



计算机程序设计与 Visual C++

第五、六章 指针与函数



本章主要内容

- C++中指针、指针变量的基本概念
- 指针与地址运算符
- 指针与一维数组
- 函数的值传递
- 递归函数

3.3.3 用户自定义函数 P₈₂

1. 函数的定义

```
函数类型  函数名(类型名 形式参数1, 类型名 形式参数2,...)
{
    说明部分    //要用到的变量、数组、指针及要调用的其它函数等说明
    语句部分    //实现某项特定功能的代码块
}
```

2. 函数的调用 P₈₆

函数的使用通过函数调用实现。 函数调用指定了被调用函数的名字和调用函数所需的参数。 函数调用的一般形式：

函数名(实参表);

p=fact(7);

3. 函数声明(原型) P₇₉

函数和变量一样，在使用之前要先声明。函数声明的一般形式：

函数类型 函数名(类型名 形式参数1, 类型名 形式参数2,...) ;

```

#include <iostream>
using namespace std;
void main()
{
    long fact(int);          /*函数声明*/
    int n; long p;
    cout<<"Please input the n:";
    cin>>n;
    p=fact(n);               /*函数调用*/
    cout<<"The n! is: " <<p<<endl;
}
long fact(int m)            /*函数定义*/
{ int i; long s=1;
  for(i=1;i<=m;i++)
    s*=i;
  return(s);                /*函数返回*/
}

```

结论: 被调函数在后, 需在主调函数中先声明后调用。

```
#include <iostream>
using namespace std;
long fact( int m)    /* 函数定义*/
{ int i; long s=1;
  for(i=1;i<=m;i++)
    s*=i;
  return(s);
}
```

```
void main()
{
  int n; long p;    /*不需函数声明*/
  cout<<"Please input the n:";
  cin>>n;
  p=fact(n);        /*函数调用*/
  cout<<"The n! is: "<<p<<endl;
}
```

结论：被调函数先于主调函数被编译，因此在编译主调函数时已知被调函数的类型等信息。故不需函数声明。

问题：如何定义一个函数

- 第一步：分析函数需要的参数，包括参数的个数以及每个参数的类型。
- 第二步：分析函数返回值的类型，若无返回值，则为void。函数的返回值可看作是函数执行完后需输出的一个数据。
- 第三步：编写函数体（编写功能代码）。
- 求n!

要处理的数据是n，因此必须有一个参数n，类型为int。返回值为int型。即

```
int fact( int n)
{
.....
}
```

```
int fact(int n)
{
    int t=1,i=1;
    while(i<=n)
    {
        t=t*i;
        i=i+1;
    }
    return t;
}
```

```
#include <iostream>
using namespace std;
void main( )
```

```
{
    int fact(int n);                //函数声明
```

```
    int n,m;
```

```
    int y;
```

```
    cout<<"Enter the positive integer m and n(m>n):";
```

```
    cin>>m>>n;
```

```
    y=fact(m)/fact(m-n);           //函数调用
```

```
    cout<<m<<"!/("<<m<<"-"<<n<<"")!="<<y<<endl;
```

```
}
```

```
int fact(int n)
{
    int t=1,i=1;
    while(i<=n)
    {
        t=t*i;
        i=i+1;
    }
    return t;
}
```

输入m, n

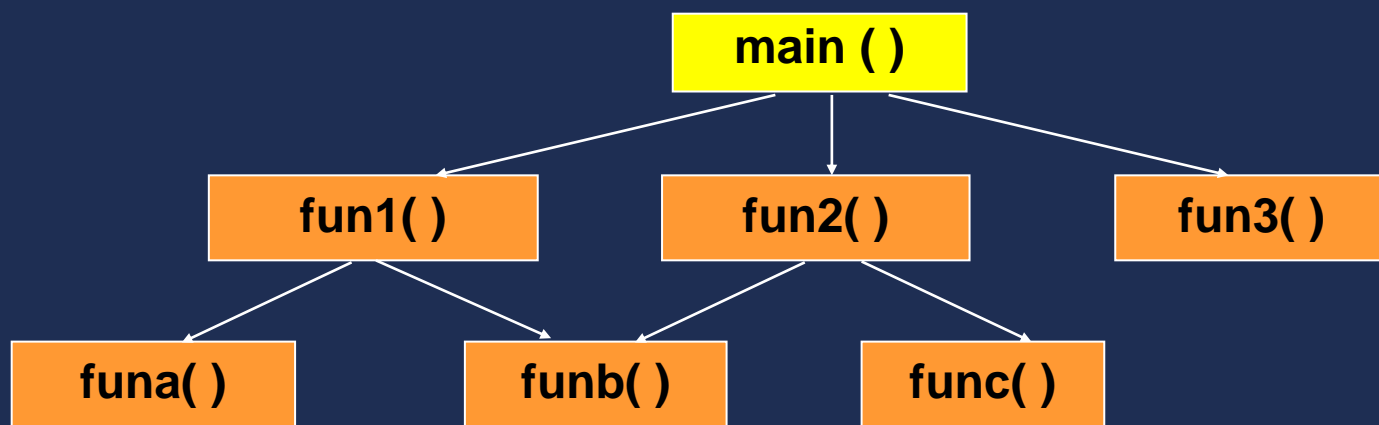
调用子函数求
m!和(m-n)!

输出m!/(m-n)!

主函数流程图

函数的调用层次关系

- 组成C++程序的若干函数中，有且仅有一个称为主函数的main()函数，它是程序执行的入口，主函数可以调用任何子函数，子函数不能调用主函数；
- 子函数可以调用任何子函数，当子函数调用它自身时，称为直接递归调用；
- 函数不能嵌套定义，即不能在一个函数中再定义另一个函数。



函数调用层次关系

6.3 函数的递归调用 P₂₃₄

递归函数即自调用函数，在函数体内部直接或间接地自己调用自己。

递归调用有直接递归调用和间接递归调用两种形式。

- 直接递归即在函数中出现调用函数本身。
- 间接递归调用是指函数中调用了其他函数，而在其他函数却又调用了本函数。

6.5 递归函数

P₂₃₆

●递归过程的两个阶段：

➤递推：

$$4!=4 \times 3! \rightarrow 3!=3 \times 2! \rightarrow 2!=2 \times 1! \rightarrow 1!=1 \times 0! \rightarrow 0!=1$$

未知

已知

➤回归：

$$4!=4 \times 3!=24 \leftarrow 3!=3 \times 2!=6 \leftarrow 2!=2 \times 1!=2 \leftarrow 1!=1 \times 0!=1 \leftarrow 0!=1$$

未知

已知

●递归条件：

1) 必须具有递归结束条件及结束时的值，即必须具有递归的“出口”

2) 求解的问题要能用递归形式表示，即有“递归公式”

例：求n!

P₂₃₄

分析：计算n!的公式如下：

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

递归结束条件

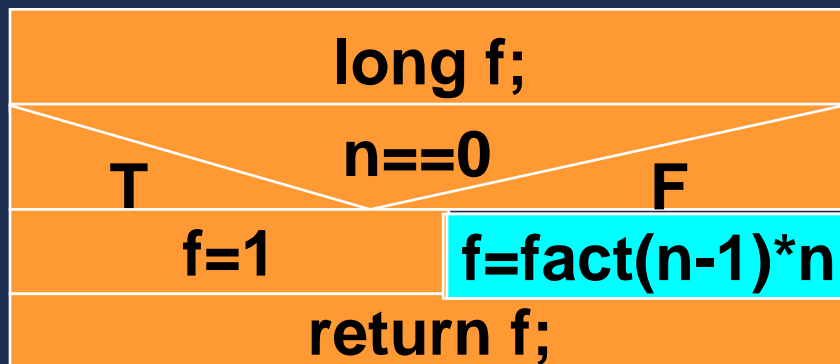
规律

这是一个递归形式的公式，应该用递归函数实现。

- 子函数的定义：

函数接收一个整型参数，返回一个整数值long int，即

```
long fact(int n) //函数定义
{
.....
}
```

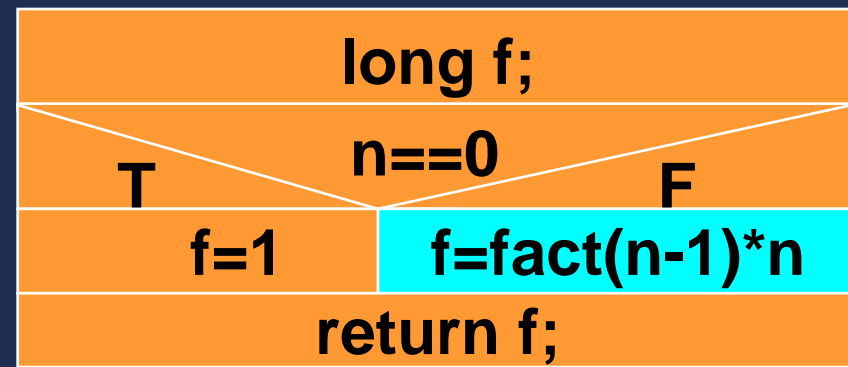


```

#include <iostream>
using namespace std;
long fact(int n)
{
    long f;
    if (n==0)
        f=1;
    else
        f=fact(n-1)*n; //函数递归调用
    return(f);
}

void main()
{
    int n;
    long y;
    cout<<"Enter a positive integer:";
    cin>>n;
    y=fact(n); // 函数调用
    cout<<n<<"!="<<y<<endl;
}

```



用递归方法求 m^n

P₂₃₆

如何定义子函数？

要计算 m^n ，则函数必须有2个参数 m, n (形参名字不重要，甚至可以省略，但是形参的类型不能省略！)，一个为`double`，一个为`int`。返回类型为`double`，即

```
double power(double m, int n) //函数定义
{
    ..... //求 $m^n$ ,递推公式  $m^n = m * m^{n-1}$ 
}
```

double f;		
T 输出0的任何 次方均为0	m==0	
	T	F
	n==0	
	T	F
f=0	f=1	f=power(m,n-1)*m;
return f;		

```
#include <iostream>
using namespace std;
double power(double m,int n)
```

```
{
```

```
    double f=0;
```

```
    if (m==0)
```

```
{
```

```
        cout<<"0的任何次幂均为0"<<endl;    f=0;
```

```
}
```

```
    else
```

```
        if (n==0) f=1; //递归出口
```

```
        else f=power(m,n-1)*m; //函数递归调用
```

```
    return(f);
```

```
}
```

```
void main()
```

```
{
```

```
    int n;    double x,y;
```

```
    cout<<"请输入x(基数)和n(幂次): ";
```

```
    cin>>x>>n;
```

```
    y=power(x,n); // 函数调用
```

```
    cout<<x<<"的"<<n<<"次幂="<<y<<endl;
```

```
}
```

double f;		
T	m==0	F
	n==0	
输出0的任何次方均为0	T	F
f=0	f=1	f=power(m,n-1)*m;
return f;		

Fibonacci数列 P_{152}

1202年，意大利数学家斐波那契出版了他的《算盘全书》。他在书中提出了一个关于兔子繁殖的问题

时间(月)	初生兔子(对)	成熟兔子(对)	兔子总数(对)
1	1	0	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
7	5	8	13
8	8	13	21
9	13	21	34
10	21	34	55

Fibonacci数列 P₂₅₂

- 问题：用**递归方法**计算Fibonacci数列第n项的值。
- 分析：该问题可以用递归形式来表示，即有递归公式：

$$\text{fibo} = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$$

递归结束条件为：当n=1或n=2, fibo=1

可以构建一个递归子函数**int fibonacci(int n)**来实现求第n项的值，主函数中调用该递归子函数。


```

#include <iostream>
using namespace std;
int fibonacci(int n)
{
    int fibo;
    if(n==1||n==2)
        fibo=1;
    else
        fibo=fibonacci(n-1)+fibonacci(n-2); //函数递归调用
    return fibo;
}

```

```

void main( )
{

```

```

    int n,fibon;
    cout<<"Please input a positive integer:";
    cin>>n;

```

```

    fibon=fibonacci(n); //函数调用

```

```

    cout<<"The "<<n<<" number of fibonacci is:"<<fibon;

```

```

}

```

int fibonacci(int n);

int fibo;		
T	n==1 n==2	F
fibo=1		fibo=fibonacci(n-1) +fibonacci(n-2);
return fibo;		

6.13 十进制转换为二进制 P255

```
#include <iostream>
using namespace std;
void DecToBin(int num, int base);
void main(void)
{
    int decimalNum;
    int base=2;
    cout << "Enter number in decimal: ";
    cin >> decimalNum;
    cout << "Decimal " << decimalNum << " = ";
    DecToBin(decimalNum, base);      //函数调用
    cout << " binary" << endl;
}
void DecToBin(int num, int base)
{
    if (num > 0)
    {
        DecToBin(num/base, base);    //函数递归调用
        cout << num%base;
    }
}
```

若输入数据为**10**，则主函数中函数调用为：**DecToBin(10,2);**
执行到调用语句时，主函数的执行转向子函数，同时，**实参的值传递给形参**，子函数开始执行：

```
void DecToBin(int num, int base)
{
    if (num > 0)
    {
        DecToBin(num/base, base);
        //递归调用
        cout << num%base;
    }
}
```

递推过程：

- 1、num=10>0 成立, DecToBin(10/2, 2)
↓
- 2、num=5, 5>0 成立, DecToBin(5/2, 2)
↓
- 3、num=2, 2>0 成立, DecToBin(2/2, 2)
↓
- 4、num=1, 1>0 成立, DecToBin(1/2, 2)
↓
- 5、num=0, 0>0 不成立，结束递推！

1、子函数调用结束，**执行其后语句**cout << num%base; 此时**num=10**，即输出10%2，**亦即输出0。整个调用结束！**

↑

2、子函数调用结束，**执行其后语句**cout << num%base; 此时**num=5**，即输出5%2，**亦即输出1。**

↑

3、子函数调用结束，**执行其后语句**cout << num%base; 此时**num=2**，即输出2%2，**亦即输出0。**

↑

4、子函数调用结束，**执行其后语句**cout << num%base; 此时**num=1**，即输出1%2，**亦即输出1。**

↑

(回归过程)

6.14推算年龄 P251

```
#include <iostream>
using namespace std;
int age(int n, int m)
```

$$age(n) = \begin{cases} 10 & , n = 1 \\ age(n - 1) + m & , n > 1 \end{cases}$$

```
{
    int year;
    if ( n==1 )
        year=10;
    else
        year=age(n-1,m)+m;
    return year;
}
```

