6.1 指针基础

6.1.4 指针与数组

一个数组包含若干元素，每个元素占用内存单元，都有相应的地址

指针变量也可以指向数组元素

如对于一维数组，

```
int a[10];
int *p;
p = &a[3];  //p指向a[3]
p = &a[0];  //p指向a[0]
```

C/C++中数组名代表数组首元素地址，因此p = &a[0];等价于p = a;

6.1 指针基础

6.1.4 指针与数组

6.1.4.1 指针运算

（1）加减整数

如：p++,p--,++p,--p,p+i,p-i,p+=i,p-=i等

p±i的含义：得到一个新的地址值，不是将p代表的地址值直接加/减i，而是增加/减少i个基类型的大小，即：p的值±n*sizeof(基类型)

例：假设指向int型数据的指针变量p值为1000，则：

p++的值为：1000+sizeof(int)=1004

p+3的值为：1000+3*sizeof(int)=1000+3*4=1012

指针加减整数用于普通变量，无语法错误，但无实际意义

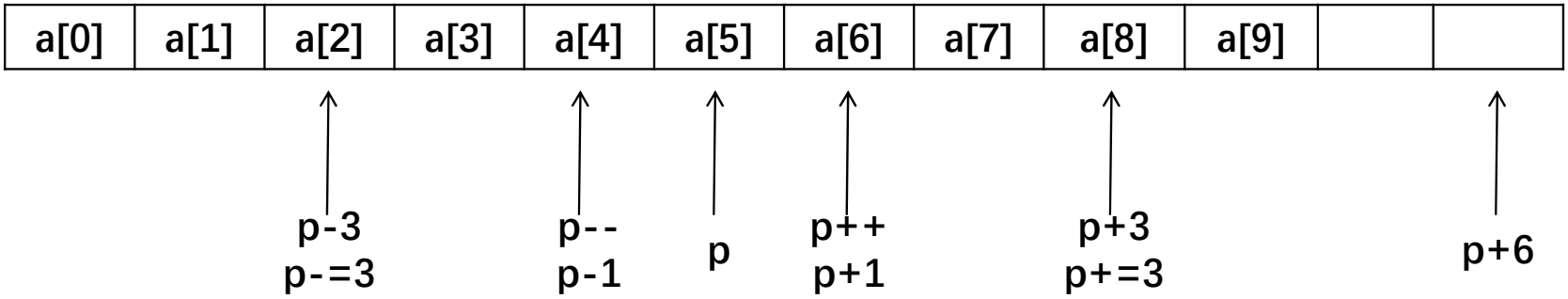
6.1 指针基础

6.1.4 指针与数组

6.1.4.1 指针运算

指针变量指向数组元素时，p±i得到的新指针指向当前元素后面/前面的第i个元素

例：假设有int a[10],*p=&a[5];



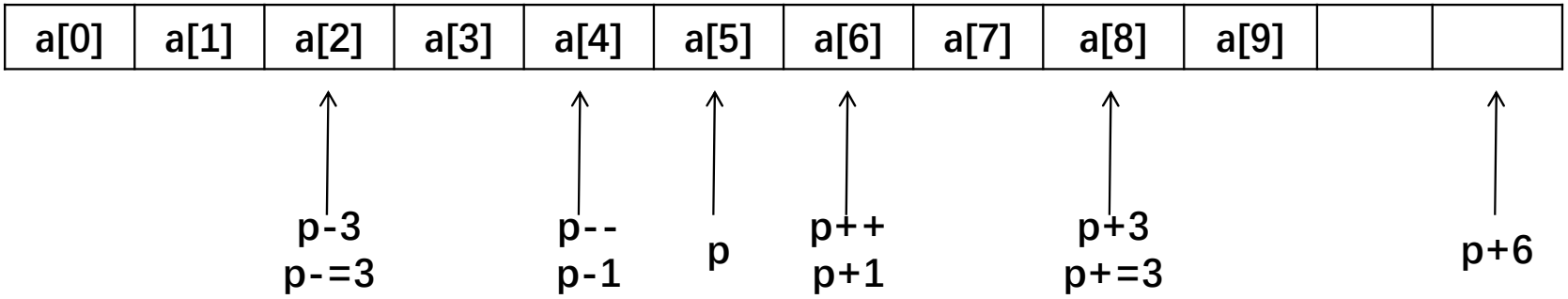
6.1 指针基础

6.1.4 指针与数组

6.1.4.1 指针运算

指针变量指向数组元素时，p±i得到的新指针指向当前元素后面/前面的第i个元素

例：假设有int a[10],*p=&a[5];



