第4章 选择结构和循环结构

C++语句分为：单语句、复合语句、结构性语句

单语句

```
int  a,b=3;
a=b*5;
cout<<a<<endl;
;   ;
```

空语句

一个语句

复合语句

```
{ t=a; a=b; b=t; }
{  cout<<x<<"\t";
   k++;
   if(k%5==0）
        cout<<endl;
}
```

一个语句

结构性语句

一个语句

```
if(score<60)
{cout<<"不及格"<<endl;}
else
{cout<<"及格"<<endl;}

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

4.1 if语句及其嵌套

if 语句的一般形式：

```
if( 表达式 )
        一个内嵌语句
else
        一个内嵌语句
```

可以省略

if语句的常用嵌套
- if-else if 形式嵌套
- if-if-else 形式嵌套

if-else if 嵌套的一般形式：

```
if( 表达式1 )
        内嵌语句1
else if( 表达式2 )
        内嵌语句2
……
else if( 表达式n )
        内嵌语句n
else
        内嵌语句 n+1
```

例C4.1 输入3个实数，输出最大数(使用if-else if)。

```
if( x>=y&&x>=z ) cout<<"最大数是： "<<x<<endl;
else if(y>=z)    cout<<"最大数是： "<<y<<endl;
else             cout<<"最大数是： "<<z<<endl;
```

一般形式的if 嵌套的例子

例A4.2 输入3个实数，输出中间大小的数(使用if嵌套)。

```
if( x<y )      //考虑区间 [x,y]
{       if( z<x )       cout<<x<<endl;
        else if( y<z )  cout<<y<<endl;
        else            cout<<z<<endl;
}
else           //考虑区间 [y,x]
{       if( z<y )       cout<<y<<endl;
        else if( x<z )  cout<<x<<endl;
        else            cout<<z<<endl;
}
```

if-if-else 嵌套的一般形式：

```
if( 表达式1 )
    if( 表达式2 )
        内嵌语句1
    else
        内嵌语句2
```

自然配对

```
if( 表达式1 )
{   if( 表达式2 )
        内嵌语句1
}
else
    内嵌语句2
```

用括号强制配对

```
y=0;
if( x!=0 )
    if( x>0 )  y=1;
    else       y= -1
```

```
y= -1;
if( x!=0 )
{    if( x>0 )  y=1;  }
else      y= 0
```

4.2 条件表达式

条件表达式的一般形式：

表达式1？表达式2：表达式3

例：a>b?a:b　　x>0?1:0　　k%5==0?'\n':'\t'

例A4.3 输入一个实数x,输出符号函数值sgn(x)(使用条件表达式)。

sgn= (x<0)?-1: (x>0)?1:0;　两个条件运算

sgn=(x<0)? -1: (x>0)?1:0 ;　等价于 (x<0)?-1:((x>0)?1:0)

4.3 ○ switch语句

switch语句的一般形式：

```
switch( 整型表达式 )
{
case  常量表达式1:
                语句序列1
case  常量表达式2:
                语句序列2
 … … … … … … … …
case  常量表达式n:
                语句序列n
default :
                语句序列n+1
}
```

通常后跟**break;**
用于跳出swicth块

可以省略

注1：switch语句中的整数表达式包括int及其派生类型、字符类型和逻辑类型，常量表达式的值应该是对应的整数类型值。

```
switch( grade)
{
case  'A':
   cout<<"90~100"<<endl;  break;
case  'B':
   cout<<"80~89"<<endl;   break;
case  'C':
   cout<<"70~79"<<endl;   break;
case  'D':
   cout<<"60~69"<<endl;   break;
case  'E':
   cout<<"0~59"<<endl;
}
```

含break语句

```
switch( grade)
{
case  'A':
   cout<<"90~100"<<endl;
case  'B':
   cout<<"80~89"<<endl;
case  'C':
   cout<<"70~79"<<endl;
case  'D':
   cout<<"60~69"<<endl;
case  'E':
   cout<<"0~59"<<endl;
}
```

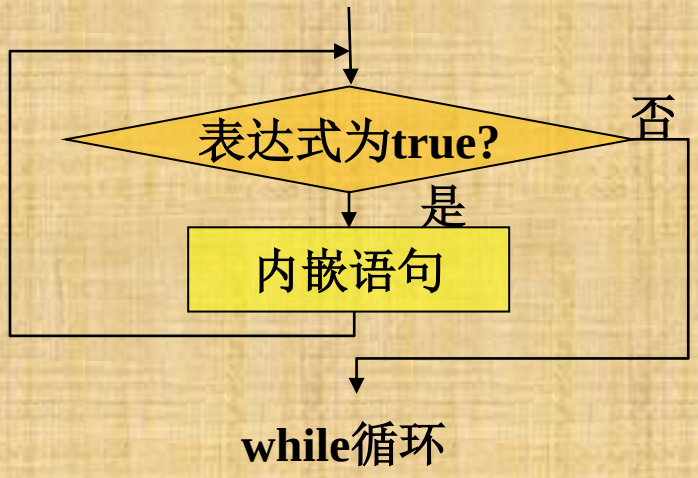
不含break语句

4.4 while语句和do-while语句

while语句的一般形式：

表示循环条件

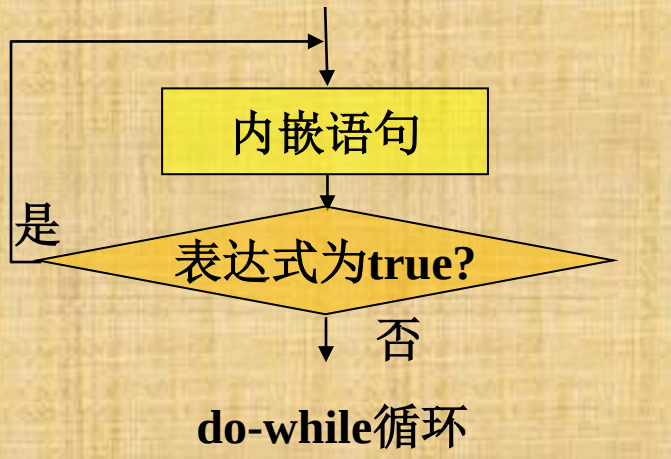
```
while(表达式 )
    一个内嵌语句
```



do-while语句的一般形式：

```
do
    一个内嵌语句
while(表达式 )
```

表示循环条件



例C4.5 输入一个弧度数x，$0 \le x \le \pi/2$，计算sin*x*的值(使用while)。

计算公式： $\sin x \approx x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \cdots + \frac{(-1)^n}{(2n+1)!}x^{2n+1}$

```
SIN=0;  s=1;  t=x;  n=1;
while( t>=1e-7)
{   SIN+=s*t;  t=t*x*x/(2*n*(2*n+1));  s=-s;  n++;  }
```

例C4.6 输入一个非负实数*a*，求 $\sqrt{a}$ 的近似值(使用do-while)。

求$\sqrt{a}$ 的迭代公式： $x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n}), n = 1, 2, 3, \cdots$

```
x2=a;
do
{   x1=x2;  x2=(x1+a/x1)/2;  }
while( fabs(x2-x1)>=1e-7 );
```

4.5 for语句

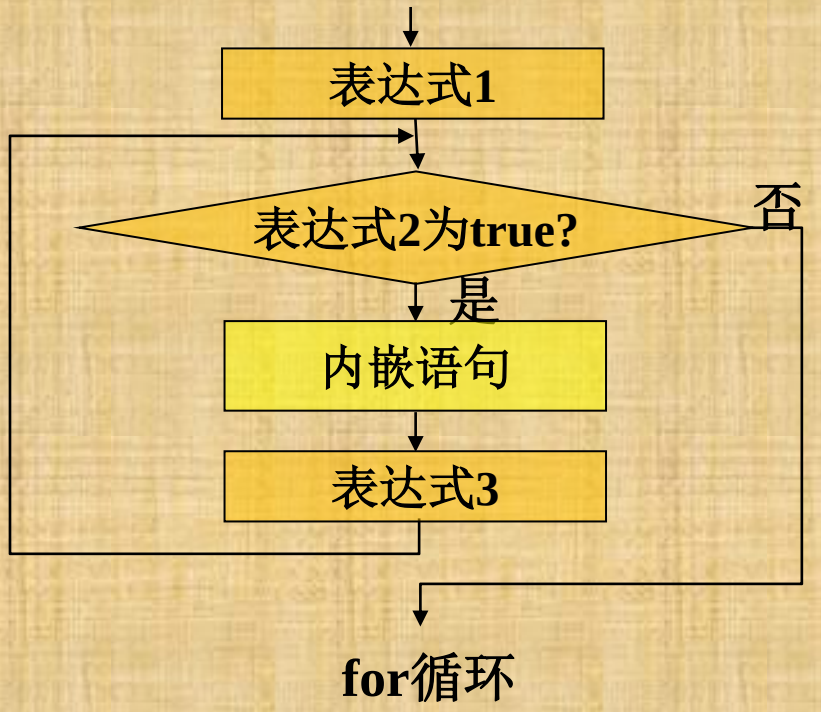
for语句的一般形式：

表示初始设置　　表示循环量变化

for(表达式1；表达式2；表达式3)

一个内嵌语句

表示循环条件



例B4.7 计算1+2+3+…+100和两个输入数的最大公约数。

