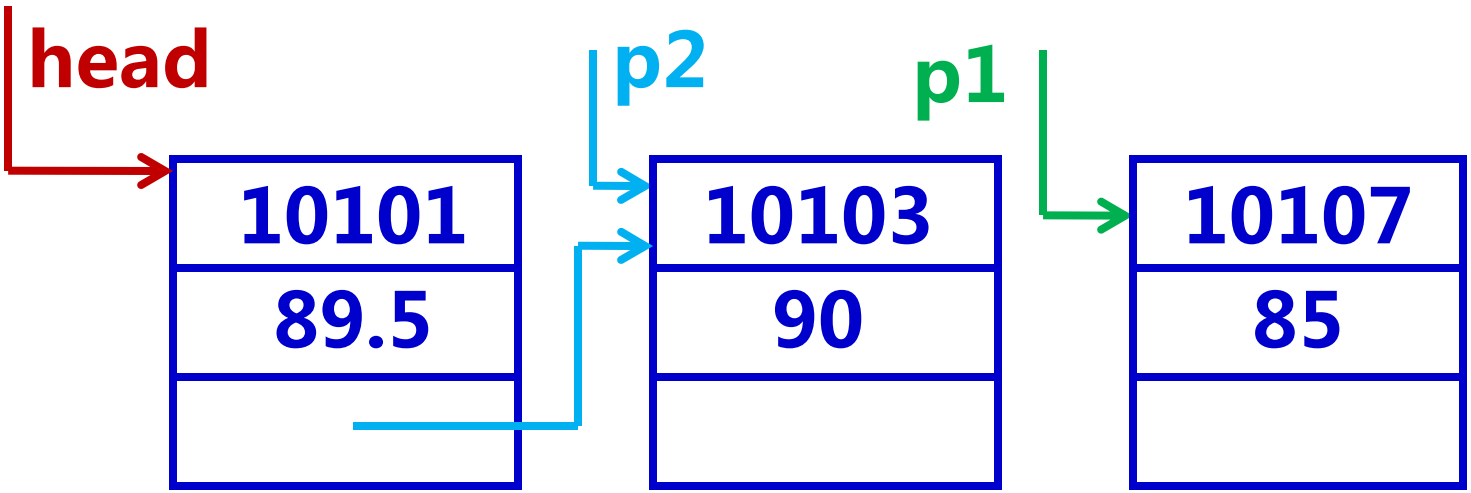


- 解题思路：
  - 再开辟另一个结点并使**p1**指向它，接着输入该结点的数据



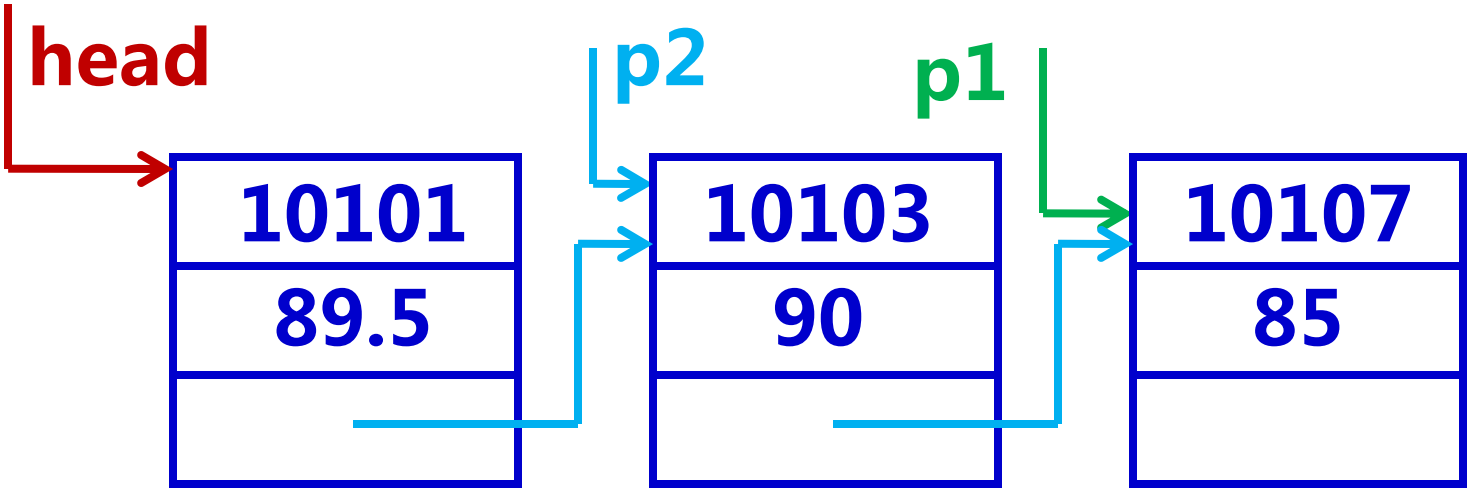
- 解题思路：
  - 再开辟另一个结点并使**p1**指向它，接着输入该结点的数据

```
p1=(struct Student*)malloc(LEN);
scanf("%ld,%f",&p1->num,&p1->score);
```



