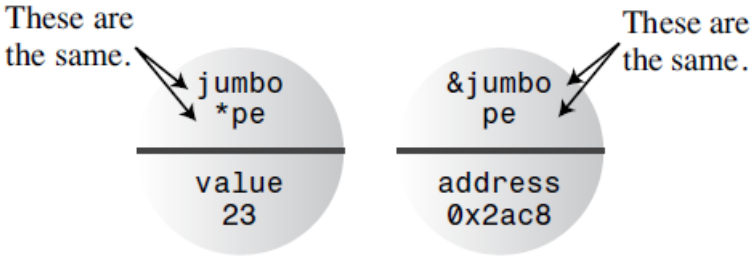
# 指针变量

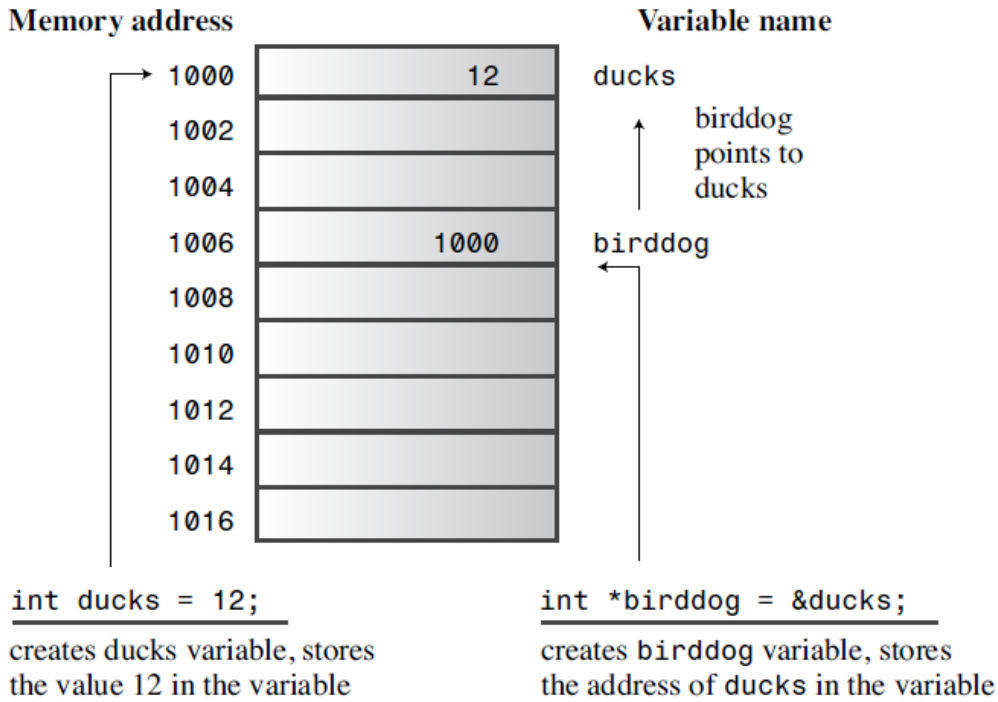
- 指针也是一种数据类型，指针变量也是一种变量
- 指针变量指向谁，就把谁的地址赋值给指针变量
- `*`操作符操作的是指针变量指向的内存空间
- 使用`sizeof()`测量指针的大小，取决于操作系统

```
int jumbo = 23;
int * pe = &jumbo;
```




```
char ch = 'A';
int a = 1;

printf("%p %p\n", &ch, &a);
```




```
int a = 1;
int* b = &a;
*b = 2;
cout << a << " " << *b << endl;
```

# 野指针和空指针

野指针：C++中创建指针时，计算机将分配用来存储地址的内存，但是不会分配用来存储指针所指向数据的内存。比如：p指针指向的空间是不确定的，通过间接操作修改了p指向空间的数据，很可能是操作系统中重要的数据，就发生不可预知的危险，因此，声明指针时一定要初始化。

```
int *p;
*p = 100;
cout << *p << endl;
```

空指针：野指针和有效指针变量保存的都是数值，为了标志此指针变量没有指向任何变量(空闲可用)，C语言中，可以把NULL赋值给此指针，这样就标志此指针为空指针，没有任何指针。

- `int *p = NULL`
- NULL是一个值为`0`的宏常量：`#define NULL((void *)0)`

## 万能指针`void *`

有时候，一个指针根据不同的情况，指向的内容是不同类型的值，我们可以先不明确定义它的类型，只是定义一个`无类型的指针`，以后根据需要再用强制类型转换的方法明确它的类型。

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

int a = 10;
double b = 3.5;
void* p;
int main(){
    p = &a;
    cout << *(int*)p << endl;
    p = &b;
    cout << *(double*)p << endl;
    cout << *(long long*)p << endl;
    return 0;
}

输出：
10
3.5
4615063718147915776
```

## const修饰的指针变量

```
int a = 100, b = 200;

// 指向常量的指针
// 修饰*，指针指向可以改变，指针指向内存区域不能修改
const int * p1 = &a;
// *p1 = 111;
p1 = &b;
cout << *p1 << endl;

```

```cpp
// 指针常量
// 修饰p2，指针指向内存区域可以改变，指针指向不可以改变
int * const p2 = &a;
// p2 = &b;
*p2 = 222;
cout << *p2 << endl;
```

## 指针和数组

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

int a[5];
int* pa = a;

int main() {

    for(int i = 0; i < 5; i++)
        scanf("%d", a + i);

    for(int i = 0; i < 5; i++)
        printf("a[%d]=%d\n", i, *(a + i));

    return 0;
}
```

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

int a[5];
int* pa = a;

int main() {

    for(int i = 0; i < 5; i++)
        scanf("%d", a + i);

    for(int i = 0; i < 5; i++){
        printf("%d ", *pa);
        pa++;
    }

    return 0;
}
```

## 指针数组

数组的每个元素都是指针类型。

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

int main() {

    int a = 1, b = 2, c = 3;
    int* p[] = {&a, &b, &c};

    for(int i = 0; i < sizeof(p) / sizeof(p[0]); i++)
        cout << *p[i] << " ";

    return 0;
}
```

## 多重指针

```cpp
#include <cstdio>

int a = 10;
int* p;
int** pp;   // 定义双重指针
int*** ppp; // 定义三重指针

int main() {
    p = &a;     // 将p指向a
    pp = &p;    // 将pp指向p
    ppp = &pp;  // 将ppp指向pp
    printf("a=%d=%d=%d\n", *p, **pp, ***ppp);
    return 0;
}
```

# 引用

> 引用是给变量起别名，比指针更加简洁。

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

int main() {

    // 引用的本质是常指针，因此必须初始化
    int a = 10;
    int& b = a; // int* const b = &a;

    b = 20; // *b = 20;

    cout << a << " " << b << endl;

    return 0;
```

```
}
```

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

int f(int& x){
    x *= 2;
    return x * 3;
}

int main() {

    int a = 10;
    cout << f(a) << " " << a << endl;

    return 0;
}
```

## 数组引用

```cpp
int a[5] = { 1, 2, 3, 4, 5 };
int(&aref)[5] = a;

aref[0] = 6;

for (int i = 0; i < 5; i++) cout << a[i] << " "; // 6 2 3 4 5
```

