C++初学者必看的50个建议

- 38.不要漏掉书中任何一个练习题——请全部做完并记录下解题思路；
- 39.C++语言和C++的集成开发环境要同时学习和掌握；
- 40.既然决定了学C++,就请坚持学下去，因为学习程序设计语言的目的是掌握程序设计技术，而程序设计技术是跨语言的；
- 41.就让C++语言的各种平台和开发环境去激烈的竞争吧，我们要以学习C++语言本身为主；

C++初学者必看的50个建议

- 42.当你写C++程序写到一半却发现自己用的方法很拙劣时，请不要马上停手；请尽快将余下的部分粗略的完成以保证这个设计的完整性，然后分析自己的错误并重新设计和编写；

- 44.决不要因为程序“很小”就不遵循某些你不熟练的规则——好习惯是培养出来的，而不是一次记住的；

- 45.每学到一个C++难点的时候，尝试着对别人讲解这个知识点并让他理解——你能讲清楚才说明你真的理解了；

C++初学者必看的50个建议

- 46.记录下在和别人交流时发现的自己忽视或不理解的知识点；
- 47.请不断的对自己写的程序提出更高的要求,哪怕你的程序版本号会变成Version 100.XX；
- 48.保存好你写过的所有的程序——那是你最好的积累之一；
- 49.请不要做浮躁的人；
- 50.请热爱C++!

第6章 指针

6.1 指针的概念

6.2 变量与指针

6.3 数组与指针

6.4 字符串与指针

6.5 函数与指针

6.6 返回指针值的函数

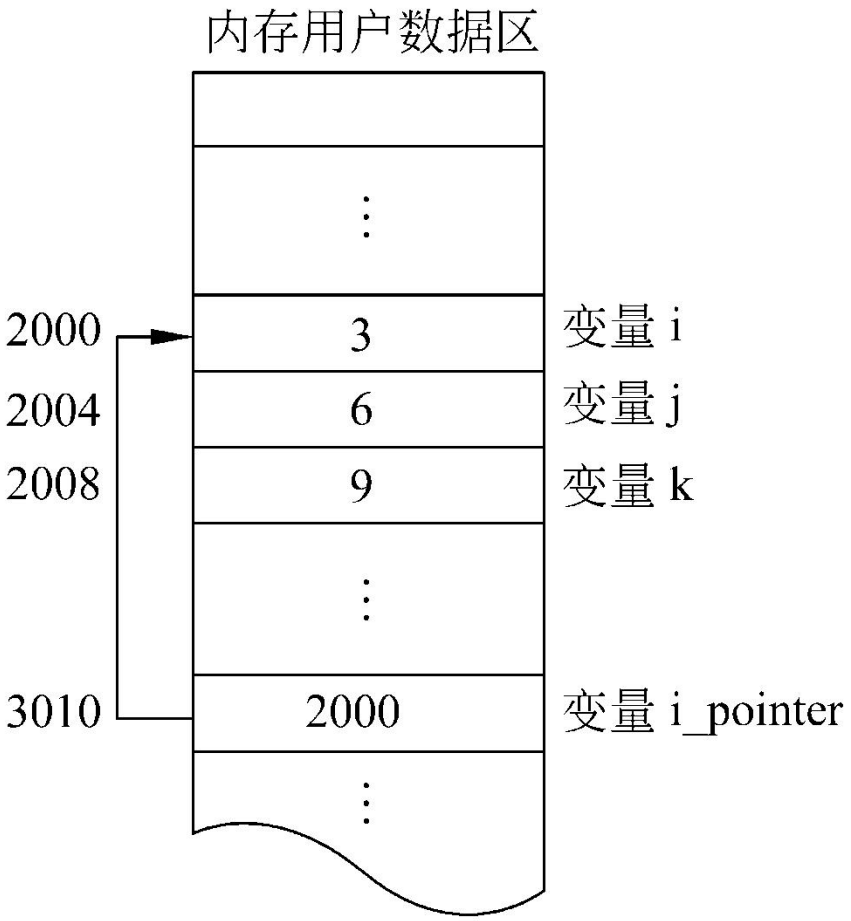
6.7 指针数组和指向指针的指针

6.8 有关指针的数据类型和指针运算的小结

*6.9 引用

6.1 The concept of Pointer

- **Definition**
  - **Pointer：** Address of other variable used to access the variable indirectly.
  - **Pointer variable：** An variable used to store address



一个变量的地址称为该变量的指针。

指针变量用来存放另一变量地址；

指针变量的值（即指针变量中存放的值）是地址（即指针）。

直接存取（直接访问）方式

通过变量名（对应变量的地址）对变量值进行存取。

间接存取（间接访问） 方式

通过指针变量对变量值进行存取。

6.2 Variables and Pointers

有两个与指针变量有关的运算符：

(1) & 取地址运算符（*ampersand*）address of 。

(2) * 指针运算符（ dereference operator，称间接访问运算符）。

例如： &a为变量a的地址，*p为指针变量p所指向的存储单元。

Declaration & Reference

Declaration：

For example： int i;

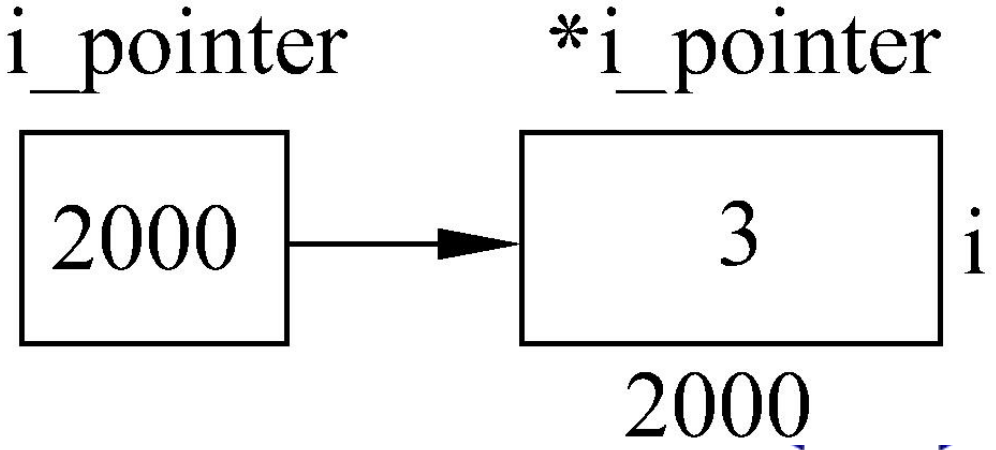
int *i_pointer=&i;//此处*为指针变量定义符号

Reference：

Example1： i=3;

Example2： *i_pointer=3;

//此处*为指针间接引用符号



定义指针变量的一般形式为

基类型 *指针变量名；

```
float *pointer_3;
// pointer_3是指向单精度型数据的指针变量
char *pointer_4;
// pointer_4是指向字符型数据的指针变量
```

注意：“*”不是指针变量名的一部分。

将被指向的变量的地址赋给指针变量，即可使一个指针变量指向另一个变量。例如：

```
pointer_1=&i;        //将变量i的地址存放到指针变量pointer_1中
pointer_1=&j;        //将变量j的地址存放到指针变量pointer_1中
```

Initialization of pointer variable

- Syntax Form
- 存储类型 数据类型 *指针名＝初始地址；
- For example： int *pa=&a;
- Note:
- (1)指针必须初始化，不能用一个整数给一个指针变量赋初值。但可以赋值为整数0，表示空指针。
- (2) 在定义指针变量时必须指定基类型。
- (3)用变量地址作为初值时，该变量必须在指针初始化之前已说明过，且变量类型应与指针类型一致。
- (4)可以用一个已赋初值的指针去初始化另一 个指针变量。
- (5)不要用一个内部 auto 型变量去初始化 static 型指针。

- 一般的C++编译系统为每一个指针变量分配4个字节的存储单元，用来存放变量的地址。
- 64位CPU可以支持更大的内存寻址空间。它打破了32位下4GB内存的限制，理论上内存访问空间可以达到1600万（16777216）TB。
- 实际上采用EM64T技术的Intel处理器支持36位物理内存寻址和48位虚拟内存寻址；
- AMD 64位处理器支持40位物理内存寻址和48位虚拟内存寻址；
- Windows XP x64版最高可以支持16GB物理内存和16TB虚拟内存(Vista 128G，Windows 7 192G)，使得应用程序可以快速处理大量数据集合。

一些符号的读法

- &: address of，and，取地址
  *: dereference，星，取数据，乘
- ~ 取反
  # 井
  ^ 异或
  | 位或
  || 逻辑或（或）
  && 并，与
- @ 英文中原意是A-Tail“带尾巴的a”，缩写就是AT：爱特（AT），常用来表示“在”。

例6.1 通过指针变量访问整型变量

```
#include <iostream>
using namespace std;
int main( )
{
    int a,b;                              //定义整型变量a,b
    int *pointer_1,*pointer_2;            //定义指针变量
    a=100;b=10;                           //对a,b赋值
    pointer_1=&a;                  //把变量 a 的地址赋给pointer_1
    pointer_2=&b;                  //把变量 a 的地址赋给pointer_2
    cout<<a<<" "<<b<<endl;              //输出a和b的值
    cout<<*pointer_1<<" "<<*pointer_2<<endl;
    //输出*pointer_1和*pointer_2的值
    return 0;
}
```

Example： 指针的声明、赋值与使用

```
#include<iostream.h>
using namespace std;
int main()
{
    int *i_pointer;       //声明int型指针i_pointer
    int i;                //声明int型数i
    i_pointer=&i;         //取i的地址赋给i_pointer
    i=10;
    cout<<"Output int i="<<i<<endl;
    cout<<"Output int pointer i="<<*i_pointer<<endl;
    return 0;
}
```

对“&”和“*”运算符的说明

设有：pointer_1=&a;