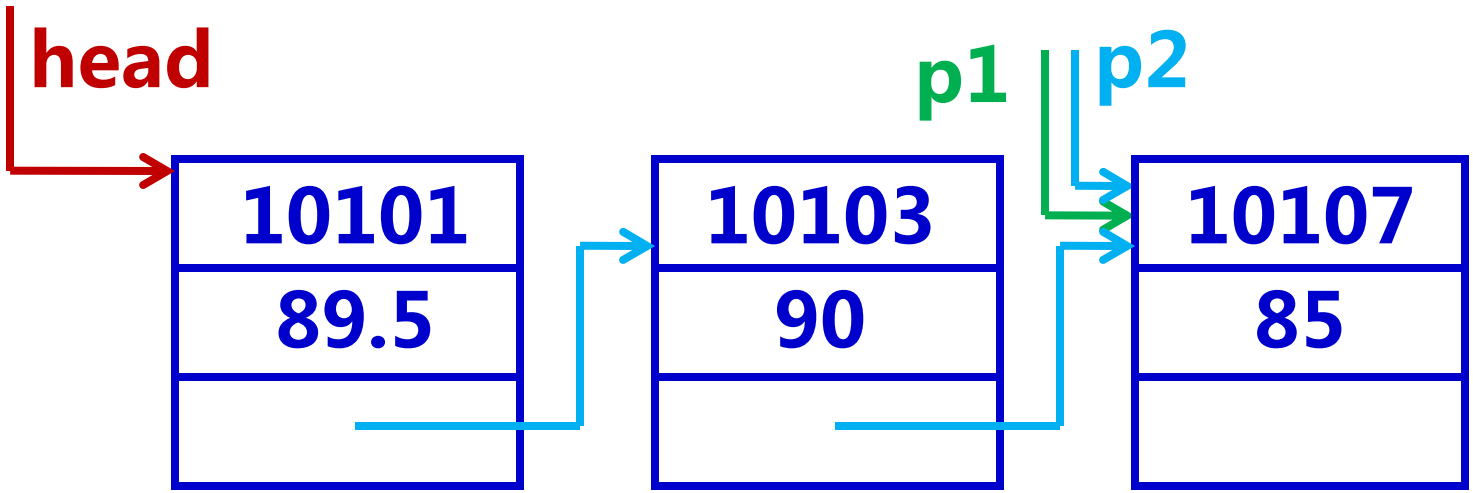
- 解题思路：
  - 使第二个结点的**next**成员指向第三个结点，即连接第二个结点与第三个结点