注意：p+6已越界，编译器并不检查越界问题，超过范围结果不确定

6.1 指针基础

6.1.4 指针与数组

6.1.4.1 指针运算

注意以下数组元素的表示方法

*(p+1)	取p所指元素的下一元素的值，p不变
*p+1	取p所指元素的值，值加1，p不变
*p++	等价于*(p++)，取p所指元素的值，p再指向下一元素
(*p)++	取p所指元素值，该元素值自增，p不变
*++p	等价于*(++p)，p先指向下一元素，再取该元素值

6.1 指针基础

6.1.4 指针与数组

6.1.4.1 指针运算

（2）两个指针相减

两个类型相同的指针可以相减，结果为这两个指针在内存中的距离（以基类型长度为单位），即：地址差/sizeof（基类型）

例： int *p1，*p2；

且p1值为2012，p2值为2020，则：p2-p1=（2020-2012）/4=2

两个指针指向普通变量或不同数组中元素时，无语法错误，但无实际意义

6.1 指针基础

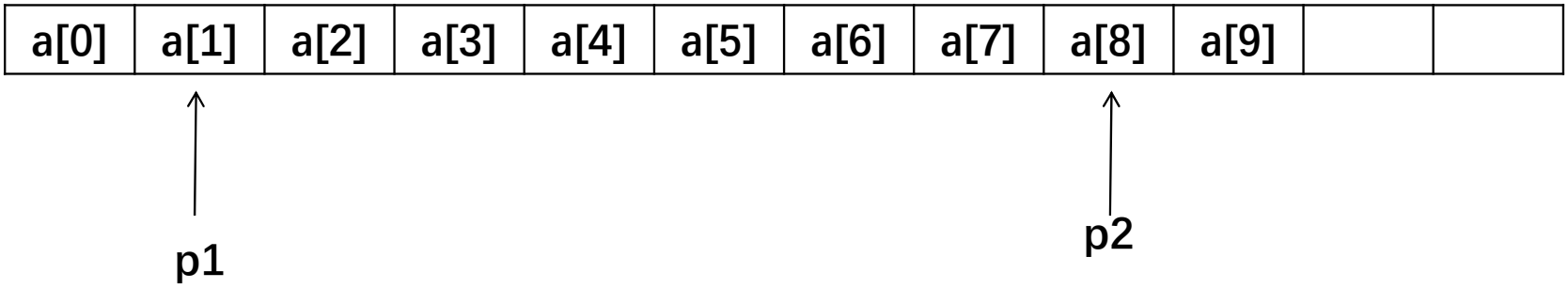
6.1.4 指针与数组

6.1.4.1 指针运算

指针相减运算仅当两个指针指向同一个数组中的元素时才有意义，表示两个数组元素的间隔。

如有：int a[10],*p1=&a[1]，*p2=&a[8];