(1) 问&*pointer_1的含义是什么？

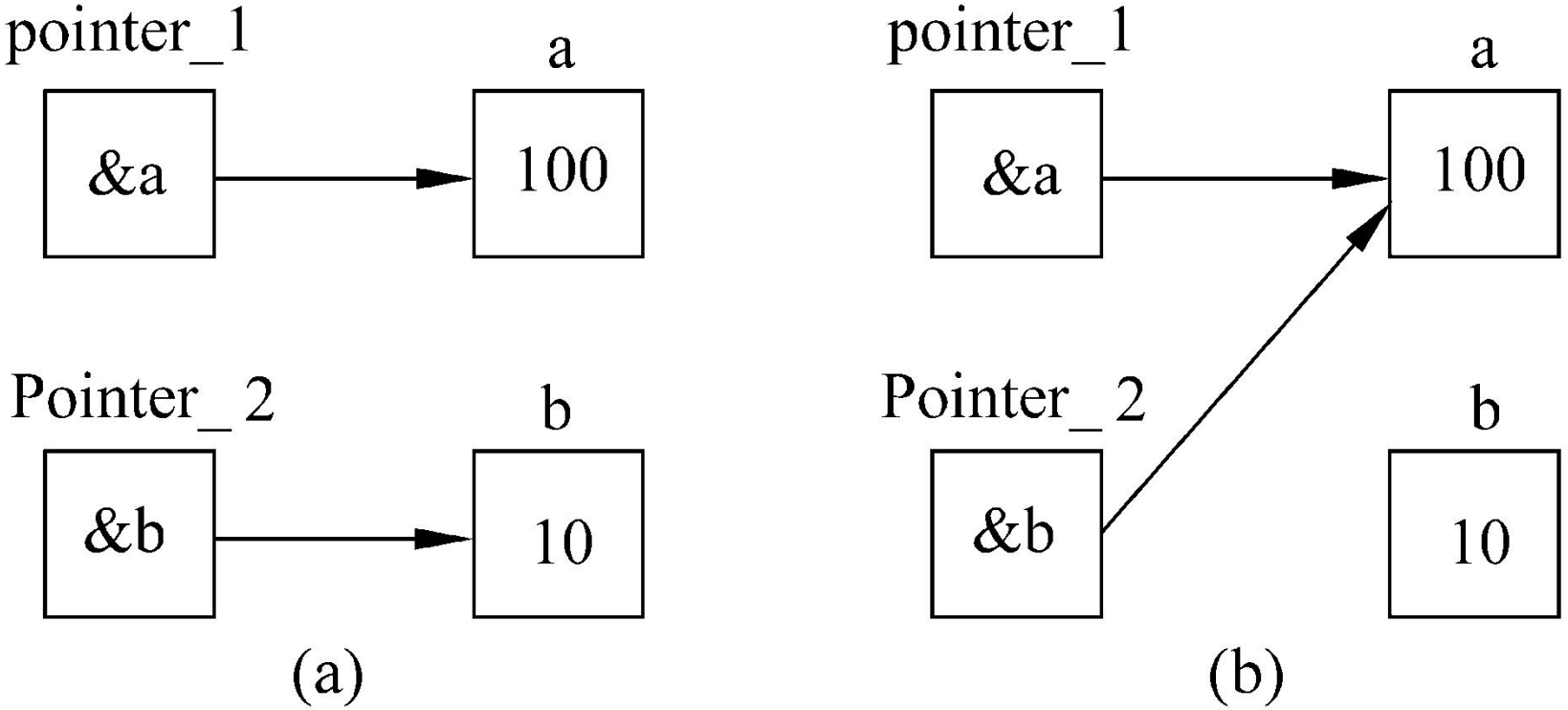
&和*优先级相同，但按自右至左方向结合。

因此先进行*pointer_1运算，它就是变量a;

再执行&运算。

因此，&*pointer_1与&a相同，即变量a的地址。

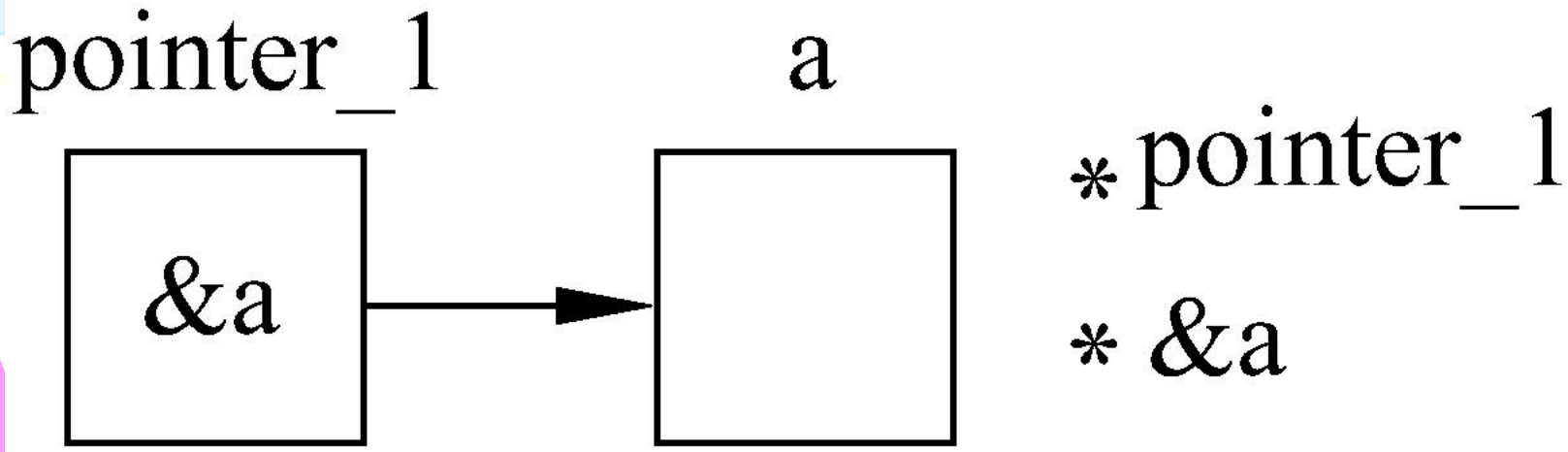
如果有pointer_ 2 =&*pointer_1；
它的作用是将&a(a的地址)赋给pointer_2。



(2) *&a的含义是什么？

先进行&a的运算，得a的地址，再进行*运算，即&a所指向的变量，即*&a与a等价。

*&a和*pointer_1的作用是一样的。



Example：空间实体的理解

```
#include<iostream>
using namespace std;
int main()
{
    float f = 34.5;
    int* ip = reinterpret_cast<int*>(&f);// int* ip = (int*)&f;
    cout<<"float address: "<<&f<<"=>"<<f<<endl;
    cout<<"  int address: "<<ip<<"=>"<<*ip<<endl;
    *ip = 100;
    cout<<"  int: "<<*ip<<endl;
    cout<<"float: "<<f<<endl;
    return 0;
}
```

例6.2 输入a和b两个整数，按先大后小的顺序输出a和b

```
#include <iostream>
using namespace std;
int main( )
{
    int *p1, *p2, *p, a, b;
    cin>>a>>b;
    p1=&a;                          //使p1指向a
    p2=&b;                          //使p2指向b
    if(a<b) {p=p1;p1=p2;p2=p;}
    //p1与p2的值交换
    cout<<“a=“<<a<<“ b=“<<b<<endl;
    cout<<“max=“<<*p1<<“ min=“<<*p2<<endl;
    return 0;
}
```

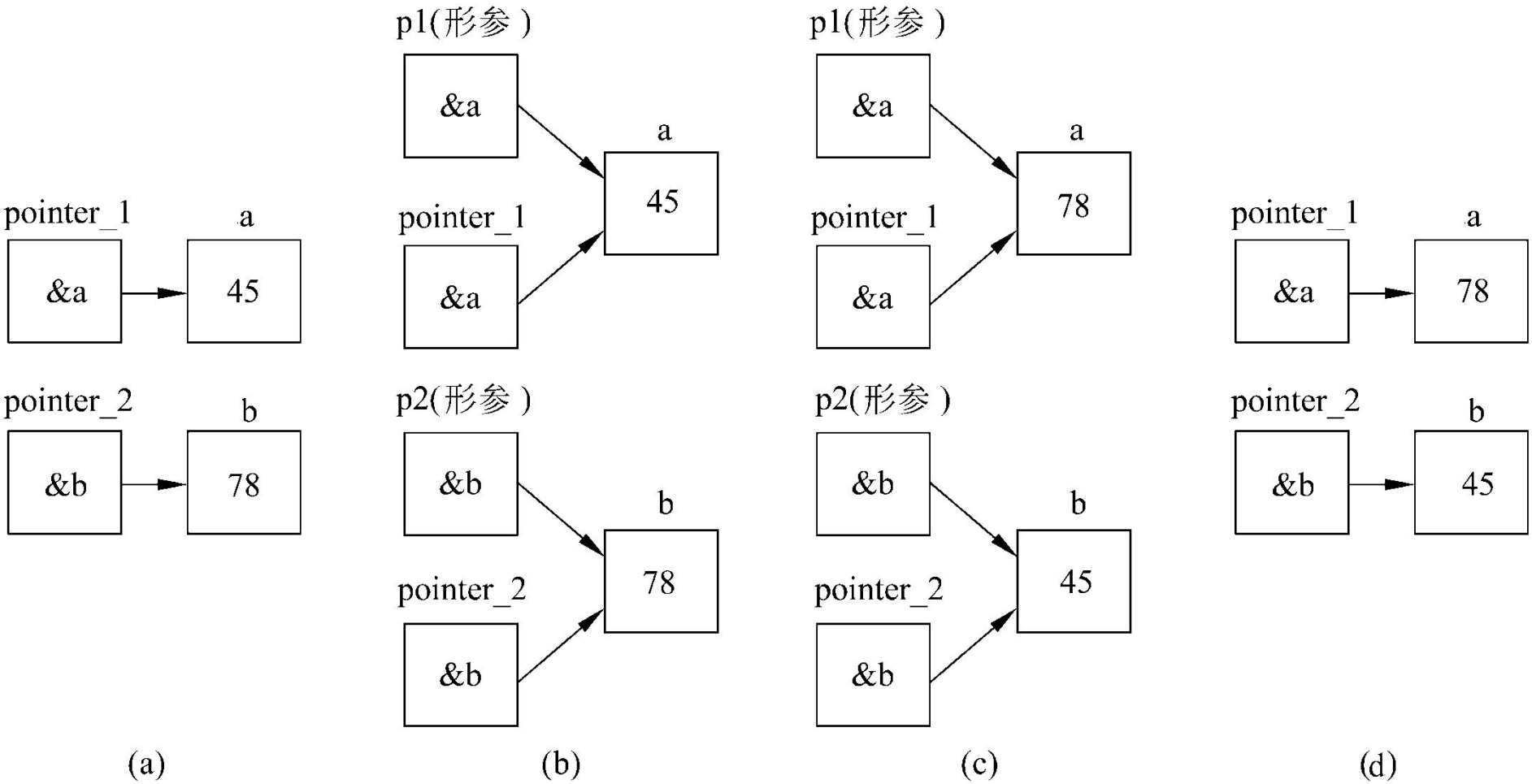
6.2.3 指针作为函数参数

函数的参数可以是指针类型。它的作用是将一个变量的地址传送给被调用函数的形参。

例6.3，同例6.2，要求用函数处理，函数参数为指针。

```
#include  <iostream>
using namespace std;
int main( )
{
    void swap(int *p1,int *p2);     //函数形参为指针
    int *pointer_1,*pointer_2,a,b;
    cin>>a>>b;
    pointer_1=&a;                   //使pointer_1指向a
    pointer_2=&b;                   //使pointer_2指向b
    if(a<b) swap(pointer_1,pointer_2);
    cout<<"max="<<a<<" min="<<b<<endl;
    return 0;
}
```