```
i=1; s1=0;  for( ; i<=100 ; i++) s1+=i;
```
表达式1空白

```
for(i=1,s2=0; i<=100;  ) { s2+=i; i++; }
```
表达式1为逗号式，表达式3空白

```
s3=0; for(int j=1; j<=100; j++) s3+=j;
```
表达式1为变量定义

```
for(cin>>a>>b; b!=0; r=a%b, a=b, b=r ) ;
```
表达式1为输入,表达式3为逗号式,循环体为空语句

注1: for表达式1中如果是变量定义，所定义的变量的使用范围按照C++标准只在for语句中，for语句后面的地方就不能使用这些变量。例如下面的程序片段有错

```
for(int sum=0,i=1;i<=100;i++) sum+=i;
cout<<sum<<endl; //错误,不能使用for语句的内部变量sum
```

注2: for表达式1中只能定义一种类型的变量，但可以象注1的for语句那样定义多个变量。若表达式1定义了多种类型变量则不合法。

例C4.8 输出所有的水仙花数。　　如：$153=1^3+5^3+3^3$

```
a100=n/100;          取百位数字
a10=n/10%10;         取十位数字，或 n%100/10
a=n%10;              取个位数字
a100*a100*a100+a10*a10*a10+a*a*a==n    判断水仙花数
```

例A4.9 输出九九乘法表(嵌套for循环)。

```
for( i=1 ; i<=9 ; i++ )                    输出9行
{       cout<<i<< "\t";
        for(j=1; j<=i; j++ )   输出一行的各列，每列4字符
                if(j*i>=10)   cout<< "□□"<<j*i;
                else          cout<<"□□□"<<j*i;
```

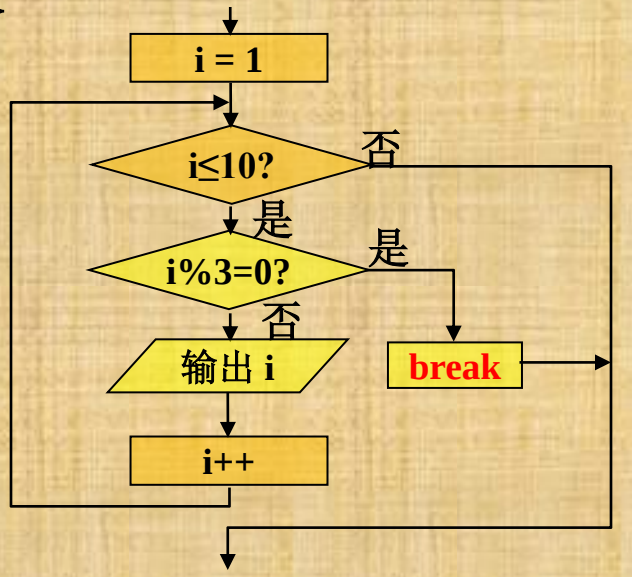
4.6 break和continue语句

break和continue都是控制流程跳转的语句

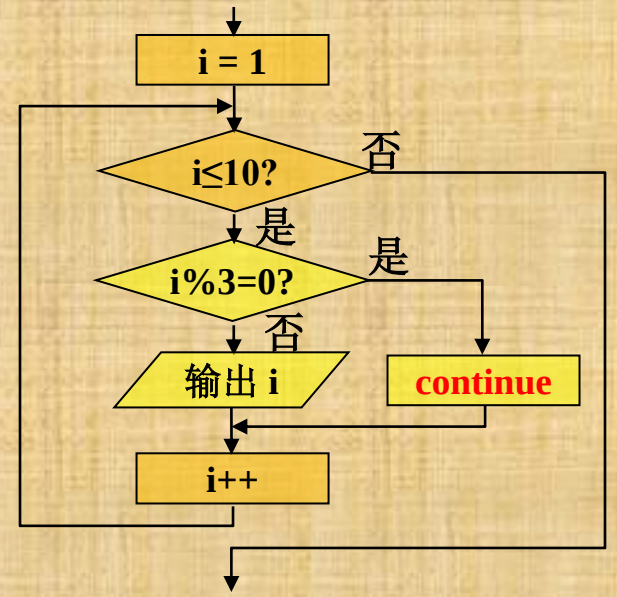
break使用在switch块内或循环体内

continue使用在循环体内

```
for( i=1; i<=10; i++)
{   if( i%3==0 )  break;
    cout<<i<<"\t";
}
```




```
for( i=1; i<=10; i++)
{   if( i%3==0 )  continue;
    cout<<i<<"\t";
}
```



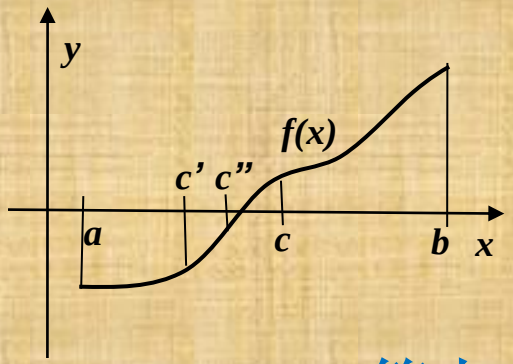
例B4.10 ○演示break和continue的作用。

```
for(i=1;i<=10;i++)
{       if(i%3==0)  break;    //break跳出循环
        cout<<i<<"\t";
}
for(i=1;i<=10;i++)
{       if(i%3==0)  continue;   //continue跳过本次循环
        cout<<i<<"\t";
}
for(i=1; i<=5; i++)
{       for(j=1; j<=10; j++)
        {       cout<< "("<<i<< ","<<j<< ")\t";
                if( (i+j)%3==0 )  break;  //break的跳转层数
        }                                  break只跳出1层
        cout<<endl;
}
```

例C4.11 输入一个正整数，判断其是否为素数。　　用试除法

```
for(sqrtn=sqrt((double)n),i=2; i<=sqrtn; i++ )   i取2,…,sqrt(n)试除
        if(n%i==0)  break;
if(n>=2&&i>sqrtn)  cout<<n<<"是素数"<<endl;
else                        cout<<n<<"不是素数"<<endl;
```

例C4.12 用二分法解方程：$\sin x-5+e^{2x-1}=0$ 。



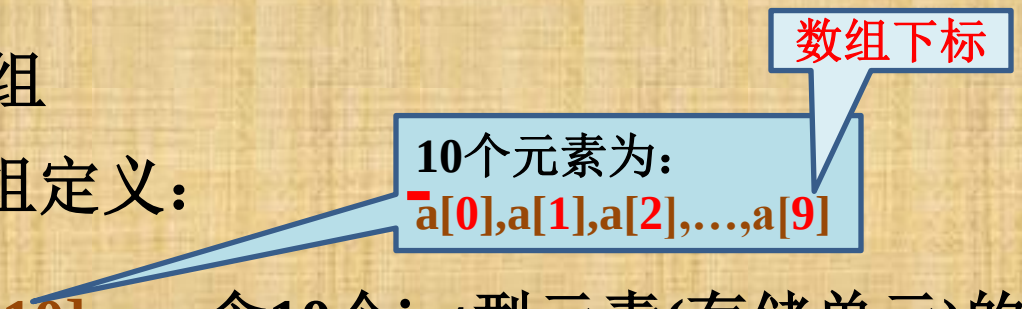