则：p2-p1结果为7，p1-p2结果为-7



6.1 指针基础

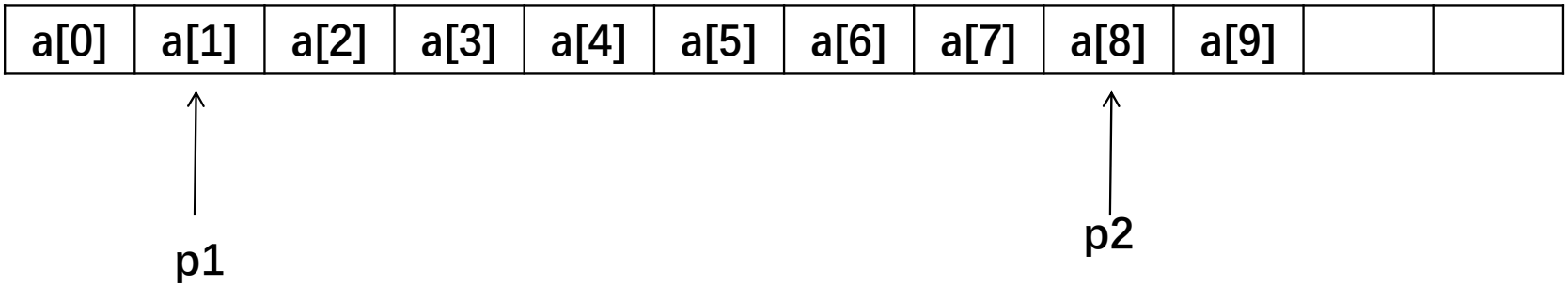
6.1.4 指针与数组

6.1.4.1 指针运算

指针相减运算仅当两个指针指向同一个数组中的元素时才有意义，表示两个数组元素的间隔。

如有：int a[10],*p1=&a[1]，*p2=&a[8];

则：p2-p1结果为7，p1-p2结果为-7



注意：不能进行两个指针相加运算

6.1 指针基础

6.1.4 指针与数组

6.1.4.1 指针运算

（3）关系运算

两个指针指向同一个数组的元素，或者指向数组尾元素的下一位置，可以利用关系运算符进行比较。例：

如果p1==p2，结果为真，则表明两个指针指向同一元素，否则就指向不同的元素。

如果p1<p2结果为真，则p1指向的元素在前，p2指向的元素在后。

6.1 指针基础

6.1.4 指针与数组

6.1.4.1 指针运算

（3）关系运算

两个指针指向同一个数组的元素，或者指向数组尾元素的下一位置，可以利用关系运算符进行比较。例：

如果p1==p2，结果为真，则表明两个指针指向同一元素，否则就指向不同的元素。

如果p1<p2结果为真，则p1指向的元素在前，p2指向的元素在后。

注意：不要对指针进行乘法、除法、取余等运算，会发生语法错误，也无实际意义

6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

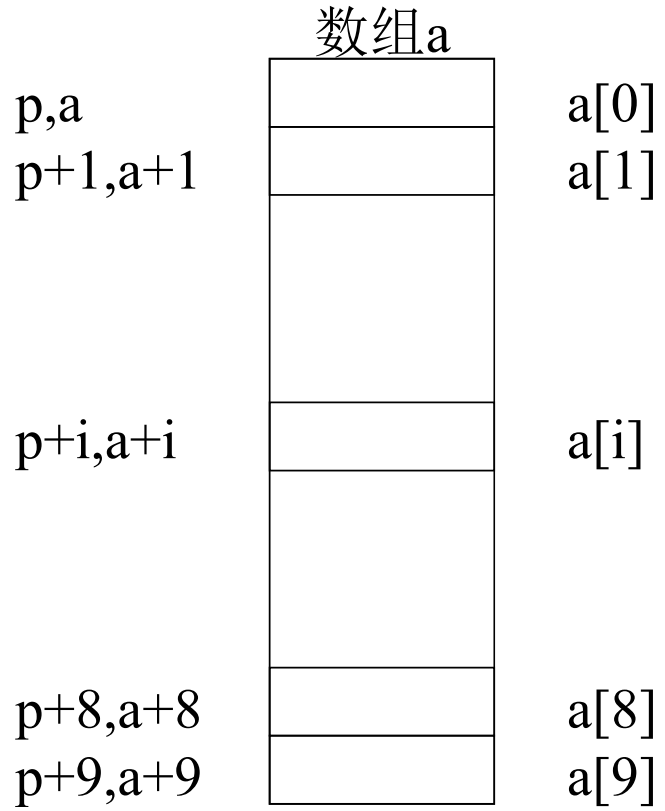
假设有定义

int a[10],*p=a;

则：

(1)p+i和a+i就是a[i]的地址，或者说，它们指向a数组的第i个元素

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



6.1 指针基础
6.1.4 指针与数组
6.1.4.2 指针与一维数组

数组元素的两种表示法

(1)下标表示法：a[i]或p[i]

