移动平台应用软件开发
C/C++/JAVA基础

指针以及指针操作

主讲：张齐勋

zhangqx@ss.pku.edu.cn

《移动平台应用软件开发》课程建设小组

北京大学

二零一五年

北京大学

一、关于内存

- 数据存储
  - 变量分类
    - 局部变量
    - 全局变量
    - 静态变量
  - 请看下面代码，分析变量类型?

```
int  pi = 3;
int  Area(int r, int *sum)
{
        int b;
        static int c =0;
        b = pi * r * r;
        c += b;
        *sum = c;
        return b ;
}
```

- 变量的存储
  - 内存：变量的存储位置
  - 不同类型的变量，存储在不同内存区域；
- 内存类型
  - 内存分为静态内存和动态内存，动态内存又可以分为堆内存和栈内存。内存分配方式相应的分为静态存储区域分配、栈创建、堆分配。
  - 静态存储区域分配
    - 程序在编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。
  - 栈创建
    - 函数执行的时候，函数内的局部变量（包括函数参数）的存储单元都是在栈上创建的，函数执行后占用栈的存储单元会自动被释放。这种分配效率很高，但是分配的内存容量有限。占用栈内存多了，会出现栈溢出；
  - 堆分配
    - 也称动态存储分配，是指程序在运行时候用malloc申请的内存，程序员自己负责在何时用free释放内存。