```
while( true )                构造循环
{   mid=(a+b)/2;   fmid=sin(mid)-5+exp(2*mid-1); 算中点
    if(fabs(fmid)<eps || fabs(b-a)<eps) break;  找到近似解
    else if(fa*fmid>0)   a=mid;        取右半区间
    else                 b=mid;        取左半区间
}
```

第5章 复合类型数据

5.1 数组

5.1.1 一维数组

一维数组定义：




```
int  a[10];     含10个int型元素(存储单元)的数组a
double  b[20];  含20个double型元素的数组b
char   c[80];   含80个char型元素的数组c
```

注1：C++规定，数组的大小必须是正的整数类型的常量表达式，一般是正整数常量。在程序中，数组大小通常取一个够用的大小值，如100、200、500等。

```
int  s1[2*10]; bool g['$'];    数组大小为常量，合法
const int size=100; float  ff[size]; 数组大小为常量，合法
int s2[ ];  int  n=100;  double d[n];  char ss[100.0];  float x[-20];
```

错误:数组大小不定、数组大小是变量、实数和负整数

例A5.1 输入10个数，求平均值(使用一维数组)。

int a[10];　　定义int型数组a

for(i=0;i<10;i++) cin>>a[i]; 输入10个数到a,下标i须整数

a的下标范围为0,1,…,9，
下标使用超界程序不报错

注2: 数组下标总是从0开始。数组下标如果超出实际下标的范围，则称为数组超界。超界使用数组时编译过程不会报错，甚至在运行时也可能不产生明显的错误而报错，但会产生难以预料的后果，因为超界使用数组可能改动数组之外的其它数据。

注3: 若有定义“int a[10];”，则a[10]是一个超界的元素，不表示数组。

一维数组定义时可以赋初值：

int a[5]={1,2,3,4,5}; 5个元素分别赋初值1,2,3,4,5

例A5.2 一维数组赋初值。

初值个数不能多于元素个数

double d[6]={0.1,0.2,0.3,0.4,0.5,0.6}; 6个元素赋上初值

double d1[ ]={0.1,0.2,0.3,0.4,0.5,0.6};初值个数确定元素个数

double d2[6]={1.11}; 初值赋值给前面元素，后面元素默认0

初值个数可以少于元素个数

double d3[4]; 未赋初值，元素值不确定

sizeof(d1)/sizeof(d1[0]); 表示数组元素个数

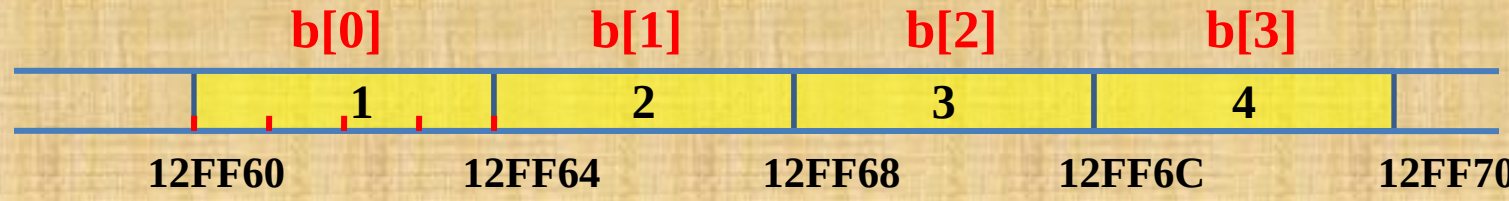
d1所占字节数

d1[0]所占字节数

注4: 初始值的个数始终不能超过数组元素的个数,否则会出错。

注5: 数组的处理需要扫描整个数组，一维数组扫描通常是循环“for(i=0;i<n;i++) 对i号元素进行处理”。

一维数组在内存中占一整块区域，数组元素按地址值从小到大依次存放，即：若 int b[4]={1,2,3,4}; 元素分布如下



例B5.3 ○演示数组的存储。

```
int a[4],b[4]={1,2,3,4};定义int型数组a和b
for(i=0;i<4;i++) cout<<&a[i]<<"\t";
cout<< "&a[0]="<<&a[0]<< ",a="<<a<<endl;
```

表示取a[i]的地址

a表示&a[0]

a[0],a[1],a[2],a[3]首字节地址为：

0106FB8C,0106FB90,0106FB94,0106FB98

&a[0]为0106FB8C，a也为0106FB8C

注6：数组的输入/输出必须用循环逐个元素输入/输出，不能直接用>>输入数组。若有定义“int a[10];”，则语句“cin>>a;”出错，而语句“cout<<a<<endl;”不是输出整个数组，而是输出数组a的首地址，即&a[0]的值。

例C5.4 求最大、最小值及最大、最小值的个数。

```
for(max=min=d[0],i=0;i<n;i++)
        if(d[i]>max)       max=d[i];  max记录更大的数
        else if(d[i]<min)  min=d[i];  min记录更小的数

for(numMax=numMin=i=0;i<n;i++)
{       if(d[i]==max)  numMax++; numMax统计max个数
        if(d[i]==min)   numMin++; numMin统计min个数
}
```

5.1.2 多维数组

多维数组定义：

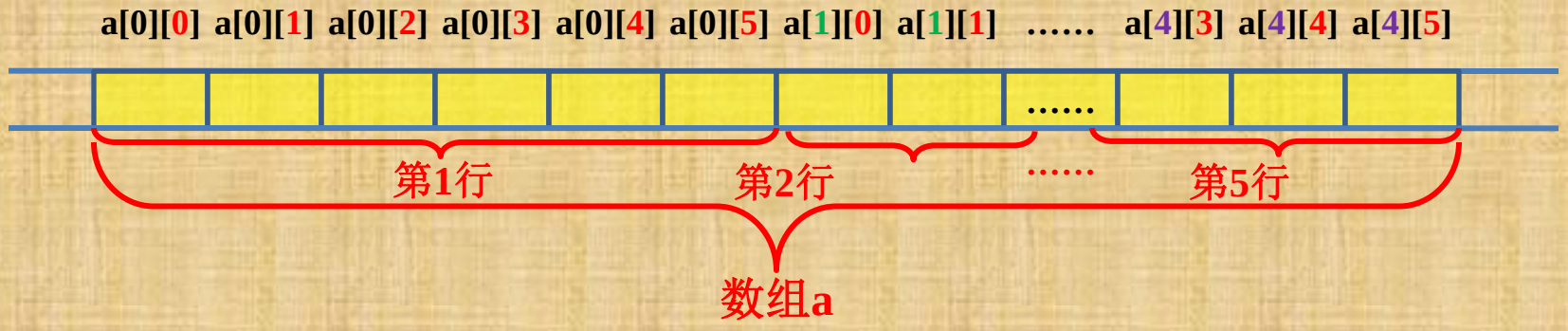
二维数组　int a[5][6];　含5行6列int型元素的数组a

5行

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]	a[2][5]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]	a[3][5]
a[4][0]	a[4][1]	a[4][2]	a[4][3]	a[4][4]	a[4][5]

6列

二维数组在内存中按行存放



3维数组 double b[2][3][4]; 含2页3行4列double型元素的数组b

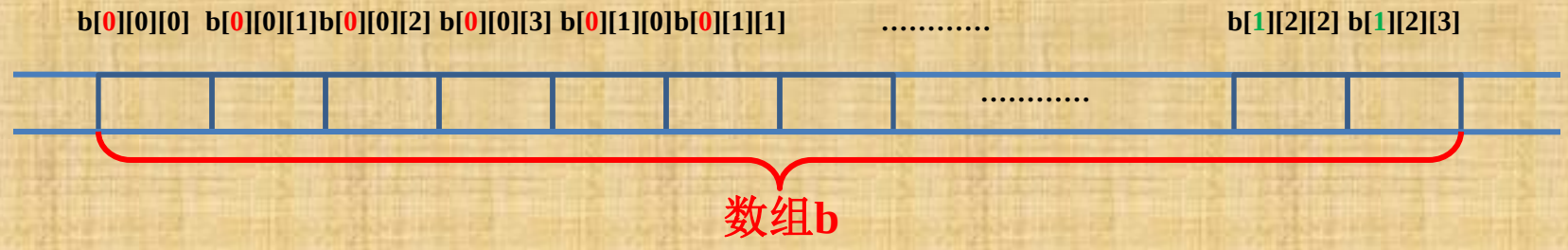
0页

b[0][0][0]	b[0][0][1]	b[0][0][2]	b[0][0][3]
b[0][1][0]	b[0][1][1]	b[0][1][2]	b[0][1][3]
b[0][2][0]	b[0][2][1]	b[0][2][2]	b[0][2][3]

1页

b[1][0][0]	b[1][0][1]	b[1][0][2]	b[1][0][3]
b[1][1][0]	b[1][1][1]	b[1][1][2]	b[1][1][3]
b[1][2][0]	b[1][2][1]	b[1][2][2]	b[1][2][3]

3维数组在内存中先第1维、再第2维、再第3维存放



注7：多维数组的每一维的下标都是从0开始。

多维数组赋初值(以二维数组为例)：两种形式

按行形式：