void swap(int *p1,int *p2)
{



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

*temp虽然指一个内存单元中的值，但temp 未初始化，指向一个不确定的内存单元；程序编译将提出警告，运行则出错。

能否通过调用下面的函数实现a和b互换？

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

虚实结合：函数调用中实参对形参的数据传递是“值传递”，即单向传递，只能从实参向形参传数据，形参值的改变无法反向回传给实参。

- 为了使在函数中改变了的变量值能被main函数所用，不能把要改变值的变量作为参数，而应该用指针变量作为函数参数。
- 在函数执行过程中使指针变量所指向的变量值发生变化，函数调用结束后，这些变量值的变化依然保留下来，这样就实现了“通过调用函数使变量的值发生变化，在主调函数中使用这些改变了的值”的目的。

通过函数调用得到n个要改变的值，可采取以下步骤：

①在主调函数中设n个变量，用n个指针变量指向它们；

②编写被调用函数，其形参为n个指针变量，这些形参指针变量应当与主调函数中的n个指针变量具有相同的基类型；

③在主调函数中将n个指针变量作实参，将它们的值(是地址值)传给所调用函数的n个形参指针变量，这样，形参指针变量也指向这n个变量；

④通过形参指针变量的指向，改变该n个变量的值；

⑤在主调函数中就可以使用这些改变了值的变量。

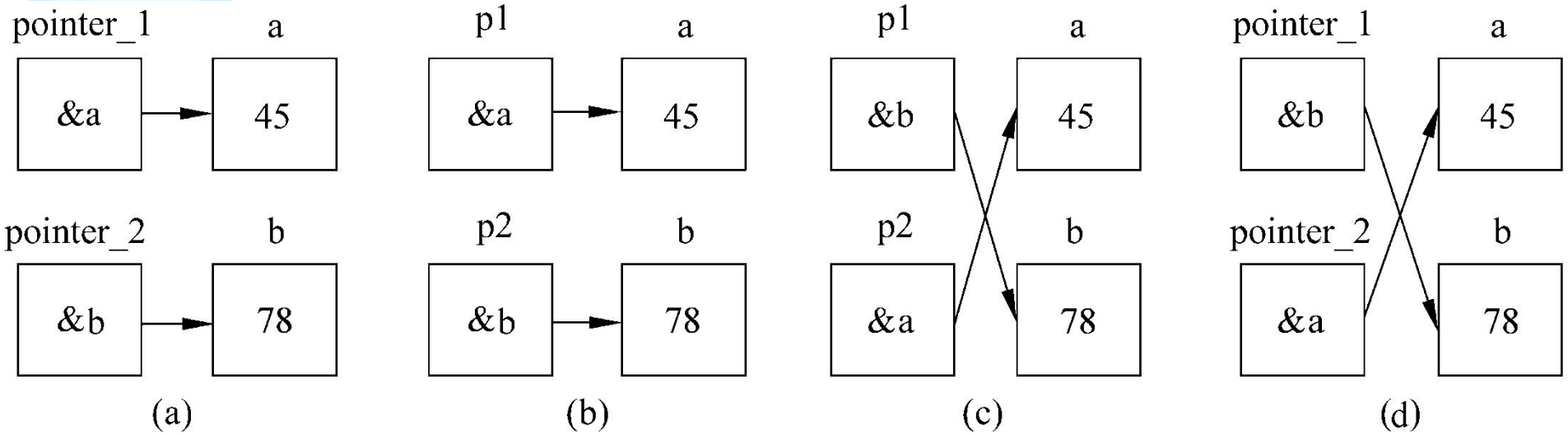
例题2、3的变型：

```
#include  <iostream>
using namespace std;
int main( )
{
    void swap(int *p1,int *p2);
    int *pointer_1,*pointer_2,a,b;
    cin>>a>>b;
    pointer_1=&a;
    pointer_2=&b;
    if(a<b) swap(pointer_1,pointer_2);
    cout<<"max="<<a<<" min="<<b<<endl;
    return 0;
}
```

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

//不能通过改变形参指针变量的值而使实参指针变量的值改变。

实参变量和形参变量之间的数据传递是单向的“值传递”方式。指针变量作函数参数也要遵循这一规则。调用函数时不会改变实参指针变量的值，但可以改变实参指针变量所指向变量的值。



函数的调用可以而且只可以得到一个返回值，即函数值；

使用指针变量作函数参数，就可以通过指针变量改变主调函数中变量的值，相当于通过函数调用从被调用的函数中得到多个返回值。如果不用指针变量是难以做到这一点的。

两数互换

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

```
swap(pt1,pt2);
void swap(int *p1,int *p2)
{
  int *temp;
  temp=p1;
  p1=p2;
  p2=temp;
}
```

两数互换

```
swap(pt1,pt2);
void swap(int *p1,int *p2)
{
  int temp;
  temp=*p1;
  *p1=*p2;
  *p2=temp;
}
```

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

例5.7 多个数交换

```
void select_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]=t;
    }
}
```

例6.4

- 输入a,b,c 3个整数，按由大到小的顺序输出。
- 用上面介绍的方法，用3个指针变量指向3个整型变量，然后用swap函数来实现互换3个整型变量的值。

```
#include <iostream>
using namespace std;
int main( )
{
    void exchange(int *,int *,int *);
    int a,b,c,*p1,*p2,*p3;
    cin>>a>>b>>c;                    //输入3个整数
    p1=&a;
    p2=&b;
    p3=&c;                           //指向3个整型变量
    exchange(p1,p2,p3);              //交换3个整型变量的值
    cout<<a<<" "<<b<<" "<<c<<endl;
}
```

```
void exchange(int *q1,int *q2,int *q3)
{
   void swap(int *,int *);
   if(*q1<*q2) swap(q1,q2);      //将q1与q2所指向的变量的值互换
   if(*q1<*q3) swap(q1,q3);      //将q1与q3所指向的变量的值互换
   if(*q2<*q3) swap(q2,q3);      //将q2与q3所指向的变量的值互换
}
```

```
void swap(int *pt1,int *pt2)
//将pt1与pt2所指向的变量的值互换
{
    int temp;
    temp=*pt1;
    *pt1=*pt2;
    *pt2=temp;
}
```

6.3 数组与指针

6.3.1 Pointers to an array element

```
int a[10];          //定义一个整型数组a
int * p;            //定义一个整型的指针变量p
p=&a[0];            //将a[0]的地址赋给指针p，p指向a[0]
p=a;                //这两句等效
```

在定义指针变量时初始化：

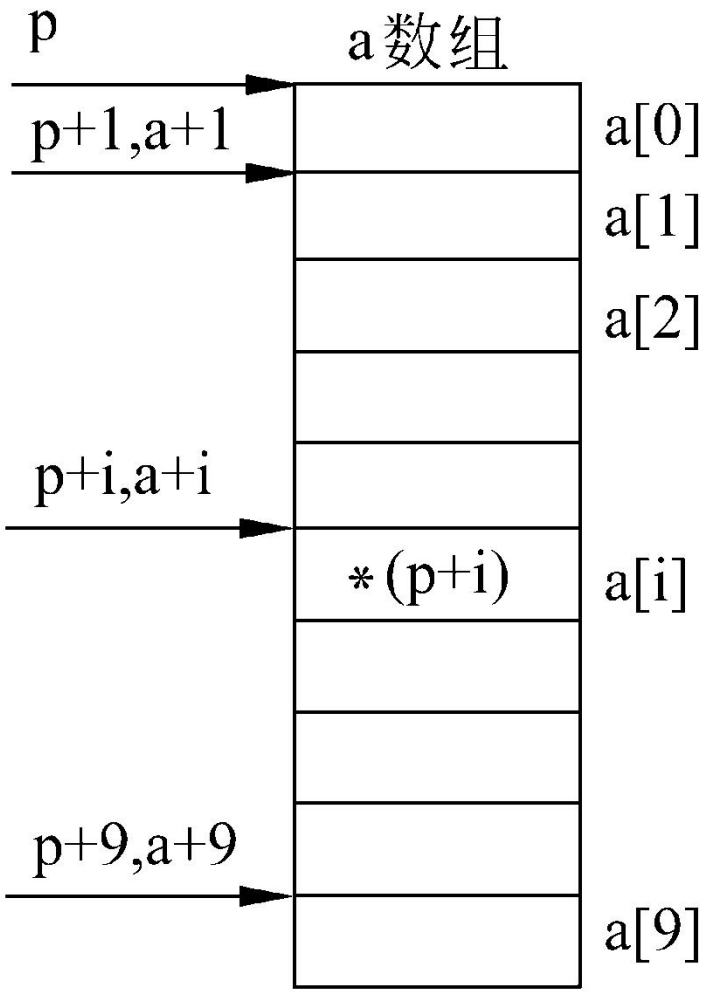
```
int * p=&a[0];
int * p=a;          //这两句等效
```