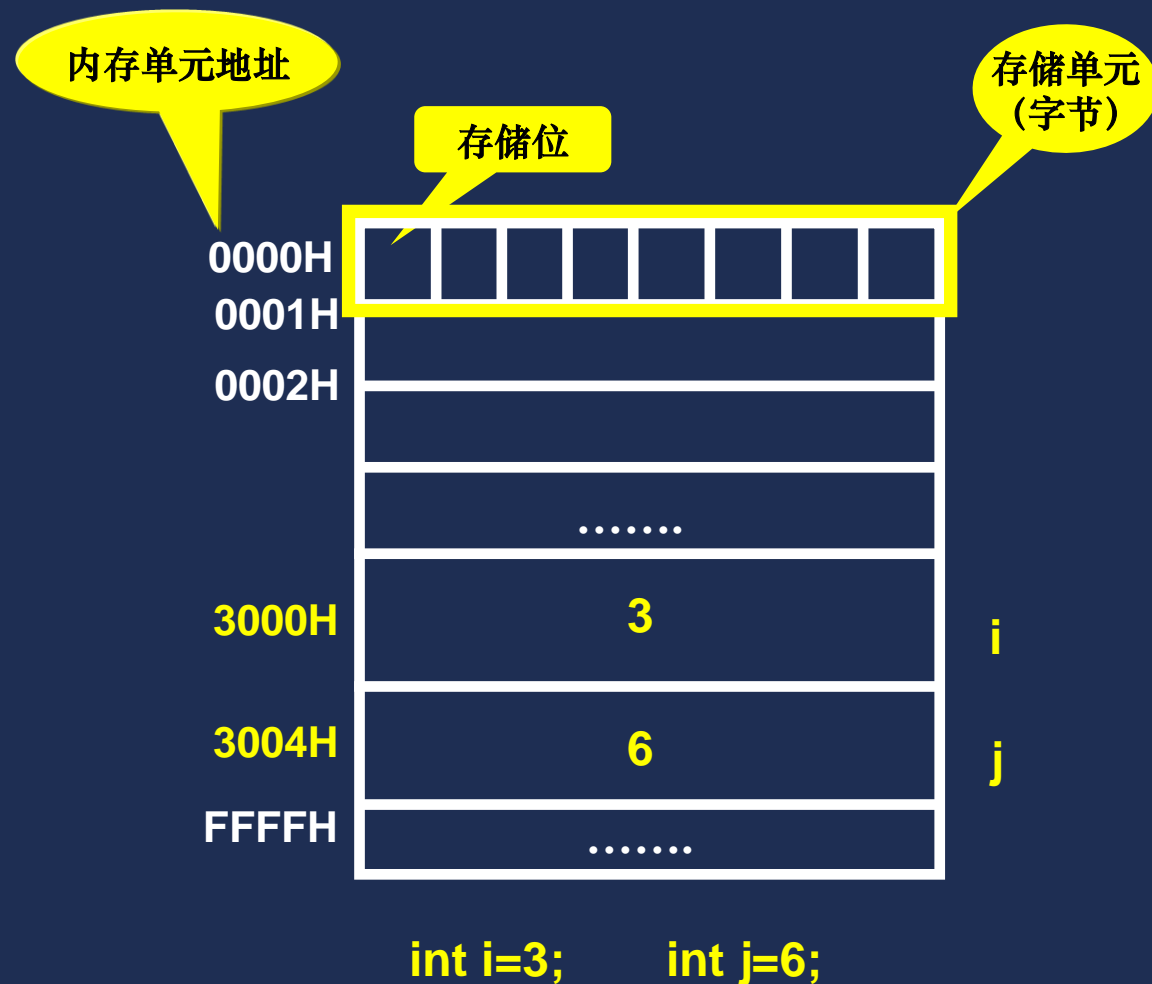
```
void main(void)
```

```
{ int n, m;
    cout<<"请输入参加年龄推算的人数 (一般不超过20人) : ";
    cin>>n;
    cout<<"请输入相邻两人的年龄差 (一般不超过5岁) : ";
    cin>> m;
    cout<<"第"<<n<<"个人的年龄为: "<<age(n,m)<<endl;
}
```

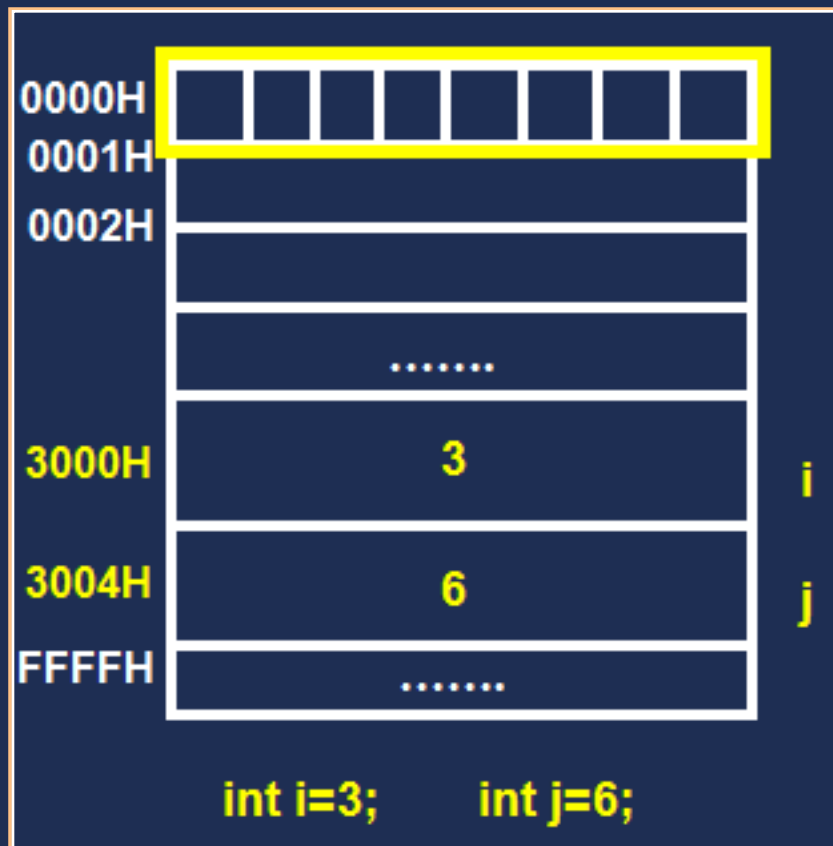
5.1 内存地址与指针 P₁₈₆

●内存地址

计算机的内存被划分为一个个的存储单元，简称**内存单元**。内存单元按一定的规则编号，这个编号就是存储单元的**地址**。



内存地址



在C++中，为某个变量或者函数分配内存的工作由编译程序完成。程序中定义的变量或声明的数组等要占用一定的内存空间，不同的数据类型占用的内存空间大小不一样。

内存单元的数据存取方法 P₁₈₆

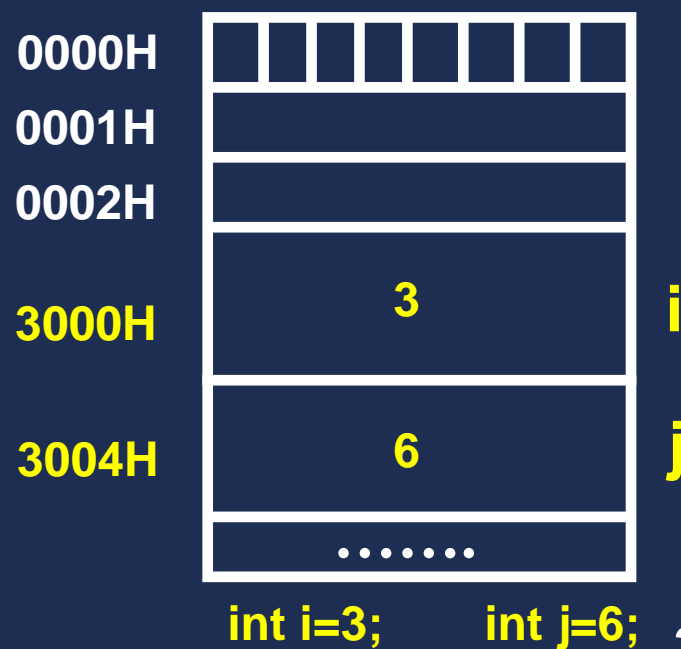
C++程序中从内存单元中存取数据的方法有两种：**直接访问方式**和**间接访问方式**。

●直接访问方式

通过变量名，直接对变量的存储单元进行存取访问（前三章）。

变量获得内存空间的同时，**变量名**也就成为了**相应内存空间的名称**，可以用该变量名访问其对应的内存空间，表现在程序语句中就是通过变量名存取变量内容。

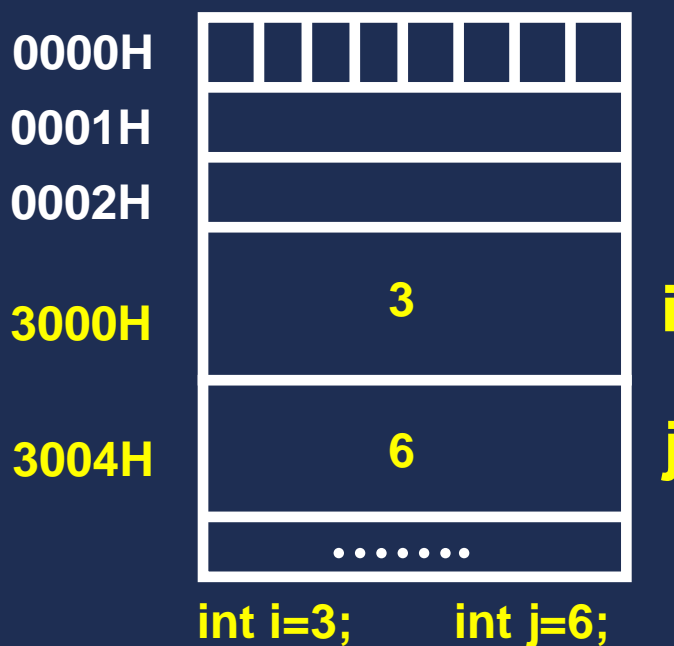
$y=i+j$ $// y=3+6$



从内存单元中存取数据的方法

●间接访问方式

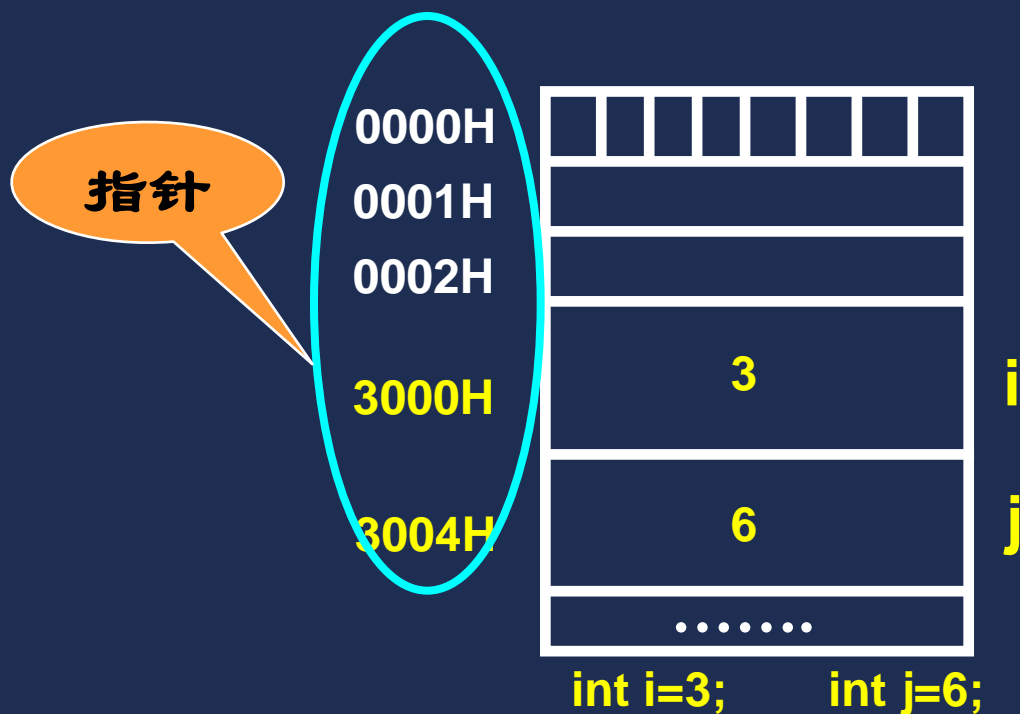
先找到存放变量的地址，再根据变量的地址找到变量的存储单元，对它进行存取访问。



指针

一个变量在内存空间中占用的地址就称为该变量的“指针”。

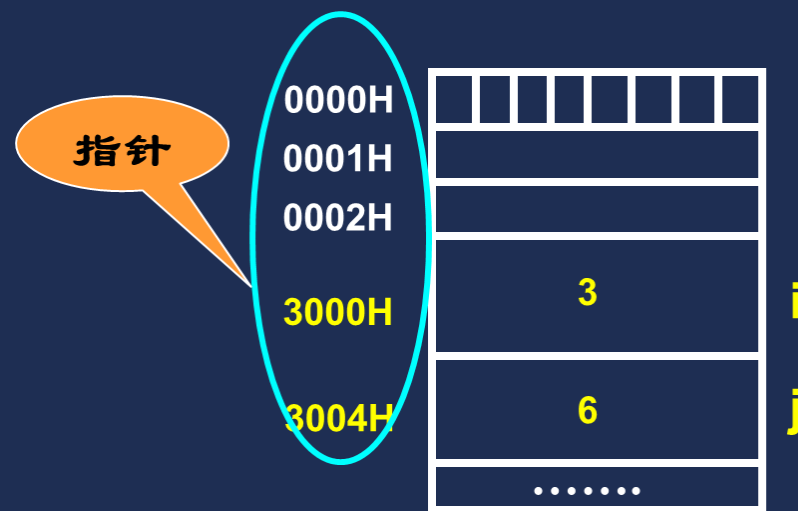
在C++中专门用来存放内存单元地址的变量类型，就是指针类型。



指针变量 P₁₈₆

指针变量是一种特殊的变量，用于存放内存单元的地址，即存放地址的变量就是指针变量。指针变量具有变量的三个要素：

- (1) 变量名，这与一般变量取名相同，由英文字符开始。
- (2) 指针变量的类型，是指针所指向的变量的类型，而不是自身的类型。
- (3) 指针变量的值是某个变量的内存地址。



指针变量定义 P₁₈₇

指针变量定义的一般格式如下：

数据类型 *指针变量名;

- 定义语句中的“*”表示该变量为指针变量。
- 数据类型标识符规定了指针变量指向的变量的数据类型。

例如：**int *p;**

表示变量p为指针变量，p指向的是整型数据，
即指针变量p只能操作整型数据。

指针与指针变量的区别

- **指针**：变量在内存空间中占用的地址。
- **指针变量**：用于存储内存单元地址的变量，指针变量的值是某个变量的内存地址。
- 通把指针变量简称为“指针” P187