    p2->next=p1;
  - 使**p2**指向刚才建立的结点

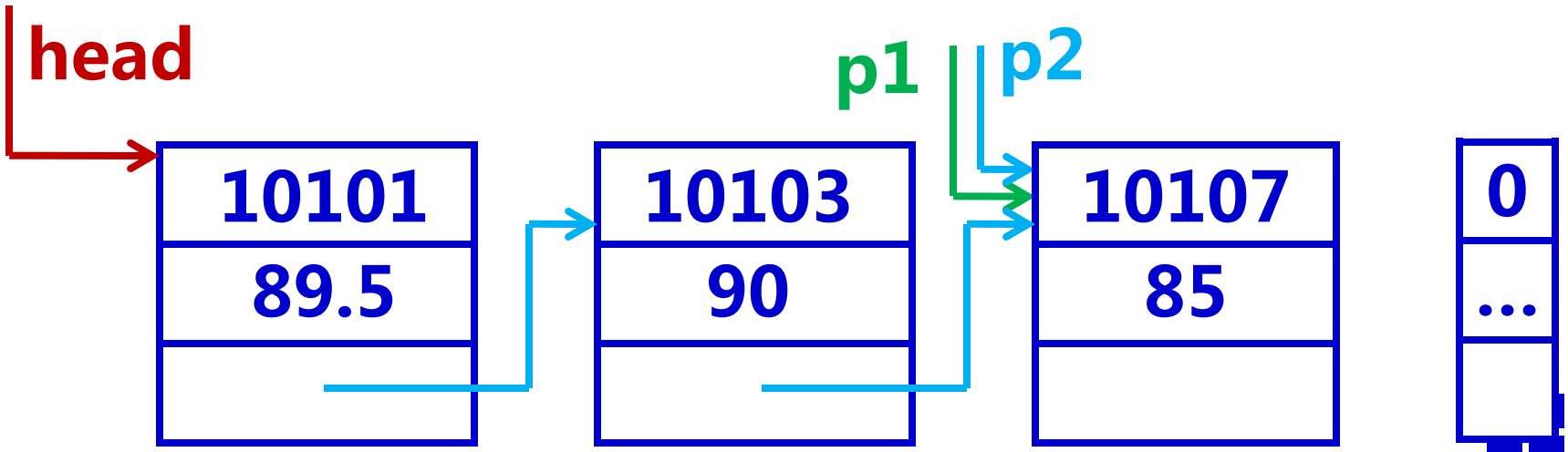


- 解题思路：
  - 使第二个结点的**next**成员指向第三个结点，即连接第二个结点与第三个结点

    p2->next=p1;
  - 使**p2**指向刚才建立的结点　　**p2=p1;**

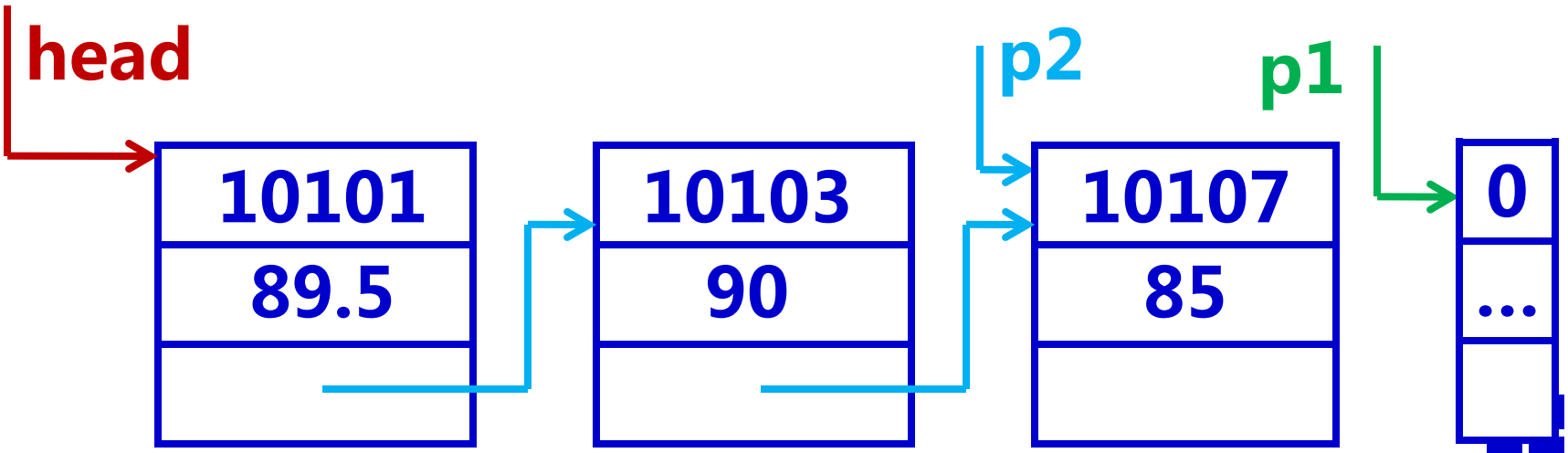


- 解题思路：
  - 再开辟另一个结点并使**p1**指向它，接着输入该结点的数据



- 解题思路：
  - 再开辟另一个结点并使**p1**指向它，接着输入该结点的数据

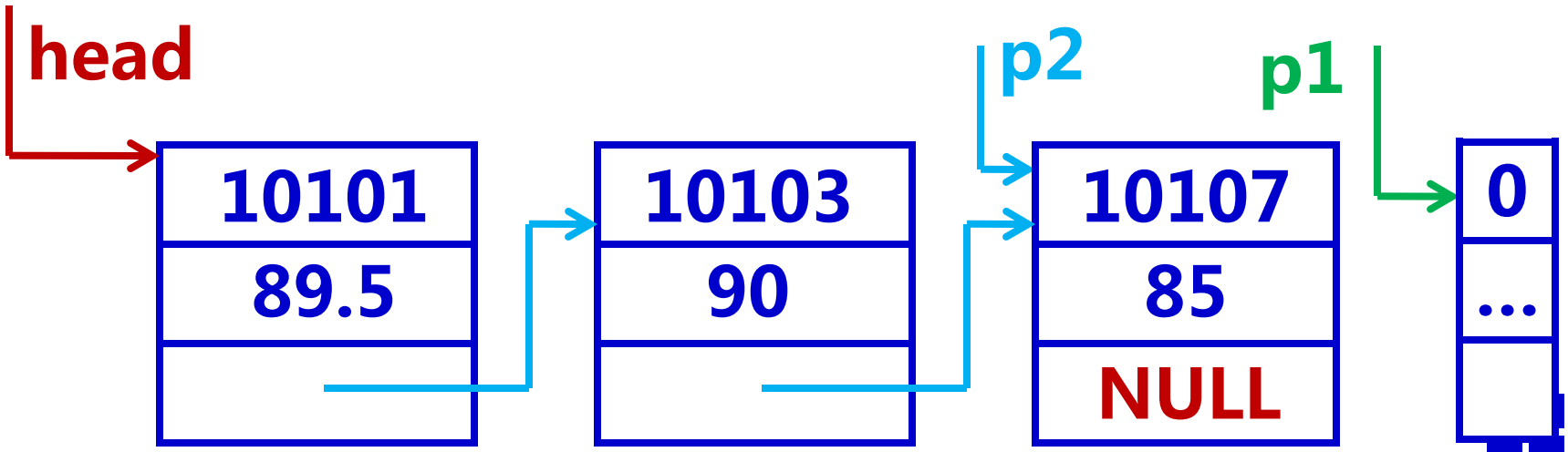
```
p1=(struct Student*)malloc(LEN);
scanf("%ld,%f",&p1->num,&p1->score);
```



解题思路：

- 输入的学号为0，表示建立链表的过程完成，该结点不应连接到链表中

```
p2->next=NULL;
```



struct Student类型数据的长度

```
#include <stdio.h
#include <stdli .h>
#define LEN sizeof(struct Student)
struct Student
{ long num;
   float score;
   struct Student *next;
};
int n;
```

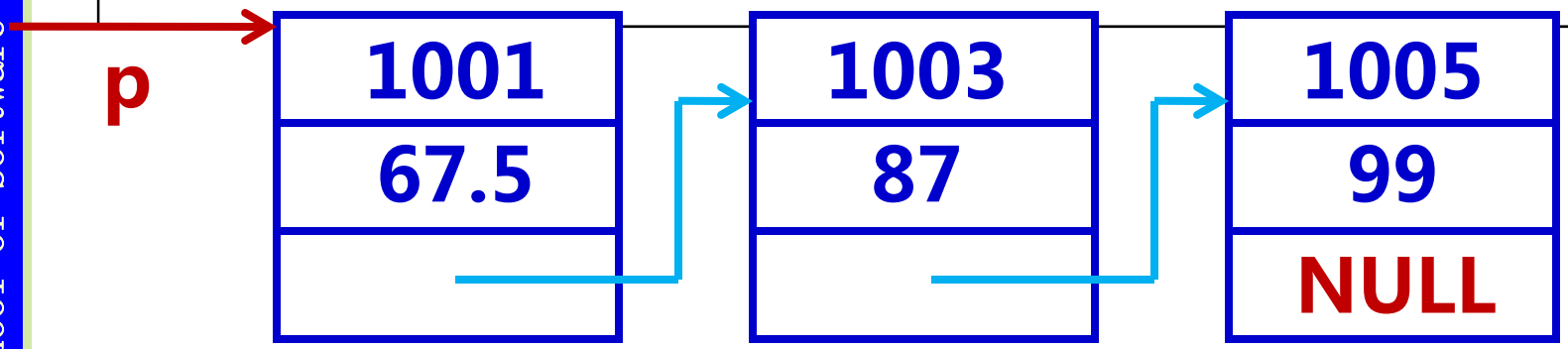
```
struct Student *creat(void)
{ struct Student *head,*p1,*p2;  n=0;
   p1=p2=( struct Student*) malloc(LEN);
   scanf( “%ld,%f” ,&p1->num,&p1->score);
   head=NULL;
   while(p1->num!=0)
   {n=n+1;
     if(n==1) head=p1;
     else  p2->next=p1;
     p2=p1;
     p1=(struct Student*)malloc(LEN);
     scanf( “%ld,%f” ,&p1->num,&p1->score);
   }
   p2->next=NULL;    return(head);
}
```