内存区通常分为4个区域：

栈区

存放程序的局部变量

静态区

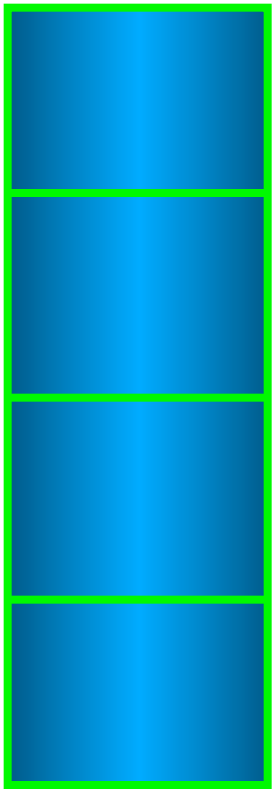
存放程序的全局数据和静态数据

堆区

存放程序动态申请的数据

代码区

存放程序的代码



数据区

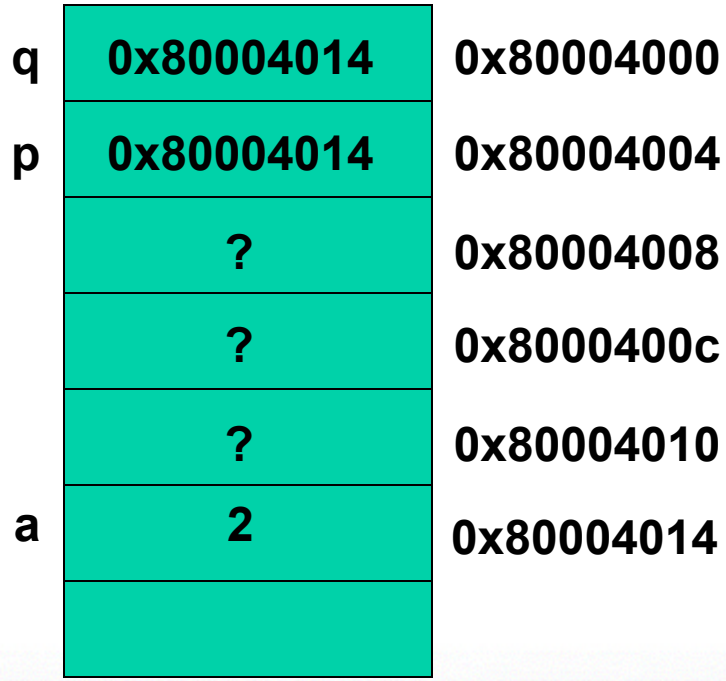
代码区

memory

二、指针概念

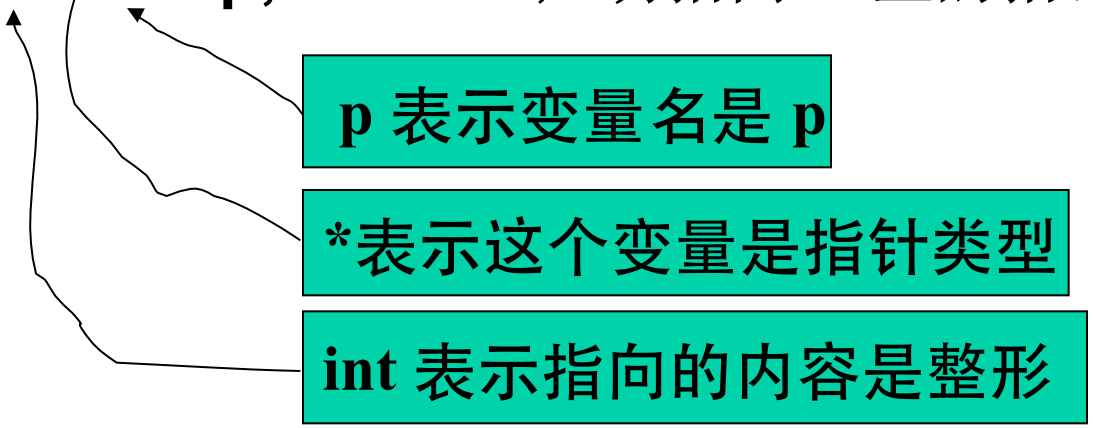
- 指针变量是一种数据类型，内容存放的是另外一个变量的内存地址。

```
int a = 2;
int *p = &a;
int *q = p;
```



指针声明和赋值操作

int *p;　　// 声明指向int型的指针变量p

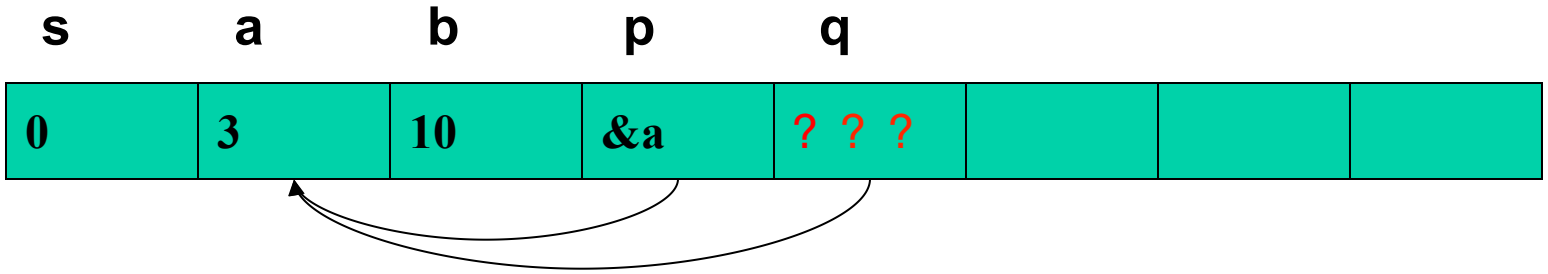


p = &a;　　// 把指针 p 指向 a

把变量 a 的地址赋给指针 p，即把指针 p 指向 a

指针声明和赋值操作

```
int s = 0, a = 3, b = 10;
int *p;          // 声明指向int型的指针
p = &a;          // 把指针 p 指向 a
int *q = p;      // 两个指针指向内容类型一致时才能赋值
```



例：查看指针变量值

```
1 #include <stdio.h>
 2 int main(void)
 3 {
 4     int a=2;
 5     int *p=&a;
 6     int *q=p;
 7     printf("&a=%p\n",&a);
 8     printf("   p=%p\n",p);
 9     printf("   q=%p\n",q);
10     printf("&p=%p\n",&p);
11     printf("&q=%p\n",&q);
12     printf(“   a=%d\n",a);
13     return 0;
14 }
```