地址运算符：& P₁₈₇

取地址运算符&，是一个一元运算符，用来得到一个对象的地址。

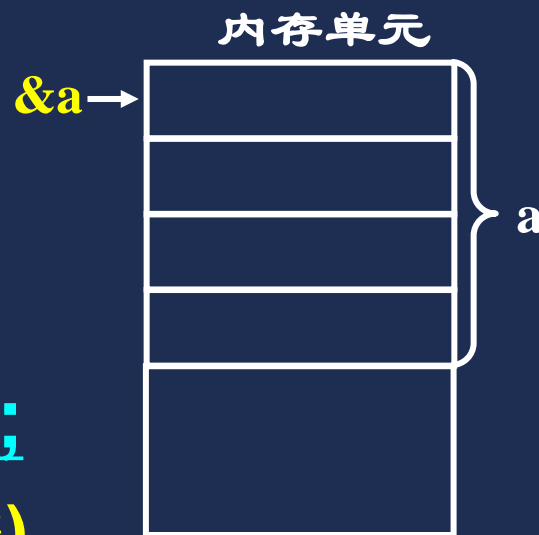
例如 `int a;` 则 `&a` 表示变量 `a` 在内存中的起始地址。
取地址运算符的操作数必须是变量名。

C++规定：

- 变量的地址可以使用地址运算符&求得。

例如，`&a` 表示变量 `a` 在内存的地址；

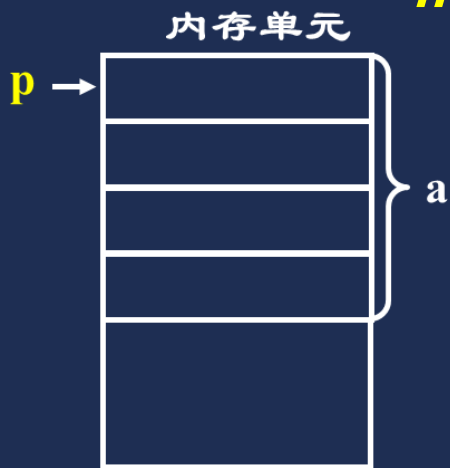
- 数组的地址，即数组第一个元素的地址，可以直接用数组名表示，例如 `a` 或 `&a[0]`；
- 函数的地址用函数名表示(第六章详细讲解)



指针运算符 * P₁₈₇

指针运算符 * 是一个一元运算符，表示指针变量所指向的变量的值（取值）。

如：int a=3,b,*p; // 定义整型指针变量p
 p=&a; // p指向变量a在内存中的地址
 b=*p; // *p, 取指针变量p所指向的 内存单元的内容
 // 等价于 b=a;



注意

```
#include<iostream>
using namespace std;
void main( )
{
    int x,*p; //定义指针变量p
    x=3;
    p=&x ;    //p指向变量x
    *p=*p+10; //x=x+10;
    cout<<*p<<endl;
}
```

*** 出现在定义语句中和执行语句中其含义是不同的。**

例如：

int x,*p; 在该语句中，*p是一个int型指针 ***在定义语句中，表示定义的是指针变量。**

*p=*p+10; 在该语句中，*p是指针p的内容，***在执行语句表示指针变量所指对象的内容（取值）。**

指针变量的初始化

P₁₈₇

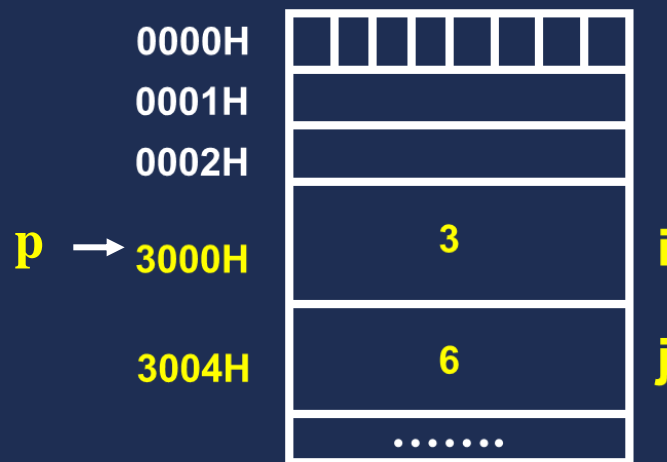
在定义指针变量的同时进行赋值，语法形式

数据类型 *指针名 = 地址;

其中的“地址”可以是变量的地址、数组名、函数名等。

例： `int i=3;`
`int *p=&i;`

注意：用变量地址作为初值
时，该变量必须在指针初
始化之前定义。



指针变量的赋值

P₁₈₇

指针变量=地址;

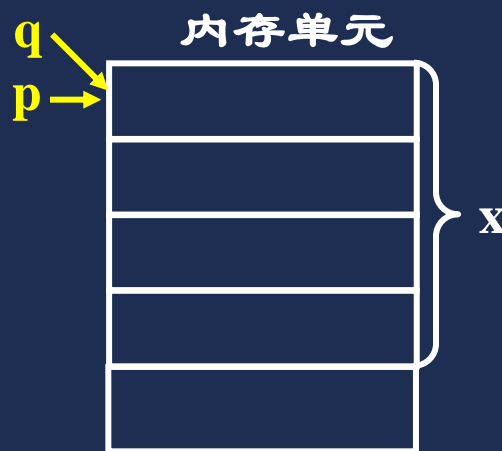
例如：

```
int *p,*q;
```

```
int x=4;
```

```
p=&x; //将x变量的地址赋给p, 即p指向x
```

```
q=p; //可以用已赋值的指针赋值给另外一个指针变量。
```



注意事项

P₁₈₇

- 不能把常量或表达式的地址赋给指针变量。
如：P=&67；P=&(i+5)是非法的
- 不能将一个整数赋给指针变量，但可以赋整数值0，表示该指针空指针。0是可以直接赋给指针变量的唯一整数值。

int *p; p=0; //p为空指针，不指向任何地址

- 允许声明指向 void 类型的指针。该指针可以被赋予任何类型对象的地址。

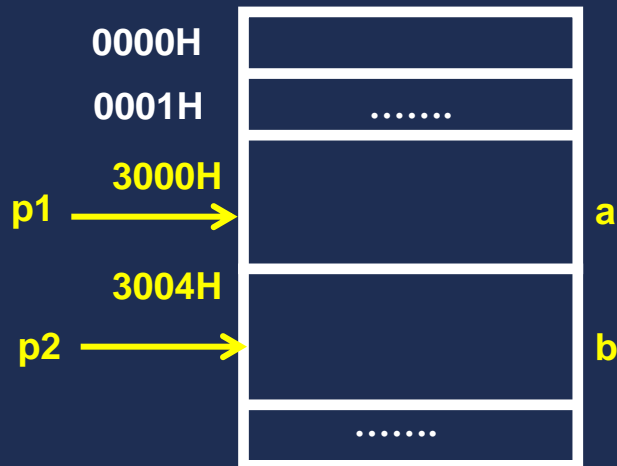
例： void *general;
 int i;
 general=&i;

示例

- 输入a和b两个整数，运用指针变量，输出a和b。
- 思路分析：输入两个整数，头文件为<bits/stdc++.h>，输出a，b；若a小输出b，a。

注意：用变量地址作为指针初值时，该变量必须在指针初始化之前定义。

int a,b;	
cin>>a>>b;	
T	F
a<b;	
cout<<b<<a;	cout<<a<<b;



int a,b;	
int *p1,*p2;	
p1=&a;	
p2=&b;	
cin>>*p1>>*p2;	
T	F
*p1<*p2;	
cout<<*p2<<*p1;	cout<<*p1<<*p2;

```
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
```

```
{
```

```
    int a,b;
```

```
    int *p1=&a;
```

```
    int *p2=&b;
```

```
    cout<<"input a and b:"<<endl;
```

```
    cin>>*p1>>*p2;
```

```
    if(*p1<*p2)
```

```
        cout<<"按先大后小输出a和b:"<<*p2<<setw(4)<<*p1<<endl
    else
```

```
        cout<<"按先大后小输出a和b:"<<*p1<<setw(4)<<*p2<<endl;
```

```
}
```

int a,b; int *p1, *p2;		
p1=&a; P2=&b;		
cin>>*p1>> *p2;		
T	*p1<*p2;	F
cout<<*p2<< *p1;		cout<<*p1<< *p2;

5.4 一维数组与指针 P₁₈₈

1. 指向数组元素的指针

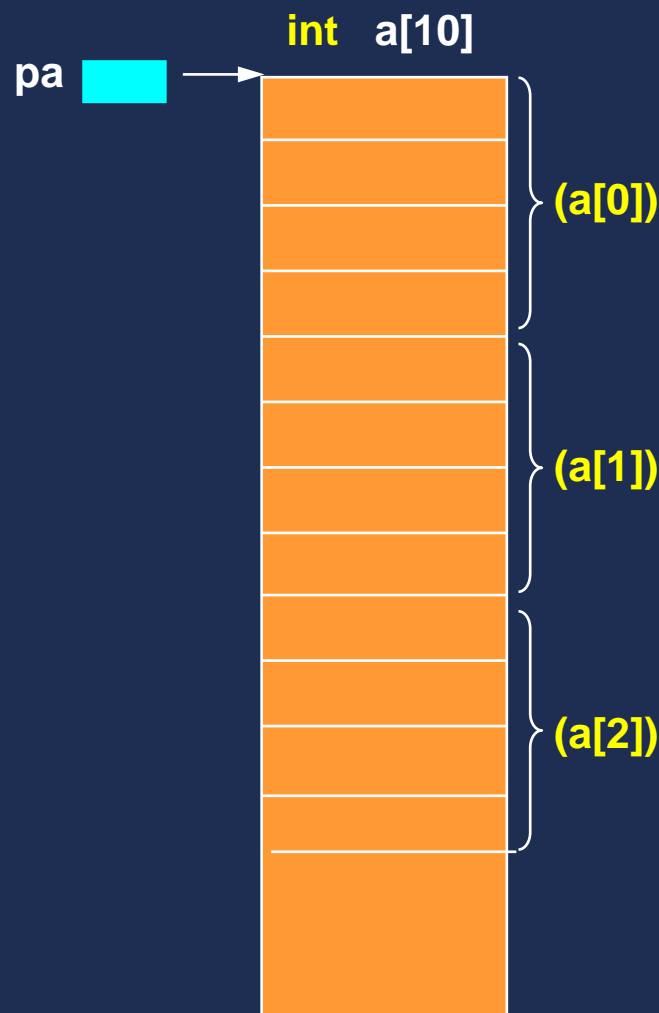
❖ 定义与赋值

例： `int a[10], *pa;`

`pa=&a[0];`

或 `pa=a;`

pa指向数组第一个元素，
或数组首地址。



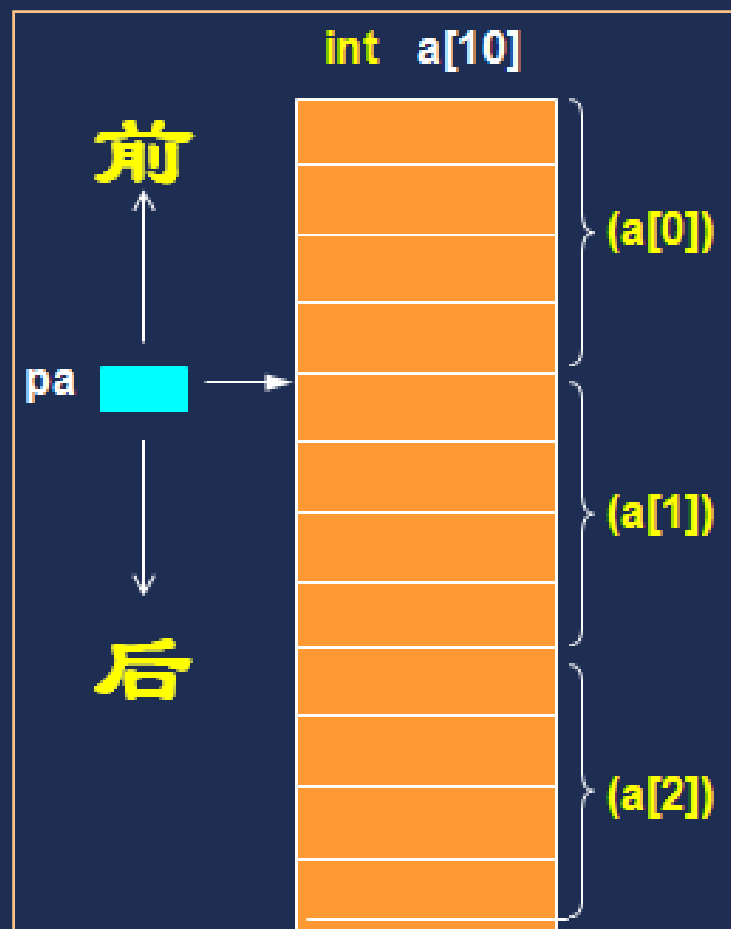
2. 指针变量的算术运算 P₁₉₀

❖ 指针加一，减一运算

- “加一”：指向**后**一个数据；“减一”：指向**前**一个数据。

❖ 指针与整数的加减运算

- 指针 p 加上或减去 n，其意义是指针当前指向位置的后方或前方第 n 个数据的地址。
- 这种运算的结果值取决于指针指向的数据类型。



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

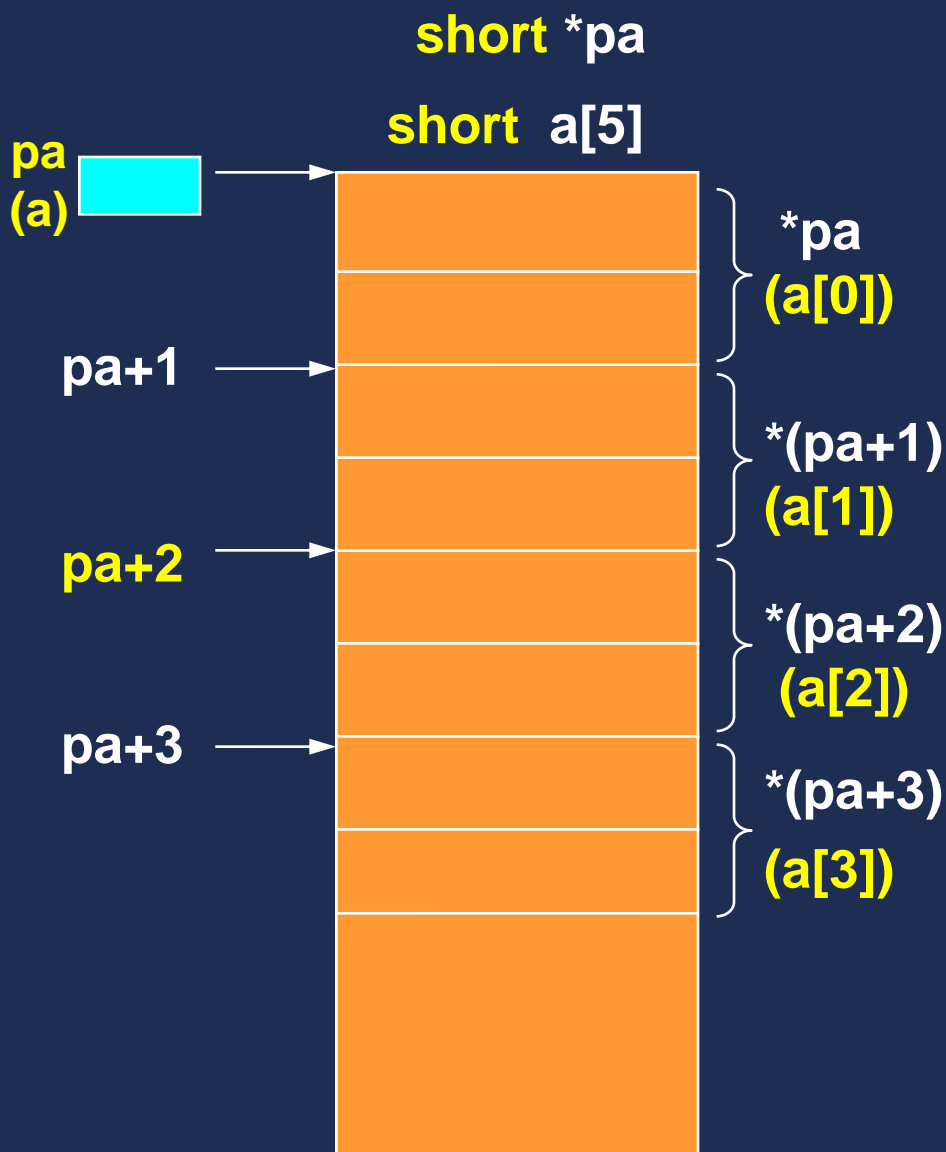
```
short *pa;
```

```
pa=a;
```

```
pa+=2;
```