(2)指针表示法：*(a+i)或*(p+i)

数组下标表示法等同于指针解引用，即a[i] ⇔ *(a+i)

对指向数组元素的指针变量，也有同样的关系成立，即p[i] ⇔ *(p+i)

6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

例：

```
#include <iostream>
using namespace std;
int main()
{
    int a[10] = { 1,2,3,4,5,6,7,8,9,10 }, * p, i;
    p = a;                                          //p指向a[0]
    cout << *(a + 1) << " " << *(p + 1) << endl;    //都表示a[1]
    cout << a[3] << " " << p[3] << endl;            //都表示a[3]
    p = a + 5;                                      //p指向a[5]
    cout << *a << " " << *p  << endl;               //分别表示a[0]和a[5]
    cout << p[-1] << endl;                          //表示a[4]
```

```
    p++;                                    //p指向a[6]
    *(p + 1) = -5;                          //a[7]赋值为-5
    *(a + 3) = -1;                          //a[3]赋值为-1
    for (i = 0; i < 10; i++)
        cout << a[i] << " ";
    cout << endl;
    p = a + 11;                             //p指向a[11]，超出数组范围
    cout << *p <<" ";
    cout << endl;
    return 0;
}
```

```
    p++;                                    //p指向a[6]
    *(p + 1) = -5;                          //a[7]赋值为-5
    *(a + 3) = -1;                          //a[3]赋值为-1
    for (i = 0; i < 10; i++)
        cout << a[i] << " ";
    cout << endl;
    p = a + 11;                             //p指向a[11]，超出数组范围
    cout << *p <<" ";
    cout << endl;
    return 0;
}
```

要注意指针的当前值，
防止产生指针越界！

6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

例：输出数组中的全部元素

方法1：

```
int a[10]={1,2,3,4,5,6,7,8,9,10},i;
for(i=0;i<10;i++)
    cout<<a[i]<<" ";
```

方法2：

```
int a[10]={1,2,3,4,5,6,7,8,9,10},i;
for(i=0;i<10;i++)
    cout<<*(a+i)<<" ";
```

方法3：

```
int a[10]={1,2,3,4,5,6,7,8,9,10},*p;
for(p=a;p<a+10;p++)
    cout<*p<<" ";
```

6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

例：输出数组中的全部元素

方法1：

```
int a[10]={1,2,3,4,5,6,7,8,9,10},i;
for(i=0;i<10;i++)
    cout<<a[i]<<" ";
```

方法2：

```
int a[10]={1,2,3,4,5,6,7,8,9,10},i;
for(i=0;i<10;i++)
    cout<<*(a+i)<<" ";
```

方法3：

```
int a[10]={1,2,3,4,5,6,7,8,9,10},*p;
for(p=a;p<a+10;p++)
    cout<*p<<" ";
```

比较：

第1、2种方法执行效率相同，编译系统是将a[i]转换为*(a+i)处理的，对每个a[i]先计算元素地址a+i，然后访问该元素。下标法比较直观。第3种方法通过移动指针变量直接指向元素，速度比1、2快，具体访问哪个元素与当前指针位置有关，不直观。

6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

例：分析以下程序能否正确实现数组输入输出？

```
#include <iostream>
using namespace std;
int main()
{
    int  a[10], i, * p = a;
    for (i = 0; i < 10; i++)
        cin >> *p++;
    cout << endl;
    for (i = 0; i < 10; i++, p++)
        cout << *p << " ";
    cout << endl;
    return 0;
}
```

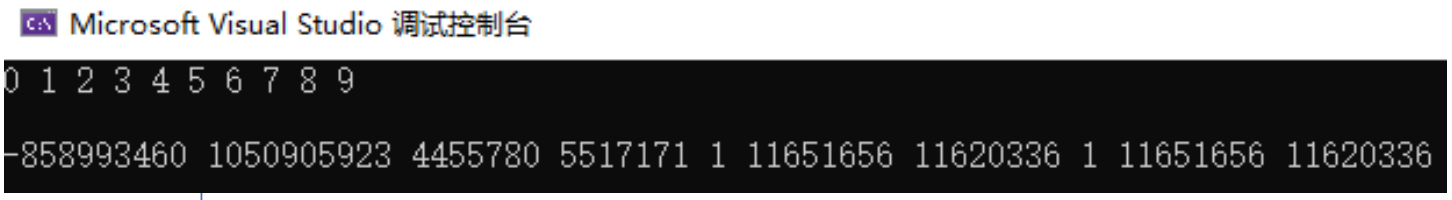
6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