p1总是开辟新结点
p2总是指向最后结点
用p2和p1连接两个结点

```
int main()
{ struct Student *pt;
   pt=creat();
   printf( "\nnum:%ld\nscore:%5.1f\n" ,
                          pt->num,pt->score);
   return 0;
}
```

9.4.4 输出链表

例9.10 编写一个输出链表的函数print。

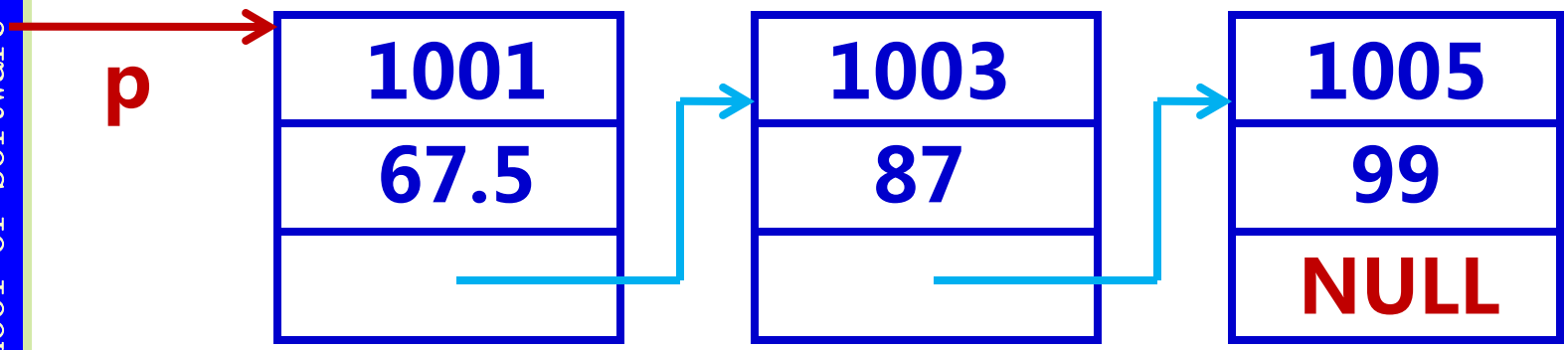


1001 67.5

- 解题思路：
  - 输出p所指的结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

  - 使p后移一个结点



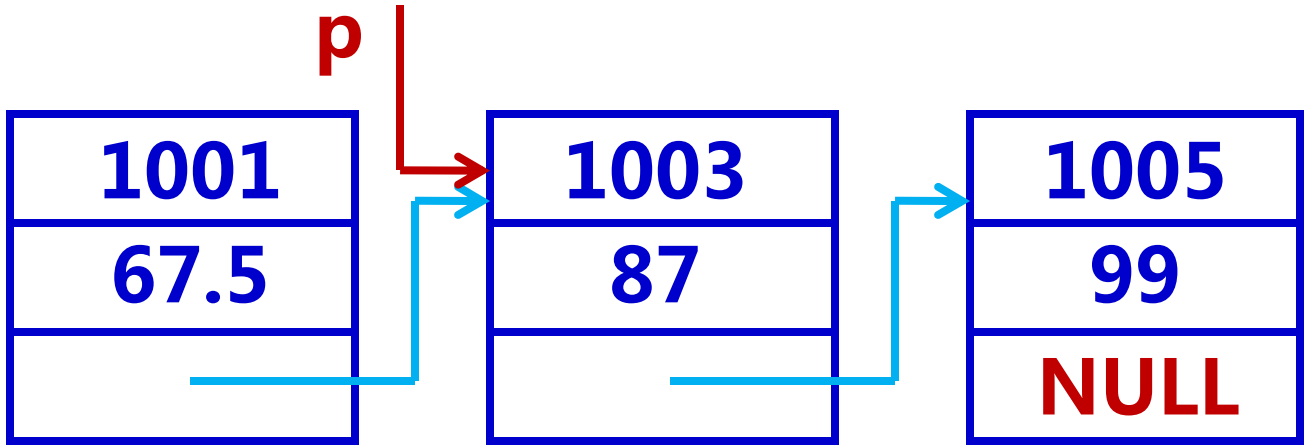
1001 67.5

- 解题思路：
  - 输出p所指的结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

  - 使p后移一个结点

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