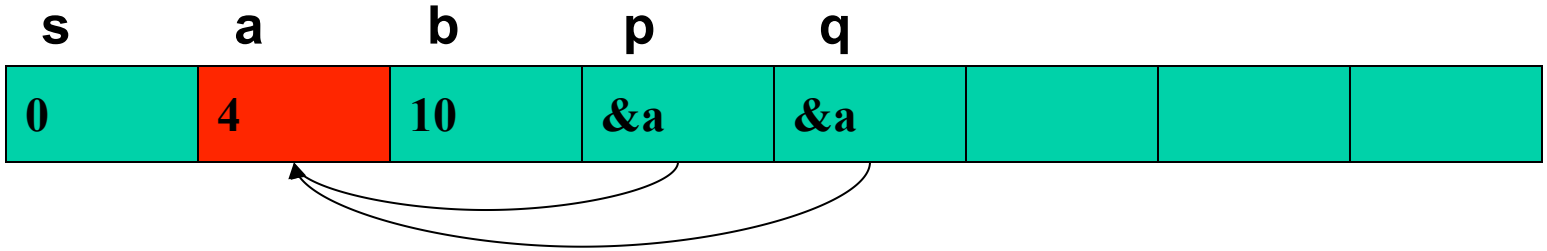
例：通过指针变量访问整型变量

```
1 #include<stdio.h>
 2 int main(void)
 3 {
 4     int a,b;
 5     int *pointer_1,*pointer_2;
 6     a=100;b=10;
 7     pointer_1=&a;
 8     pointer_2=&b;
 9     printf("%d,%d\n",a,b);
10     printf("%d,%d\n",*pointer_1,*pointer_2);
11     return 0;
12 }
13
```

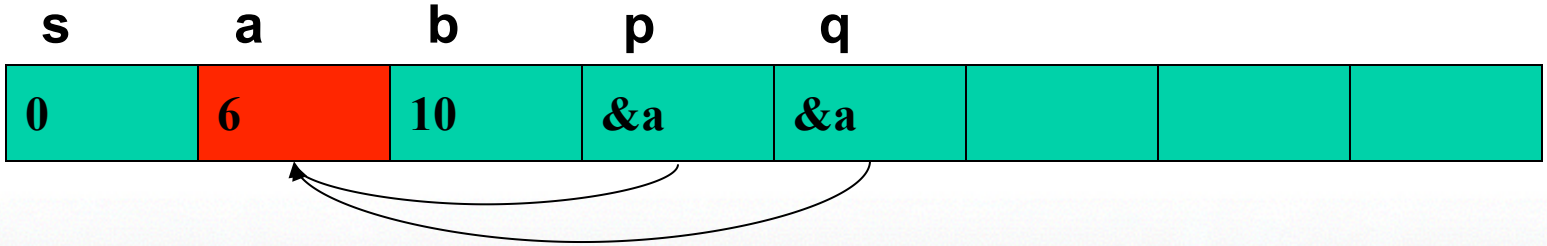
运行结果：
100,10
100,10

访问指针指向的变量内容

(*p) ++; //等价于a++, 能不能去掉括号？



*p = 6; //等价于a=6



使用指针注意事项

- int *p1, *p2; 与 int *p1, p2;
- 指针变量名是p1,p2 ,不是*p1,*p2
- 指针变量只能指向定义时所规定类型的变量
- 指针变量定义后，变量值不确定，应用前必须先赋值。

思考题

- 思考：下面代码的含义和对内存的修改

```
char c1 = 'A', c2 = 'B', tmp;
char *p = &c1, *q = &c2;
tmp = *p;
*p = *q;
*q = tmp;
// c1 和 c2 的值变成了什么？
// tmp 值呢？
```

指针类型的函数参数和返回值

```
1 #include <stdio.h>
 2 void inc(int ai)
 3 {
 4     printf("ai=%d\n",ai);
 5     ai++;
 6     printf("ai=%d\n",ai);
 7 }
 8
 9 int main(void)
10 {
11     int a=0;
12     printf("a=%d\n",a);
13     inc(a);
14     printf("a=%d\n",a);
15     return 0;
```

- 左边的代码能把a值进行加1操作吗？为什么？
- 函数传值调用时，a值会被复制一份，副本被传入函数内部，a值本身并没有修改
- 使用传址调用，把a的地址传进函数内部，可以修改a值

指针类型的函数参数和返回值

```
1 #include <stdio.h>
2 void inc(int ai)
3 {
4     printf("ai=%d\n",ai);
5     ai++;
6     printf("ai=%d\n",ai);
7 }
8
9 int main(void)
10 {
11     int a=0;
12     printf("a=%d\n",a);
13     inc(a);
14     printf("a=%d\n",a);
15     return 0;
```

```
1 #include <stdio.h>
2 void inc(int *ai)
3 {
4     printf("*ai=%d\n",*ai);
5     (*ai)++;
6     printf("*ai=%d\n",*ai);
7 }
8
9 int main(void)
10 {
11     int a=0;
12     printf("a=%d\n",a);
13     inc(&a);
14     printf("a=%d\n",a);
15     return 0;
```