Reference array elements by pointer

若 p=&a[0]，则：

(1) p+i和a+i就是a[i]的地址。

(2) *(p+i)或*(a+i)是p+i或a+i所指向的数组元素，即a[i]。



Reference array elements by pointer

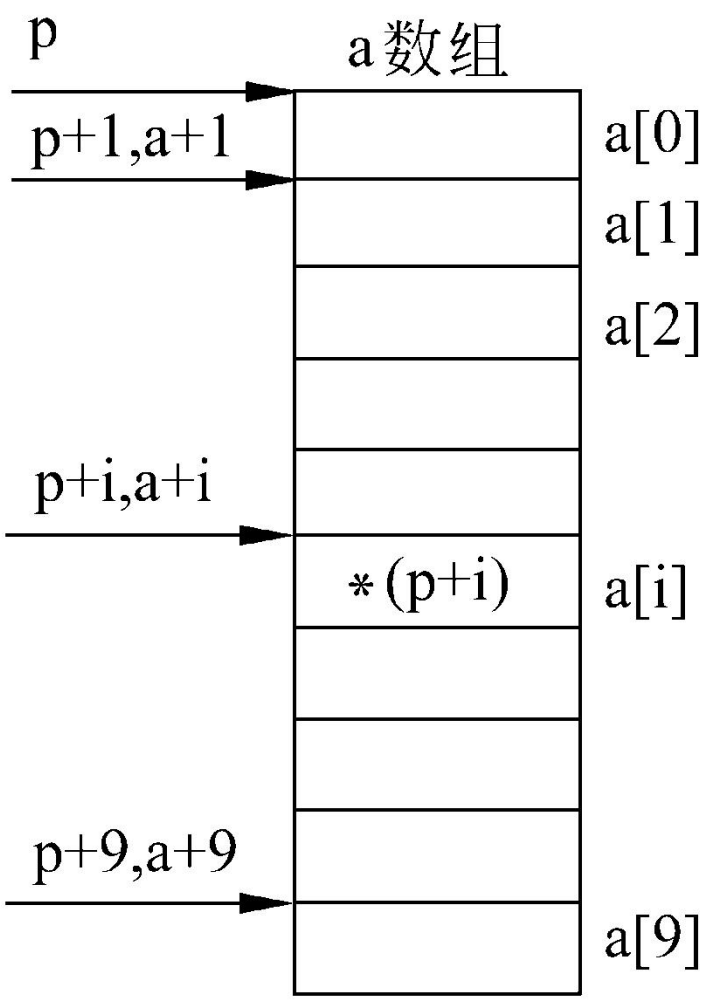
(3) 指向数组元素的指针变量

也可带下标，如

p[i]与*(p+i)等价,进一步有

p[i]与*(a+i)等价

p[i]与a[i]等价



引用数组元素的方法
Reference of array elements

- (1) 下标法
  - a[i]
  - p[i]
- (2) 指针法
  - *(a+i)
  - *(p+i) //又称指针变量法
- 其中a是数组名,p是指向数组a的指针变量。则有p=a，
  (p+i)=(a+i)=a[i]=p[i]

指针的运算
Pointer Operation

如果有p=a,则：

(1) p++（或p+=1）

使p指向下一元素，即a[1]。

(2) *p++

++和*同优先级，结合方向为自右而左，因此它等价于*(p++)。

(3) *(p++)与*(++p)作用不同。

指针的运算
Pointer Operation

(4) (*p)++

表示p所指向的元素值加 1 。

(5) 如果p指向a[i]，

(p--)　先作“”运算，得到a[i]，再使p减1。

*(++p)　先使p自加1，再作*运算，得到a[i+1]。

*(--p)　先使p自减1，再作*运算，得到a[i-1]。

(6) 不能写 a++，因为a是数组首地址是常量。

例6.5 输出数组中的全部元素

```
#include <iostream>
using namespace std;
int main( )
{
    int i,a[10]={1,2,3,4,5,6,7,8,9,10};
    int * p=a;
    for(i=0;i<10;i++)cout<<a[i]<<“ ”;                    //下标法
    cout<<endl;
    for(i=0;i<10;i++)cout<<*(a+i)<<" ";                  //指针法
    cout<<endl;
    for(p=a;p<(a+10);p++) cout<<*p<<“ ”;                 //指针变量法
    cout<<endl;
    return 0;
}
```

注意

指针变量p可以指向有效的数组元素，也可以指向数组以后的内存单元。

如

```
int a[10],*p=a;          //指针变量p的初值为&a[0]
cout<<*(p+10);           //要输出a[10]的值
```

在使用指针变量指向数组元素时，应切实保证指向数组中有效的元素。

将一个数组逆序输出

```
#include <iostream>
#include <iomanip>
using namespace std;
const int N=5;
int main()
{
        int a[N]={9,6,5,4,1},i,temp;
        cout<<"\nOriginal array:\n";
        for(i=0;i<N;i++) cout<<setw(4)<<a[i];
        for(i=0;i<N/2;i++)
        {
                temp=a[i];
                a[i]=a[N-i-1];
                a[N-i-1]=temp;
        }
        cout<<"\nSorted array:\n";
        for(i=0;i<N;i++) cout<<setw(4)<<a[i]; cout<<"\n\n";
        return 0;
}
```

将一个数组逆序排列后输出-指针法

```
#include <iostream>
#include <iomanip>
using namespace std;
const int N=5;
int main()
{
        int a[N]={9,6,5,4,1},i,j,temp;
        cout<<"\nOriginal array:\n";
        for(i=0;i<N;i++) cout<<setw(4)<<a[i];
        i=0;j=N-1;
        while(i<j)
        {
                int t;
                t=*(a+i);
                *(a+i)=*(a+j);
                *(a+j)=t;
                i++; j--;
        }
        cout<<"\nSorted array:\n";
        for(i=0;i<N;i++) cout<<setw(4)<<a[i]; cout<<"\n\n";
        return 0;
}
```

6.3.2 用指针变量作函数参数接收数组地址

Using a pointer variable as a function parameter

用指针变量作函数形参，同样可以接收从实参传递来的数组首元素的地址(此时，实参是数组名)。

例6.6 将10个整数按由小到大的顺序排列

```
#include <iostream>
using namespace std;
int main( )
{
    void select_sort(int *p,int n);          //函数声明
    int i, a[10]={10,9,8,7,6,5,4,3,2,1};
    select_sort(a,10);                       //函数调用，数组名作实参
    cout<<"the sorted array:"<<endl;
    for(i=0;i<10;i++)
        cout<<a[i]<<"  ";
    cout<<endl;
    return 0;
}
```

```
void select_sort(int * p,int n)
//用指针变量作形参
{
    int i,j,k,t;
    for(i=0;i<n-1;i++)
    {
        k=i;
        for(j=i+1;j<n;j++)
        if(*(p+j)<*(p+k)) k=j;
        //用指针法访问数组元素
        t=*(p+k); *(p+k)=*(p+i); *(p+i)=t;
    }
}
```

例6.6与例5.7两个程序在编译以后是完全相同的。

C++编译系统将形参数组名一律作为指针变量来处理。实际上在函数调用时并不存在一个占有存储空间的形参数组，只有指针变量。

实参与形参的结合，有以下4种形式：

实 参	形 参	
数组名	数组名	(如例5.7)
数组名	指针变量	(如例6.6)
指针变量	数组名	
指针变量	指针变量	

实参数组名a代表一个固定的地址，或者说是指针型常量，因此要改变a的值是不可能的。如

```
a++;                    //语法错误，a是常量，不能改变
```

而形参数组名是指针变量，并不是一个固定的地址值。它的值是可以改变的。如

```
f(array[],int n)
{
  cout<<*array;                 //输出array[0]的值
  array=array+3;                //array指向array[3]
  cout<<*array<<endl;           //输出array[3]的值
}
```

6.3.3 多维数组与指针
Multidimensional arrays and pointers

用指针变量也可以指向多维数组中的元素。

1. 多维数组元素的地址

设有一个3行4列的二维数组 a ，

```
int a[3][4]=
{     { 1,  3,  5,  7},
      { 9, 11, 13, 15},
      {17, 19, 21, 23} };
```

a →

a[0]	=	1	3	5	7
a[1]	=	9	11	13	15
a[2]	=	17	19	21	23

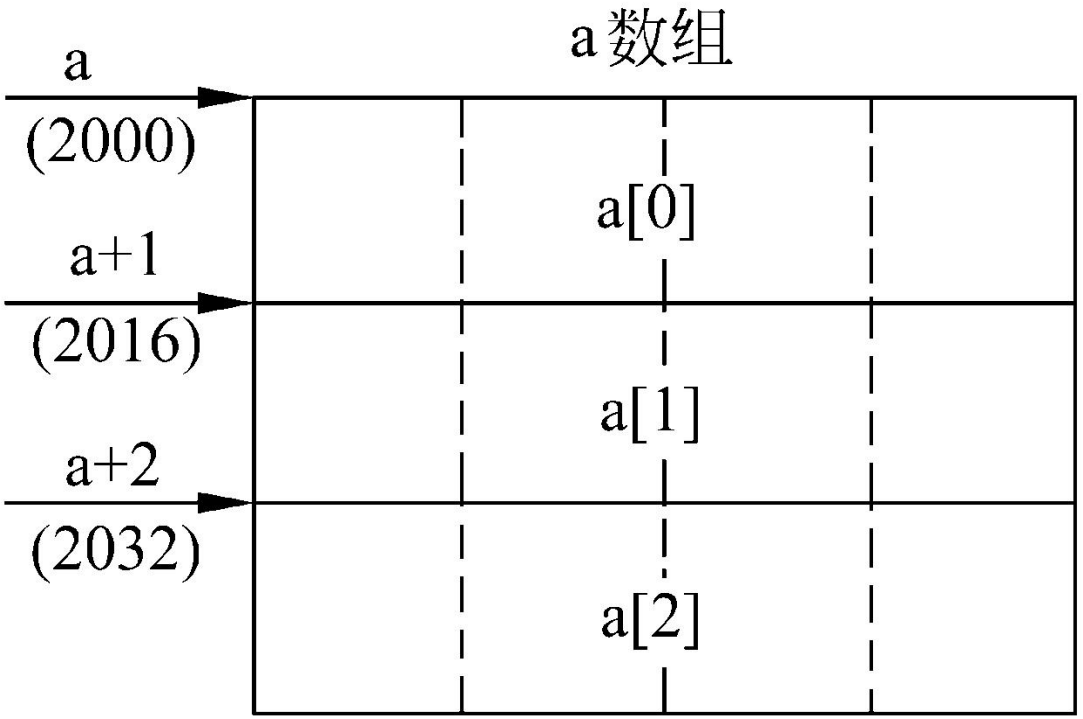
数组a包含3个元素:a[0],a[1],a[2]。

各元素是包含4个元素的一维数组。

a[i]的值

a　代表数组第一个元素即首行的地址　&a[0]，
a+1代表数组第二个元素即第二行的地址&a[1]。
a[0],a[1],a[2] 是一维数组名，
因此

a[0]的值是&a[0][0]，
a[1]的值是&a[1][0]；
a[2]的值是&a[2][0]。



a[i][j]的值

a[0] 的值是 &a[0][0]，

* (a[0]) 的值是 a[0][0]