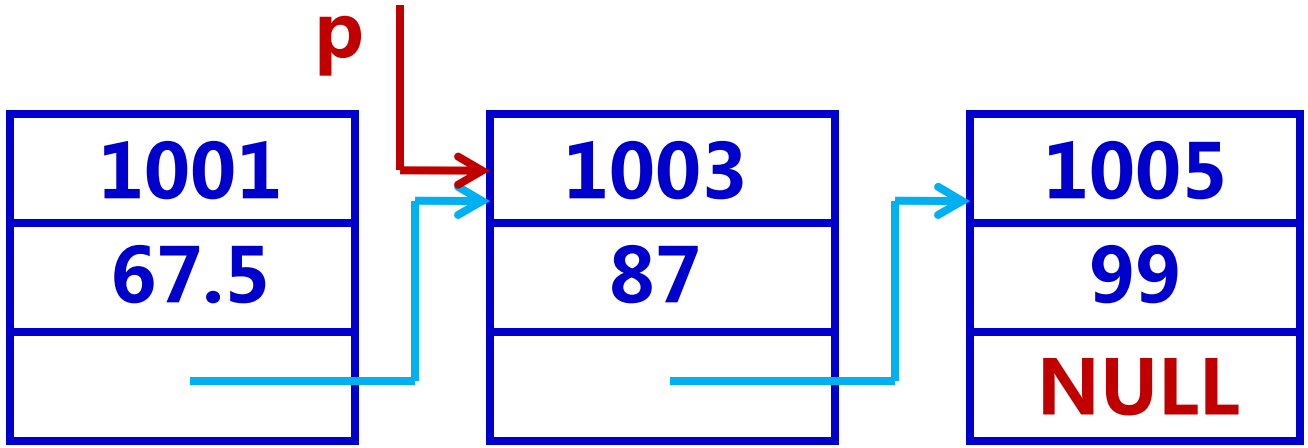


```
1001   67.5
1003   87.0
```

- 解题思路：
  - 输出p所指的新结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

  - 使p后移一个结点



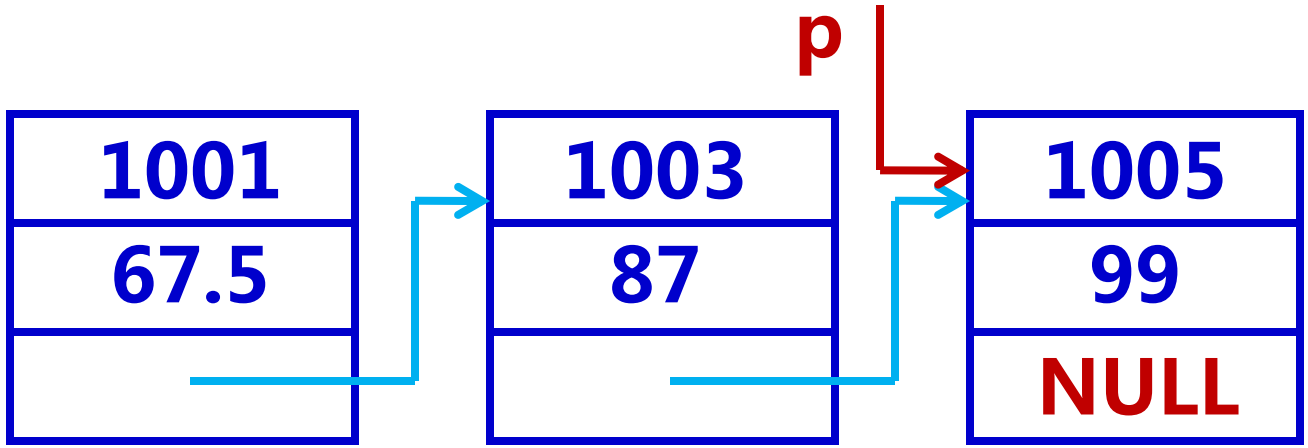
```
1001   67.5
1003   87.0
```

- 解题思路：
  - 输出p所指的新结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

  - 使p后移一个结点

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



```
1001   67.5
1003   87.0
1005   99.0
```

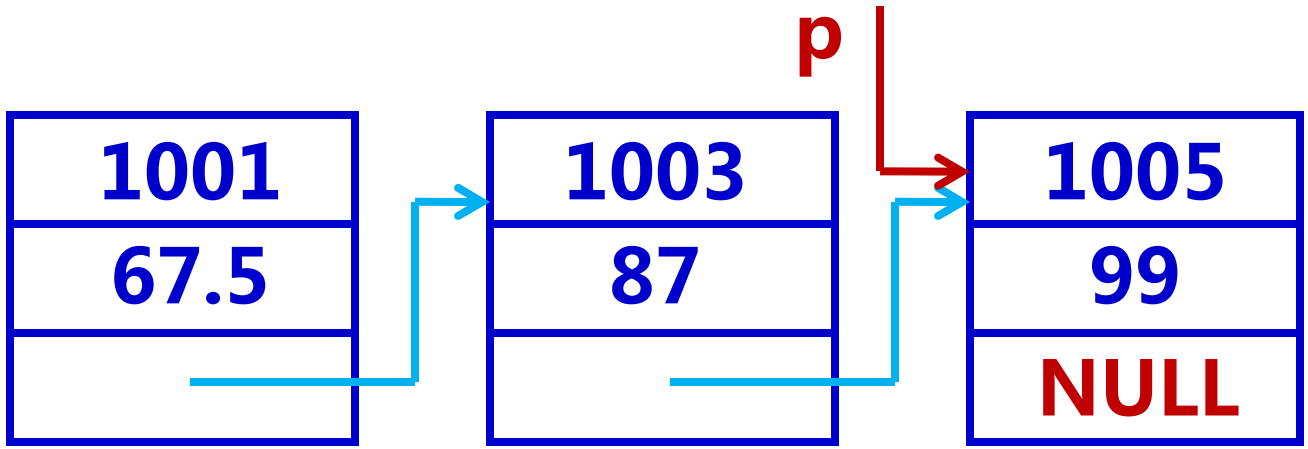
- 解题思路：
  - 输出p所指的新结点

```
printf("%ld %5.1f\n",p->num,p->score);
```