指针类型的函数参数和返回值

指针参数可以把变量的地址传给函数，从而在被调用函数里面修改调用者内部的变量。

右边的代码用于交换主函数里面的 i 和 j 变量的值，但交换过程是在子函数 **swap()** 中进行的。

返回了交换后的**px**指针，即原来第二个变量的地址。

```
1 #include <stdio.h>
2 int *swap(int *px,int* py)
3 {
4     int temp;
5     temp=*px;
6     *px=*py;
7     *py=temp;
8     return px;
9 }
10 int main(void)
11 {
12     int i=10, j=20;
13     int *p=swap(&i,&j);
14     printf("Now
              i=%d,j=%d,*p=%d
\n",i,j,*p);
15     return 0;
```

指针类型的函数参数和返回值

- 常见错误：返回函数局部变量的地址

```
int *getNum(){
    int a = 3;
    return &a;
} //出此函数后a已被释放
```

- 变量 a 在函数 getNum() 内部定义，生命周期只在函数的执行时期内。函数退出后变量 a 消亡，成为非法的内存空间，值可能被别的函数修改，再次访问 a 得到的值不可预测。
- int *swap(int *px, int *py) 返回的 px 是从外部传进来的，不是在函数内部定义的，swap() 退出后仍然存在。

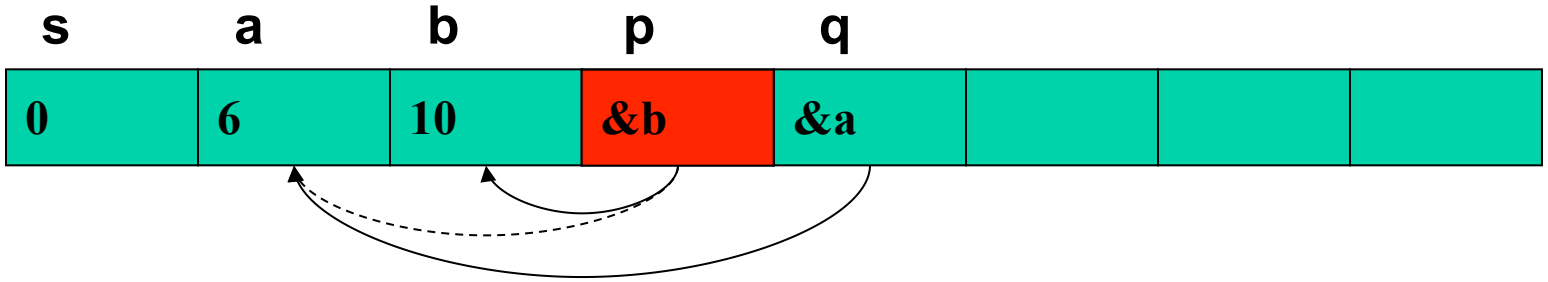
三、指针的基本操作

修改指针本身

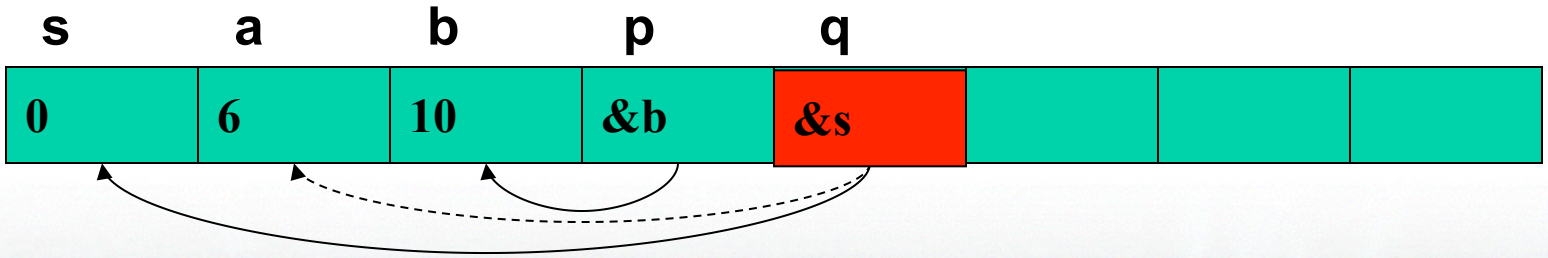
p ++;