# 函数参数实现变量交换

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

void swap1(int a, int b){
    int t = a;
    a = b;
    b = t;
    cout << "函数内部: " << a << " " << b << endl;
}

void swap2(int* a, int* b){
    int t = *a;
    *a = *b;
    *b = t;
}

void swap3(int& a, int& b){
    int t = a;
```

```
    a = b;
    b = t;
}

int main() {

    int a = 1, b = 2;

    // swap1(a, b);
    // swap2(&a, &b);
    swap3(a, b);

    cout << "函数外部：" << a << " " << b << endl;

    return 0;
}
```

## 动态开辟空间

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

/*
const int N = 1e8 + 10;
int a[N];
*/

int main() {

    int* p = new int(3);
    cout << *p << endl;

    delete p;   // 释放空间

    cout << *p << endl;

    int* p2 = new int;
    cout << *p2 << endl;

    return 0;
}
```

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

int main() {

    int n;
    cin >> n;
    int* a = new int[n + 1];
```

```
    for(int i = 1; i <= n; i++)
        cin >> a[i];

    for(int i = 1; i <= n; i++)
        cout << a[i] << " ";

    delete [] a;

    return 0;
}
```

# 数组做函数参数

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

int n;
int a[110];

void f(const int a[]){
    // a[1] *= 2;
    for(int i = 1; i <= n; i++)
        cout << a[i] << " ";
}

void f2(const int* a){
    // a[1] *= 2;
    for(int i = 1; i <= n; i++)
        cout << a[i] << " ";
}

int main() {

    cin >> n;

    for(int i = 1; i <= n; i++)
        cin >> a[i];

    f(a);

    cout << endl << "---------------" << endl;

    for(int i = 1; i <= n; i++)
        cout << a[i] << " ";

    return 0;
}
```

# 字符串指针

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

int main() {

    // char str[] = "hello cat";
    const char * str = "program";

    printf("%s\n", str);
    cout << str << endl;

    str = "hello";

    printf("%s\n", str);
    cout << str << endl;

    int len = strlen(str);
    for(int i = 0; str[i]; i++)
        cout << *(str + i);

    // str[0] = 'z';

    return 0;
}
```

# 图解指针

黑猫编程

blackcat1995.com

例如：

**int a , b ;**

**int *p1 , *p2 ;**

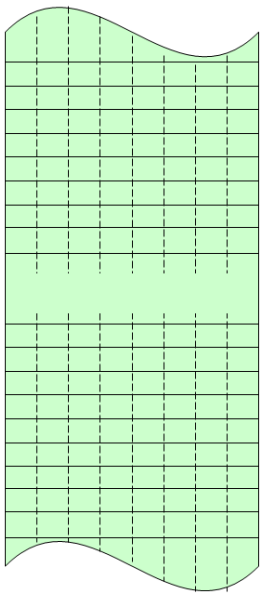
**p1 = &a ;**

**p2 = &b ;**

**a = 10 ;**

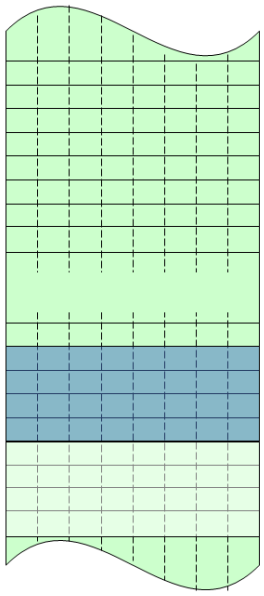
**b = 20 ;**

**a = *p1 + *p2 ;**



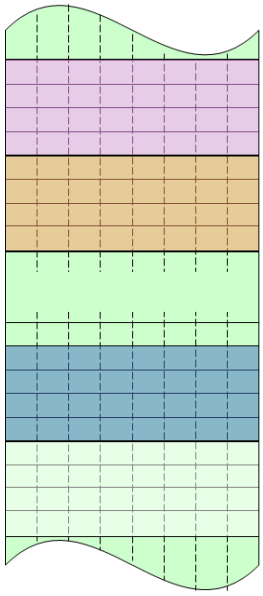
黑猫编程
blackcat1995.com

```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```



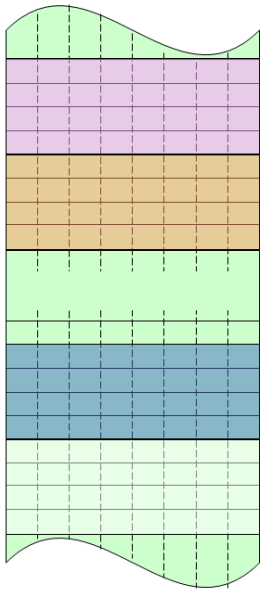

黑猫编程
blackcat1995.com

```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```



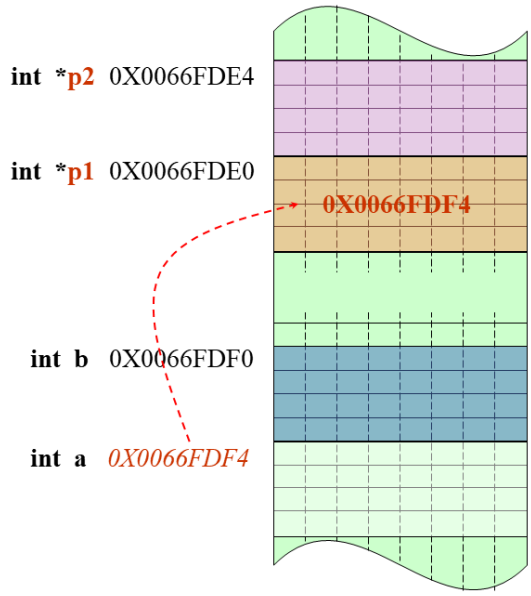

黑猫编程
blackcat1995.com

```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```



黑猫编程 blackcat1995.com

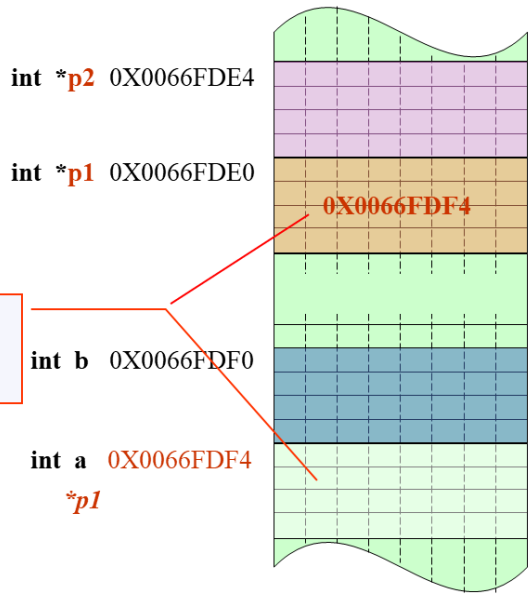
```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```




黑猫编程 blackcat1995.com