/* 指针 p 加上或减去 n, 即
指针当前指向位置的后方或
前方第 n 个数据的地址。

结果值取决于指针指向的数
据类型。*/



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

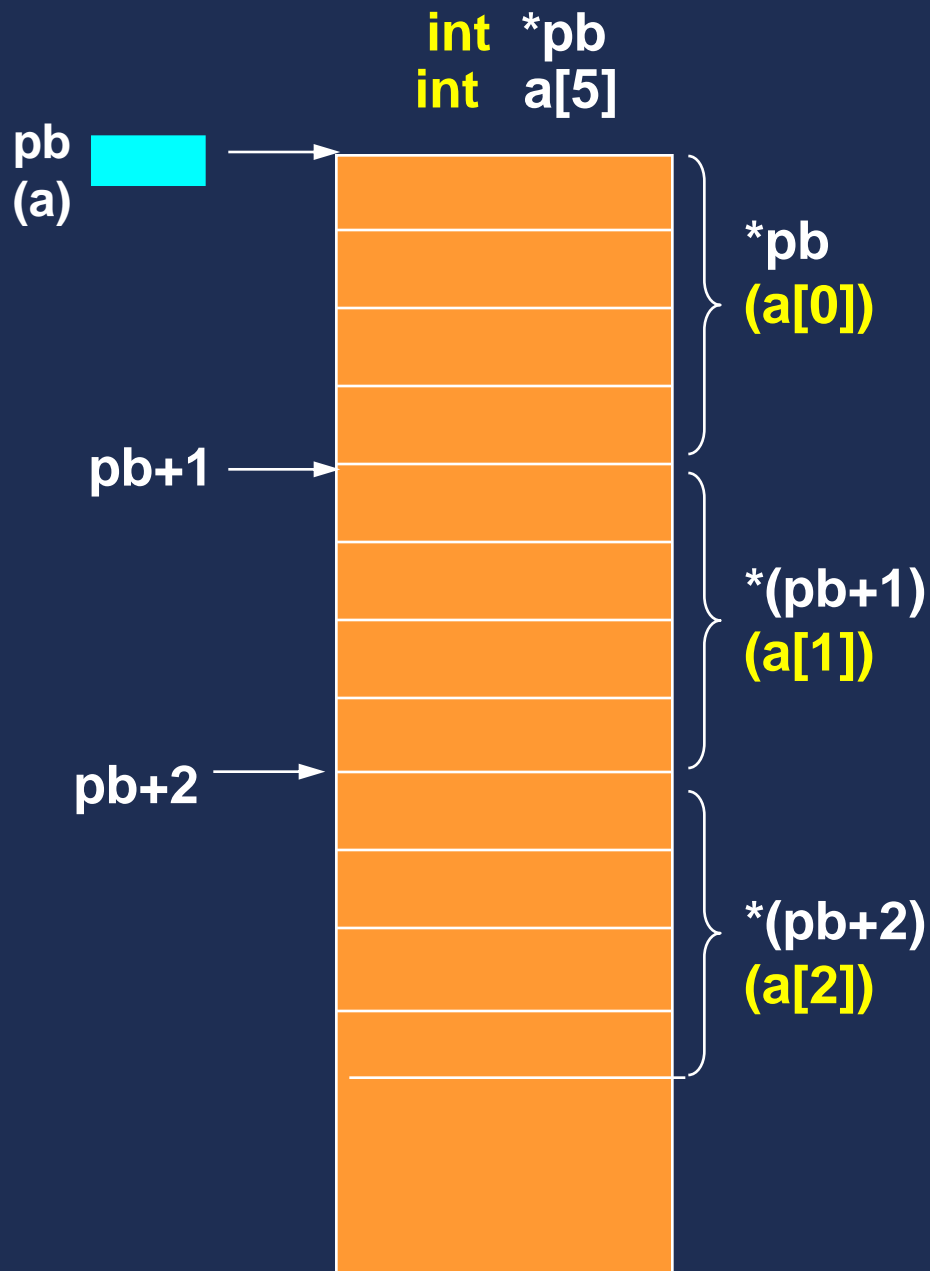
```
int *pb;
```

```
pb=a;
```

```
pb+=2;
```

/*指针 p 加上或减去 n，即
指针当前指向位置的后方或
前方第 n 个数据的地址。

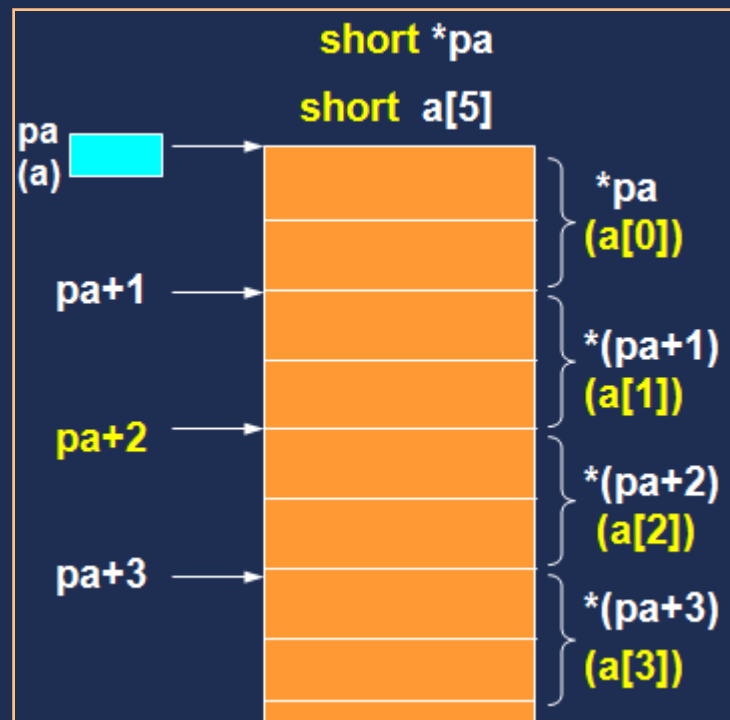
结果值取决于指针指向的数
据类型。*/



通过指针引用数组元素

对数组中第 $i+1$ 个元素可表示成以下四种：

- $a[i]$, $*(pa+i)$, $*(a+i)$, $pa[i]$ 都是等效的。这四个等价关系使得数组与指针相互转换非常灵活。
- $*pa$ 就是 $a[0]$ 。
- 不能写 $a++$, 因为 a 是数组首地址是常量。



与上面相对应的有下面四个地址等价关系：

- $\&a[i]$, $pa+i$, $a+i$, $\&pa[i]$ 均表示第 i 个数组元素在内存中的地址。

应用举例 P₁₈₉、P₁₉₁ 例5.1

在C++中，有了指针和地址的概念，在操作数组时，就可以用如下的四种方法来操作数组。

- 使用数组名和下标 (**a[i]**)
- 使用指针变量的下标表示法 (**pa[i]**)
- 使用数组名和指针运算 (***(a+i)**)
- 使用指针变量 (***(pa+i)**)

```
#include <iostream>
using namespace std;
void main( )
{ int a[10];
  int *p, i;
```

设有一个int型数组a，有10个元素。用上述四种方法访问数组的各个元素。

前4种方法有元素的下标信息，如果需要处理位置的问题可选用其中任何一种，第5种方法没有下标信息，通过移动指针操作所有数组元素。在进行下一次处理时，需要回溯指针到数组开始位置，即p=a;

```
for(i=0; i<10; i++)
    cin>>a[i];    //①数组名和
```

```
p=a;    //指针变量指向数组首地址
for(i=0; i<10; i++)
    cout<<p[i];    //②指针变量
```

```
for(i=0; i<10; i++)
    cout<<*(a+i); //③数组名和指针运算访问数组
```

```
p=a;    //指针变量指向数组首地址
for(i=0; i<10; i++)
    cout<<*(p+i); //④数组名和指针运算访问数组
```

```
for(p=a; p<a+10; p++)
    cout<<*p;    //⑤使用指针变量访问数组 (重点掌握)
p=a;    //将指针回溯到数组开始位置，以便下一次处理。
```

```
}
```

示例5.8

P_{208~209}

- 利用指针技术，将键盘输入的N个整数按相反的顺序存放并输出。

- 思路分析：

分别取出数组最前面和最后面的元素，进行交换，即 $a[0]$ 与 $a[N-1]$ 交换；然后再分别取出 $a[1]$ 与 $a[N-2]$ 交换；直到交换完毕。

定义数组a[N]、输入数组的初始值

```
int *p; p=a;
```

```
int i=0, j=N-1;
```

```
for(i=0,j=N-1;i<j;i++,j--)
```

```
temp=*(p+i);
```

```
*(p+i)=*(p+j);
```

```
*(p+j) =temp;
```

输出交换后数组的各元素

a[i], a[j]如何用指针变量表示

*(p+i)

*(p+j)

```

#include <iostream>
#include <iomanip>
using namespace std;
void main()
{
    const N=10;
    int a[N],i,j,temp,*p;
    cout<<"Input ten interger:"<<endl;
    for(p=a;p<a+N;p++) cin>>*p; //用⑤指针变量操作数组元素
    p=a; //回溯指针, 让其指到数组的首地址
    for(i=0,j=N-1;i<j;i++,j--)
    {
        temp=*(p+i); *(p+i)=*(p+j); *(p+j)=temp; //实现反序存储
    }
    for(p=a;p<a+N;p++)
        cout<<setw(4)<<*p; //用方法⑤输出反序后的数组元素
    cout<<endl;
}

```

定义数组a[N]、输入数组的初始值

Int *p; p=a;

int i=0, j=N-1;

for(i=0,j=N-1;i<j;i++,j--)

temp=*(p+i);

(p+i)=(p+j);

*(p+j)=temp;

输出交换后数组的各元素

6.2 函数的参数传递机制 P₂₂₆

- 在子函数未被调用时，函数的形参并不占有实际的内存空间，也没有实际的值。

- 只有在子函数被调用时：

- 才分配形参的存储单元；
- 并将实参与形参结合。

- 实参与形参结合有3种：

- 值传递

单向传递

- 引用传递

- 地址传递

双向传递

✦ 数组

✦ 指针

```
#include <iostream>
using namespace std;
void main()
{
    int max(int a, int b); //函数声明
    int m, n, max_num;
    cout<<" input m,n:"<<endl;
    cin>>m>>n;
    max_num=max(m,n); //函数调用
    cout<<max_num<<endl;
}

int max(int a, int b) //函数定义
{
    return (a>b?a:b);
}
```

6.2.1 值传递的特点 P₂₂₉

调用时实参仅将其值赋给形参，在函数中
对形参值的任何修改都不会影响到实参的值，
传递时是传递参数值，即“单向传递”。

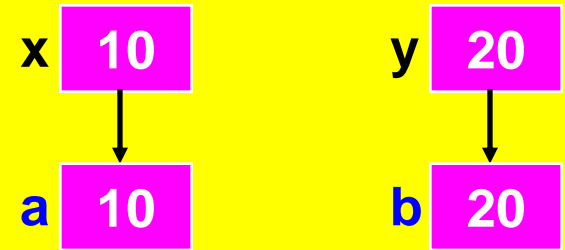

```
#include <iostream>
using namespace std;
void main()
{
    void swap(int a, int b);
    int x=10,y=20;
    cout<<"(1)x="<<x<<"y="<<y<<endl;
    swap(x, y);
    cout<<"(4)x="<<x<<"y="<<y<<endl;
}
```

程序运行结果如下：

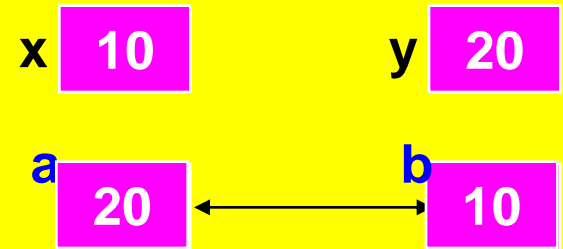
```
(1)x=10 y=20
(2)a=10 b=20
(3)a=20 b=10
(4)x=10 y=20
```

```
void swap(int a, int b)
{
    int t;
    cout<<"(2)a="<<a<<"b="<<b<<endl;
    t=a;a=b;b=t;
    cout<<"(3)a="<<a<<"b="<<b<<endl;
}
```

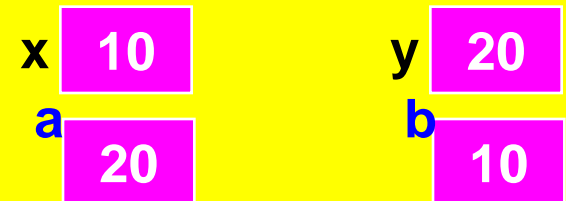
结论：参数的传递是单向的，即只能由实参传给形参，在被调函数中对形参的改变的不影响实参的值。