```
int a[4][4]={{1,2,3,4},{5,6},{9},{0}};
```

每行不足补0

按整体形式：

```
int a[4][4]={1,2,3,4,5,6,9};
```

按内存存放次序,不足补0

例C5.5 二维数组赋初值及其使用：计算矩阵乘列向量y=Ax。

```
int A[4][4]={{1,2,3,4},{5,6,7,8},{1,1,1,1},{2,1,0,-1}};
int y[4],x[4]={1,2,-1,1};

for(i=0;i<4;i++)    双重循环扫描二维数组，进行输出
{       for(j=0;j<4;j++)  cout<<A[i][j]<<"\t";
        cout<<endl;
}
for(i=0;i<4;i++)      每一行计算一个y[i]
        for(y[i]=0,j=0;j<4;j++)  y[i]+=A[i][j]*x[j];
```

例C5.6 输入两个4 × 4矩阵，求它们的乘积。

```
for(i=0;i<4;i++) for(j=0;j<4;j++) cin>>a[i][j];     输入a
for(i=0;i<4;i++) for(j=0;j<4;j++) cin>>b[i][j];     输入b

for(i=0;i<4;i++) for(j=0;j<4;j++)
{       s=0;
        for(k=0;k<4;k++) s+=a[i][k]*b[k][j];    计算c[i][j]
        c[i][j]=s;
}
```

例B5.7 ○二维数组中行的演示。　　进一步说明见下页

```
int a0[3]={1,2,3};    int a[3][3]={{1,2,3},{4,5,6},{7,8,9}};
```

a[0]、a[1]、a[2]类似于a0，是一维数组，
a[0]、a[1]、a[2]相当于一维数组名，表示数组首地址
a可看成{a[0],a[1],a[2]}三个行构成的一维数组

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

3行
相当于
3个一维数组

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

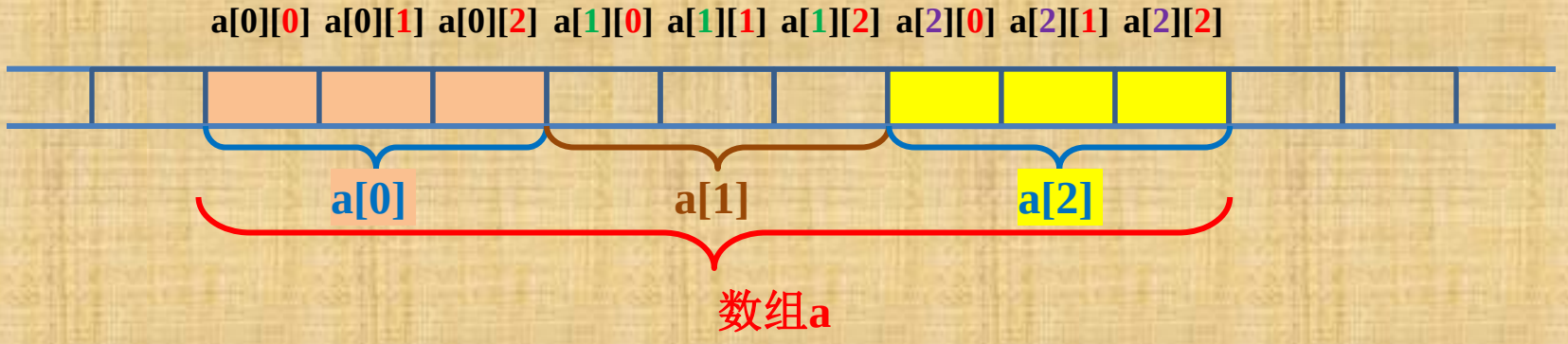
一维数组a[0]

sizeof(a0) 3*4=12

sizeof(a[0])=sizeof(a[1])=sizeof(a[2]) 3*4=12

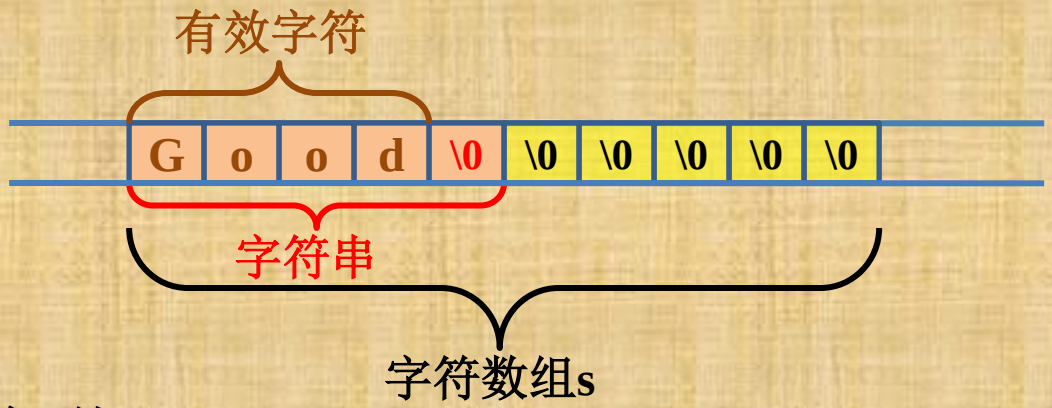
sizeof(a[0][0])=sizeof(a[0][1]) 4

二维数组(或一维行数组)的内存分布



5.1.3 字符数组及字符串处理

字符数组用于存放和处理字符串。字符串由一系列字符构成并以零ASCII码字符 '\0'作为字符串结束标志，称为C风格字符串。 '\0'前的字符称为有效字符，有效字符个数为字符串长度。C++标准提倡使用专门的字符串类型string，在12.6小节中将作说明。



字符数组赋初值：

```
char  s1[10]={'H','e','l', 'l','o'}; 不足补 '\0',即0
char  s2[ ]={'H','e','l', 'l','o','\0'};
char  s3[ ]= "Hello";      常用形式
```

最后的初值为串结束标志'\0'

字符串的输入和输出：

```
cin>>s1;      cout<<s1<<endl;
```

字符串的处理：逐个字符处理，串结束标志控制循环

```
for(int i=0;s3[i]!='\0';i++)  cout<<s3[i];
for(int j=0;s3[j];j++)  s3[j]++;
```

例A5.8 求字符串长度。

```
char  s[ ]="This is a book";    字符数组赋初值
for(i=0;s[i]!='\0';i++) ;      字符串处理：求长度
```

注8：循环处理字符串时通常用循环 “for(i=0;s[i]!='\0';i++) …”，而不用循环 “for(i=0;i<n;i++) …”

字符串的输入：　　若 char s[80];

- 方式1　　cin>>s; 输入单词(空格、回车、TAB结束输入)
- 方式2　　cin.getline(s,80); 输入一行(仅回车结束输入)

s提供的存放字符串的最大空间

注9：字符串直接用<<输出，直接用>>输入单词，用cin.getline输入一行。只在对字符串进行其它处理时用循环。

例C5.9 输入一行,统计单词的个数(非字母字符为单词间隔符)。