```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```

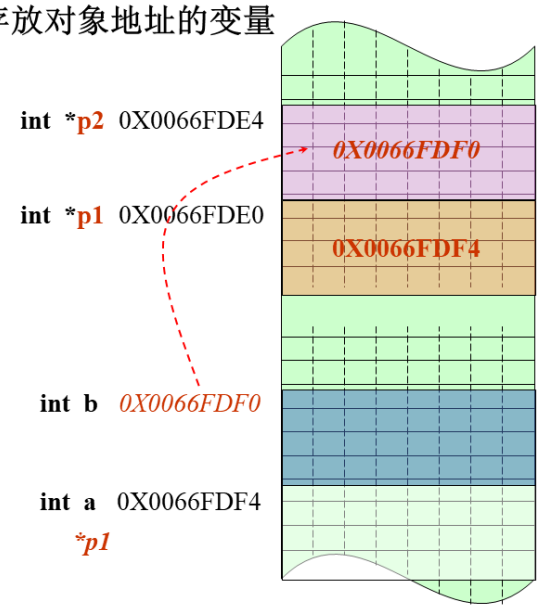
*p1
指针p1所指的对象




黑猫编程 blackcat1995.com

*指针类型变量*——能够存放对象地址的变量

```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```

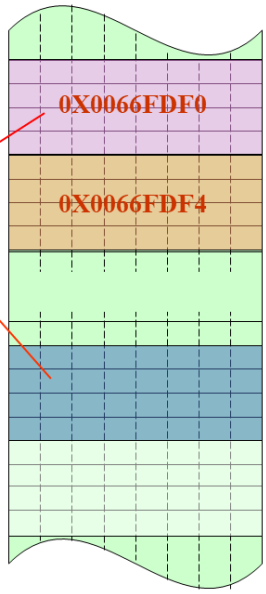


黑猫编程
blackcat1995.com

```
int a , b ;
int *p1 , *p2 ;
p1 = &
p2 = &
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```

*p2
指针p2所指的对象

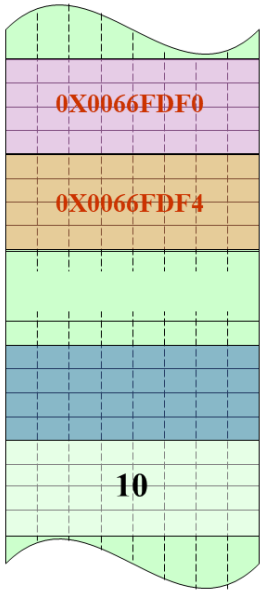
int *p2 0X0066FDE4
int *p1 0X0066FDE0
int b 0X0066FDF0
*p2
int a 0X0066FDF4
*p1




黑猫编程
blackcat1995.com

```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;  // *p1 = 10
b = 20 ;
a = *p1 + *p2 ;
```

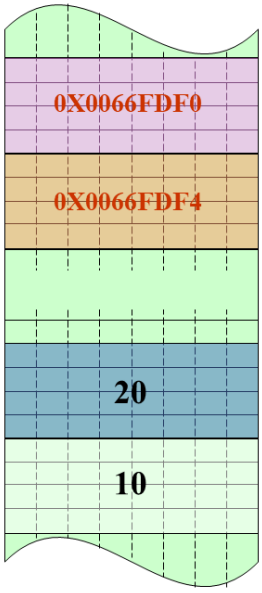
int *p2 0X0066FDE4
int *p1 0X0066FDE0
int b 0X0066FDF0
*p2
int a 0X0066FDF4
*p1




黑猫编程
blackcat1995.com

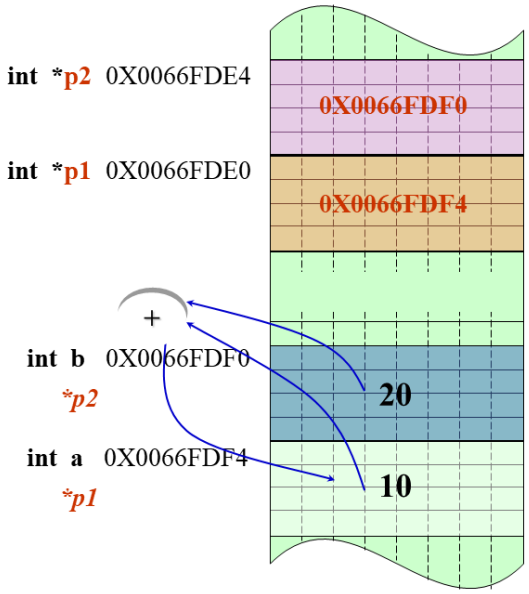
```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;  // *p2 = 20
a = *p1 + *p2 ;
```

int *p2 0X0066FDE4
int *p1 0X0066FDE0
int b 0X0066FDF0
*p2
int a 0X0066FDF4
*p1



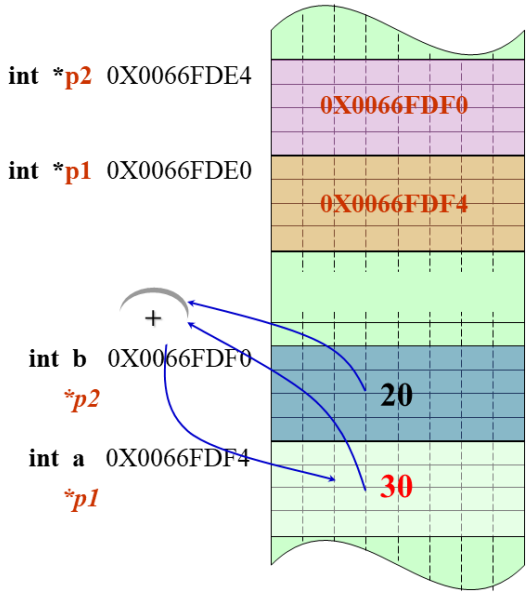
黑猫编程 blackcat1995.com

```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```




黑猫编程 blackcat1995.com