* (a[0]+1) 的值是 a[0][1]

因为

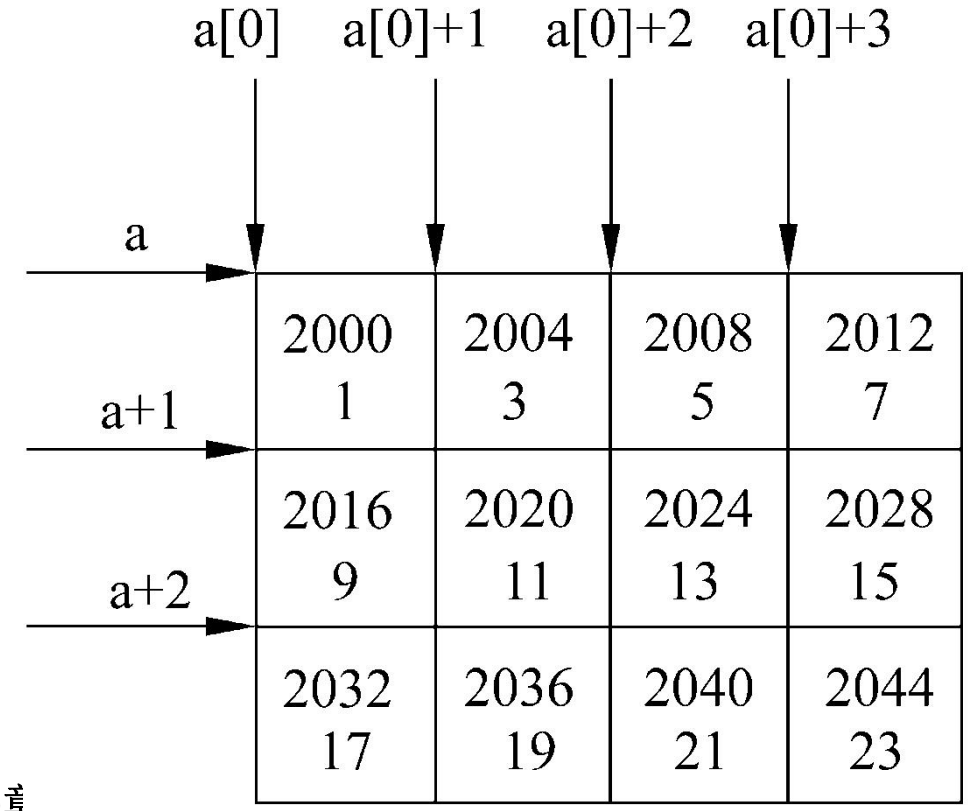
a[0]等于*(a+0)，

a[i]等于*(a+i)，

所以

a[0][1]等于*(*(a+0)+1)，

a[i][j]等于*(*(a+i)+j)。



表示形式	含义
a	二维数组名称，数组首地址
a+i, &a[i]	第 i+1 行的首地址
a[i], *(a+i)	第 i+1 行，第 1 列元素的首地址
a[i]+j, *(a+i)+j, &a[i][j]	第 i+1 行，第 j+1 列元素的首地址
**(a+i), *a[i]	第 i+1 行，第 1 列元素的值
*(a[i]+j), *(*(a+i)+j), a[i][j]	第 i+1 行，第 j+1 列元素的值

数组元素&地址打印

```
#include <iostream>
using namespace std;
int main( )
{
        int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12},i,j;
        cout<<a<<"\n\n";                //打印a
        for(i=0;i<3;i++)                //打印a+i, &a[i]
                cout<<"a+"<<i<<"="<<a+i
                <<",\t&a["<<i<<"]="<<&a[i]<<endl;
        for(i=0;i<3;i++)                //打印a[i], *(a+i)
                cout<<"a["<<i<<"]="<<a[i]
                <<",\t*(a+"<<i<<")="<<*(a+i)<<endl;
        for(i=0;i<3;i++)                //打印a[i]+j, *(a+i)+j, &a[i][j]
                for(j=0;j<4;j++)
                        cout<<"a["<<i<<"]+"<<j<<"="<<a[i]+j<<
                        ",\t*(a+"<<i<<")+"<<j<<"="<<*(a+i)+j<<
                        ",\t&a["<<i<<"]["<<j<<"]="<<&a[i][j]<<endl;
        for(i=0;i<3;i++)                //打印**(a+i), *a[i]
                cout<<"**(a+"<<i<<")="<<**(a+i)<<
                ",\t*a["<<i<<"]="<<*a[i]<<endl;
        for(i=0;i<3;i++)                //打印*(a[i]+j), *(*(a+i)+j), a[i][j]
                for(j=0;j<4;j++)
                        cout<<"*(a["<<i<<"]+"<<j<<")="<<*(a[i]+j)
                        <<",\t*(*(a+"<<i<<")+"<<j<<")="<<*(*(a+i)+j)<<
                        ",\ta["<<i<<"]["<<j<<"]="<<a[i][j]<<endl;
        return 0;
}
```

2. 指向多维数组元素的指针变量
pointer to a multidimensional array elements

(1) 指向数组元素的指针变量

例6.7 输出二维数组各元素的值。

```
#include <iostream>
using namespace std;
int main( )
{
   int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
   int *p;                          //p是基类型为整型的指针变量
   for(p=a[0];p<a[0]+12;p++)
      cout<<*p<<" ";
   cout<<endl;
   return 0;
}
```

p是整型指针变量，

p=a[0];

p=&a[0][0]; //这两句等效

p=a; //错误

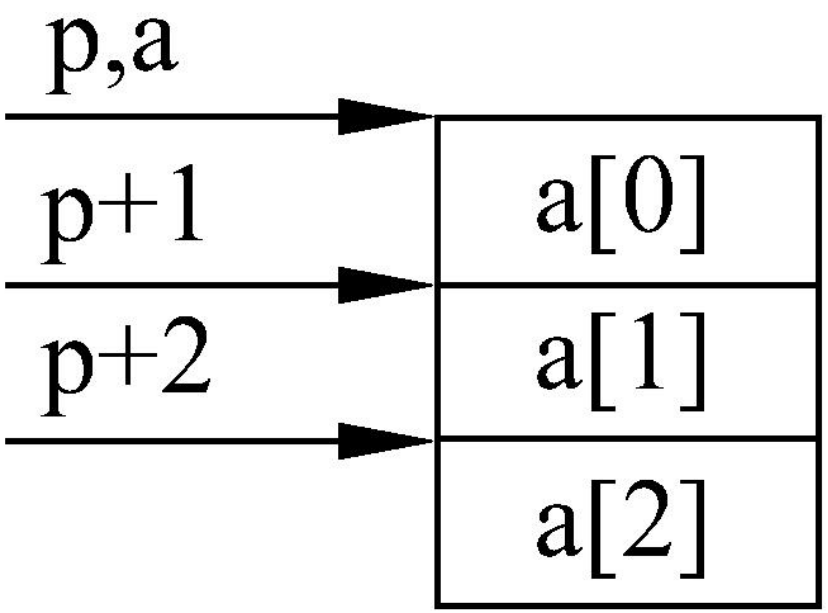
此时p指向a的首元素a[0]，而a[0] 是一维数组，与p的类型不匹配。

循环条件“p<a[0]+12”同样不能写成p<a+12，此时12代表的是12个一维数组。

(2) 指向一维数组的指针变量

该指针变量指向一个包含m个元素的一维数组。

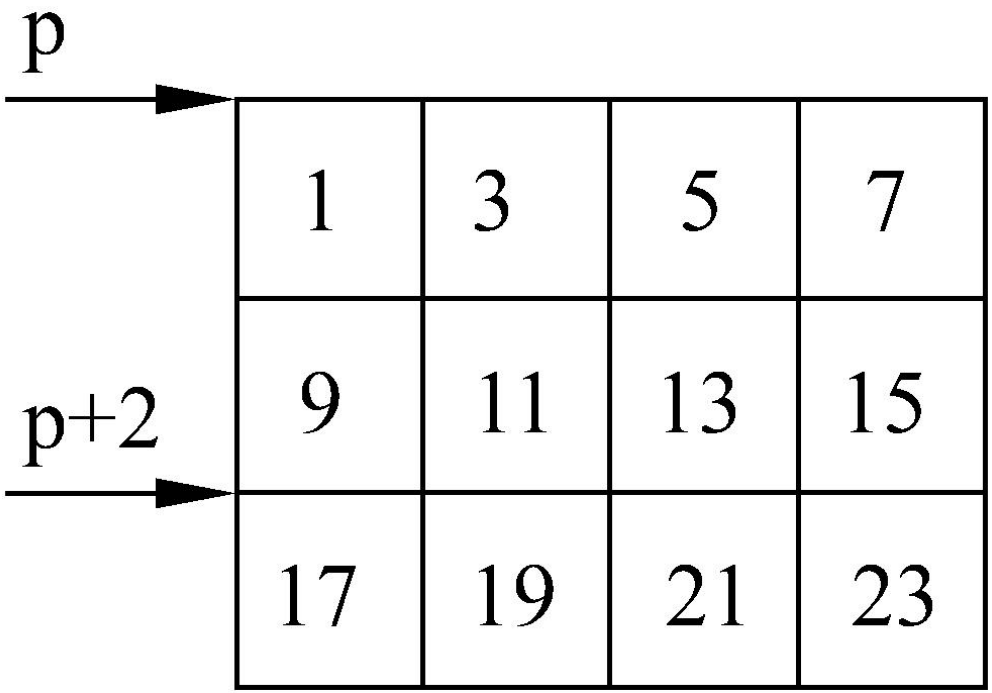
若指针p指向a[0]（即p=&a[0]），则p+1指向a[1]，p的增值以一维数组的长度为单位。



例6.8 输出二维数组任一元素的值

```
#include <iostream>
using namespace std;
int main( )
{
  int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  int (*p)[4],i,j;
  for(i=0;i<3;i++)
       for(j=0;j<4;j++)
          cout<<*(*(p+i)+j)<<" ";
  cout<<endl;
  return 0;
}
```

“p=a”，使p指向a[0]。因此p+2是二维数组a中序号为2的行的起始地址。*(p+2)+3是a数组3行4列元素地址。*(*(p+2)+3)是a[2][3]的值。