```
count=0; i=0;
while(s[i])
{  if(!(s[i]>='A'&&s[i]<='Z' || s[i]>='a'&&s[i]<='z')) i++;
   else
   { count++;
     while(s[i]>='A'&&s[i]<='Z'||s[i]>='a'&&s[i]<='z') i++; }
}
```

当前字符s[i]为间隔字符

刚遇到新的单词，统计单词数

跳过单词

注10：在定义字符数组时，通常取数组大小为80。因为程序运行的窗口每行可以显示80个字符，故80个字符大小的字符数组差不多可以存放一行字符串(最多79个有效字符)。

改动字符串的例

例C5.10 输入一行，删除字符串中的空格。

```
for(j=i=0;s[i];i++)     扫描字符串s
        if(s[i]!=' ')  s[j++]=s[i];    若非空格字符则复制
s[j]='\0';    新串后面加上串结束标志
```

主要片段的改动：　省略附加串结束标志

```
for(j=i=0;s[j]=s[i];i++)
        if(s[j]!=' ') j++;
```

注11：二维字符数组可以保存多个字符串，每一行表示一个字符串。如定义　char ss[3][80]={"Apple", "Pear", "Banana"};则ss[0]、ss[1]、ss[2]就表示3个字符串"Apple"、"Pear"、"Banana"。

5.1.4 字符串处理函数

常用的字符串处理库函数：

strlen(s)	求字符串长度
strcpy(s1,s2)	将s2复制到s1中
strcmp(s1,s2)	比较s1和s2，函数值为1,0,-1
strcat(s1,s2)	将s2合并到s1后面
strlwr(s)	将s中大写字母改成小写
strupr(s)	将s中小写字母改成大写
strstr(s1,s2)	在s1中查找子串s2
itoa(n,s,radix)	将n的值转换成radix进制的数串
atoi(s)	将十进制数串s转成int型的数

注12：库函数要合理使用，如下代码设计有差异。

```
for(i=0;i<strlen(s);i++) 处理s[i] (反复调用strlen(s),代码差)
for(n=strlen(s),i=0;i<n;i++) 处理s[i]    (代码较好)
for(i=0;s[i]!='\0';i++)  处理s[i]     (代码很好)
```

新版软件的安全版　strcpy_s(s1,len,s2) 将s2复制到s1中，s1提供len个字节空间

复制、合并库函数：strcat(s1,len,s2) 将s2合并到s1后面，s1提供len个字节空间

例C5.11 反复输入字符串直到输入^Z，求出最大、最长的单词并输出。

```
cin>>s;   输入单词
strcpy(sMax,s); strcpy(sLong,s); sMax,sLong取初始值
while(cin>>s)  反复输入单词，直到输入^Z
{       if(strcmp(s,sMax)>0)  strcpy(sMax,s); s更大取s
        if(strlen(s)>strlen(sLong))
                strcpy(sLong,s);                s更长取s
}

strcpy(s,"最大字符串\""); strcat(s,sMax);
strcat(s,"\"\t最长字符串\""); strcat(s,sLong);
strcat(s,"\"");                不同词汇合并成一个字符串
```

5.1.5 常用数组算法

数组的有序插入和删除

例C5.12 在升序数组中有序插入一个实数。

```
for(i=n-1;i>=0&&x<d[i];i--) d[i+1]=d[i];  d[i]>x则后移
d[i+1]=x;    n++   应该插入在d[i]和d[i+1]之间
```

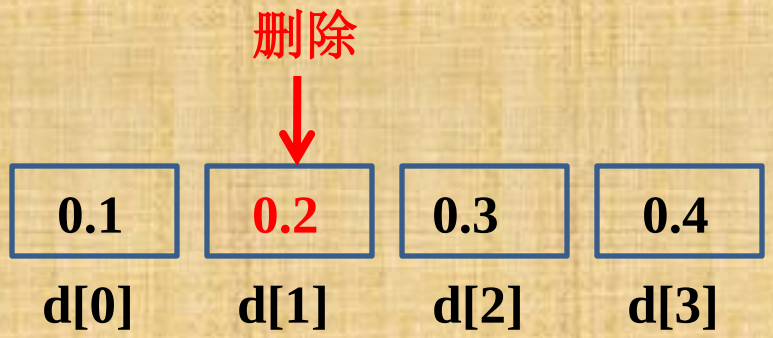
例C5.13 在数组中删除某个值。

```
for(i=k-1;i<n-1;i++)  d[i]=d[i+1];
```

从d[k-1]开始后一数覆盖前一个数

```
n--;
```

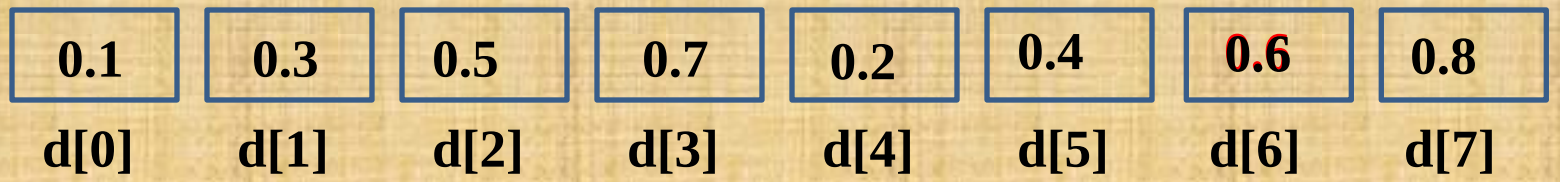
删除数据后，数据个数减去1



查找

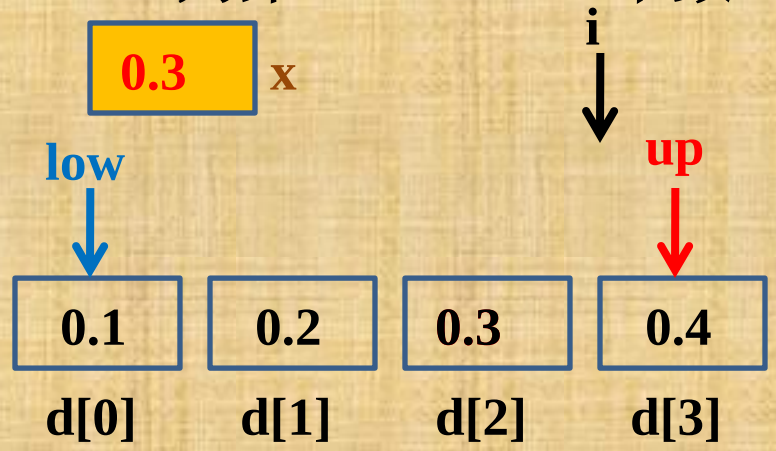
例C5.14 顺序查找。

```
for(i=0;i<n;i++)
    if(d[i]==x) break;  d[i]就是要找的数，结束循环

if(i<n)  cout<<"为第"<<i+1<<"个数"<<endl;
else     cout<<"未找到"<<endl;   没找到时，有i ≥ n
```



例C5.15 折半查找。　　升序数组

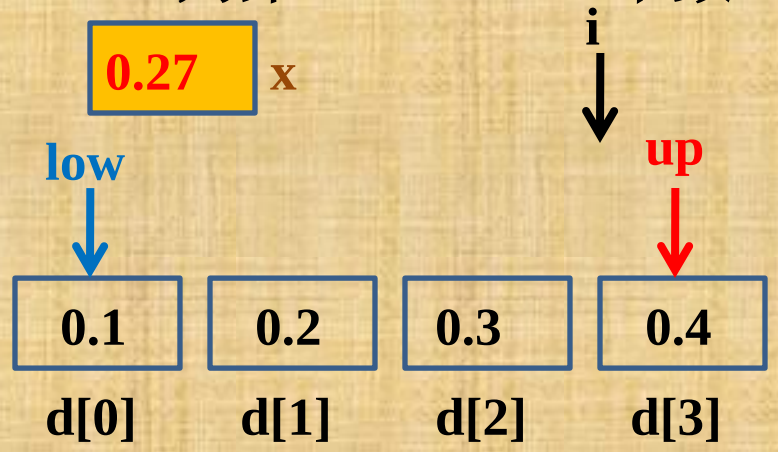
```
for(low=0,up=n-1;low<=up;  ) 在[low,up]中查找,直到空
{     i=(low+up)/2; if(d[i]==x)  break;  i取中间,找到跳出
      if(x<d[i])  up=i-1;       待查数在d[i]前
      else        low=i+1;      待查数在d[i]后
}

if(low>up)  cout<<"未找到"<<endl;   [low,up]为空,失败
else cout<<"为第"<<i+1<<"个数"<<endl;      i找到
```



注13：在上述两个查找的例子中，为了方便起见,数据相等比较用“= =”，也可以使用实数相等比较的方式“fabs(d[i]-x)<eps”。

例C5.15 折半查找。　　升序数组