```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```

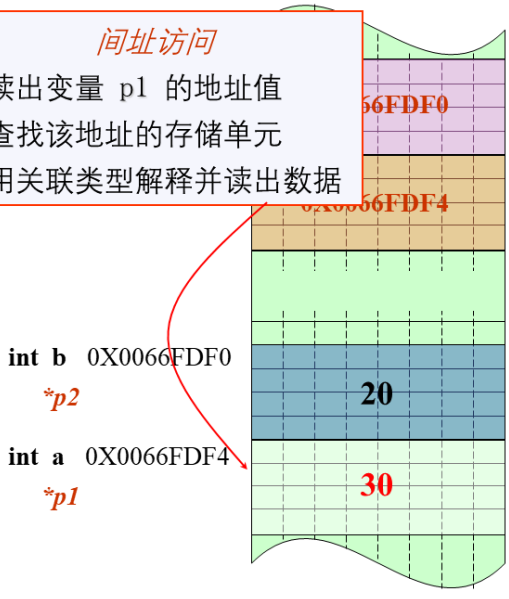



黑猫编程 blackcat1995.com

```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```

*间址访问*

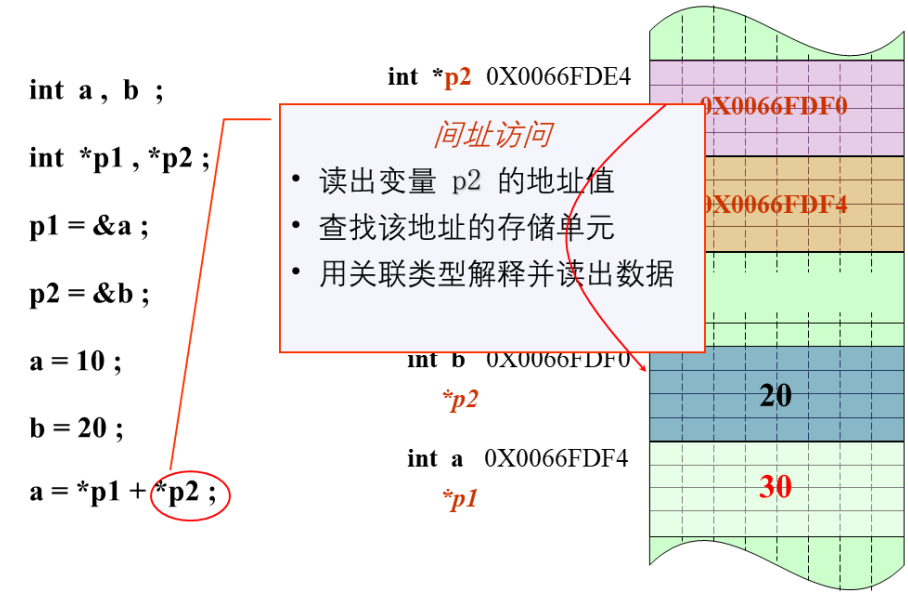
- 读出变量 p1 的地址值
- 查找该地址的存储单元
- 用关联类型解释并读出数据



```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```

*间址访问*

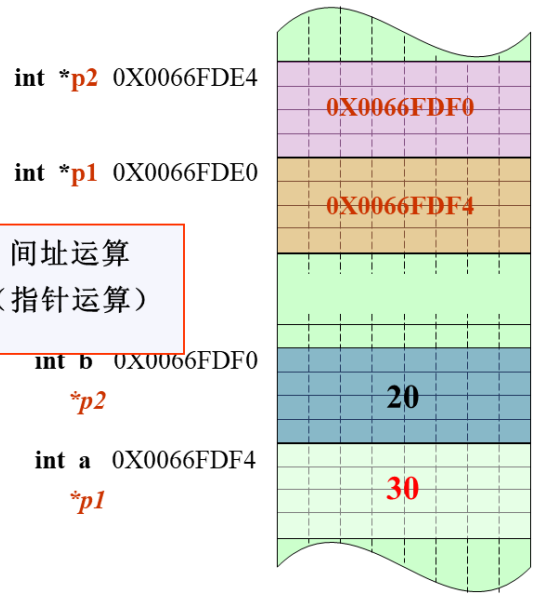
- 读出变量 p2 的地址值
- 查找该地址的存储单元
- 用关联类型解释并读出数据






```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```

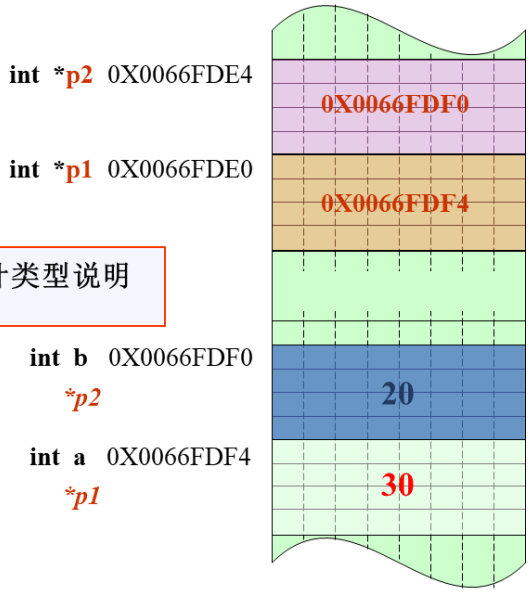
间址运算
（指针运算）






```
int a , b ;
int *p1 , *p2 ;
p1 = &a ;
p2 = &b ;
a = 10 ;
b = 20 ;
a = *p1 + *p2 ;
```

指针类型说明



## 黑猫编程 blackcat1995.com 指针与数组的关系

指向数组的指针变量称为数组指针变量。一个数组是一块连续的内存单元组成的，数组名就是这块连续内存单元的首地址。一个数组元素的首地址就是指它所占有的几个内存单元的首地址。一个指针变量即可以指向一个数组，也可以指向一个数组元素，可把数组名或第一个元素的地址赋予它。如要使指针变量指向第 i 号元素，可以把 i 元素的首地址赋予它，或把数组名加 i 赋予它。

设有数组 a，指向 a 的指针变量为 pa，则有以下关系：pa、a、&a[0]均指向同一单元，是数组 a 的首地址，也是 0 号元素 a[0]的首地址。pa+1、a+1、&a[1]均指向 1 号元素 a[1]。类推可知 pa+i、a+i、&a[i]指向 i 号元素 a[i]。pa 是变量，而 a，&a[i]是常量，在编程时应予以注意。

## 黑猫编程 blackcat1995.com 指向数组的指针

数组指针变量说明的一般形式为：

类型说明符 *指针变量名

其中类型说明符表示所指数组的类型，从一般形式可以看出，指向数组的指针变量和指向普通变量的指针变量的说明是相同的。

引入指针变量后，就可以用两种方法访问数组元素了，

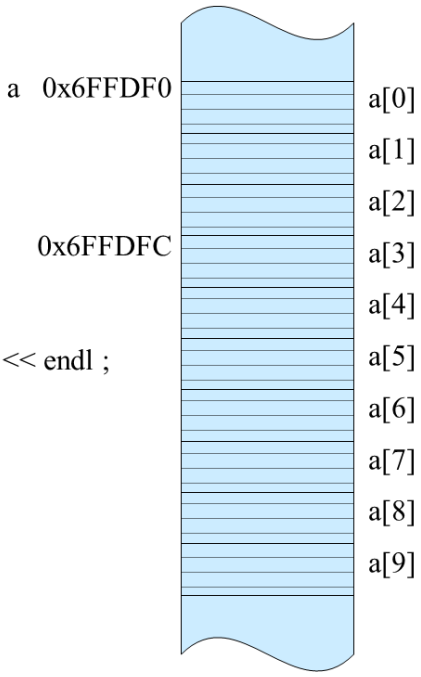
例如定义了 int a[5]; int *pa=a;

第一种方法为下标法，即用 pa[i]形式访问 a 的数组元素。

第二种方法为指针法，即采用*(pa+i)形式，用间接访问的方法来访问数组元素。

```
// 例　以指针方式访问数组
#include<iostream>
using namespace std;
int main()
{ int a[ 10 ] = { 1, 3, 5, 7, 9 } ;
  cout << "Address of array a : " << a << endl ;
  cout << "Address of element a[3] : " << &a[3] << endl ;
}
```

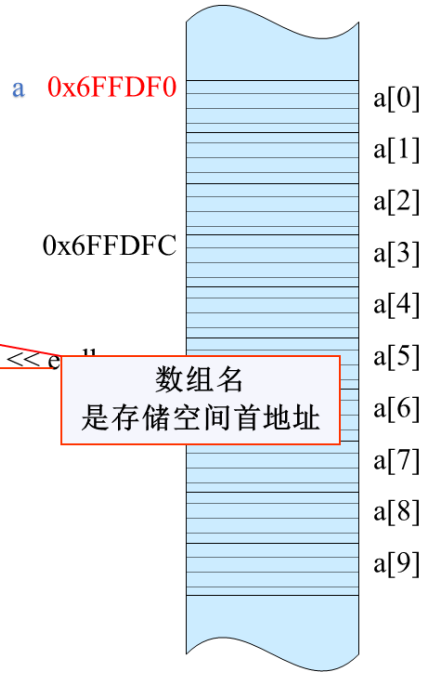
```
Address of array a : 0x6ffdf0
Address of element a[3] : 0x6ffdfc
```



*// 例　以指针方式访问数组*