  - 使p后移一个结点

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

相当于p=NULL;

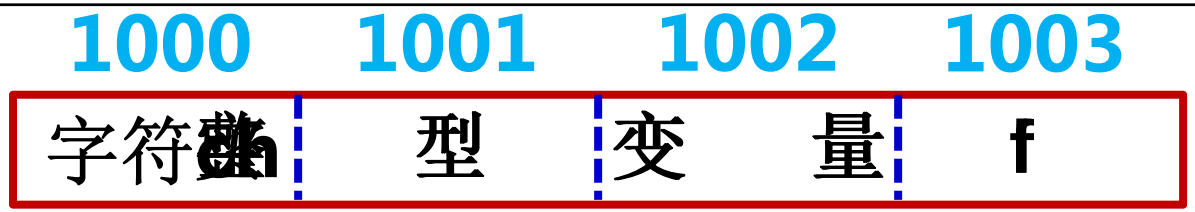


```
void print(struct Student *p)
{
   printf("\nThese %d records are:\n",n);
   if(p!=NULL)
      do
      { printf("%ld %5.1f\n",
                        p->num,p->score);
        p=p->next;
      }while(p!=NULL);
}
```

9.5 共用体类型

9.5.1 什么是共用体类型

- 有时想用同一段内存单元存放不同类型的变量。
- 使几个不同的变量共享同一段内存的结构，称为“共用体”类型的结构。



- 定义共用体类型变量的一般形式为：

```
union   共用体名囍
{ 成员表列囍
}变量表列；  囍
```

例如：

```
union Data瀍
{ int i;瀍
   char ch;瀍
   float f; 瀍
}a,b,c;
```

```
union Data瀍
{ int i;
  char ch;瀍
  float f; 瀍
};
union Data a,b,c;
```

- “共用体”与“结构体”的定义形式相似，但它们的含义是不同的。
- 结构体变量所占内存长度是各成员占的内存长度之和，每个成员分别占有其自己的内存单元。而共用体变量所占的内存长度等于最长的成员的长度。

9.5.2 引用共用体变量的方式

- 只有先定义了共用体变量才能引用它，但应注意，不能引用共用体变量，而只能引用共用体变量中的成员。

例如，前面定义了a,b,c为共用体变量，下面的引用方式是正确的：

a.i　a.ch　a.f

9.5.3 共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

  (1) 同一个内存段可以用来存放几种不同类型的成员，但在每一瞬时只能存放其中一个成员，而不是同时存放几个。

9.5.3 共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

  (2)可以对共用体变量初始化，但初始化表中只能有一个常量。

9.5.3 共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

  (3)共用体变量中起作用的成员是最后一次被赋值的成员，在对共用体变量中的一个成员赋值后，原有变量存储单元中的值就取代。

9.5.3 共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

  (4) 共用体变量的地址和它的各成员的地址都是同一地址。

  (5) 不能对共用体变量名赋值，也不能企图引用变量名来得到一个值。

9.5.3 共用体类型数据的特点

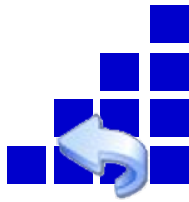
- 在使用共用体类型数据时要注意以下一些特点：

(6) 以前的C规定不能把共用体变量作为函数参数，但可以使用指向共用体变量的指针作函数参数。C99允许用共用体变量作为函数参数。

9.5.3 共用体类型数据的特点

- 在使用共用体类型数据时要注意以下一些特点：

  (7) 共用体类型可以出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可以出现在共用体类型定义中，数组也可以作为共用体的成员。



9.5.3 共用体类型数据的特点

例9.11 有若干个人员的数据，其中有学生和教师。学生的数据中包括：姓名、号码、性别、职业、班级。教师的数据包括：姓名、号码、性别、职业、职务。要求用同一个表格来处理。

解题思路：

- 学生和教师的数据项目多数是相同的，但有一项不同。现要求把它们放在同一表格中

num	name	sex	job	class(班) / position(职务)
101	Li	f	s	501
102	Wang	m	t	prof

- 解题思路：
  - 如果**job**项为**s**，则第 5 项为**class**。即**Li**是**501**班的。如果**job**项是**t**，则第 5 项为**position**。**Wang**是**prof**（教授）。

num	name	sex	job	class(班) / position(职务)
101	Li	f	s	501
102	Wang	m	t	prof

- 解题思路：
  - 对第5项可以用共用体来处理（将class和position放在同一段存储单元中）

num	name	sex	job	class(班) / position(职务)
101	Li	f	s	501
102	Wang	m	t	prof

```
#include <stdio.h>
struct
{ int num;
   char name[10];
   char sex;
   char job;
   union
   { int clas;
      char position[10];
   }category;
}person[2];
```

共用体变量

外部的结构体数组


```
#include <stdio.h>
union  Categ
   { int clas;
     char position[10];
   };
struct
{ int num;
   char name[10];
   char sex;
   char job;
   union  Categ  category
}person[2];
```