例：分析以下程序能否正确实现数组输入输出？

```
#include <iostream>
using namespace std;
int main()
{
    int  a[10], i, * p = a;
    for (i = 0; i < 10; i++)
        cin >> *p++;
    cout << endl;
    for (i = 0; i < 10; i++, p++)
        cout << *p << " ";
    cout << endl;
    return 0;
}
```



指针越界！

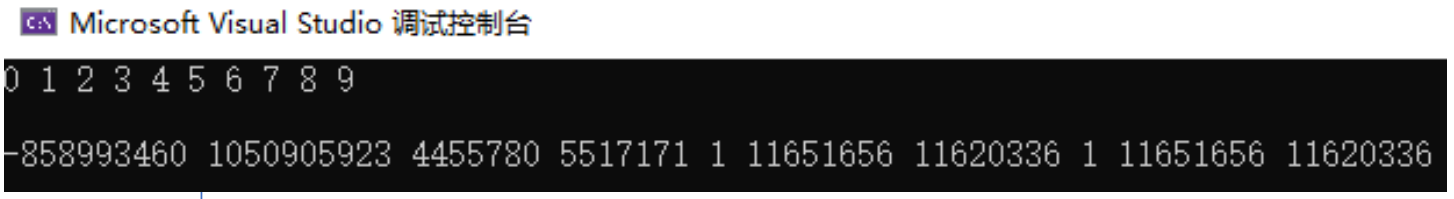
6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

例：分析以下程序能否正确实现数组输入输出？

```
#include <iostream>
using namespace std;
int main()
{
    int  a[10], i, * p = a;
    for (i = 0; i < 10; i++)
        cin >> *p++;
    cout << endl;
    for (i = 0; i < 10; i++, p++)
        cout << *p << " ";
    cout << endl;
    return 0;
}
```



指针越界！

怎么改？
第二个for循环前p指向数组首元素，
如加语句p=a;

6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

指针变量p和数组名a的区别：

- ✓ p是指针变量，可以实现使本身的值改变，a是数组名，是指针常量，其值在程序运行期间是固定不变的

  如： p++、p=p+3 √

  a++、a=a+3 ×

6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

指针变量p和数组名a的区别：

- ✓ p是指针变量，可以实现使本身的值改变，a是数组名，是指针常量，其值在程序运行期间是固定不变的

  如： p++、p=p+3 √

  a++、a=a+3 ×

- ✓ 对数组名使用sizeof运算符得到的是数组的长度，而对指针应用sizeof得到的是指针的长度

  如有：int a[10],*p=a;则sizeof(a)为40，sizeof(p)为4

6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

a和&a的区别：

a　表示数组首元素地址

&a表示数组地址

二者数字相同，但从概念上意义不同，对于int a[10]，a（即&a[0]）是一个4字节内存块的地址，&a是一个40字节内存块的地址，即二者基类型不同

6.1 指针基础

6.1.4 指针与数组

6.1.4.2 指针与一维数组

例：

```
int a[10] = { 1,2,3,4,5,6,7,8,9,10 };
cout << a << " " << &a << endl;
cout << a+1 << " " << &a+1 << endl;
```

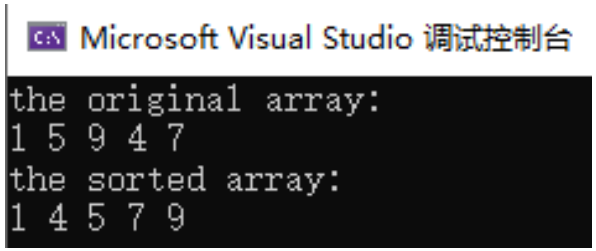
数组地址概念可用于二维数组中，本课程不涉及。

6.1 指针基础
6.1.4 指针与数组
6.1.4.3 用指针变量做函数形参接收数组地址

数组做函数参数（回顾模块5.1）

例：

```
int main()
{
    int a[5], i;
    cout << "the original array:" << endl;
    for (i = 0; i < 5; i++)
        cin >> a[i];
    select_sort(a, 5);
    cout << "the sorted array:" << endl;
    for (i = 0; i < 5; i++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```




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

例：