```
#include<iostream>
using namespace std;
int main()
{ int a[ 10 ] = { 1, 3, 5, 7, 9 } ;
  cout << "Address of array a : " << a << endl ;
  cout << "Address of element a[3] : " << &a[3] << e
}
```

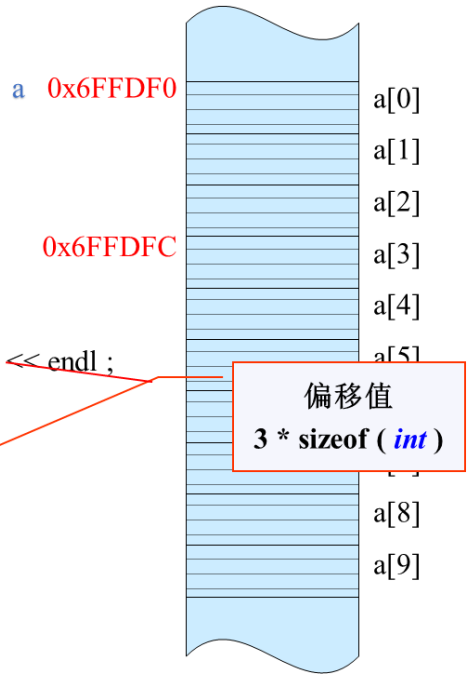
```
Address of array a : 0x6ffdf0
Address of element a[3] : 0x6ffdfc
```




*// 例　以指针方式访问数组*

```
#include<iostream>
using namespace std;
int main()
{ int a[ 10 ] = { 1, 3, 5, 7, 9 } ;
  cout << "Address of array a : " << a << endl ;
  cout << "Address of element a[3] : " << &a[3] << endl ;
}
```

```
Address of array a : 0x6ffdf0
Address of element a[3] : 0x6ffdfc
```




*以指针方式访问数组*

数组名是隐含意义的常指针（直接地址）

其关联类型是数组元素的类型

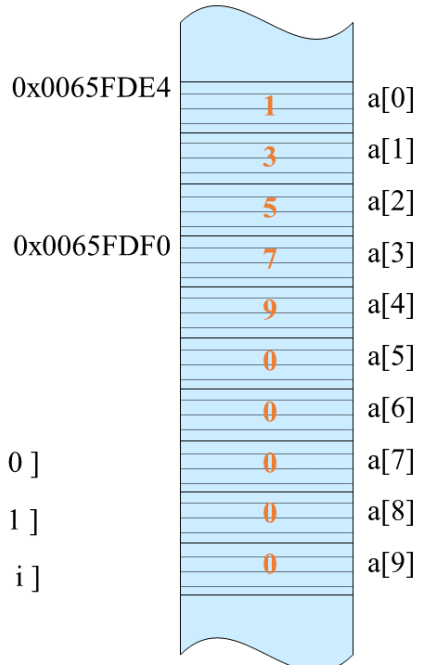
int a [ 10 ] = { 1, 3, 5, 7, 9 } ;

则：

a == & a [ 0 ]　　*a == a [ 0 ]

a + 1 == & a [ 1 ]　　* ( a + 1 ) == a [ 1 ]

a + i == & a [ i ]　　* ( a + i ) == a [ i ]



## 以指针方式访问数组

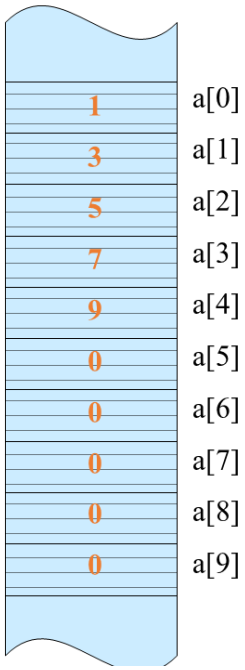
数组名是隐含意义的常指针（直接地址）

其关联类型是数组元素的类型

int a [ 10 ] = { 1, 3, 5, 7, 9 } ;

则：

| | |
|---|---|
| **a == & a [ 0 ]** | *a == a [ 0 ] |
| a + 1 == & a [ 1 ] | * ( a + 1 ) == a [ 1 ] |
| a + i == & a [ i ] | * ( a + i ) == a [ i ] |




## 以指针方式访问数组

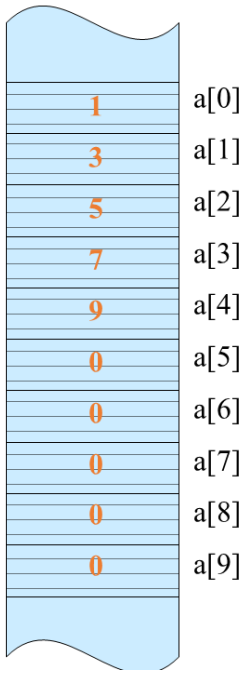
数组名是隐含意义的常指针（直接地址）

其关联类型是数组元素的类型

int a [ 10 ] = { 1, 3, 5, 7, 9 } ;

则：

| | |
|---|---|
| a == & a [ 0 ] | *a == a [ 0 ] |
| **a + 1 == & a [ 1 ]** | * ( a + 1 ) == a [ 1 ] |
| a + i == & a [ i ] | * ( a + i ) == a [ i ] |




## 以指针方式访问数组

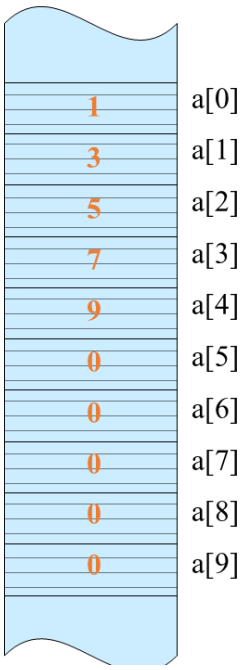
数组名是隐含意义的常指针（直接地址）

其关联类型是数组元素的类型

int a [ 10 ] = { 1, 3, 5, 7, 9 } ;

则：

| | |
|---|---|
| a == & a [ 0 ] | *a == a [ 0 ] |
| a + 1 == & a [ 1 ] | * ( a + 1 ) == a [ 1 ] |
| **a + i == & a [ i ]** | * ( a + i ) == a [ i ] |



## 以指针方式访问数组

数组名是隐含意义的常指针（直接地址）

其关联类型是数组元素的类型

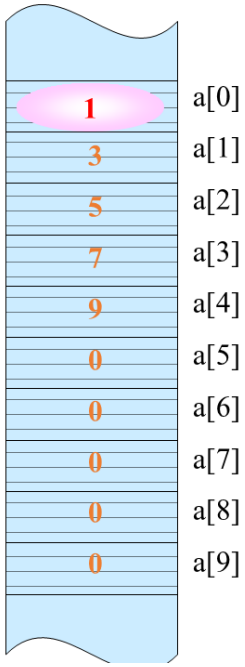
int a [ 10 ] = { 1, 3, 5, 7, 9 } ;

则：

a == & a [ 0 ]　　***a == a [ 0 ]**

a + 1 == & a [ 1 ]　　* ( a + 1 ) == a [ 1 ]

a + i == & a [ i ]　　* ( a + i ) == a [ i ]




## 以指针方式访问数组

数组名是隐含意义的常指针（直接地址）

其关联类型是数组元素的类型

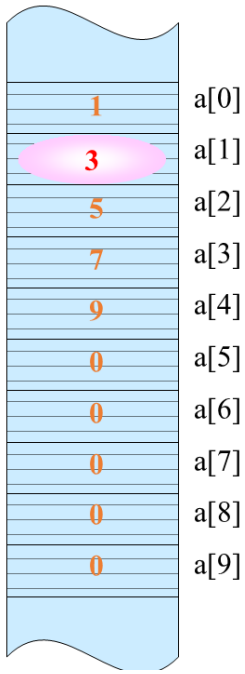
int a [ 10 ] = { 1, 3, 5, 7, 9 } ;

则：

a == & a [ 0 ]　　*a == a [ 0 ]

a + 1 == & a [ 1 ]　　*** ( a + 1 ) == a [ 1 ]**

a + i == & a [ i ]　　* ( a + i ) == a [ i ]




## 以指针方式访问数组

数组名是隐含意义的常指针（直接地址）

其关联类型是数组元素的类型

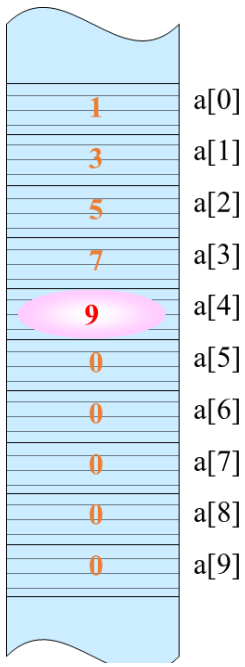
int a [ 10 ] = { 1, 3, 5, 7, 9 } ;

则：

a == & a [ 0 ]　　*a == a [ 0 ]

a + 1 == & a [ 1 ]　　* ( a + 1 ) == a [ 1 ]

a + i == & a [ i ]　　*** ( a + i ) == a [ i ]**



## 以指针方式访问数组

数组名是隐含意义的常指针（直接地址）

其关联类型是数组元素的类型

int a [ 10 ] = { 1, 3, 5, 7, 9 } ;

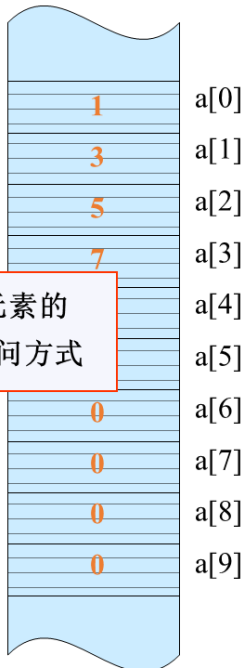
则：

a == & a [ 0 ]　　*a == a [ 0 ]

a + 1 == & a [ 1 ]　　* ( a + 1 ) == a [ 1 ]

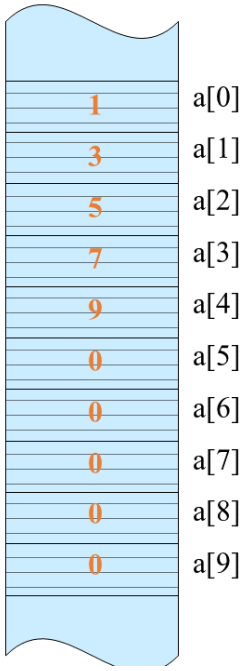
a + i == & a [ i ]　　*** ( a + i )** == a [ i ]

数组元素的指针访问方式



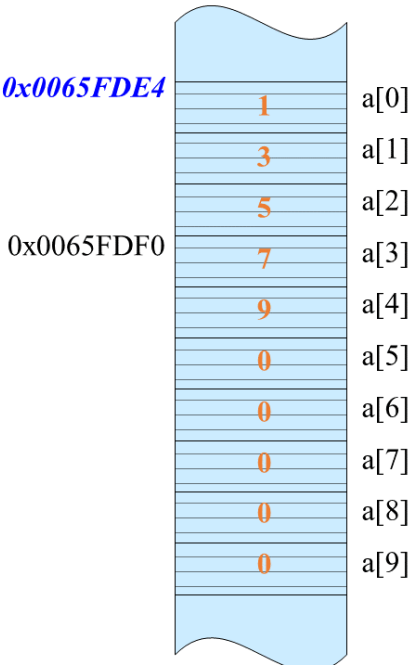

## 指针变量与数组