3. 用指向数组的指针作函数参数
Using pointers to arrays as function parameters

一维数组名可以作为函数参数传递，多维数组名也可作函数参数传递。

例6.9 输出二维数组各元素的值。

```
#include <iostream>
using namespace std;
int main( )
{
   void output(int (*p)[4]);                    //函数声明
   int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
   output(a);                                   //多维数组名作函数参数
   return 0;
}
```

```
void output(int (*p)[4])
//形参是指向一维数组的指针变量
{
    int i,j;
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            cout<<*(*(p+i)+j)<<" ";
    cout<<endl;
}
```

6.4 字符串与指针

```
#include <iostream>          //访问字符串的3种方法
#include <string>
using namespace std;
int main( )
{
    char str1[]="I love C++!";       //字符数组
    cout<<str1<<endl;
    string str2="I love C++!";       //字符串变量
    cout<<str2<<endl;
    char *str3="I love C++!";        //字符指针
    cout<<str3<<endl;
    return 0;
}
```

对字符串中字符的存取

例6.13 将字符串str1复制为字符串str2。

```
#include <iostream>
using namespace std;
int main( )
{
    char str1[]="I love CHINA!",str2[20],*p1,*p2;
    p1=str1;p2=str2;
    for(;*p1!='\0';p1++,p2++)
        *p2=*p1;
    *p2='\0';
    p1=str1;p2=str2;
    cout<<"str1 is: "<<p1<<endl;
    cout<<"str2 is: "<<p2<<endl;
    return 0;
}
```

课堂练习

1. 输入某年某月某日，判断这一天是这一年的第几天？
2. 一个5位数，判断它是不是回文数。即12321是对称（回文）数，个位与万位相同，十位与千位相同。
3. 有1、2、3、4个数字，能组成多少个互不相同且无重复数字的三位数？都是多少？
4. 一个整数，它加上100后是一个完全平方数，再加上168又是一个完全平方数，请问该数是多少？
5. 求s=a+aa+aaa+aaaa+aa...a的值，其中a是一个数字。例如2+22+222+2222+22222(此时共有5个数相加)，几个数相加由键盘控制。
6. 输出9*9口诀。

s=a+aa+aaa+aaaa+aa...a

```
#include<iostream>
#include<cmath>
using namespace std;
void add(int a[],int s[]);
int main()
{
        int n, i;
        int a[256]={0};
        int s[256]={0};
        cout<<"请输入N\n ";
        cin>>n;
        for(i=0;i<n;i++)
        {
                a[i] = 2;
                add(a,s);
        }
        for(i=n;i>=0;i--)
                cout<<s[i];
    return 0;
}
```

```
#include<iostream>
using namespace std;
int main()
{int a,b;
cout<<"请输入一个自然数:";
cin>>a;
b=a*a;
if(b<1||b>9999)
{cout<<"数据输入有误！请重新再输:";
}
else if((b%10==a||b%100==a))
{cout<<a<<' '<<"yes"<<' '<<a<<"*"<<a<<"="<<b;
}
else
cout<<a<<' '<<"no"<<' '<<a<<"*"<<a<<"="<<b;
cout<<endl;
return 0;
}
```

习题19

```
#include <iostream>
using namespace std;
int main ( )
{
    int i,j,k,n;
    cout<<"narcissus numbers are:"<<endl;
    for (n=100;n<1000;n++)
    {
      i=n/100;
      j=n/10-i*10;
      k=n%10;
      if (n == i*i*i + j*j*j + k*k*k)
        cout<<n<<" ";
    }
    cout<<endl;
    return 0;
}
```

习题20

```
#include <iostream>
using namespace std;
int main( )
{
    int m,s,i;
    for (m=2;m<1000;m++)
    {
        s=0;
        for (i=1;i<m;i++)
        if ((m%i)==0) s=s+i;
        if(s==m)
        {
            cout<<m<<" is a完数"<<endl;
            cout<<"its factors are:";
            for (i=1;i<m;i++)
                if (m%i==0)  cout<<i<<" ";
            cout<<endl;
        }
    }
    return 0;
}
```

习题20-1

```
#include <iostream>
using namespace std;
int main( )
{
    int k[11];
    int i,a,n,s;
    for (a=2;a<=1000;a++)
    {
        n=0;
        s=a;
        for (i=1;i<a;i++)
          if ((a%i)==0)
          {
              n++;
              s=s-i;
              k[n]=i;          // 将找到的因子赋给k[1]---k[10]
          }
```

```
        if (s==0)
        {
        cout<<a<<" is a 完数"<<endl;
        cout<<"its factors are:";
        for (i=1;i<n;i++)
            cout<<k[i]<<" ";
        cout<<k[n]<<endl;
        }
    }
    return 0;
}
```

题目：计算字符串中子串出现的次数

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
        char str1[20],str2[20],*p1,*p2;
        int sum=0;
        cout<<"please input two strings\n";
        cin>>str1>>str2;
        p1=str1;p2=str2;
        while(*p1!='\0')
        {
                if(*p1==*p2)
                {
                        while(*p1==*p2&&*p2!='\0') {p1++; p2++;}
                }
                else p1++;
                if(*p2=='\0') sum++;
                p2=str2;
        }
        cout<<sum<<endl;
        return 0;
}
```

6.5 函数与指针

6.5.1 用函数指针变量调用函数

指针变量也可以指向一个函数。
函数入口地址就称为函数的指针。
通过函数指针变量可调用此函数。
函数指针的一般定义形式为

函数类型 （*指针变量名）（函数形参表）；

```
int  (*max) (int,int);          //指向函数的指针
int  max(int,int);              //max函数原型
int  *max (int,int);            //返回指针的函数
```

例6.14 求 a 和 b 中的大者

```
#include <iostream>
using namespace std;
int main( )
{
    int max(int x,int y);        //函数声明
    //int (*p)(int,int);          //定义函数指针p
    int a,b,m;
    //p=max;                      //使p指向函数max
    cin>>a>>b;
    m=max(a,b);                   //调用函数max，m=p(a,b);
```

```
    cout<<"max="<<m<<endl;
    return 0;
}
int max(int x,int y)
{
    int z;
    if(x>y) z=x;
    else z=y;
    return(z);
}
```


HOME BACK NEXT

6.5.2 用指向函数的指针作函数参数

函数指针作为函数的参数，将函数名传给其他函数的形参。这样在调用一个函数的过程中可根据给定的不同实参调用不同的函数。

例：编写求定积分的通用函数，用它分别求5个函数的定积分，有3个形参： 下限a、上限b，以及函数指针fun。函数原型为

double integral (double a，double b，double (*fun)(double));

- 编写一个函数，
- 输入n为偶数时，调用函数求1/2+1/4+...+1/n；
- 输入n为奇数时，调用函数求1/1+1/3+...+1/n。)
- 1.程序分析：
- 2.程序源代码：

```
#include <iostream>
using namespace std;
int main()
{       float peven(int n),podd(int n),dcall(float (*fp)(int n),int n);
        float sum;
        int n;
        while (1)
        {  cin>>n;
                if(n>1)
                        break;      }
        if(n%2==0)
        {               cout<<"Even=";
                        sum=dcall(peven,n); }
        else
        {               cout<<"Odd=";
                        sum=dcall(podd,n);  }
        cout<<sum;
        return 0;
}
```

```
float peven(int n)
{       float s;
        int i;
        s=1;
        for(i=2;i<=n;i+=2)
                s+=1/(float)i;
        return(s);}
float podd(int n)
{       float s;
        int i;
        s=0;
        for(i=1;i<=n;i+=2)
                s+=1/(float)i;
        return(s);}
float dcall(float (*fp)(int n), int n)
{       float s;
        s=(*fp)(n);
        return(s);}
```

6.6 返回指针值的函数

返回指针值的函数简称为指针函数。

定义指针函数的一般形式为

类型名 *函数名（参数表列）；

例如

```
int *a(int x,int y);
```

6.7 指针数组和指向指针的指针

6.7.1 指针数组的概念

数组元素均为指针的数组称为指针数组。
一维指针数组的定义形式为

类型名 *数组名［数组长度］；

例如

int *p[4];

例6.15 若干字符串按字母顺序输出

```
#include <iostream>
using namespace std;
int main( )
{
    void sort(char *name[],int n);              //声明函数
    void print(char *name[],int n);             //声明函数
    char *name[]=
      {"BASIC","FORTRAN","C++","Pascal","COBOL"};
    //定义指针数组
    int n=5;
    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 print(char *name[],int n)
{
   int i;
   for(i=0;i<n;i++)
   cout<<name[i]<<endl;
}
void print(char *name[],int n)
{
   int i=0
   char *p;
   p=name[0];
   while(i<n)
   { p=*(name+i++); cout<<p<<endl;}
}
```

6.7.2 指向指针的指针

name是指针数组，其元素是指针。

name+i是name[i]的地址，name[i]的值是指针，故name+i就是指向指针的指针。

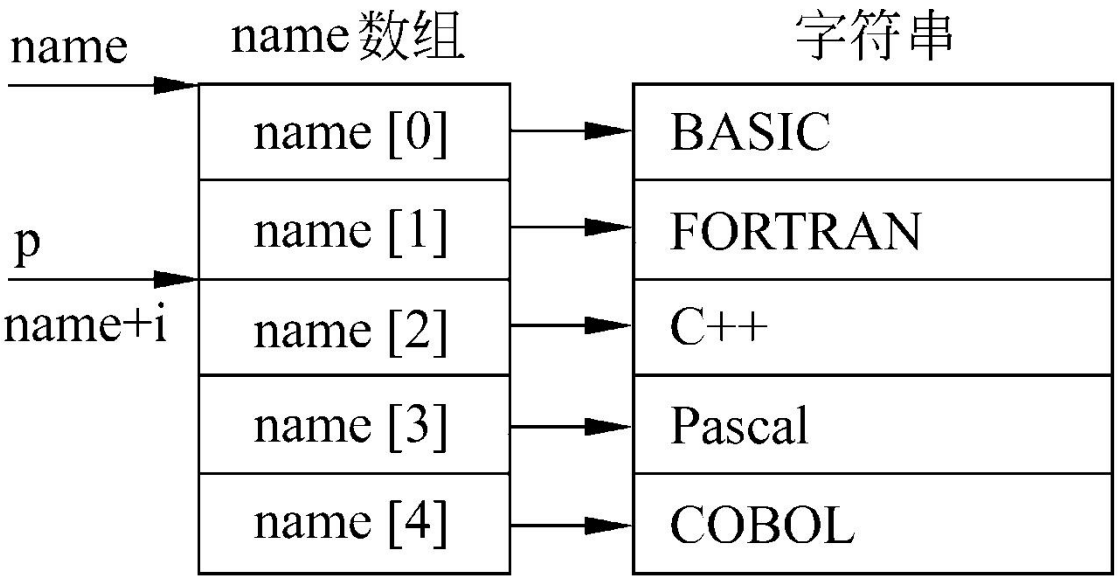
设置指针变量p，它指向指针数组的元素。

p就是指向指针型数据的指针变量。

定义一个指向指针

char *(*p);

char **p;



例6.16 指向字符型数据的指针变量