指针运算 p++

相当于：地址 += sizeof(指向的变量类型)



q --;



北京大学

运算符*和&的优先级

- 单目运算符： ++ -- ！ * &, 优先级仅次于括号，并从右到左结合
- *p ++ // ++ 优先，与 p 结合
- (*p) ++ // 括号优先级高于 ++, 先行运算
- ++ *p // *p 先运算
- if(*p != 5) // 单目运算符优先级高于 !=
- b = *p + 10 // + 是加法运算符，优先级低于 *
- b = *(p + 10) // 先运算括号里面加法

零指针与空类型指针

零指针：(空指针)

定义:指针变量值为零

表示： int * p=0;

p指向地址为0的单元，
表示指针变量值没有意义

```
#define   NULL   0
int   *p=NULL:
```

p=NULL与未对p赋值不同

用途:

避免指针变量的非法引用

在程序中常作为状态比较

void *类型指针

表示：void *p;

void*类型指针与其他类型指针可以隐式转换

四、指针与数组

指向数组元素的指针变量

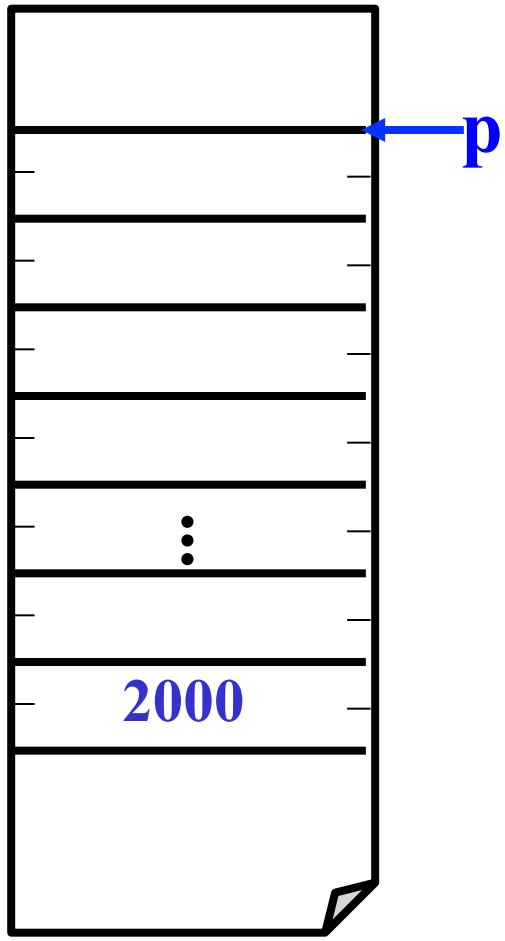
定义　　int　a[10];

　　int　*p;

赋值　　p=&a [0];

等价于

　　p=a;