```
int  a [ 10 ] ;
int  * p ;
p = a ;
cout << *p ;
p ++ ;
cout << *( p++ ) ;
p += 3 ;
cout << &p ;
```



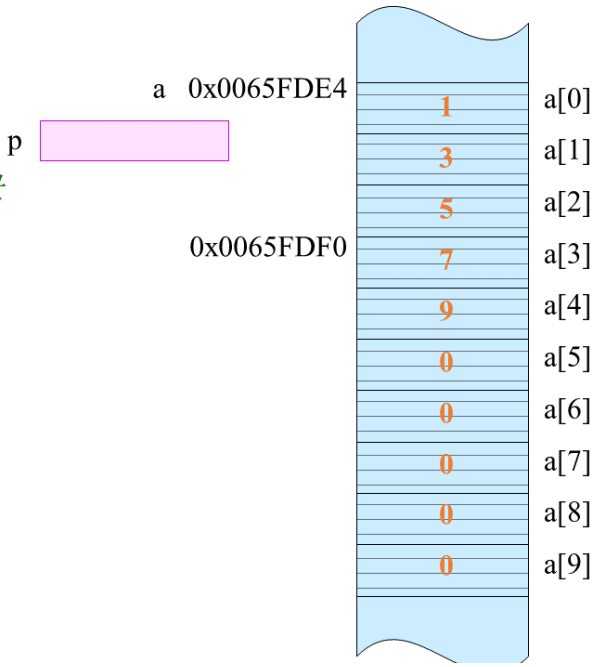

## 指针变量与数组

```
int  a [ 10 ] ;    // a 是内存的直接地址
int  * p ;
p = a ;
cout << *p ;
p ++ ;
cout << *( p++ ) ;
p += 3 ;
cout << &p ;
```



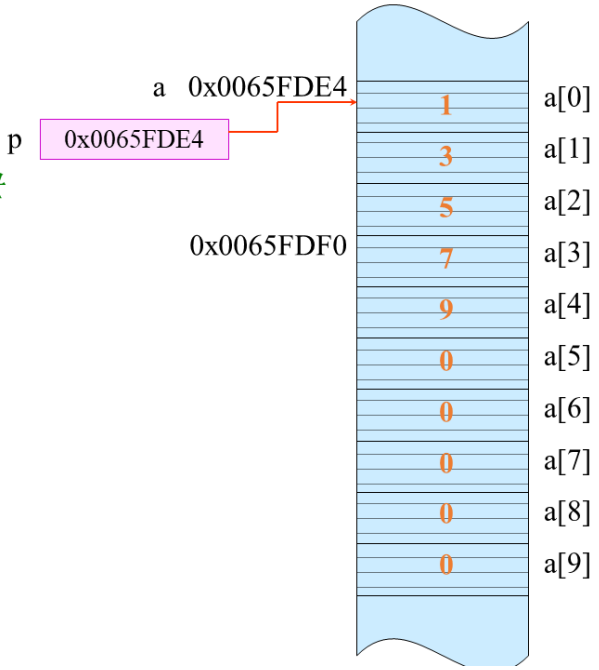
## 指针变量与数组