```
#include <iostream>
using namespace std;
int main( )
{
   char **p;                    //定义指向字符指针的指针变量p
   char *name[]={"BASIC","FORTRAN","C++",
   "Pascal","COBOL"};
   p=name+2;                    //p指向"C++"
   cout<<*p<<endl;              //输出name[2]指向的字符串
   cout<<**p<<endl;             //输出name[2]指向的字符串的第一个字符
  return 0;
}
```

通过指针函数，输入一个1～7之间的整数， 输出对应的星期名

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    char * day_name(int n);
    cout<<"input Day No:\n";
    cin>>i;
    if(i<0) exit(1);
    cout<<"Day No:"<<i<<" is "<<day_name(i)<<"\n";
    return 0;
}
```

```
char * day_name(int n)
{
    static char *name[]={ "Illegal day",
    "Monday",       "Tuesday",
    "Wednesday",    "Thursday",
    "Friday",       "Saturday",     "Sunday"};
    for(int i=0;*(name[n]+i)!='\0';i++)
            cout<<*(name[n]+i);
    return((n<1||n>7) ? name[0] : name[n]);
}
```

6.8 Pointers and Constants
--Pointer to Constant

- Syntax Form:
- const 类型 *指针名=地址
- You can't change value of the variable which the pointer pointing at. But the pointer itself can be changed, and can be changed to point at another variable.
- For example:
- int n1=3;
- int const n2=5;
- const int *pn= &n1;
- pn=&n2; //ok
- *pn=6; //Error

Pointers and Constants
--Constant Pointer

- Syntax Form:
- 类型 * const 指针名=地址
- If declared a constant pointer, the value of pointer itself can't be changed.
- For example:
- int n1=3;
- int const n2=5;
- int *const pn= &n1;
- pn=&n2; //Error
- *pn=6; //OK

Pointers and Constants

- When using a pointer, two objects are involved: the pointer itself and the object pointed to. ‘‘Prefixing’’ a declaration of a pointer with const makes the object, but not the pointer, a constant.

- To declare a pointer itself, rather than the object pointed to, to be a constant, we use the declarator operator *const instead of plain *.

- For example:

Pointers and Constants

```
void f1 (char * p){
    char s[] = "Gorm";
    const char * pc = s; // pointer to constant
    pc[3] = ´g ´; // error: pc points to constant
    pc = p ; // ok
    char *const cp = s ; // constant pointer
    cp [3 ] = ´a ´; // ok
    cp =p; / / error: cp is constant
    const char *const cpc = s ; // const pointer to const
    cpc [3 ] = ´a ´; // error: cpc points to constant
    cpc = p ; // error: cpc is constant
}
```

Pointers and Constants

- The declaratory operator that makes a pointer constant is *const . There is no const * declaratory operator, so a const appearing before the * is taken to be part of the base type. For example:
- *char *const cp ; // const pointer to char*
- *char const * pc ; // pointer to const char*
- *const char * pc2 ; // pointer to const char*
- Some people find it helpful to read such declarations right-to-left.
- For example, "cp is a const pointer to a char " and "pc2 is a pointer to a char const ."
- The C++ Programming Language Third Edition, ch5.4

常量指针VS指针常量

const int * pn= &n1; // *pointer to constant*

指向常量的指针，指针所指向的地址的内容是不可修改的。

int * const pn= &n1; // *constant pointer*

指针本身的值是常量，是不可改变地址的指针，但是可以对它所指向的内容进行修改。

指向常量的指针

常量指针就是本身是常量的指针//本书的提法

常量指针就是指向常量的指针

指针常量就是指针类型的常量//其它书的提法

常量指针VS指针常量

类型	常量	变量	常变量
字符	A	char c;	const char c; 常字符变量VS字符常变量
整型	25	int i;	const int i; 常整型变量VS整型常变量
逻辑型	true、false	bool flag;	const bool flag; 常逻辑变量VS逻辑常变量
实型	3.14	float f;	const float f; 常实型变量VS实型常变量
指针	0x0000ff00	int * p;	int * const p; 常指针变量VS指针常变量

指向常量的常量指针

```
#include <iostream>
using namespace std;
int main()
{
    int a = 0;
    const int b = 1;
    int* const p = &b; //为什么不能把常量地址赋给常量指针
    int* const p = &a; //这样就可以
    cout<<*p<<endl;
    return 0;
}
```

Example of pointer application

```
#include <iostream>
using namespace std;
int main（ ）
{
    int   * pt_int;
    float * pt_float;
    int   pig = 7, dog = 27;
    float x = 1.2345, y = 32.14;
    void *general;
    pt_int = &pig;
    *pt_int += dog;
    cout << "Pig now has the value of " << *pt_int <<"\n";
    general = pt_int; //指向int型的指针赋值给void型指针
```

Example of pointer application

```
    pt_float = &x;
    y += 5 * (*pt_float);
    cout << "y now has the value of " << y << "\n";
    general = pt_float;  //指向float型的指针赋值给void型指针
    const char *name1 = "John";
    //指向常量的指针，所指对象之值不能改变
    char *const name2 = "John";
    //常量指针，指针本身不能被改变
    return 0;
}
```

6.9 void类型指针

- void类型除用于不返回值的函数类型之外，也可用于说明一个指针变量是一个不定类型的指针变量，一个基本类型是void的指针称为void类型的指针。任何类型的指针都可以赋给void类型的指针变量，例如：
- int val＝5;
- int *pi＝&val;
- void *pv ＝ pi;
- 但将void类型的指针赋给其他类型的指针时必须使用强制类型转换，例如：
- pi＝(int *)pv;
- void类型的指针主要用于通用函数中函数的形式参数。

Example： void类型指针的使用

```
void vobject;        //错，不能声明void类型的变量
void *pv;            //对，可以声明void类型的指针
int  *pint; int i;
void main（ ）       //void类型的函数没有返回值
{
   pv = &i;          //void类型指针指向整型变量
   pint = (int *)pv; // void指针赋值给int指针需要类型
   强制转换
}
```

6.10 有关指针的数据类型和指针运算的小结

- ## 6.10.1 有关指针的数据类型的小结

定义	含义
int i;	定义整型变量
int *p;	p为指向整型数据的指针变量
int a[n];	定义整型数组a，它有n个元素
int *p[n];	定义指针数组p，它由n个指向整型数据的指针元素组成
int (*p)[n];	p为指向含n个元素的一维数组的指针变量
int f( );	f为带回整型函数值的函数
int *p( );	p为带回一个指针的函数，该指针指向整型数据
int (*p)( );	p为指向函数的指针，该函数返回一个整型值
int **p;	p是一个指向指针的指针，它指向一个指向整型数据的指针变量

const char * pc	pointer to constant
char *const cp	constant pointer
const char *const cpc	const pointer to const
void *pv	不定类型的指针变量

6.10.2 指针运算小结

(1) 指针变量加/减 一个整数

p+i的地址计算： p+i*d，d为p所指向的变量单元所占用的字节数。p+i指向p下面的第i个元素。

(2) 指针变量赋值

将一个变量地址赋给一个指针变量。

(3) 指针变量可以有空值，即该指针变量不指向任何变量。注意，p的值等于NULL和p未被赋值是两个不同的概念。

(4) 两个指针变量可以相减

如果两个指针变量指向同一个数组的元素，则两个指针变量值之差是两个指针之间的元素个数。

p1+p2并无实际意义。

(5) 两个指针变量比较

如果p1和p2不指向同一数组则比较无意义。

(6) 对指针变量的赋值应注意类型问题。

使用指针的优点是：

①提高程序效率；

②在调用函数时，得到多个可改变的值；

③可以实现动态存储分配。

指针使用太灵活，十分容易出错，而且这种错误往往难以发现。

*6.11 引用

6.11.1 什么是变量的引用

“引用”(reference) 是变量的别名(alias)。

```
int a,c;                //定义a、c是整型变量
int &b=a;               //声明的同时初始化，b是a的引用
int &b=c;               //企图使b又变成c的引用是不行的
```

经过这样的声明后，a或b都代表同一变量。 a和b 占内存中的同一个存储单元，它们具有同一地址。

& 在此是引用声明符，并不代表地址。

6.11.2 引用的简单使用

例6.15 引用和变量的关系。

```
#include <iostream>
#include <iomanip>
using namespace std;
int main( )
{
    int a=10;
    int &b=a;                  //声明b是a的引用
    a=a*a;                     //a的值变化了，b的值也应一起变化
    cout<<a<<setw(6)<<b<<endl;
    b=b/5;                     //b的值变化了，a的值也应一起变化
    cout<<b<<setw(6)<<a<<endl;
    return 0;
}
```

6.11.3 引用作为函数参数

C++增加引用类型是把它作为函数参数，以扩充函数传递数据的功能。

函数参数传递的两种情况：

(1)传送变量的值。将变量名作为实参和形参。在执行函数期间形参的值发生变化，并不传回给实参。因为在调用函数时，形参和实参不是同一个存储单元。

(2) 传送变量的地址。形参是指针变量，实参是一个变量的地址，调用函数时，形参(指针变量)指向实参变量单元。本质上仍然是值传递。

(3) 传送变量的别名。引用作为函数形参。

两个变量互换1

```
#include <iostream>
using namespace std;
int main( )
{
    void swap(int,int);
    int i=3,j=5;
    swap(i,j);
    cout<<i<< “ ” <<j<<endl;
    return 0;
}
void swap(int a,int b)
{
    int temp;
    temp=a; a=b;b=temp;
}
```

两个变量互换2

```
#include <iostream>
using namespace std;
int main( )
{
    void swap(int *,int *);
    int i=3,j=5;
    swap(&i,&j);                          //实参是变量的地址
    cout<<i<<" "<<j<<endl;                //i和j的值已互换
    return 0;
}
void swap(int *p1,int *p2)                //形参是指针变量
{
    int temp;
    temp=*p1;
    *p1=*p2;
    *p2=temp;
}
```

两个变量互换3

```
#include <iostream>
using namespace std;
int main( )
{
    void swap(int &,int &);
    int i=3,j=5;
    swap(i,j);
    cout<<"i="<<i<<"  "<<"j="<<j<<endl;
    return 0;
}
void swap(int &a,int &b)          //形参是引用类型
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
```