也可，在定义指针变量时赋初值

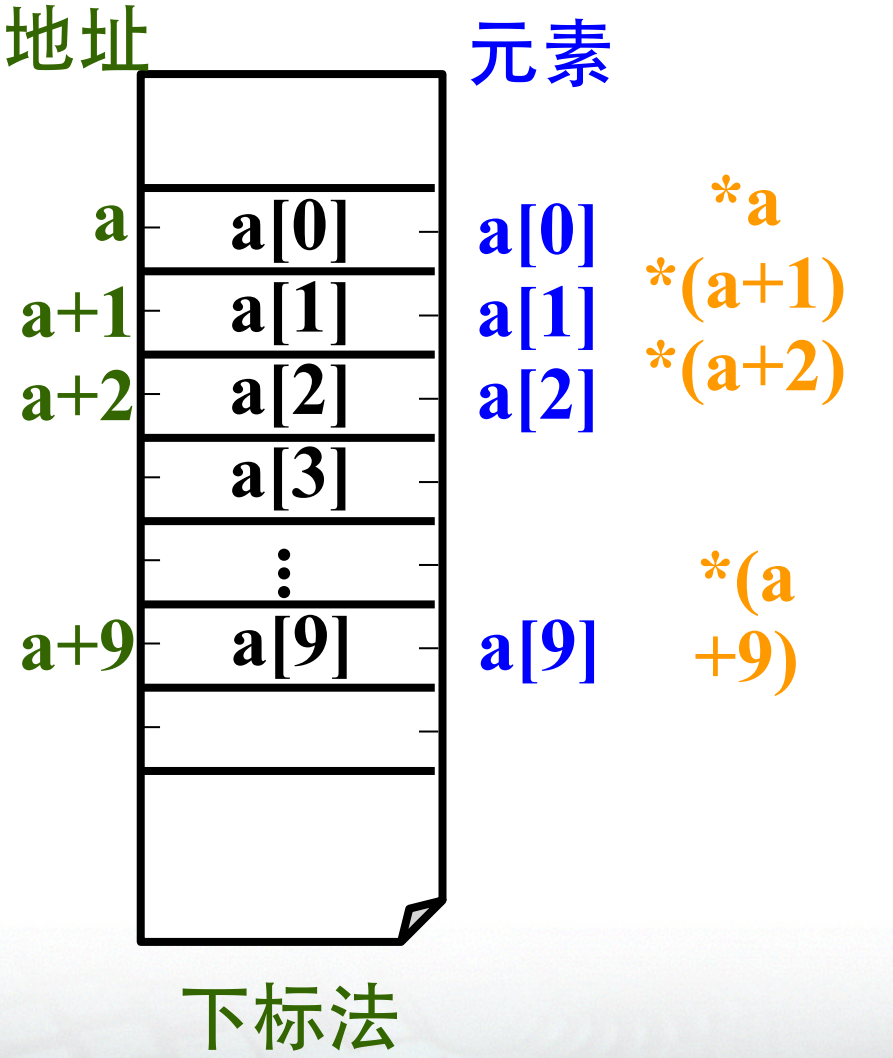
int *p＝a;