```
int  a [ 10 ] ;     // a 是内存的直接地址
int  * p ;          // p 是指针变量
p = a ;
cout << *p ;
p ++ ;
cout << *( p++ ) ;
p += 3 ;
cout << &p ;
```



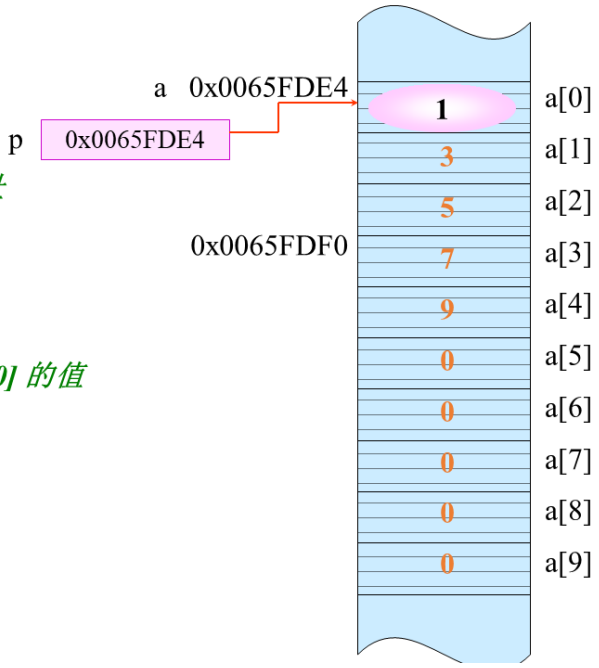

## 指针变量与数组

```
int  a [ 10 ] ;     // a 是内存的直接地址
int  * p ;          // p 是指针变量
p = a ;             // p的值是a[0]的地址
cout << *p ;
p ++ ;
cout << *( p++ ) ;
p += 3 ;
cout << &p ;
```



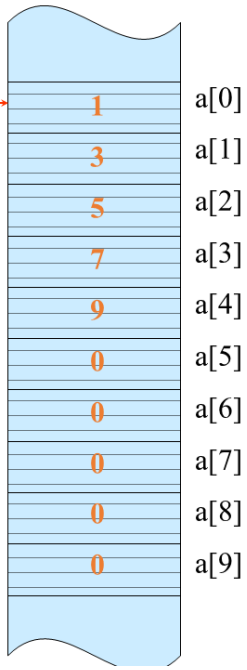

## 指针变量与数组

```
int  a [ 10 ] ;     // a 是内存的直接地址
int  * p ;          // p 是指针变量
p = a ;             // p的值是a[0]的地址
cout << *p ;        // 间址访问，输出a[0] 的值
p ++ ;
cout << *( p++ ) ;
p += 3 ;
cout << &p ;
```



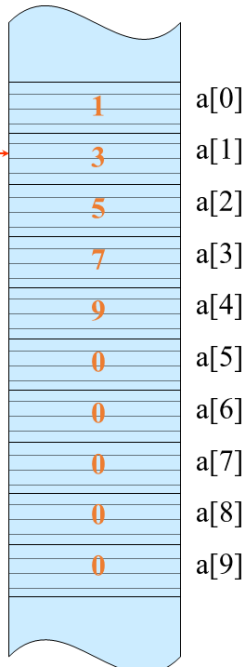
## 指针变量与数组

```
int  a [ 10 ] ;    // a 是内存的直接地址
int  * p ;              // p 是指针变量
p = a ;         // p的值是a[0]的地址
cout << *p ;    // 间址访问，输出 a[0] 的值
p ++ ;          // p 指向 a[1]
cout << *( p++ ) ;
p += 3 ;
cout << &p ;
```



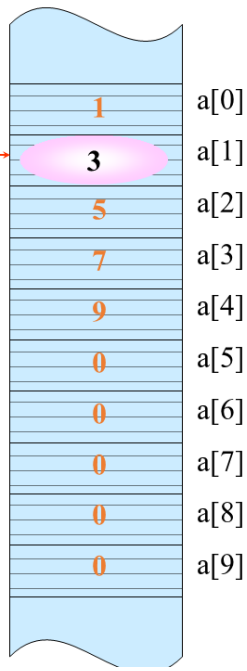

## 指针变量与数组

```
int  a [ 10 ] ;    // a 是内存的直接地址
int  * p ;              // p 是指针变量
p = a ;         // p的值是a[0]的地址
cout << *p ;    // 间址访问，输出 a[0] 的值
p ++ ;          // p 指向 a[1]
cout << *( p++ ) ;
p += 3 ;
cout << &p ;
```



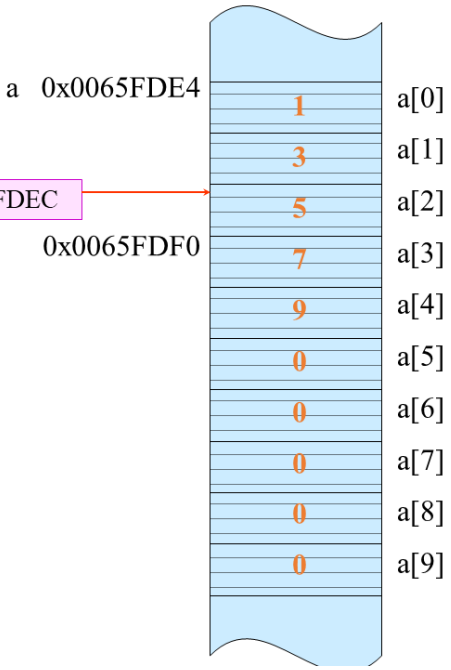

## 指针变量与数组

```
int  a [ 10 ] ;    // a 是内存的直接地址
int  * p ;              // p 是指针变量
p = a ;         // p的值是a[0]的地址
cout << *p ;    // 间址访问，输出 a[0] 的值
p ++ ;          // p 指向 a[1]
cout << *( p++ ) ;  // 输出 a[1] , p 指向 a[2]
p += 3 ;
cout << &p ;
```



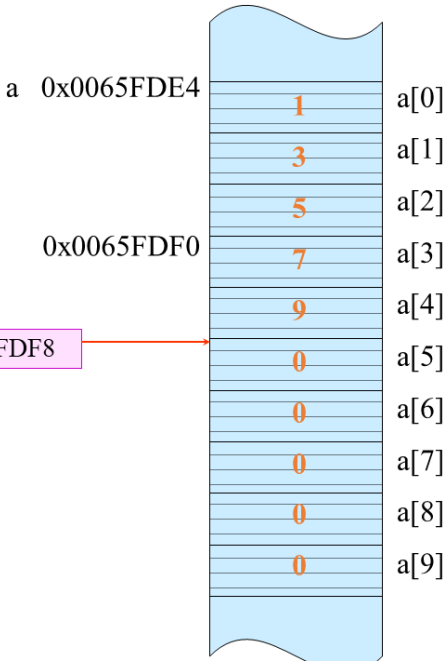
## 指针变量与数组