传值和传址的对比

①值传递方式传送的是实参的值，地址传递方式传送的是实参的地址。

②值传递方式在调用函数时，对形参要分配内存单元。指针形参同样要另外开辟内存单元，只是其内容是实参的地址。引用不是一个独立变量，不单独占内存单元。

③地址传递方式在调用函数时，实参不必用变量的地址(在变量名的前面加&)，而直接用变量名，系统向形参传送的是实参的地址而不是实参的值。

使用引用类型比使用指针变量简单、直观、方便，可以部分代替指针的操作。从而降低了程序设计的难度。

例6.19 对3个变量按由小到大的顺序排序

```
//例6.19 对3个变量按由小到大的顺序排序。
#include <iostream>
using namespace std;
int main( )
{
    void sort(int &,int &,int &); //函数声明，形参是引用类型
    int a,b,c;                    //a,b,c是需排序的变量
    int a1,b1,c1;                 //a1,b1,c1最终的值是已排好序的数列
    cout<<"Please enter 3 integers:";
    cin>>a>>b>>c;                 //输入a,b,c
    a1=a;b1=b;c1=c;
    sort(a1,b1,c1);               //调用sort函数，以a1,b1,c1为实参
    cout<<“sorted order is ”<<a1<<“ ”<<b1<<“ ”<<c1<<endl;
    return 0;
}
```

```
void sort(int &i,int &j,int &k)  //对i,j,k 3个数排序
{
   void change(int &,int &);     //函数声明，形参是引用类型
   if (i>j) change (i,j);        //使i<=j
   if (i>k) change (i,k);        //使i<=k
   if (j>k) change (j,k);        //使j<=k
}
void change (int &x,int &y)              //使x和y互换
{
   int temp;
   temp=x;
   x=y;
   y=temp;
}
```

输出数组元素的内容和地址

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    void Array_Ptr(long * P, int n);
    long list[5] = {50, 60, 70, 80, 90};
    cout<< "In main, address of array is "
        << unsigned long(list) << endl;
    cout << endl;
    Array_Ptr(list, 5);
    return 0;
}
```

```
void Array_Ptr(long * P, int n)
{
    int i;
    cout << "In func, address of array is "
         << unsigned long(P) << endl;
    cout << "Accessing array in the function using pointers"
<< endl;
    for (i = 0; i < n; i++)
    {
        cout << "   Address for index " << i << " is "
             << unsigned long(P+i);
        cout << "  Value is " << *(P+i)  << endl;
    }
}
```

Dynamic allocation of memory —heap memory allocated

- 堆(heap):
- 区别于栈区、数据区、代码区的内存空间。堆允许程序在运行时(而不是在编译时)，申请某个大小的存储空间。
- 开辟堆内存的原因：
  - 通常情况下，数组的大小在程序编译时必须是已知的，即必须用一个常数对数组的大小进行申明。
  - 但是如果在定义数组时不知道数组应该定义多大，则需要程序在运行时自动从系统中获取存储空间。
- 堆内存的实质
- 程序在编译和连接时不需要确定内存空间的大小，随着程序的运行自动的分配时大时小的内存空间，这种内存就是堆内存，所以堆内存是动态的，所以也称为动态内存。

Dynamic memory allocated function

- 函数原型： void *malloc(size);
- 参数size：欲分配的字节数
- 返回值：成功，则返回void型指针。失败，则返回空指针。
- 头文件：<stdlib.h> 和 <malloc.h>
  (alloc.h abandoned)
- 作用：
  - 从对内存中“切下”一块size大小的内存，将指向该内存空间的首地址返回，该内存空间的内容是未知的。

Dynamic memory released function

- 函数原型：void free(void *memblock);
- 参数memblock：指针，指向需释放的内存。
- 返回值：无
- 头文件：<stdlib.h> 和 <malloc.h>(alloc.h abandoned)
- 作用：
  - 堆内存是按照要求进行分配的存储空间，对于程序在运行中不再需要的由malloc()函数开辟的内存空间，必须在程序还未结束前及时释放，以供程序其它部分使用。
- 注意：
  - 用free释放的内存必须是先前用malloc分配的， 同一内存只能释放一次，重复释放会出错。

Example：从堆中获取一个整数数组，赋值并打印

```
#include <iostream.h>
#include <malloc.h> //堆内存开辟头文件
using namespace std;
int main()
{
        int arraysize;  //定义元素个数
        int * array;
        cout<<"please input a number of array:"<<endl;
        cin>>arraysize;
        if((array=(int *)malloc(arraysize*sizeof(int)))==NULL)//c++标准中规定，malloc()函数的返回值必须经过强制转换成整形指针才能使用
        {
                cout<<"can't allocate more memory!"<<endl;
                return(1);
        }
        for(int count=0;count<arraysize;count++)
                array[count]=count*2;
        for(count=0;count<arraysize;count++)
                cout<<array[count]<<"  ";
        cout<<endl;
        free(array);//释放堆内存
        return 0;
}
```

Dynamic memory applied operator new

- Syntax Form:
- new 类型名T[初值列表]
- Function：在程序执行期间，申请用于存放T类型对象的内存空间，并依初值列表赋以初值。
- Result：成功：返回一个T类型的指针，指向新分配的内存。失败：0（NULL）
- For Example:
- new int; //开辟一个存放整数的存储空间，返回一个指向该存储空间的地址(即指针)
- new char[10]; //开辟一个存放字符数组(包括10个元素)的空间，返回首元素的地址

float *p=new float(3.14159); //开辟一个存放单精度数的空间，并指定该实数的初值为3.14159，将返回的该空间的地址赋给指针变量p

int arraysize; //元素个数，即需申明的空间大小

int *array;

array=new int[arraysize] //因为返回的是一个自己定义的指针类型，不需转换

- Note：Needn't head-file declaration.
- Note：用new分配数组空间时不能指定初值。如果由于内存不足等原因而无法正常分配空间，则new会返回一个空指针NULL，用户可以根据该指针的值判断分配空间是否成功。

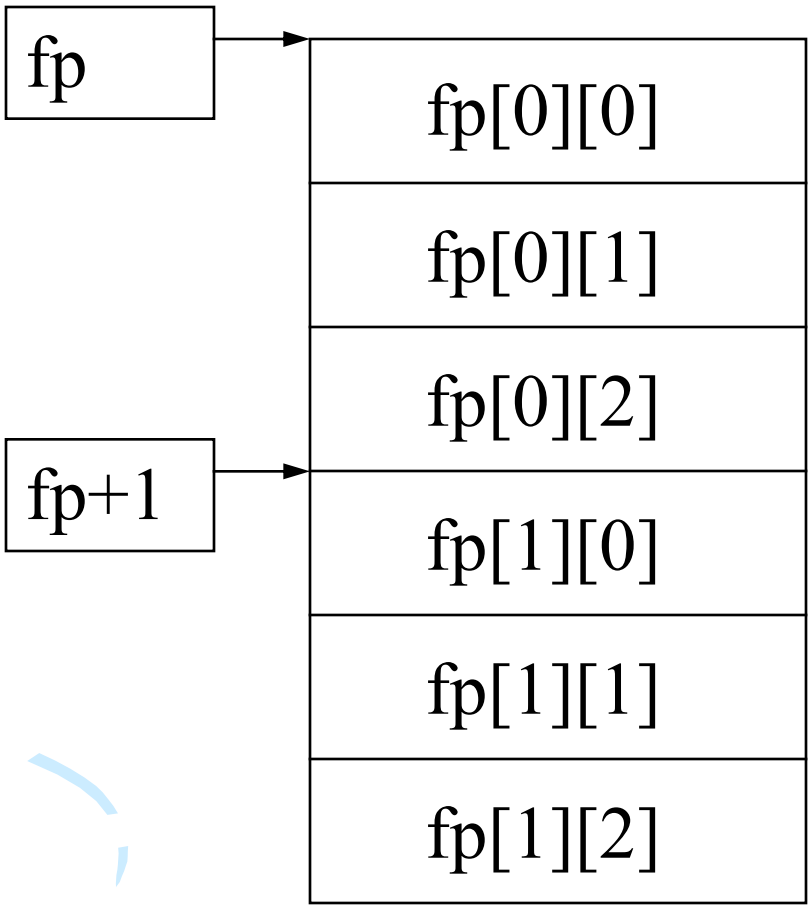
Dynamic memory released operator delete

- Syntax Form:
- delete 指针P
- or
- delete []P //对数组
- Function:
- Releasing the memory which the pointer "P" pointed at. "P" must be return value of new operation.
-

Attention while allocating dynamic array

- 用new创建多维数组：
- new 类型名T[下标表达式1][下标表达式2]…;
- 如果内存申请成功，new运算返回一个指向新分配内存首地址的指针，是一个T类型的数组，数组元素的个数为除最左边一维外各位下标表达式的乘积。例如：
- char (*fp)[3];
- fp = new char[2][3];

char (*fp)[3];



Example 动态创建多维数组

```
#include<iostream>
using namespace std;
int main()
{
    float (*cp)[9][8];
    int i,j,k;
    cp = new float[8][9][8];
    for (i=0; i<8; i++)
        for (j=0; j<9; j++)
            for (k=0; k<9; k++)
                *(*(*(cp+i)+j)+k)=i*100+j*10+k;
//通过指针访问数组元素
```

Example 动态创建多维数组

```
    for (i=0; i<8; i++)
    {
        for (j=0; j<9; j++)
        {
            for (k=0; k<8; k++)
//将指针cp作为数组名使用
//通过数组名和下标访问数组元素
                cout<<cp[i][j][k]<<"  ";
            cout<<endl;
        }
        cout<<endl;
    }
    return 0;
}
```

例7.6 开辟空间以存放一个结构体变量

```
#include <iostream>
#include <string>
using namespace std;
struct Student            //声明结构体类型Student
{
string name;
int num;
char sex;
};
int main( )
{
  Student *p;           //定义指向结构体类型Student的数据的指针变量
  p=new Student;        //用new运算符开辟一个存放Student型数据的空间
  p->name="Wang Fun";   //向结构体变量的成员赋值
  p->num=10123;//要访问用new所开辟的结构体空间，只能通过指针p进行访问
  p->sex='m';
  cout<<p->name<<endl<<p->num<<endl<<p->sex<<endl; //输出各成员的值
  delete p;          //撤销该空间
  return 0;
}
```

作业

- **P192：3，4，6，7，12，17**