```
for(low=0,up=n-1;low<=up;  ) 在[low,up]中查找,直到空
{     i=(low+up)/2; if(d[i]==x)  break;  i取中间,找到跳出
      if(x<d[i])  up=i-1;       待查数在d[i]前
      else        low=i+1;      待查数在d[i]后
}

if(low>up)  cout<<"未找到"<<endl;   [low,up]为空,失败
else cout<<"为第"<<i+1<<"个数"<<endl;      i找到
```

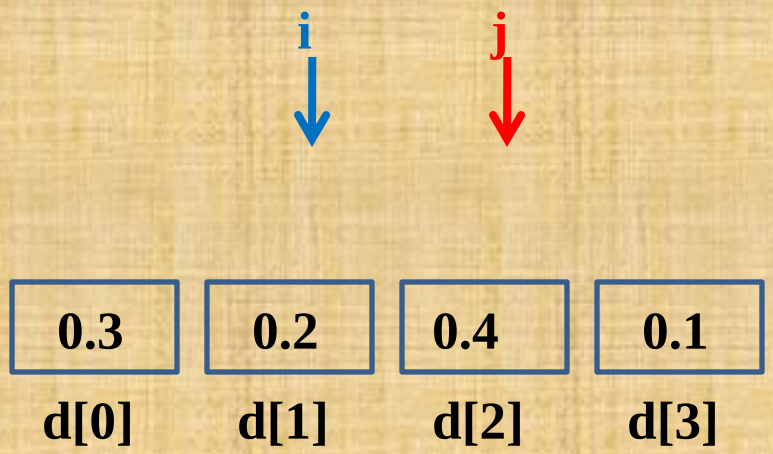


注13：在上述两个查找的例子中，为了方便起见,数据相等比较用“= =”，也可以使用实数相等比较的方式“fabs(d[i]-x)<eps”。

排序

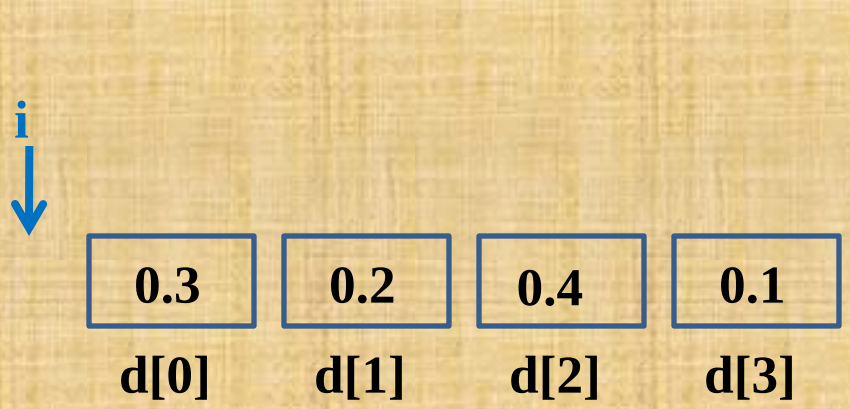
例C5.16 用选择法排序。　　升序排序

```
for(i=0;i<n;i++)
{  每次找d[i],d[i+1],…,d[n-1]中最小的数d[k]与d[i]交换
   for(k=i,j=i+1;j<n;j++)  if(d[j]<d[k]) k=j;   d[k]是最小值
   if(k!=i)  {  t=d[k]; d[k]=d[i]; d[i]=t;  }      d[k],d[i]交换
}
```



例C5.17 用冒泡法排序。　　升序排序

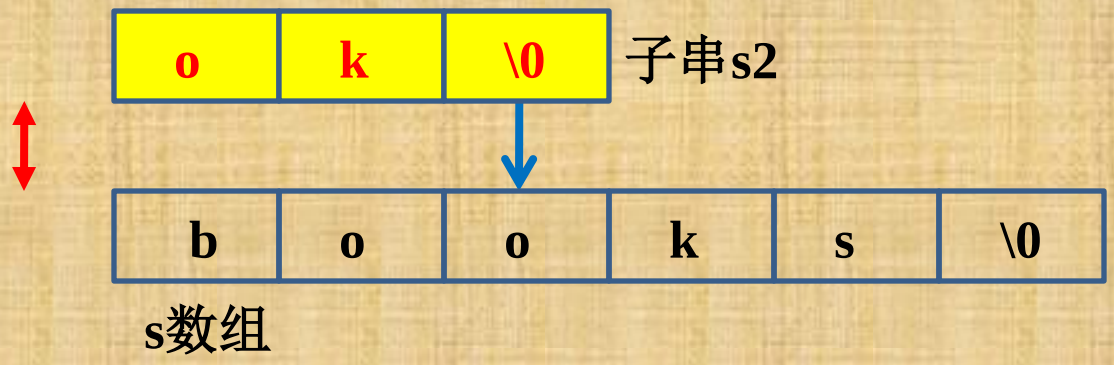
```
for(k=n-1;k>0;k--)
{  每次扫描d[0],d[1],…,d[k]，两两顺次序
   for(i=0;i<k;i++)
      if(d[i]>d[i+1])  若d[i],d[i+1]不是升序，则交换
      { t=d[i]; d[i]=d[i+1]; d[i+1]=t; }
}
```



子串匹配

例C5.18 子串匹配。

```
for(i=0;s[i];i++)
{   对字符串s逐个位置探测是否含子串s2
    for(j=0;s2[j]&&s[i+j]==s2[j];j++) ; 从s[i]开始对照子串
    if(s2[j]=='\0')  break;   子串s2的字符都对照成功,匹配
}

if(s[i]) cout<<"第"<<i+1<<"个字符处匹配"<<endl;
else   cout<<"匹配失败"<<endl;  串s扫描完,则子串未匹配
```



子串匹配

例C5.18 子串匹配。

```
for(i=0;s[i];i++)
{   对字符串s逐个位置探测是否含子串s2
    for(j=0;s2[j]&&s[i+j]==s2[j];j++) ; 从s[i]开始对照子串
    if(s2[j]=='\0')  break;   子串s2的字符都对照成功,匹配
}

if(s[i]) cout<<"第"<<i+1<<"个字符处匹配"<<endl;
else   cout<<"匹配失败"<<endl;  串s扫描完,则子串未匹配
```

5.2 指针

指针就是**地址**

为什么要使用指针：

- 指针反映了真实的机器指令的操作
- 指针可以使我们使用无名的变量

机器指令使用地址

```
static int  a,b,c;
a=5;
b=7;
c=a+b;
```

机器指令

```
[002D714C]⇐5
[002D7148] ⇐ 7
EAX ⇐ [002D714C]
EAX ⇐ EAX+[002D7148]
[002D7150] ⇐ EAX
```

指针使用无名变量

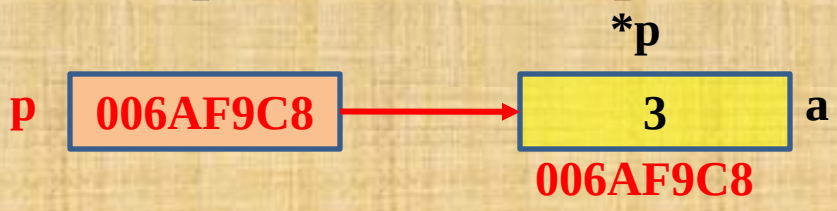
```
int  *p,i;
p=new int;
*p=12037;  //*p为无名变量
for(i=2;*p%i;i++);
if(i==*p)  cout<<"素数";
```

p保存无名变量的地址

C++使用指针：

```
int  a;
int  *p;
p=&a;
*p=3;
```

图示：称p指向变量a，*p就是a



注1：指针变量p保存了某个变量a的地址(即变量a的指针)后称为指针变量p指向变量a，于是*p就是p指向的变量a。

注2：指针变量p没有指向任何变量时称为野指针，不能使用*p。

注3：在定义指针变量时每个变量前加*，表示该变量是指针变量，否则不是。例如：