```
int  a [ 10 ] ;    // a 是内存的直接地址
int  * p ;              // p 是指针变量
p = a ;          // p的值是a[0]的地址
cout << *p ;    // 间址访问，输出a[0] 的值
p ++ ;          // p 指向a[1]
cout << *( p++ ) ;  // 输出a[1] , p 指向a[2]
p += 3 ;
cout << &p ;
```



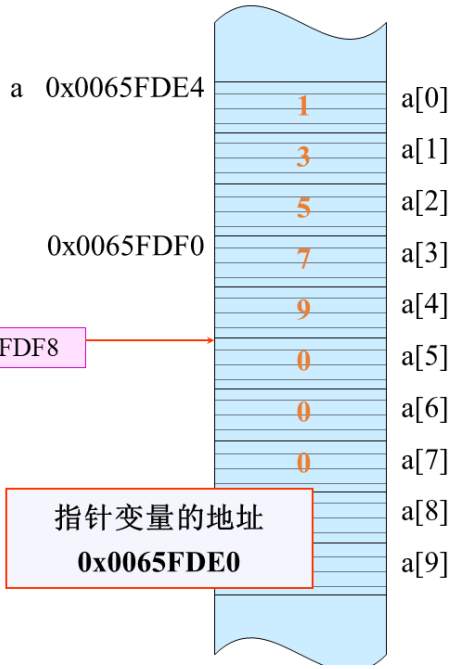

## 指针变量与数组

```
int  a [ 10 ] ;    // a 是内存的直接地址
int  * p ;              // p 是指针变量
p = a ;          // p的值是a[0]的地址
cout << *p ;    // 间址访问，输出a[0]的值
p ++ ;          // p 指向a[1]
cout << *( p++ ) ;  // 输出a[1] , p 指向a[2]
p += 3 ;        // p 指向a[5]
cout << &p ;
```



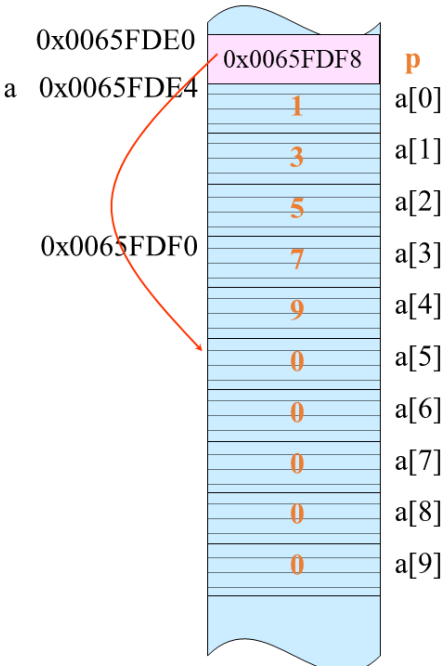

## 指针变量与数组

```
int  a [ 10 ] ;    // a 是内存的直接地址
int  * p ;              // p 是指针变量
p = a ;          // p的值是a[0]的地址
cout << *p ;    // 间址访问，输出a[0]的值
p ++ ;          // p 指向a[1]
cout << *( p++ ) ;  // 输出a[1] , p 指向a[2]
p += 3 ;        // p 指向a[5]
cout << &p ;    // 输出p 的地址
```



## 指针变量与数组

```
int  a [ 10 ] ;     // a 是内存的直接地址
int  * p ;          // p 是指针变量
p = a ;             // p的值是a[0]的地址
cout << *p ;        // 间址访问，输出a[0] 的值
p ++ ;              // p 指向a[1]
cout << *( p++ ) ;  // 输出a[1] , p 指向a[2]
p += 3 ;            // p 指向a[5]
cout << &p ;        // 输出p 的地址
```




## 黑猫编程 动态数组

blackcat1995.com

例 计算前缀和数组，b 是数组 a 的前缀和的数组定义：b[i]=a[1]+a[2] +…+a[i]，即 b[i]是 a 的 i 个元素的和。

```
#include <cstdio>
int main() {
    int n;
    scanf("%d", &n);
    int* a;         // 定义指针变量a，作为数组名
    a = new int[n+1];
    for(int i=1;i<=n;i++)
        scanf("%d", &a[i]);
    for(int i=2;i<=n;i++)
        a[i] += a[i-1];
    for(int i=1;i<=n;i++)
        printf("%d ", a[i]);

    return 0;
}
```

```
5
1 2 3 4 5
1 3 6 10 15
```

使用“动态数组”，在确保小数据没有问 题前提下，尽量满足大数据需求

## 黑猫编程 字符串的表示形式

blackcat1995.com

在 C++中，我们可以用两种方式访问字符串。

- 用字符数组存放一个字符串，然后输出该字符串。
  - char str[]="I love China!";
  - printf("%s\n", str);
- 用字符指针指向一个字符串。可以不定义字符数组，而定义一个字符指针。用字符指针指向字符串中的字符。
  - const char *str="I love China!";
  - printf("%s\n", str);

在这里，我们没有定义字符数组，而是在程序中定义了一个字符指针变量 str，用字符串常量"I love China!"，对它进行初始化。C++对字符串常量是按字符数组处理的，在内存中开辟了一个字符数组用来才存放该字符串常量。对字符指针变量初始化，实际上是把字符串第 1 个元素的地址（即存放字符串的字符数组的首元素地址）赋给 str。有人认为 str 是一个字符串变量，以为在定义时把"I love China!"这几个字符赋给该字符串变量，这是不对的。

实际上，const char *str="I love China!";
等价于：const char *str;
str="I love China!";

可以看到，str 被定义为一个指针变量，指向字符型数据，请注意它只是指向了一个字符变量或其他字符类型数据，不能同时指向多个字符数据，更不是把"I love China!"这些字符存放到 str 中（指针变量只能存放地址）。只是把"I love China!"的第一个字符的地址赋给指针变量 str。



在输出时，要用：printf("%s\n", str);

其中"%s"是输出字符串时所用的格式符，在输出项中给出字符指针变量名，则系统先输出它所指向的一个字符数据，然后自动是 str 加 1，使之指向下一个字符，然后再输出一个字符……如此知道遇到字符串结束标志"\0"为止。

注意：可以通过字符数组名或者字符指针变量输出一个字符串。而对一个数值型数组，是不能企图用数组名输出它的全部元素的。

例如：
int i[10];
……
printf（"%d\n", i）；

这样是不行的，只能逐个输出。显然 %s 可以对一个字符串进行整体的输入和输出。



## 字符串指针作函数参数

将一个字符串从一个函数传递到另外一个函数，可以用地址传递的方法，即用字符数组名作参数或用指向字符的指针变量做参数。在被调用的函数中可以改变字符串内容，在主调函数中可以得到改变了的字符串。

例：输入一个长度最大为 100 的字符串，以字符数组的方式储存，再将字符串倒序储存，输出倒序储存后的字符串。(这里以字符指针为函数参数)

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

char str[110];

void myswap(char& a, char& b){
    char t = a;
    a = b;
    b = t;
}

void work(char* str){
    int len = strlen(str);
    for(int i = 0; i < len / 2; i++)
        myswap(str[i], str[len - i - 1]);
}

int main() {

    char * pstr = str;
    cin.getline(pstr, 110);
    work(pstr);
    cout << pstr << endl;

    return 0;
}
```