```
int main()
{
    int a[5], i;
    cout << "the original array:" << endl;
    for (i = 0; i < 5; i++)
        cin >> a[i];
    select_sort(a, 5);
    cout << "the sorted array:" << endl;
    for (i = 0; i < 5; i++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

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

数组做函数参数，参数传递时，将实参数组首元素地址传递给形参，形参数组的改变会影响实参数组

改用指针变量做函数形参，接收从实参传递来的数组首元素地址

```
int main()
{
    int a[5], i;
    cout << "the original array:" << endl;
    for (i = 0; i < 5; i++)
        cin >> a[i];
    select_sort(a, 5);
    cout << "the sorted array:" << endl;
    for (i = 0; i < 5; i++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
the original array:
1 5 9 4 7
the sorted array:
1 4 5 7 9
```

```
void select_sort(int *p, int n)
{
    int i, j, min, t;
    for (i = 0; i < n - 1; i++)
    {
        min = i;
        for (j = i + 1; j < n; j++)
            if (*(p+j) < *(p+min))
                min = j;
        if (i != min)
        {
            t = *(p + i);
            *(p + i) = *(p + min);
            *(p + min) = t;
        }
    }
}
```

对比两种实现：

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

```
void select_sort(int *p, int n)
{
    int i, j, min, t;
    for (i = 0; i < n - 1; i++)
    {
        min = i;
        for (j = i + 1; j < n; j++)
            if (*(p+j) < *(p+min))
                min = j;
        if (i != min)
        {
            t = *(p + i);
            *(p + i) = *(p + min);
            *(p + min) = t;
        }
    }
}
```

6.1 指针基础

6.1.4 指针与数组

6.1.4.3 用指针变量做函数形参接收数组地址

编译系统将形参数组名作为指针变量名来处理，通过参数传递得到实参数组首元素地址

形参数组名本质上是一个指针变量，只分配指针变量的空间，不分配形参数组空间，因此形参数组大小可不指定或指定任意正整数，再通过另一个参数传递数组长度

6.1 指针基础

6.1.4 指针与数组

6.1.4.3 用指针变量做函数形参接收数组地址

函数调用：select_sort(a, 5);

自定义函数头：

```
void select_sort(int array[],int n)
void select_sort(int array[5],int n)
void select_sort(int array[10],int n)
void select_sort(int *array, int n)
```

等价，本质都是指针变量

6.1 指针基础
6.1.4 指针与数组
6.1.4.3 用指针变量做函数形参接收数组地址

怎么证明?

main函数中用sizeof(a)对实参数组a求长度结果为20

select_sort函数中用sizof(array)对形参数组array求长度结果为4

6.1 指针基础

6.1.4 指针与数组

6.1.4.3 用指针变量做函数形参接收数组地址

注意：

实参数组名是指针常量，值是固定不变的，

形参数组名是指针变量，值是可以改变的。

例：

```
void f(int array[], int n)
{
    cout << *array;
    array = array + 3;
    cout << *array << endl;
}
```

如果有函数调用：f(a, 5);，输出a[0]和a[3]

6.1 指针基础

6.1.4 指针与数组

6.1.4.3 用指针变量做函数形参接收数组地址

实参数组也可以用指向它的指针变量来代替

前例主函数中函数调用也可以改成：

```
int main()
{
    int a[5], * p;
    p = a;
    ...
    select_sort(p, 5);
    ...
}
```

6.1 指针基础

6.1.4 指针与数组

6.1.4.3 用指针变量做函数形参接收数组地址

数组做参数，进行参数传递，实参与形参的结合的4种形式

实参	形参
数组名	数组名
数组名	指针变量
指针变量	数组名
指针变量	指针变量

注意：形参实参类型必须一致