调用swap时



调用swap中



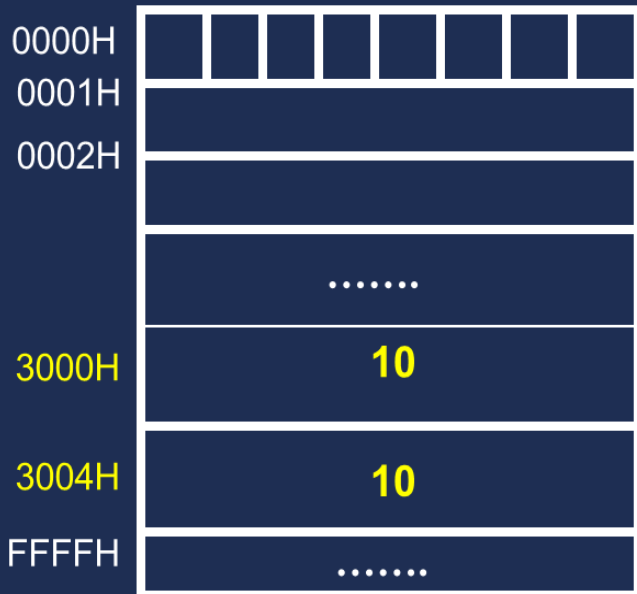
调用swap后

6.2.2 引用传递的特点 P₂₂₉

- 引用(&)是一种特殊类型的变量，某变量的引用可以看成是该变量的一个别名，引用传递的是变量的地址。

```
int i=3,j=6;
```

```
int &ri=i; /*建立一个int型的引用ri,将其初始化为变量i的地址*/
```

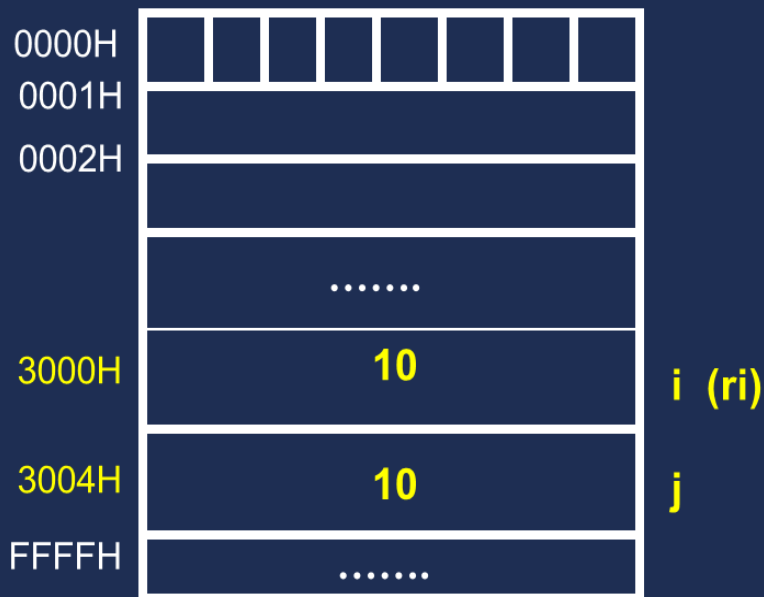


- 对引用的改动实际就是对目标的改动。

```
j=10;    ri=j;
```

```
/*相当于 i=10;*/
```

6.4.2 引用传递的特点 P₂₂₉



```
int i=3,j=6;  
int &ri=i;
```

- 引用不是真正意义上的变量，它不占存储空间。引用只有声明。
- 声明一个引用时，必须同时对它进行初始化，使它指向一个已存在的对象。一旦一个引用被初始化后，就不能改为指向其它的对象。

特别注意的问题：

- 不允许对void类型进行引用；
- 出现在声明定义语句中的‘&’符号用来说明引用，除此之外，在任何其他位置出现都属于地址运算符。

6.4.2 引用传递的特点 P₂₂₉

```
#include<iostream>
using namespace std;
void main()
{ //函数声明, 形参是引用
  void swap(int &a, int &b);
  int x=5, y=10;
  cout<<"x="<<x<<"   y="<<y<<endl;
  swap(x,y);    //函数调用时实参是普通变量
  cout<<"x="<<x<<"   y="<<y<<endl;
}
```

//用引用作参数

```
void swap(int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}
```

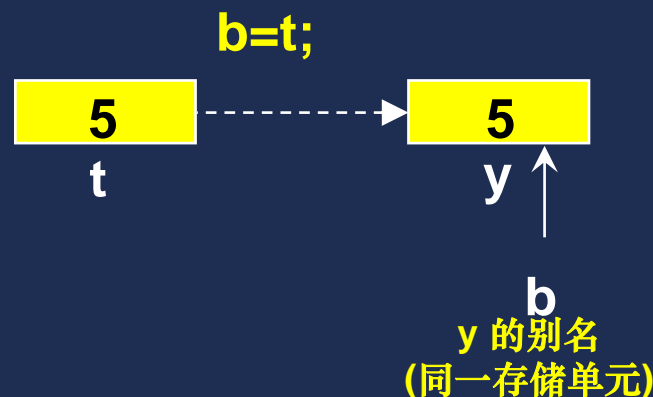
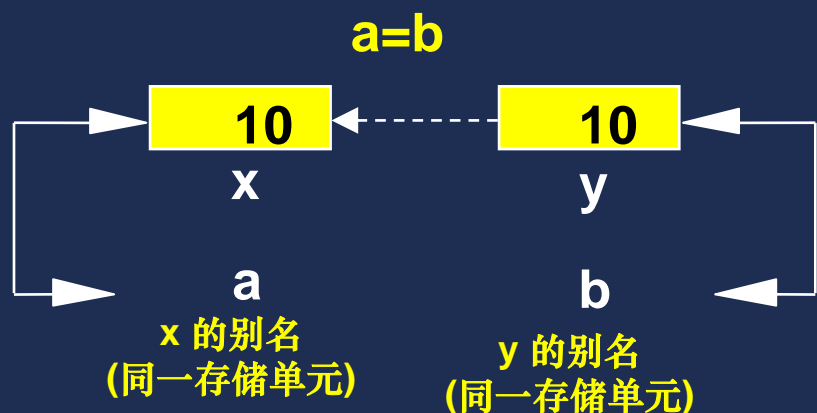
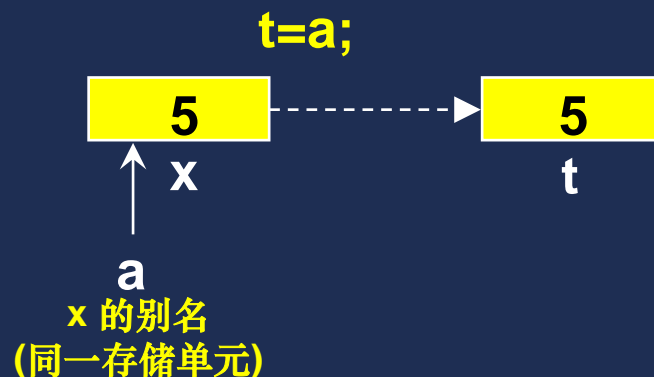
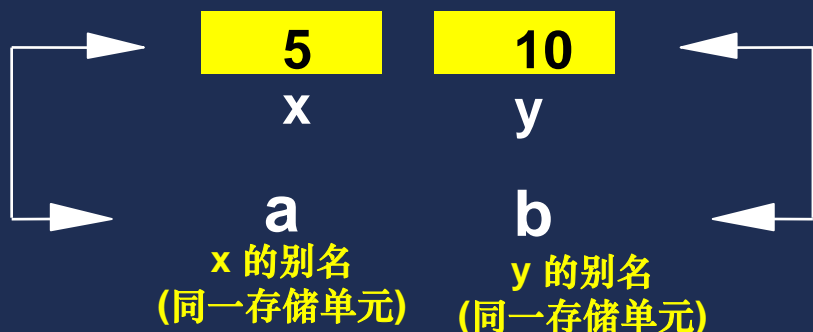
格式:

- **形参是引用**, 写为: **&变量名**。实参是普通变量。
- 参数传递时, 将实参变量的地址,也就是实参变量的别名传递给形参变量。这时实参和形参两个变量对应同一个存储单元, 对形参的任何操作也就是直接作用于实参, 为双向值传递。

```
void swap(int &a, int &b)
{
    int t;
    t=a; a=b; b=t;
}
```

当希望在执行被调用函数时，能够改变主调用函数中某个实参的值，可在对应的被调用函数的形参变量名前加上引用符号&。

swap(x,y); //x, y为实参



示例

设计一个加法程序，**用子函数分别实现输入和计算输入的两数之和**，子函数的返回类型均为**void**，主函数通过调用子函数实现输入，并计算结果输出。

```
void input(double &a, double &b)
{ cout<<"请输入两个加数："<<endl;
  cin>>a>>b;
}
```

```
void add(double m, double n, double &result)
{ result=m+n; }
```

```
void main()
{ double x,y,a;
  input(x,y); //函数调用，实参为普通变量
  add(x,y,a); //函数调用，实参为普通变量
  cout<<x<<"+"<<y<<"="<<a<<endl;
}
```

示例

实现功能：统计任意一个字符串中**元音字母**的数量并输出。

(1) **主函数功能**：从键盘输入一个字符串，通过调用子函数得到元音字母的数量并输出。

(2) **子函数功能**：定义一个**void型子函数**，实现统计字符串中元音字母数量的**功能**。

```
void count(string s, int &sum);
```

```
void count(string s, int &sum)
{
    int i=0;
    for(i=0;i<s.size();i++)
        if(元音字符)
            sum=sum+1;
}
```

```
void main()
{
    string s;
    int sum;
    getline(cin,s); //可接收空格
    count(s,sum);
    cout<<"串中的元音字符个数
为："<<sum<<endl;
}
```

//辗转法求最大公

r=p%q;

while(r!=0)

{ p=q;

q=r;

r=p%q;

} //辗转法

如何设计子函数实现求最大公
因数和最小公倍数？

int gys(int x, int y) {.....}

int gbs(int x, int y) {.....}

值传递

int gys(int x, int y)

{

int r;

r=x%y;

while(r!=0)

{ x=y; y=r; r=x%y; }

return y;

}

cout <<"两个正整数的最大公因数是"<<q<<endl;

cout <<"两个正整数的最小公倍数是"<<a*b/q<<endl;

}

int gbs(int x, int y)

{ return (x*y/gys(x,y)); }

//辗转法求最大公约数:

r=p%q;

while(r!=0)

{ p=q;

q=r;

r=p%q;

} //辗转法

}

如果子函数的返回类型为void, 该如何设置子函数求最大公因数和最小公倍数?

```
void gys_gbs(int x, int y, int &m, int &n)
{.....}
```

```
void gys_gbs(int x, int y, int &m, int &n)
{
    int r,t;
    t=x*y; //为求最小公倍数, 保存原数乘积
    r=x%y;
    while(r!=0)
    { x=y; y=r; r=x%y; }
    m=y; n=t/y;
}

void main()
{ .....
    gys_gbs(x,y,m,n);
    cout<<"最大公约数为: "<<m<<endl;
    cout<<"最小公倍数为: "<<n<<end;
}
```

6.4.3 地址传递

P₂₂₉

地址传递：指针、数组

在地址传递中，实参将自己的地址（即内存空间中的位置）传递给形参。此时形参和实参在内存中占用相同的内存空间，相当于是一个数据。因此形参不但可以获得实参的数据，还可以修改该数据，即自定义函数中对形参的修改相当于在修改实参的值，所以地址传递是一种“双向传递”。

1. 指针变量做函数参数

用指针变量作为函数参数的作用是将一个变量的地址传送到另一个函数中。

例：输入两个整数，将两个数据交换后输出。 P227

```
void swap(int *p1,int *p2)
{ int t;
  t=*p1; *p1=*p2; *p2=t;
}
```

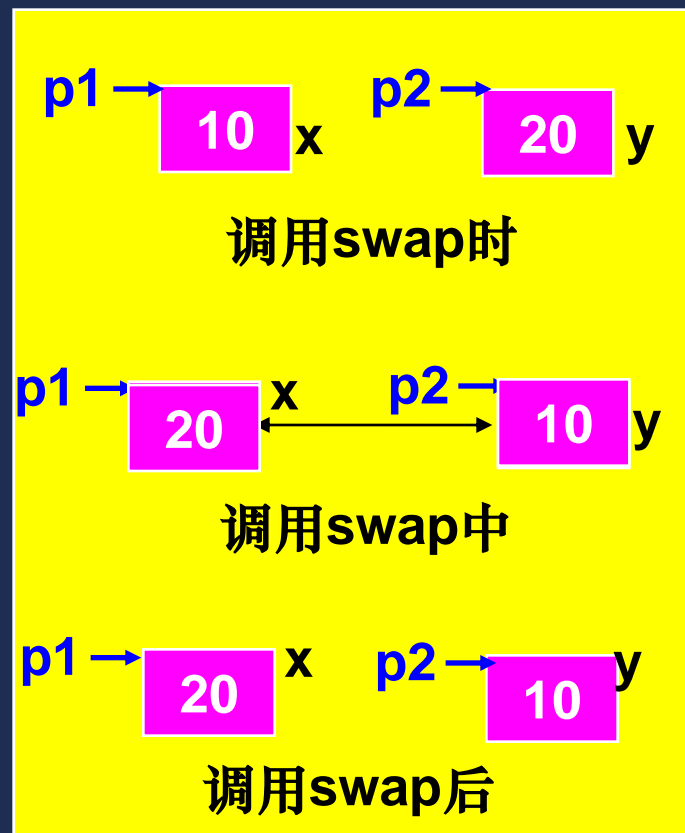
```
#include <iostream>
using namespace std;
void main()
{
  void swap(int *p1,int *p2);
  int x,y;
  cout<<"Input two interger:"<<endl;
  cin>>x>>y;
  swap(&x,&y); //调用时用变量地址作实参
  cout<<x<<endl;
  cout<<y<<endl;
}
```

主函数也可以改为：

```
void main()
{
  void swap(int *p1,int *p2);
  int x,y;
  int *q1=&x,*q2=&y; //定义指针变量
  cout<<"Input two interger:"<<endl;
  cin>>x>>y;
  swap(q1,q2); //指针变量做实参
  cout<<x<<endl;
  cout<<y<<endl;
}
```

```
#include <iostream>
using namespace std;
void main()
{
    void swap(int *p1,int *p2);
    int x,y;
    cout<<"Input two interger:"<<endl;
    cin>>x>>y;
    swap(&x,&y); //调用时用变量地址作实参
    cout<<x<<endl;
    cout<<y<<endl;
}
```

```
void swap(int *p1,int *p2)
{
    int t;
    t=*p1;
    *p1=*p2;
    *p2=t;
}
```