```
int a,*p=&a,*q,b;
```

p、q是指针变量，而a、b只是int型变量，并且初值&a是赋给p的，不是赋给*p的。

例A5.19 指针的定义与使用。

int a=1,b=2,c=3; int *pa,*pb,*pc; pa,pb,pc为指针变量

pa=&a; pb=&b; pc=&c; pa,pb,pc指向a,b,c

*pa=3; *pb=5; *pc=*pa+*pb; 就是a=3;b=5;c=a+b;

double d=3.14,*pd=&d,x=2.71828; *pd就是d

pd=&x; 此时,*pd就是x

指针变量赋初值

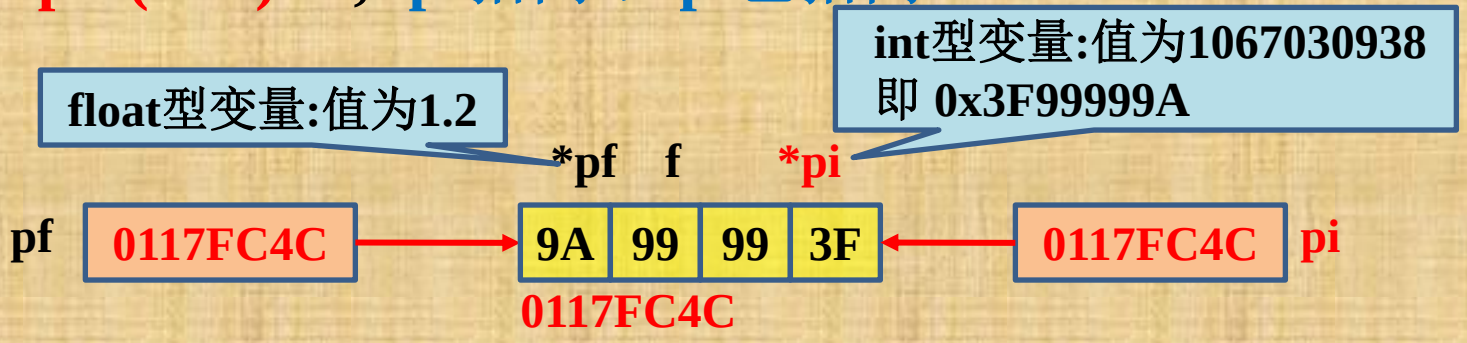
*指针变量赋初值：

double *pd=&d; 等价于 double* pd=&d;

注4: 指针变量的指向类型(即指针指向的变量类型)不同，使用结果也不同。

float f=1.2,*pf; int *pi;

pf=&f; pi=(int*)&f; pf指向f，pi也指向f



指针的算术运算

指针可以进行算术加减运算，运算时以指向的存储单元为单位。

- 指针加1表示地址增加到下1个单元的地址
- 指针减1表示地址减小到前一个单元的地址

若有定义

```
int  a[10]={0,10,20,30,40,50,60,70,80,90};
```

指针加减整数

&a[1]+3　　表示a[1]后面第3个元素a[1+3]的指针&a[4]

&a[5]-2　　表示a[5]前面第2个元素a[5-2]的指针&a[3]

&a[5]-&a[3]　表示a[5]与a[3]之间相差5-3个(即2个)元素

指针减指针

例B5.20 演示指针运算。

int a[10]={0,10,20,30,40,50,60,70,80,90},*pa,*pb;

&a[0]、&a[0]+0、a+0、a 等价，因为a表示&a[0]

&a[0]+1、&a[0+1]、&a[1]、a+1等价

(&a[0]+1)、a[1]、(a+1)等价

&a[5]-&a[3]、(a+5)-(a+3)、5-3等价

pa=a; pa++; pb=pa+3;

pa=a;即pa=&a[0]; pa指向a[0]。

pa++;后pa指向a[1]，pb=pa+3;pb指向a[1+3]即a[4]

pa=a;

即pa=&a[0]，于是a数组就是pa数组，并有关系

pa=a≡&a[0],pa+4=a+4≡&a[4],pa+3=a+3≡&a[3]

pa[0]≡*pa=*a≡a[0],pa[4]≡*(pa+4)=*(a+4)≡a[4]

注5：数组名表示数组，但在使用中除了sizeof使用和取地址&使用以外，其它地方都表示数组的首地址。

注6：若指针变量pa指向某个位置，从这个位置开始的一段空间也可看成数组pa。

注7：数组及指针有以下等价写法 (若定义 “int a[10],*p=a;”):

a+i≡&a[i] , *(a+i)≡a[i] ； p+i≡&p[i] , *(p+i)≡p[i]

其中，≡表示等价，等价的表达式编译后代码完全一样。

指针更多地用于处理数组，尤其是对数组的扫描处理，

常用数组扫描方式：　　若double a[100],*p; int i,n;

下标方式　　**for(i=0;i<n;i++) 处理a[i] (等价于*(a+i))**

下标指针方式　**for(p=a,i=0;i<n;i++,p++) 处理*p**

纯指针方式　　**for(p=a;p<a+n;p++) 处理*p**

例C5.21 指针用于数组颠倒次序。

```
double a[100],t,*p,*q;
for(p=a;p<a+n;p++)  cin>>*p;
for(p=a,q=a+n-1;p<q;p++,q--)
{   t=*p;  *p=*q;  *q=t;  }
for(p=a;p<a+n;p++)  cout<<*p<<"\t";
```

指针输入数组元素

指针颠倒数组元素

交换p和q指向的元素值

指针输出数组

注8：数组元素的下标可以看成是某种相对指针，即相对于数组首地址的相对指针，而本节介绍的指针则是绝对指针。从相对指针和绝对指针的角度看数组的处理可以较好地理解指针的用法，也可以很容易地将数组的下标方式处理代码改写成指针方式处理代码。

上述程序改成下标形式的主要程序片段：

```
int  i,j;
for(i=0;i<n;i++) cin>>a[i];
for(i=0,j=n-1;i<j;i++,j--) { t=a[i];a[i]=a[j];a[j]=t; }
for(i=0;i<n;i++) cout<<a[i]<< "\t";  cout<<endl;
```

将相对指针(即下标)改成绝对指针：

```
double   *i, *j;
for(i=a;i<a+n;i++) cin>>*i;
for(i=a,j=a+n-1;i<j;i++,j--) { t=*i; *i=*j; *j=t; }
for(i=a;i<a+n;i++) cout<<*i<< "\t";  cout<<endl;
```

例C5.22 将一个十进制数串转换成有符号整数，并计算平方。

```
char s[80],*p;  int y,sign=1;
if(*p== '+'||*p== '-') { if(*p++== '-') sign=-1; } 确定符号
for(y=0;*p>= '0'&&*p<= '9';p++)    扫描数串
        y=y*10+(*p- '0');     *p转成数字添加到y后面
y=y*sign;     y保存的值中加入符号
```

注9：用<<输出字符变量的地址或字符数组的地址将输出从该地址开始的一个字符串，若要输出字符类型的地址，必须将字符类型的地址显式转换成其它类型的地址。若有定义“char s[ ]= "Hello";”，则执行语句“cout<<&s[3]<<endl;”将输出字符串“lo”，只有执行语句“cout<<(int*)&s[3]<<endl;”才输出s[3]的地址——&s[3]的值0092FC5C。

指针可用于使用无名变量，使用如下：

```
int  *p;
p=new int;      此时*p就是无名变量，实际上是用new
                生成的变量，无名，只有一个地址
p=new int(3);   p指向一个动态变量，初始值为3
delete p;       释放p指向的动态变量

p=new int[5];  表示p指向一个无名数组，含5个int元素
delete[ ] p;    释放p指向的动态数组
```

称为动态变量

称为动态数组

程序中有一块区域专门用于动态地提供变量或数组的存储空间的，称为堆。堆中的空间即动态空间需要用new来分配。由于空间有限，为了后续使用动态空间的顺利，在不用动态变量或动态数组时要用delete来释放占用的空间，保证堆中有足够的空闲动态空间。

例A5.23 用new和delete使用动态变量和动态数组。