声明共用体类型

定义共用体类型变量

```
int main()
{int i;
  for(i=0;i<2;i++)
  {scanf("%d %s %c %c “,&person[i].num,
              &person[i].name,
              &person[i].sex,&person[i].job);
   if(person[i].job == 's')
     scanf("%d “,&person[i].category.clas);
   else if(person[i].job == 't ‘)
     scanf( “%s” ,person[i].category.position);
   else   printf( “Input error!” );
  }
  printf("\n");
```

```
for(i=0;i<2;i++)
{if (person[i].job == ‘s’ )
    printf("%-6d%-10s%-4c%-4c%
        -10d\n",person[i].num,person[i].
        name,person[i].sex,person[i].job,
                person[i].category.clas);
  else
    printf("%-6d%-10s%-4c%-4c%
        -10s\n",person[i].num,person[i].
       name,person[i].sex, person[i].job,
        person[i].category.position);
}
return 0;
}
```

9.6 使用枚举类型

- 如果一个变量只有几种可能的值，则可以定义为枚举类型
- 所谓“枚举”就是指把可能的值一一列举出来，变量的值只限于列举出来的值的范围内

9.6 使用枚举类型

- 声明枚举类型用enum开头。
- 例如：

```
enum Weekday{sun,mon,tue,
             wed,thu,fri,sat};
```

枚举元素

  - 声明了一个枚举类型enum
  - 然后可以用此类型来定义变量

枚举变量

```
enum Weekday  workday,weekend;
```

9.6 使用枚举类型

```
workday=mon;        正确
weekend=sun;        正确
weekday=monday;     不正确
```


明德求新 尚用笃行

- 说明:囍

(1) C编译对枚举类型的枚举元素按常量处理，故称枚举常量。不要因为它们是标识符(有名字)而把它们看作变量，不能对它们赋值。例如:

sun=0; mon=1; 错误

- 说明:囍

(2) 每一个枚举元素都代表一个整数，C语言编译按定义时的顺序默认它们的值为0,1,2,3,4,5...

  - 在上面定义中，sun的值为0，mon的值为1,...sat的值为6
  - 如果有赋值语句: workday=mon;

    相当于workday=1;

- 说明:囍

(2) 每一个枚举元素都代表一个整数，C语言编译按定义时的顺序默认它们的值为0,1,2,3,4,5...

- 也可以人为地指定枚举元素的数值，例如:

```
enum Weekday{sun=7,mon=1,tue,
wed,thu,fri,sat}workday,week_end;
```

- 指定枚举常量sun的值为7，mon为1，以后顺序加1，sat为6。

- 说明:囍

(3) 枚举元素可以用来作判断比较。例如：

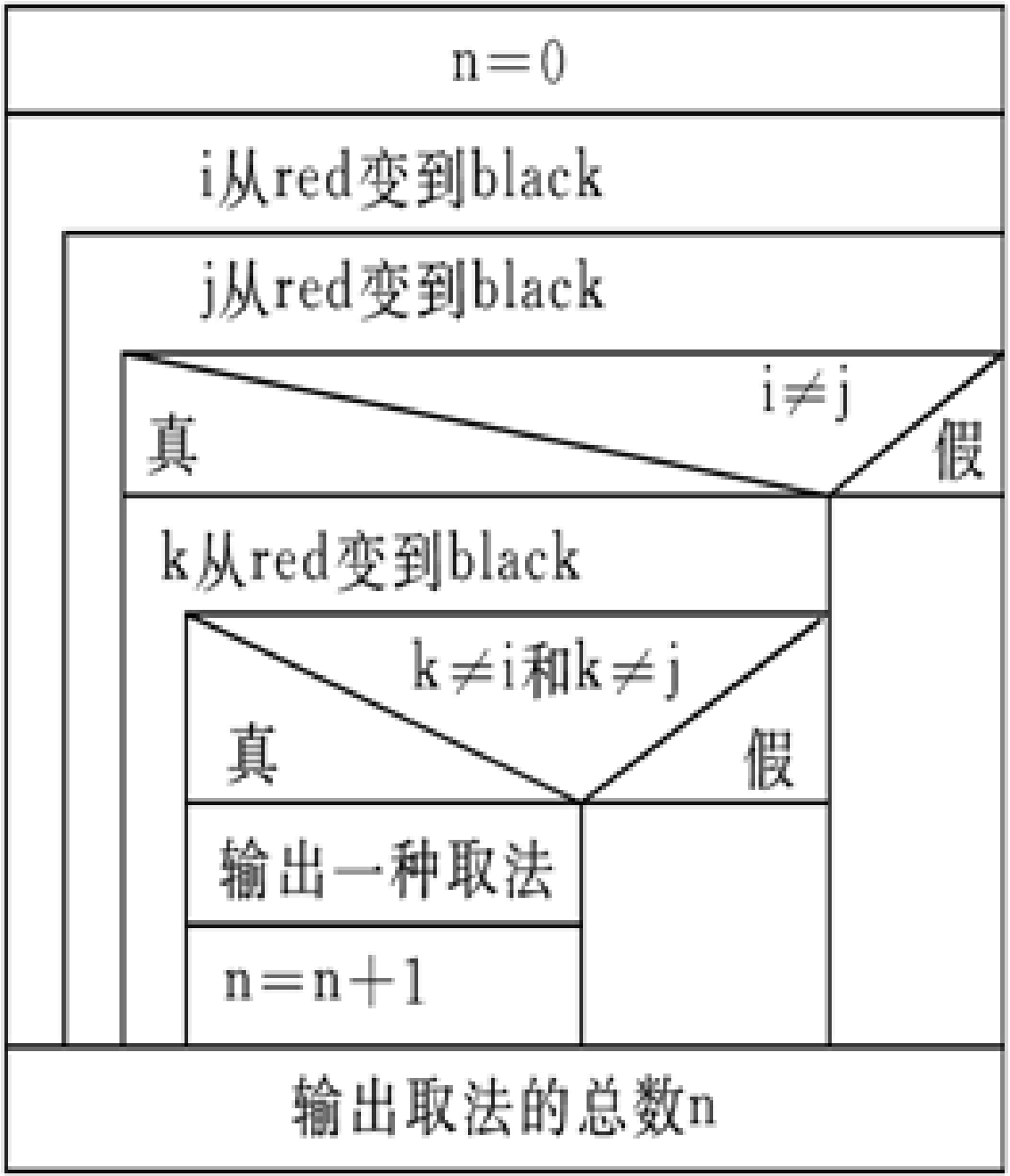
if(workday==mon)...灌

if(workday>sun)...