结论：

形参和实参在内存中占用相同的内存空间，对形参的修改相当于在修改实参的值，是一种“双向传递”。

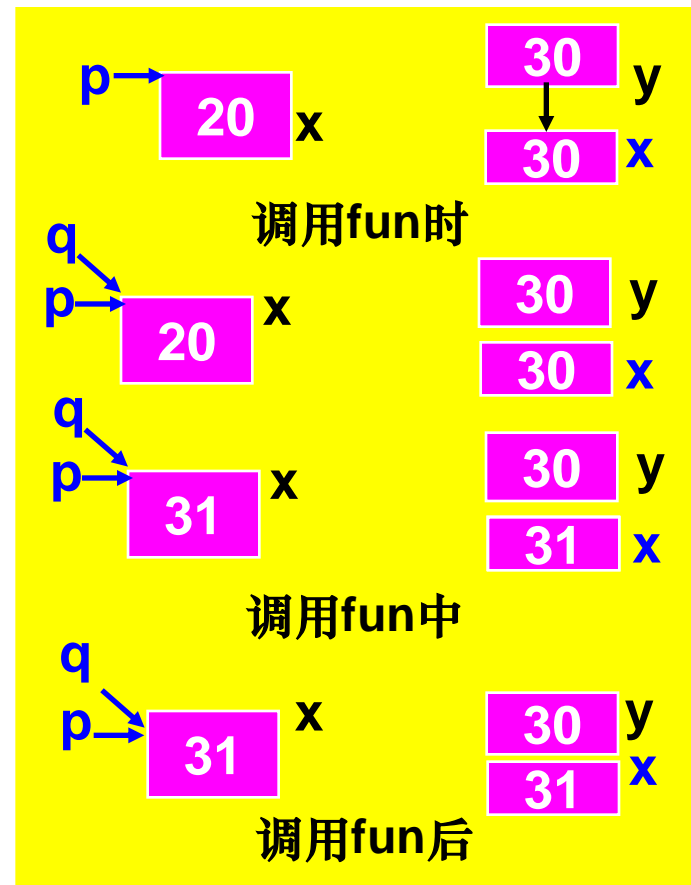
阅读程序给出运行结果

```
#include<iostream>
using namespace std;
void main()
{
    void fun(int *p,int x);
    int x=20, *p=&x;
    int y=30;
    fun(p,y); //调用
    cout<<"x="<<x<<" ,y="<<y<<endl;
}
```

x=31,y=30

```
void fun(int *p,int x)
{
    int *q;
    q=p; //指针变量p、q指向同一个内存空间
    *p=x+1;
    x=*q;
}
```

- 第1个形参为**指针变量**，是双向传递，形参的改变会影响实参；
- 第2个形参是**普通变量**，即单向传递，形参值的改变不会影响实参



2. 用数组名(地址)作为函数参数

掌握P₂₁₄ 例6.3-1

利用选择法排序，将键盘输入的10个整数按从小到大的顺序排序。

思路分析：用一个子函数实现选择法排序，主函数调用该子函数实现排序。

数组的定义，变量的定义

数组的输入

调用子函数进行排序(选择法)

输出排序后的数组

主函数算法描述

如何定义子函数？

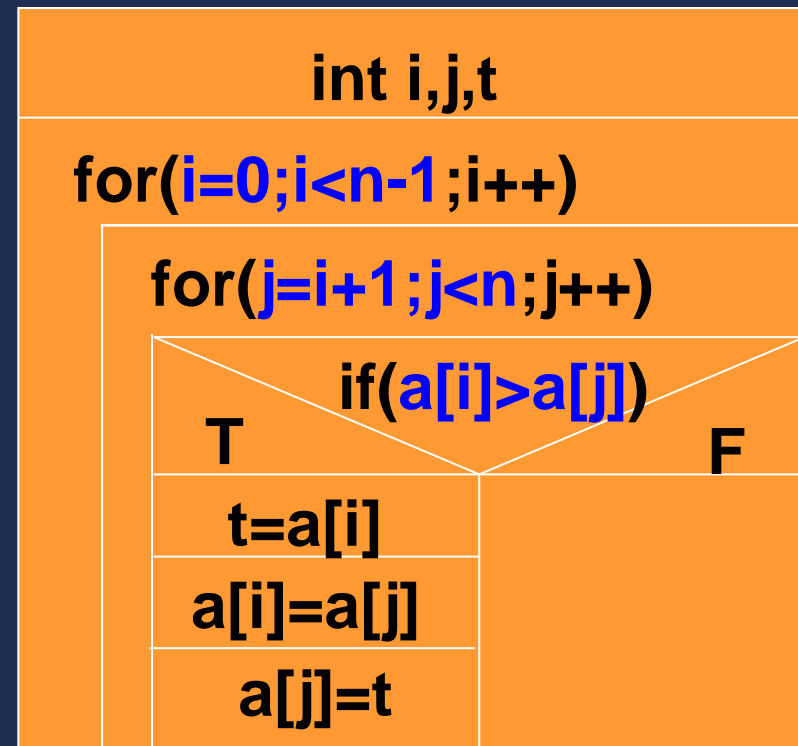
需要对一组数进行排序，
故**参数应为数组**。即

```
void sort(int a[ ], int n)
{
    .....
    //无return语句，结果由数组带回
}
```

```
void sort(int a[ ], int n)
{.....}
```

```
#include<iostream>
using namespace std;
```

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

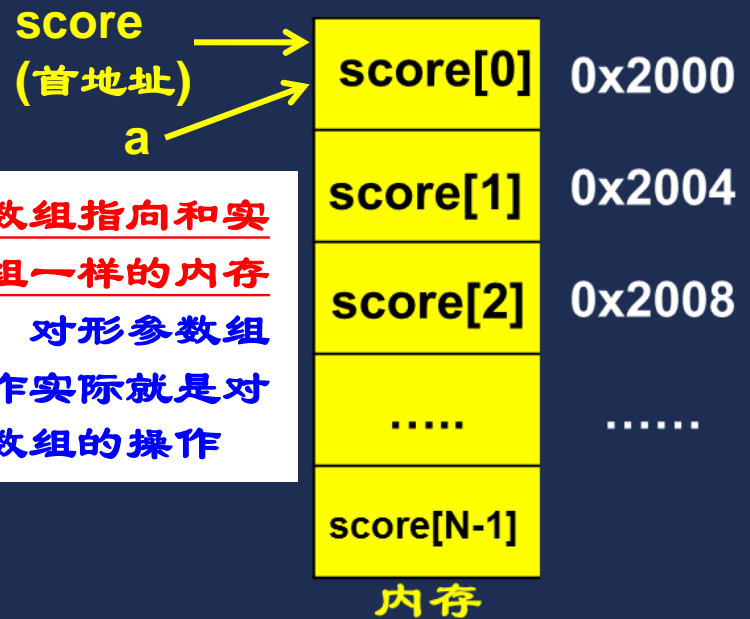


子函数算法描述

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

```
void main( )
{ const int N=10;
  int i;
  int score[N];
  cout<<"Input the numbers:"<<endl;
  for(i=0;i<N;i++)
    cin>>score[i];
  sort (score,N); //函数调用实现排序
  cout<<"The result is:"<<endl;
  for(i=0;i<N;i++)
    cout<<score[i]<<endl;
}
```

函数调用时给出数组名和数组长度



形参数组指向和实参数组一样的内存空间，对形参数组的操作实际就是对实参数组的操作

数组名作为函数参数时，应注意以下几点：

- 实参与形参都是数组名(形参数组名后的方括号不能省略掉!!!)。
- 实参数组与形参数组类型应一致，如不一致，结果将出错。

阅读程序给出运行结果

```
#include<iostream>
using namespace std;
void main()
{
    void fun_chage(int a[ ], int n);
    const int N=10;
    int a[N]={2,4,6,7,8,9,1,3,5,10};
    int i;
    fun_chage(a,N);
    for(i=0;i<N;i++)
        if(i%2==0)
            cout<<a[i]<<" ";
    cout<<endl;
}
void fun_chage(int a[], int n)
{
    int i;
    for(i=0;i<n;i++)
        a[n-i-1]=a[i];
}
```

数组原始数据：

2	4	6	7	8	9	1	3	5	10
---	---	---	---	---	---	---	---	---	----

执行子函数后的数组元素：

2	4	6	7	8	8	7	6	4	2
---	---	---	---	---	---	---	---	---	---

输出结果：

2	6	8	7	4
---	---	---	---	---

利用子函数实现求数组的最大值

```
#include <iostream>
using namespace std;
void main( )
{
    const int N=20;
    int a[N],i,m,n;
    void max(int a[ ],int n,int &max); //第三个参数为引用，即可以将最大值通过形参max
    的变化返回给实参!!!
    cout<<"请输入数组的实际长度:";          cin>>n;
    for(i=0;i<n;i++)
        cin>>a[i];
    for(i=0;i<n;i++)
        cout<<a[i]<<" ";
    cout<<endl;
    max(a,n,m); //函数调用
    cout<<"这一组数据中的最大值为："<<m<<endl;
}

void max(int a[ ],int n,int &max) //求数组中的最大元素，最大值通过形参(引用)返回
{
    max=a[0];
    for(int i=0;i<n;i++)
        if(a[i]>max)
            max=a[i];
}
```

说明

数组名作为函数参数时，应注意以下几点：

- 实参与形参都是数组名(形参数组名后的方括号不能省略掉!!!)。

```
void sort(int a[ ], int n) {.....}
```

- 实参数组与形参数组类型应一致，如不一致，结果将出错。
- 实参数组与形参数组维数大小可以不一致也可以一致。如果要求形参数组得到实参数组全部的元素值，最好指定形参数组与实参数组大小一致。
- 数组名作函数参数时，是“地址传递”，把实参数组的起始地址传递给形参数组，两个数组共同占用同一段内存单元。

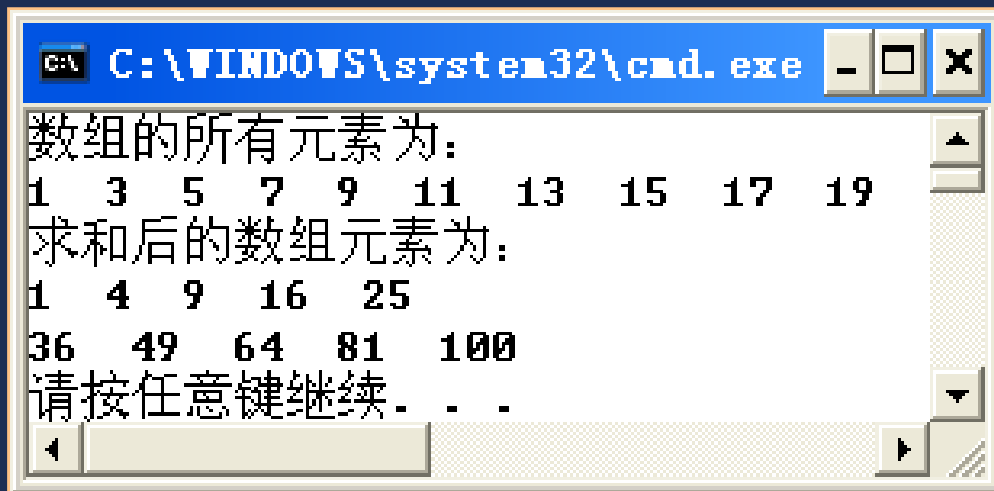
课后练习

求数组元素和值。

(1)主函数定义整型数组a[10]，数组元素值 $a[j]=2*j+1$ ，输出a数组；

(2)编一子函数，将主函数传递来的数组元素值改变为其前面所有数组元素的和值（包括该数组元素自身值），子函数头要求为sum(int a[],int n)，n用于传递数组的大小；

(3)主函数中输出改变后的a数组。



```
C:\WINDOWS\system32\cmd.exe
数组的所有元素为:
1 3 5 7 9 11 13 15 17 19
求和后的数组元素为:
1 4 9 16 25
36 49 64 81 100
请按任意键继续...
```

3. 指向数组的指针变量作为函数参数

示例： 将一个数组中的数据按相反顺序存放。

思路分析： 用一个子函数实现按相反顺序存放，主函数调用该子函数实现按相反顺序存放。

数组的定义，变量的定义

数组的输入

调用子函数,将数组元素反序

输出反序后的数组

主函数算法描述

如何定义子函数？

需要对一组数进行处理，
参数可设为数组。即

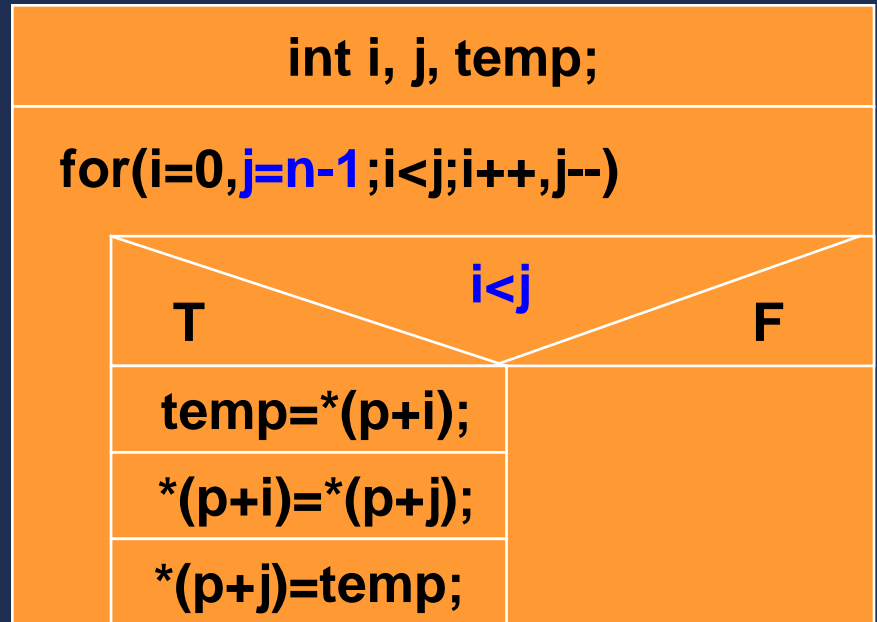
```
void invert(int a[ ], int n)
{.....}
```

需要对一组数进行处理， 参数可设为指向数组的指针

```
void invert(*p, int n) {.....}
```

P₂₀₉ 例5.8

```
void invert(int *p, int n)
{
    int i,j,temp;
    for(i=0,j=n-1;i<j;i++,j--)
    {
        temp=*(p+i);
        *(p+i)=*(p+j);
        *(p+j)=temp;
    }
}
```