```
double *pd;   int  *p,n,i;
p=new int;          p指向new开辟的动态变量
*p=20;              给动态变量*p赋值20
delete p;           用delete释放动态变量
p=new int[n];   p指向new开辟的动态数组，看成p数组
for(i=0;i<n;i++)  p[i]=rand( );  数组元素p[i]赋值随机数
for(i=0;i<n;i++) cout<<p[i]<< "\t";   输出动态数组
pd=new double(3.14);   pd指向动态变量，初值为3.14
cout<<*pd<<endl;        输出动态变量*pd的值
delete pd;              释放pd使用的动态变量占用的空间
delete[ ] p;            释放p使用的动态数组占用的空间
```

随机数函数

若new开辟动态空间失败，VS2008将产生一个异常，若不处理将提前结束程序。异常处理可如下：

将语句　　p=new int[n];

改为

```
try { p=new int[n]; }
catch (…)  处理异常的catch块首部
{    cout<<"动态空间开辟失败"<<endl;  catch块内处理异常
     return 1;
}
```

注10：0指针是一个特别的指针值，表示不指向任何变量。0可以是任何指向类型。

注11：使用0指针指向的变量是非法的，使用野指针指向的变量有严重隐患会导致不可预料的后果。

5.3 引用

引用就是取别名

定义一个变量的引用就是给这个变量再取一个变量名。

例B5.24 演示引用。

```
int  a=3;
int  &a2=a;
a2=5;
```

a2是a的引用

a2,a是同一个变量的两个名字

变量a2(即变量a)赋值5

```
double d,&d2=d,&d3=d2;
d2=10.5;
```

d,d2,d3是同一个变量的3个名字

变量d2(即变量d,d3)赋值10.5

图示如右：

5.4 结构与联合　只讲结构内容

5.4.1 结构

考虑复合型数据：人员基本数据：姓名、性别、出生年月日

可通过结构类型表示，该类型定义如下：

```
struct person
{
        char  name[20];  //保存姓名
        char  sex[4];    //保存性别：男或女
        int  year, month, day; //保存出生年月日
};
```

结构类型名：person

结构成员名：name,sex,year,month,day

于是就有了新的类型person，这是一种结构类型。

结构类型person的使用：

```
person a1={"张三", "男", 1975,5,21};  结构变量赋初值
person a2;                            定义结构变量
a2=a1;                                结构变量赋值
person a3={"李四", "女"};             初值个数不足，则补0
```

注1: 结构类型的定义不占用存储空间，所以不能在结构类型定义中赋初值。

例如:

```
struct person
{       char  name[20],sex[4];
        int year=2000,month=1,day=1;  //不能在类型定义中赋初值
};
```

这是错误的。结构类型变量的定义需要分配存储空间，此时可以给变量赋初值。**新的C++标准允许这样赋初值。**

注2: 结构类型变量中的各个成员在内存中按定义次序占据内存空间。

注3: 结构变量除了变量之间可以赋值以外，其它时候(包括输入和输出)只能使用变量的各个成员，不能直接使用变量。

结构变量正确的使用：

```
cin>>a1.name>>a1.sex>>a1.year>>a1.month>>a1.day;
cout<<a1.year<<"-"<<a1.month<<"-"<<a1.day<<endl;
strcpy(a2.name, "王五");  strcpy(a2.sex, "男");
a2.year=1976;  a2.month=10;  a2.day=31;
```

错误使用：

```
cin>>a1;  cout<<a1;  a2={"王五", "男", 1976,10,31};
```

例A5.25 结构变量赋初值及使用。

```
struct point
{   double  x,y;  };
point  c1={10,5},c2;
cin>>c2.x>>c2.y;
cout<<"c1=("<<c1.x<<","<<c1.y<<")"<<endl;
c2=c1;
c2.x=c1.x+10;  c2.y=c1.y+8;
```

结构类型定义

结构变量定义及赋初值

使用结构变量成员

直接使用结构变量

结构变量定义的其它形式：

- struct point2
  { double x,y; } c3={3,7},c4;

  类型、变量同时定义

- struct
  { double x,y; } c5={0.3,-1.2},c6;

  无类型名

  类型一次性使用

例C5.26 用结构数组处理学生的数学、语文和英语成绩，对平均分降序排序。

```
struct  scoreRecord
{   char name[10];  int mathScore,chineseScore,englishScore;
    double  averageScore;   };
scoreRecord  student[200],t;

while(cin>>t.name)
{  cin>>t.mathScore>>t.chineseScore>>t.englishScore;
   …….
   student[n++]=t;   }

for(i=0;i<n;i++)
{   for(k=i,j=i+1;j<n;j++)
    if(student[j].averageScore>student[k].averageScore)k=j;
    if(k!=i) { t=student[k]; student[k]=student[i];student[i]=t;}
}
```

更复杂的结构类型的定义和使用　　成员是结构类型

```
struct date
{    int  year , month , day;  };

struct person2
{    char  name[20] , sex[4];
     date  birthday;  //birthday成员类型是结构类型date
};

person2  b1={"张三", "男", { 1975,5,21} };

cout<<b1.birthday.year<<"年"<<b1.birthday.month
      << "月"<<b1.birthday.day<< "日\n";
```

5.5 枚举

枚举类型是一类用名字作为数据的类型，用多少名字做数据，哪些名字做数据由编程者通过定义来确定。

定义枚举类型：

枚举类型名　　枚举值(枚举常数),前小后大

```
enum color { red , green , blue };
```

上述定义了枚举类型color，可以定义和使用color类型变量

```
color  i,j=green,k;   定义color型变量i,j,k, 且j初值为green
i=red;    变量i赋值为red
if(i<blue)  cout<<"i值为red或green"<<endl;
```

color型数据比较，枚举值表中前面值小后面值大

枚举类型数据在内部表示为一个整数，即枚举类型变量的值是一个整数值。枚举值与整数值的对应关系由类型定义时确定。如定义

```
enum color { red , green , blue };
```

中，枚举值表{red,green,blue}对应于内部数据0,1,2。

枚举类型定义中可以指定枚举值与整数值的对应关系。如下列定义：

自动对应0　　自动对应加1后的数

```
enum color2 { black , blue , green , red=4 , magenta };
```

为{black,blue,green,red,maganta}对应{ 0,1,2, 4,5 }

```
enum adjust {left=0,center=1,right=2,bottom=0,top=2};
```

为 {left,center,right,bottom,top}对应{0,1,2,0,2}

```
enum comparison { Less= -1 , Equal , Greater };
```

为{ Less,Equal,Greater }对应 { -1,0,1 }

注1：不能输入枚举类型的数据，但可以输出，输出的是枚举值对应的内部值。

枚举类型数据可以自动转换成int型

如　　int m=red,n; n=j;

例C5.30 枚举类型的使用：输出红、绿、蓝3种颜色的全排列。

```
enum color { red , green , blue };    枚举类型定义
char ss[3][4]={"红\t", "绿\t", "蓝\t" };
        定义二维字符数组，初值为三行串:红、绿、蓝
color  i , j , k ;   定义枚举类型变量
for(i=red;i<=blue;i=(color)(i+1))
i循环取值red,green,blue，i+1为i自动转换成int型值, 然后
“该数”+1，得int型结果值，在赋值给i时要显式转换
if(i!=j&&i!=k&&j!=k)    表示i,j,k为不同值
        cout<<++n<<"\t"<<ss[i]<<ss[j]<<ss[k]<<endl;
```

i自动转换成int型，再取ss的行

注2: 对于枚举类型的变量i，表达式“i++”有错,因为i++等价于i=i+1,而这是将一个int型数赋值给一个枚举变量,是不被允许的。

5.6 ○用typedef定义类型

typedef可以给类型取其它名字，或组合出新的类型。

typedef定义新类型。

```
typedef   int   Integer ;        类型Integer即类型int
typedef   unsigned int     size_t ;
                    类型size_t即类型unsigned int
typedef   unsigned int     studentIDType ;
                    类型studentIDType即类型unsigned int
```

studentIDType也可以改成字符数组类型：

```
typedef   char  studentIDType[15];
                    studentIDType现在为char [15]
```

于是　studentIDType studentID;　等价于　char studentID[15];