- 枚举元素的比较规则是按其在初始化时指定的整数来进行比较的。
- 如果定义时未人为指定，则按上面的默认规则处理，即第一个枚举元素的值为 0 ，故 **mon>sun，sat>fri**

例9.12 口袋中有红、黄、蓝、白、黑5种颜色的球若干个。每次从口袋中先后取出3个球，问得到3种不同颜色的球的可能取法，输出每种排列的情况。

解题思路：


n=0
i从red变到black
j从red变到black
i≠j
真
假
k从red变到black
k≠i和k≠j
真
假
输出一种取法
n=n+1
输出取法的总数n

解题思路：

loop由1到3

loop的值

1	2	3
i⇒pri	j⇒pri	k⇒pri

pri的值

red	yellow	blue	white	black
输出"red"	输出"yellow"	输出"blue"	输出"white"	输出"black"

```
#include <stdio.h>
int main()
{enum Color{red,yellow,blue,white,black};
  enum Color i,j,k,pri;
  int n,loop;
  n=0;
  for (i=red;i<=black;i++)
    for (j=red;j<=black;j++)
      if (i!=j)
      { for (k=red;k<=black;k++)
          if ((k!=i) && (k!=j))
          { n=n+1;
            printf( "%-4d" ,n);
```

```
for (loop=1;loop<=3;loop++)
{switch (loop)
  { case 1: pri=i;break;
    case 2: pri=j;break;
    case 3: pri=k;break;
    default:break;
   }
```

```
switch (pri)
{case red: printf( “%-10s” , “red” );
                    break;
  case yellow:printf("%-10s","yellow");
                     break;
  case blue: printf( “%-10s” , “blue” );
                   break;
  case white: printf( “%-10s” , “white” );
                    break;
  case black: printf("%-10s","black");
                    break;
}
```

```
            }
            printf("\n");
        }
    }
  printf("\ntotal:%5d\n",n);
  return 0;
}
```

9.7 用typedef声明新类型名

1.简单地用一个新的类型名代替原有的类型名

typedef int Integer;

typedef float Real;

int i,j; float a,b;等 与

Integer i,j; Real a,b;等

等价

9.7 用typedef声明新类型名

2.命名一个简单的类型名代替复杂的类型表示方法

(1)命名一个新的类型名代表结构体类型：

```
typedef struct
{ int month; int day; int year; }Date;
Date birthday;
Date *p;
```


9.7 用typedef声明新类型名

2.命名一个简单的类型名代替复杂的类型表示方法

(2) 命名一个新的类型名代表数组类型

```
typedef int Num[100];
Num a;
```


9.7 用typedef声明新类型名

2.命名一个简单的类型名代替复杂的类型表示方法

(3)命名一个新的类型名代表一个指针类型

typedef char *String;　　灌

String p,s[10];

9.7 用typedef声明新类型名

2.命名一个简单的类型名代替复杂的类型表示方法

(4)命名一个新的类型名代表指向函数的指针类型

typedef int (*Pointer)(); 灌

Pointer p1,p2;

9.7 用typedef声明新类型名

- 归纳起来，声明一个新的类型名的方法是
  - ① 先按定义变量的方法写出定义体（**int i;**）
  - ② 将变量名换成新类型名（将**i**换成**Count**）
  - ③ 在最前面加**typedef**
    （**typedef int Count**）
  - ④ 用新类型名去定义变量

9.7 用typedef声明新类型名

- 以定义上述的数组类型为例来说明：囍
  - ① 先按定义数组变量形式书写：**int a[100];**
  - ② 将变量名**a**换成自己命名的类型名：**int Num[100];**囍
  - ③ 在前面加上**typedef**，得到**typedef int Num[100];**囍

  用来定义变量：**Num a;**囍

  相当于定义了：**int a[100];**

9.7 用typedef声明新类型名

- 对字符指针类型，也是：

```
char *p;
char *String;
typedef char *String;
String p;
```


9.7 用typedef声明新类型名

- 说明：甚

(1)以上的方法实际上是为特定的类型指定了一个同义字(synonyms)。例如

①typedef int Num[100];

Num a;　　Num是int [100]的同义词

②typedef int (*Pointer)();

Pointer p1; Pointer是int (*)()的同义词

9.7 用typedef声明新类型名

- 说明：囍

(2) 用**typedef**只是对已经存在的类型指定一个新的类型名，而没有创造新的类型。

(3)用**tyoedef**声明数组类型、指针类型，结构体类型、共用体类型、枚举类型等，使得编程更加方便。

(4)**typedef**与**#define**表面上有相似之处

9.7 用typedef声明新类型名

- 说明：囍

(5) 当不同源文件中用到同一类型数据时，常用**typedef**声明一些数据类型。可以把所有的**typedef**名称声明单独放在一个头文件中，然后在需要用到它们的文件中用**#include**指令把它们包含到文件中。这样编程者就不需要在各文件中自己定义**typefef**名称了。

9.7 用typedef声明新类型名

- 说明：囍

(6) 使用**typedef**名称有利于程序的通用与移植。有时程序会依赖于硬件特性，用**typedef**类型就便于移植。