子函数算法描述
(按相反顺序存放数组元素)

```
void main()
```

```
{  
    void invert(int *p, int n);  
    int a[10], i, *p;  
    cout<<"Input ten interger:"<<endl;  
    for(i=0; i<10; i++)    cin>>a[i];  
    p=a;    //p=&a[0];  
    invert(p, 10);    //函数调用方式1, 用指向数组的指针  
    for(i=0; i<10; i++)    cout<<a[i]<<endl;  
}
```

1、将一组同类型的数据（数组）从一个函数传递到另一个函数，可以采用数组名作为函数参数，也可以采用指向数组的指针变量作为函数参数

2、当函数的形参为指向数组的指针时，函数的实参既可以是数组名，也可以是指向数组起始地址的指针变量

```
void main( )
```

```
{  
    void invert(int *p, int n);  
    int a[10], i;  
    cout<<"Input ten interger:"<<endl;  
    for(i=0; i<10; i++)    cin>>a[i];  
    invert(a, 10);    //函数调用方式2, 用数组名  
    for(i=0; i<10; i++)    cout<<a[i]<<endl;  
}
```

```
void invert(int *p, int n)  
{  
    int i, j, temp;  
    for(i=0, j=n-1; i<j; i++, j--)  
    {  
        temp=*(p+i);  
        *(p+i)=*(p+j);  
        *(p+j)=temp;  
    }  
}
```

```
void main()
```

```
{
```

```
void invert(int *p, int n):
```

```
int a[10], i, *p;
```

```
cout<<"Input ten i
```

```
for(i=0; i<10; i++)
```

```
p=a; //p=&a[0];
```

```
invert(p, 10);
```

```
for(i=0; i<10; i++)
```

```
cout<<a[i]<<endl;
```

```
}
```

主函数执行到函数调用语句，
系统会做两件事：

- 1、把实参的值传递给形参；
- 2、转而执行子函数。

```
void invert(int *p, int n)
```

```
{
```

```
int i, j, temp;
```

```
for(i=0, j=n-1; i<j; i++, j--)
```

```
{
```

```
temp=*(p+i);
```

```
*(p+i)=*(p+j);
```

```
*(p+j)=temp;
```

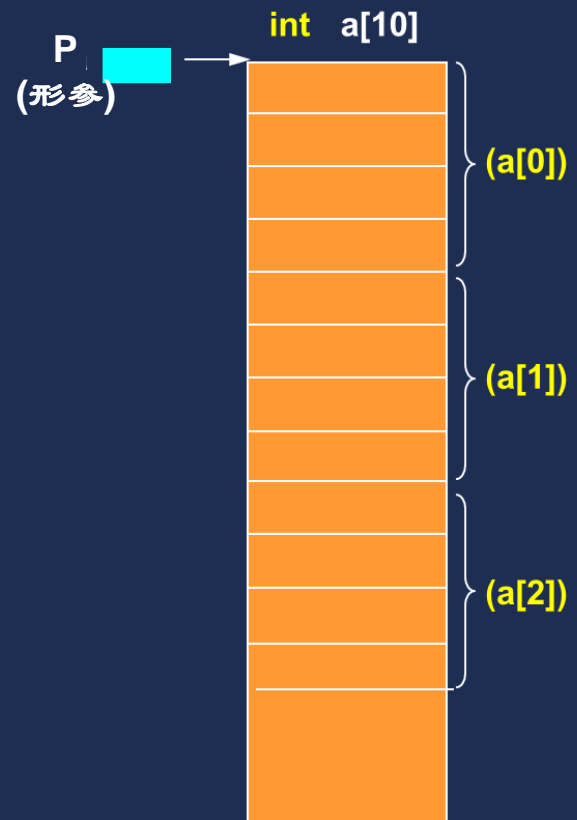
```
}
```

```
}
```

形参从实参获取数组a的首
地址，即形参p指向数组a
在内存中的开始位置。

2、利用形参p可以操作数
组a的所有元素。

3、当子函数执行完后，数
组a的所有元素均已改变。



5.1 简单变量与指针 P_{186~189}

使用指针变量与使用一般变量一样，一定要先定义后使用，使用前，指针变量一定要有明确的指向（即指向某个内存单元地址）例如，`int *p=&x;` p明确指向变量x（即P中存储的是变量x的地址）。

示例-阅读程序

```
#include<iostream>
using namespace std;
void main( )
{
    int x,*p;    //定义整型指针p
    p=&x ;      //将x在内存中的地址赋给指针变量p
    cin>>*p;    //从键盘接收指针变量p所指向上的变量x的值
    cout<<*p<<endl; //输出指针变量p所指向上的变量x的值
}
```

```
#include<iostream>
using namespace std;
void main( )
{
```

```
    int a=3;
```

```
    int *pa=&a;           //定义整型指针pa，并初始化为变量a的地址
```

```
    int *p;               //定义整型指针p
```

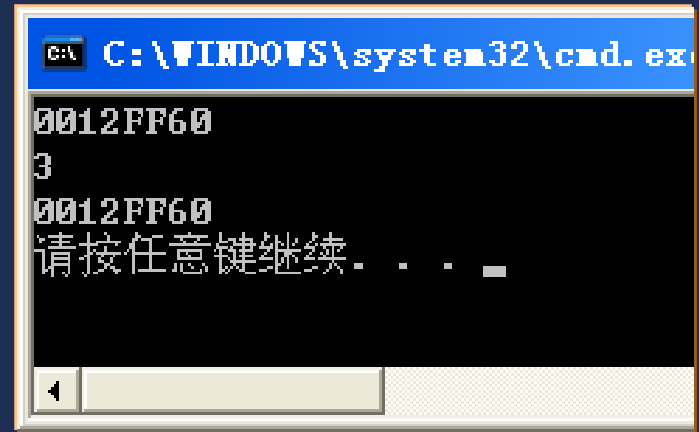
```
    p=pa ;                //将pa赋值给p，即pa和p的值都为变量a的地址值
```

```
    cout<<pa<<endl;      //输出变量a的地址值
```

```
    cout<<*pa<<endl;     //输出指针变量pa所指向的变量a的值
```

```
    cout<<p<<endl;       //输出变量a的地址值
```

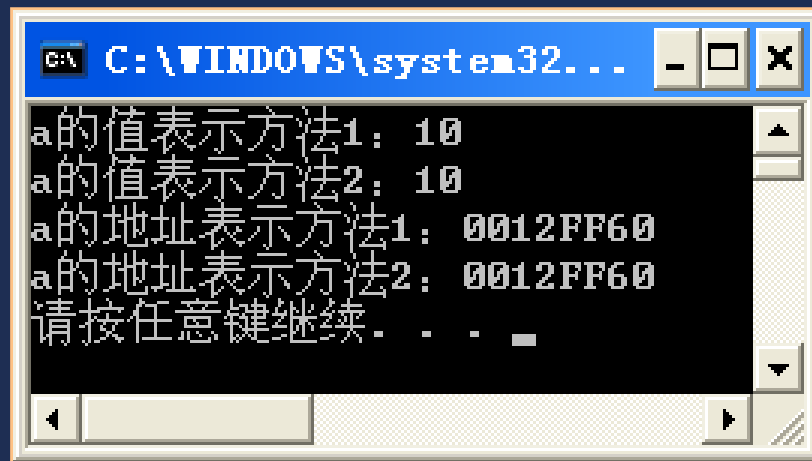
```
}
```



```
C:\WINDOWS\system32\cmd.exe
0012FF60
3
0012FF60
请按任意键继续...
```

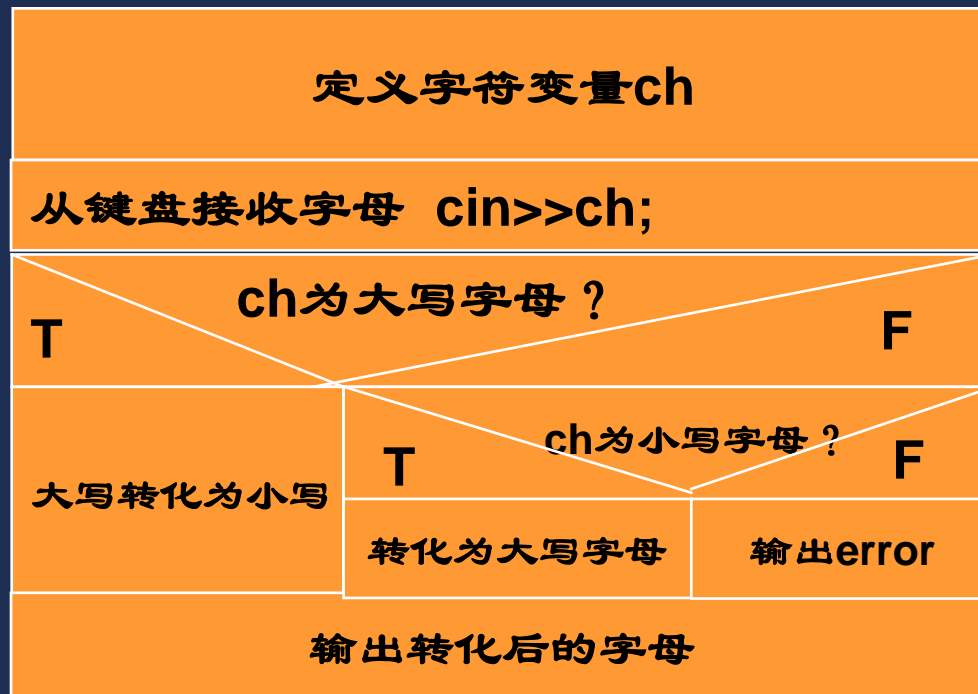
```
#include<iostream>
using namespace std;
void main( )
```

```
{
    int a;
    int *p; //定义一个整型的指针变量p
    a=10;
    p=&a; //将变量a的地址赋给指针变量p
    cout<<"a的值表示方法1: "<<a<<endl;
    cout<<"a的值表示方法2: "<<*p<<endl;
    cout<<"a的地址表示方法1: "<<&a<<endl;
    cout<<"a的地址表示方法2: "<<p<<endl;
}
```

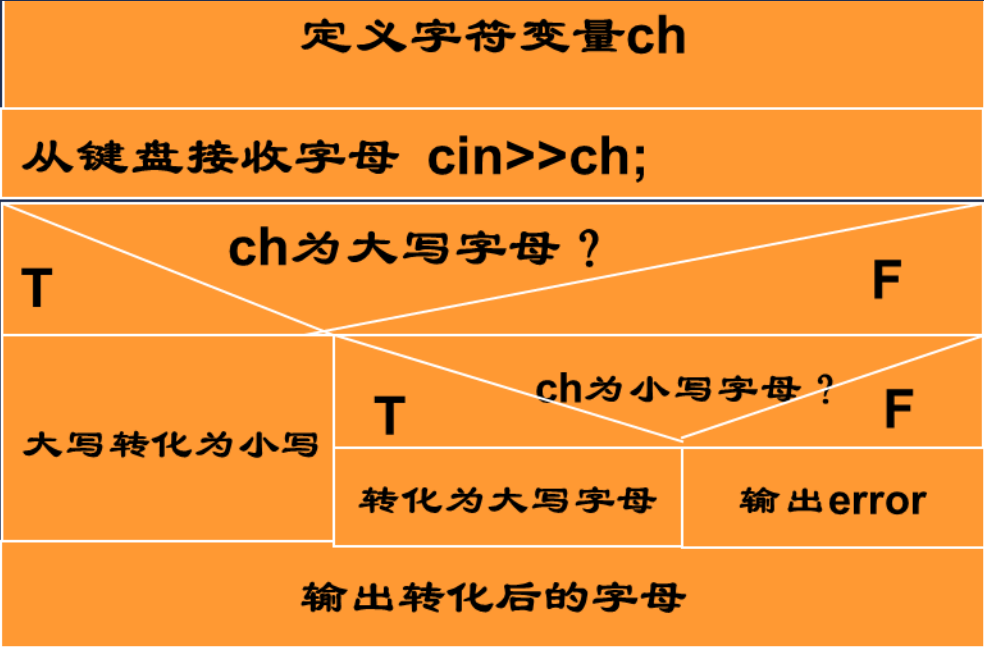
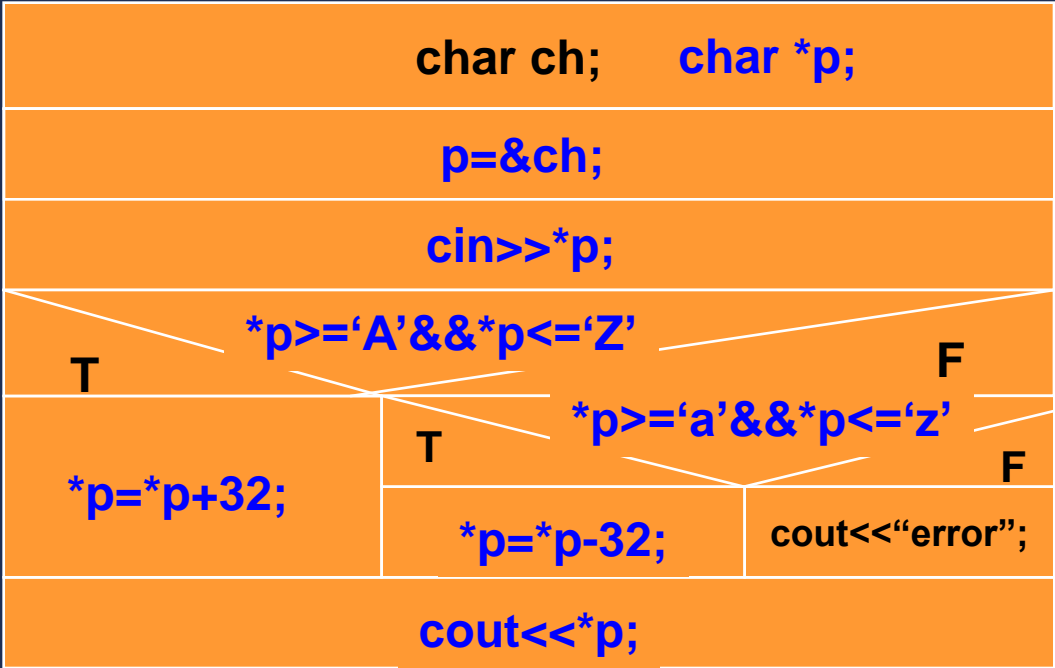


示例

- **运用指针方法**编程实现：从键盘输入一个字母，判断该字母如果是大写字母就转化成其对应的小写字母，如果是小写字母就转化成其对应的大写字母，并将结果输出显示在屏幕上。
- **思路分析：**大写字母ch转化为小写字母 $ch=ch+32$ 。
小写字母ch转化为大写字母： $ch=ch-32$



运用指针进行操作



```
#include <iostream>
#include <iomanip>
using namespace std;
void main()
```

```
{
```

```
    char ch,*p;
```

```
    p=&ch;
```

```
    cout<<"input a charactor:"<<endl;
```

```
    cin>>*p;
```

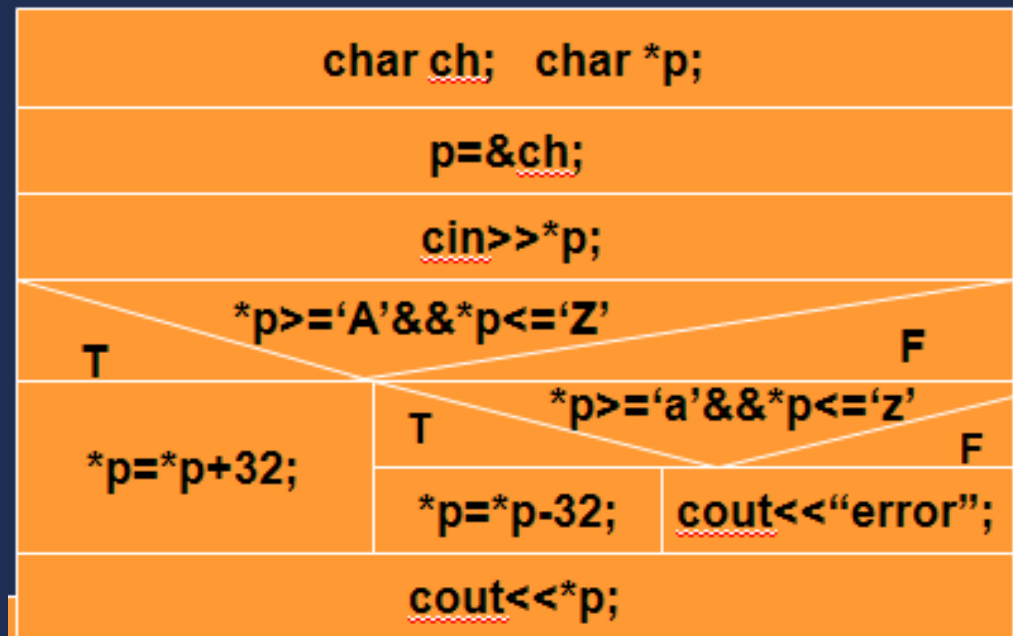
```
    if(*p>='A'&&*p<='Z') *p=*p+32;
```

```
    else if(*p>='a'&&*p<='z') *p=*p-32;
```

```
        else cout<<"Input a error character!"<<endl;
```

```
    cout<<"The character after changed is "<<*p<<endl;
```

```
}
```



指针变量的关系运算 P₁₉₀

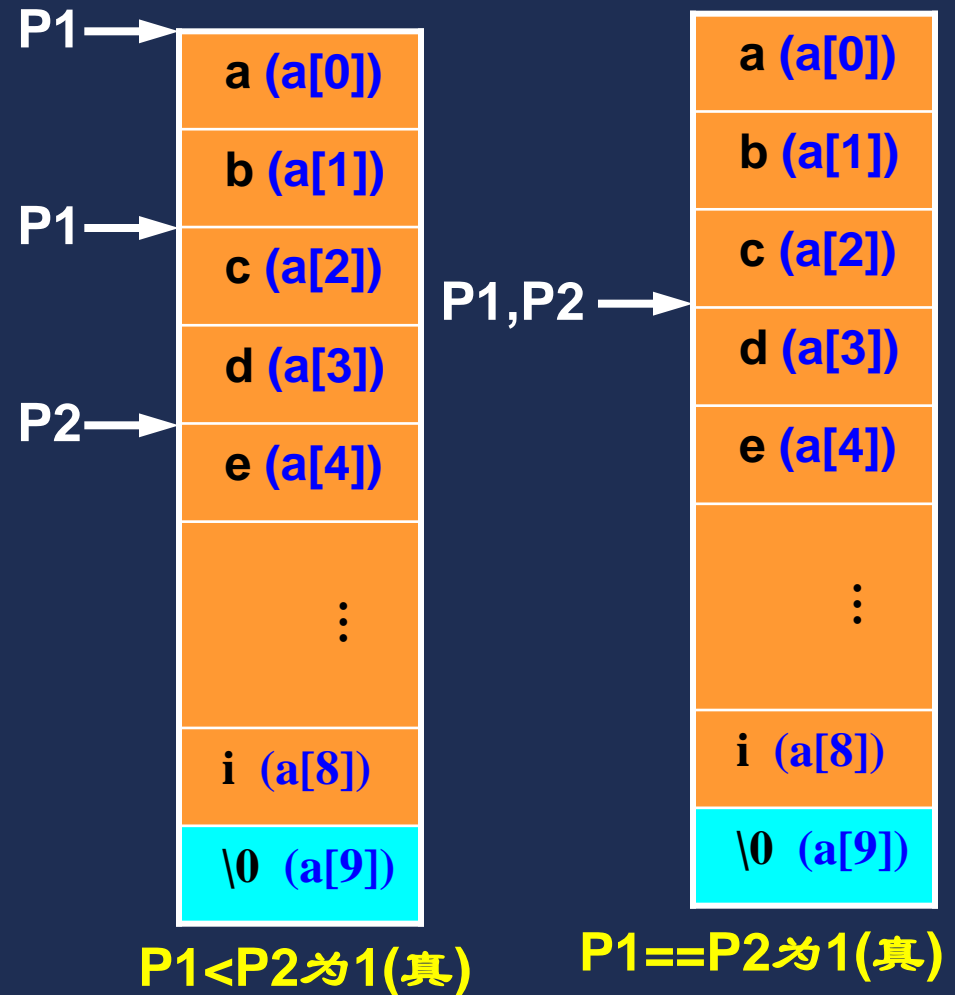
两个指针变量指向同一个数组中的元素时，其关系运算的结果表明了这两个指针变量所指向的数组元素的先后关系。

- 若 $p1 == p2$ ；表明 $p1$ 和 $p2$ 指向数组中的同一个数组元素；
- 若 $p1 < p2$ ；表明 $p1$ 所指向的数组元素在 $p2$ 所指向的数组元素前面；
- 若 $p1 > p2$ ；表明 $p1$ 所指向的数组元素在 $p2$ 所指向的数组元素后面。
- 指针可以和 0 之间进行等于或不等于的关系运算。
例如： $p == 0$ 或 $p != 0$

```

#include <iostream>
using namespace std;
void main()
{
    char a[10]="abcdefghi";
    char *p1,*p2;
    p1=a;
    p1+=2;
    p2=a+4;
    cout<<*p1<<endl;
    cout<<*p2<<endl;
    p1++;
    p2--;
    if (p1==p2)
        cout<<*p1<<endl;
    else
        cout<<"p1和p2没有指向数组中的同一个元素" <<endl;
}

```

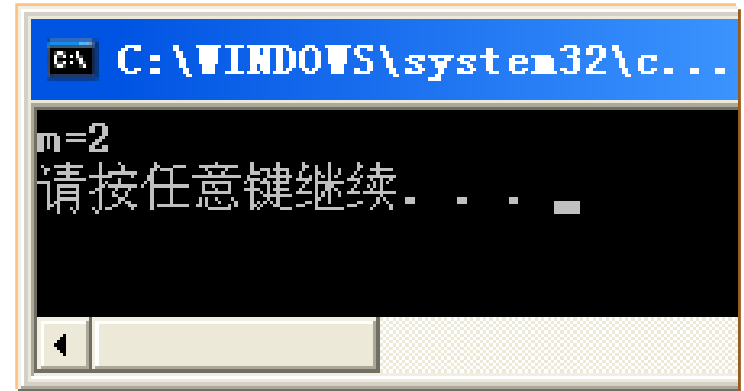


P₁₉₂ 例5.2

```
#include <iostream>
using namespace std;
void main ( )
{
    char *p, sa[20],sb[20];
    int i;
    strcpy_s(sa,"Today is Sunday.");           //字符串数组函数
    for (p=sa,i=0;*p!='\0';i++,p++)          //sa未到结束符即处理
        if(*p==' ') sb[i]='_';              //将空格用下划线替代
        else sb[i]=*p;                      //将指针指向的元素赋给sb中对应元素
    sb[i]='\0';                               //在sb中 写入结束符构成字符串数组
    p=sb;                                       //数组名为数组在内存中的首地址，指针p指向字符串数组sb
    cout<<*p<<endl;                          //输出数组首元素的值
    cout<<p<<endl;                          //输出指针p指向的字符串数组的内容
}
```

P₂₀₆ 例5.7

```
#include <iostream>
#include <iomanip>
using namespace std;
void main( )
{
    const int N=10;
    int b=0,t=9,m=0,x=3;
    int a[N]={-5,4,3,-2,-1};
    int *p=a;
    while(b<=t)
    {
        m=(t-b)/2;
        if(x==*(p+m)) break;
        else if(x>*(p+m))    b=m+1;
        else    t=m-1;
    }
    cout<<"m="<<m<<endl;
}
```



插入

P₂₀₆

例5.7

x=11

2	4	6	8	
---	---	---	---	--

↑ ↑ ↑ ↑
p=0 p=1 p=2 p=3 p=4

$x > a[p]$ $x > a[p]$ $x > a[p]$ $x < a[p]$

算法关键：

1. 如何确定正确的插入位置？
2. 如何把插入位置“空出来”？
3. 如何“插入”？

2	4	6	8		12	16	17	20	30	40
---	---	---	---	--	----	----	----	----	----	----

2	4	6	8	X	12	16	17	20	30	40
---	---	---	---	---	----	----	----	----	----	----

```
#include <iostream>
#include <iomanip>
using namespace std;
void main()
```

可以用如下的四种方法来操作数组。

- 使用数组名和下标(**a[i]**)
- 使用指针变量的下标表示法(**pa[i]**)
- 使用数组名和指针运算(***(a+i)**)
- 使用指针变量(***(pa+i)**)

```
{
    const int N=10;
    int a[N+1],p,x,i;
    cout<<"输入a数组: " <<endl;
    for(i=0;i<N;i++)    cin>>a[i];
    cout<<"输入待插入的数x: " <<endl;
    cin>>x;
    p=0;
    while(x>a[p]&&p<n)    p++; //找到x应插入的正确位置
    for(i=n-1;i>=p;i--)
        a[i+1]=a[i]; //将a[p]~a[n-1]后移
    a[p]=x; //x插入正确位置
    for(i=0;i<=n;i++)
        cout<<setw(3)<<a[i]<<endl;
}
```

```
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
    const int N=10;
    int a[N+1],p,x,*t ;    //定义指针变量
    cout<<“输入a数组： ” <<endl;
    for(t=a;t<a+N;t++)
        cin>>*t;          //用指针变量输入数组元素
    cout<<“输入待插入的数x： ” <<endl;
    cin>>x;
```

```

p=a;    //回溯指针
j=0;
while (x>*(p+j)&& j<N)
    j++;
for (i=n-1; i>=j; i--)
    *(p+i+1)=*(p+i);
*(p+j)=x; //在插入位置插入数据
for (p=a; p<=a+N; p++)
    cout<<setw(3)<<*p;
}

```

```

j=0;
while(x>a[j]&& j<n)
    j++; //①确定插入位置
for(i=n-1; i>=j; i--)
    a[i+1]=a[i]; //②空出插入位置
a[j]=x; //③插入数据
for(i=0; i<=n; i++)
    cout<<setw(3)<<a[i]; //输出
}

```

```

for (p=a+n-1; p>=a+j; p--)
    *(p+1)=*p; // ②空出插入位置
p=a; //回溯指针
*(p+j)=x; // ③在插入位置插入数据

```


示例5.11 P_{214}

从键盘输入整数集合a、b的元素个数和各个元素的值，计算并输出其**交集**。

思路分析：整数集合a、b分别用两个整型数组a和b来表示，由于集合a、b的个数需从键盘输入，**无法确定整型数组a和b的大小**，所以只有**先定义数组的长度足够大**，再根据具体的输入使用其中一部分。定义整型数组c来存放a和b的交集，**数组c的大小可取a和b中较小者**。

求交集示例图

```
int *pa,*pb,*pc;
```



↑
Pa



↑
Pb

↑
Pb

↑
Pb

↑
Pb



↑
Pc

求交集示例图



↑
Pa

```
int *pa,*pb,*pc;
```



↑
Pb



↑
Pc

求交集示例图



```
int *pa,*pb,*pc;
```



求交集示例图



↑
Pa

```
int *pa,*pb,*pc;
```



↑ ↑ ↑ ↑
Pb Pb Pb Pb



↑
Pc

求交集示例图



Pa

```
int *pa,*pb,*pc;
```



Pb

Pb

Pb

Pb

Pb

Pb



Pc

思路分析

求交集其实就是要求出a和b中的公共元素：

(1) 首先取出a中的第1个元素与b中元素从第1个开始进行比较，会有以下两种情况：

①若相等，表明是a和b中都有的元素，应该放入结果数组C中，此时数组C的长度加一，操作数组C的指针变量指向C的下一个元素。接着取出a中的下一个元素再与b中的元素从第1个开始进行比较。

②若不相等，则将a的第1个元素与b的下一个元素进行比较，若相等，重复①操作。若不相等，重复②操作，直到b中每一个元素都比较完成。

(2) 取出a中的下一个元素与b的元素从第一个开始进行比较，重复上述①②操作，直到a中的每一个元素都比较完毕。

程序代码

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
void main(void)
```

```
{
```

```
    const int M=20,N=10;
```

```
    int a[M],b[N],c[N];
```

```
    int m,n,f=0,*pa,*pb,*pc;
```

```
/* m、n为数组a、b的实际长度；f记录数组c实际长度；pa、  
    pb、pc为指向数组a、b、c的指针 */
```



```
cout<<"输入数组a中元素的个数："<<endl;
cin>>m;
cout<<"输入数组a的元素："<<endl;
for(pa=a;pa<a+m;pa++)
    cin>>*pa;    //用指针变量进行数组元素的输入
cout<<"输入数组b中元素的个数："<<endl;
cin>>n;
cout<<"输入数组b的元素："<<endl;
for(pb=b;pb<b+n;pb++)
    cin>>*pb;    //用指针变量进行数组元素的输入
```

//将a, b中所有因素进行比较

```
for( pa=a,pc=c;pa<a+m;pa++) //依次比较a中所有元素
    for( pb=b;pb<b+n;pb++) //用b的每一个元素和a的当前元素进行比较
        if(*pa==*pb )
        {
            *pc++=*pa;
            f++;
            break;
        }

cout<<"交集c的各个元素为:"<<endl;
for (pc=c;pc<c+f;pc++)
    cout<<setw(3)<<*pc<<" ";
cout<<endl;
}
```

```
f=0;
for(i=0; i<m; i++)
    for(j=0; j<n; j++)
        if(a[i]==b[j])
        {
            c[f]=a[i];
            f++;
            break;
        }
```

后移
度
比较