数组名是表示数组首地址的地址常量

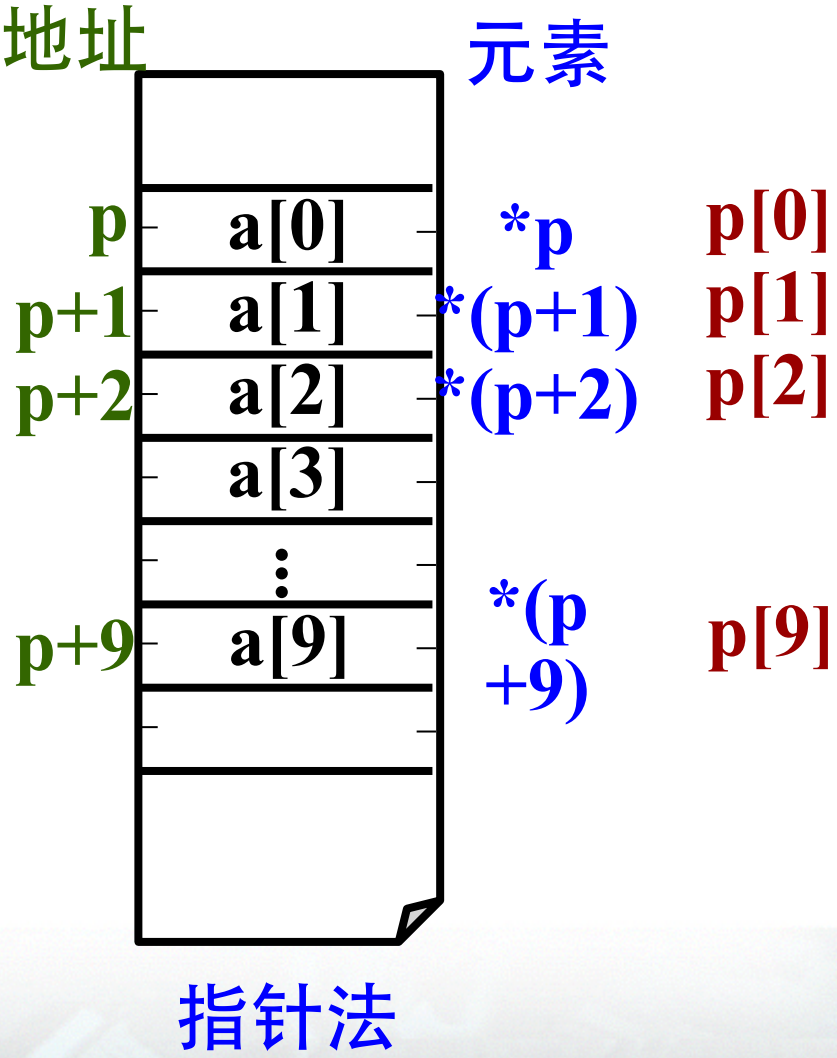
p=a的作用是把数组的首地址赋给指针变量P，

而不是把数组a的各元素赋给p。

数组元素表示方法



下标法



指针法

北京大学

	首址	偏移量	运算
下标法	a	i	a[i]
计算法	a	i	*(a+i)
指针法	p	已用p++移到当前位置	*p

- 无论是下标法（**a[i]**）还是计算法（***(a+i)**）还是指针法（*** (p +i)** ）,尽管表现形式不同，可本质都是：
  - ***（首址+ 偏移量）**

例　输出数组中的全部元素（3种方法）

（1）下标法

```
void main()
{    int a[10]
     int i;
     for(i=0;i<10;i++)
          scanf("%d",&a[i]);
     printf("\n");
     for(i=0;i<10;i++)
          printf("%d",a[i]);
}
```

（2）数组名计算数组元素的地址

```
void main()
{    int a[10]
     int i;
     for(i=0;i<10;i++)
          scanf("%d",&a[i]);
     printf("\n");
     for(i=0;i<10;i++)
          printf("%d",*(a+i));
}
```

（3）指针变量法

```
void main()
{    int a[10];
     int *p,i;
     for(i=0;i<10;i++)
          scanf("%d",&a[i]);
     printf("\n");
     for(p=a;p<(a+10);p++)
          printf("%d",*p);
}
```

指针变量的运算

p++，使p指向下一元素a[1]；

*p++，＋＋和*同优先级，自右至左，等价于*（p++）

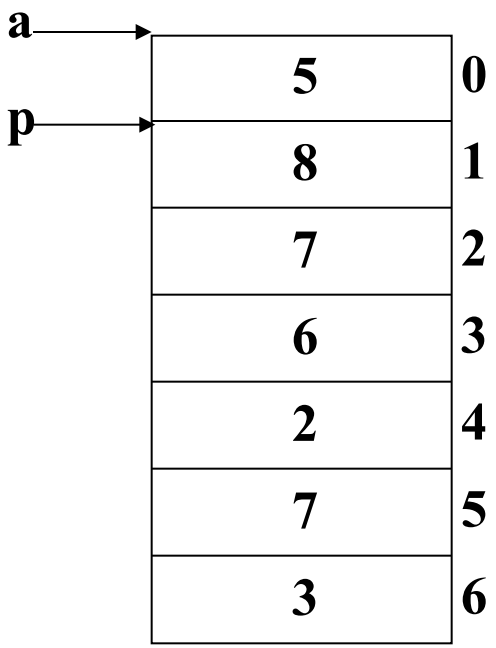
（p++）和（ ++ p）作用不同。前者，先取*p值，再使p+1。后者先使p+1，再取*p。

＋＋和－－运算符可以使指针变量向前或向后移动。

若p1与p2指向同一数组，p1-p2=两指针间元素个数⇔(p1-p2)/d

例　注意指针变量的运算

```
例
void main()
{       int   a []={5,8,7,6,2,7,3};
        int y,*p=&a[1];
        y=(*--p)++;
        printf("%d",y);
        printf("%d",a[0]);
}
```



输出：5　6

数组名作函数参数

```
main()
{int array[10];
  ……
  f(array,10);
  ……
}
```

```
f(int arr[],int n )
{
  ……
}
```

array实参数组名，代表该数组首元素的地址

arr形参数组名，用来接收从实参传递过来的数组首元素的地址，实际上，形参数组名是按指针变量处理的

```
1 #include <stdio.h>
 2 void init(int a[],int len)
 3 {
 4     while(len>0){
 5         *a++=len--;
 6     }
 7 }
 8 int main(void)
 9 {
10     int a[10];
11     init(a,10);
12     int i=0;
13     for(i=0;i<10;i++)
14         printf("%d ",a[i]);
15     printf("\n");
16     return 0;
17 }
```

输出：10 9 8 7 6 5 4 3 2 1

init函数形参int a[]，a++为什么可以？

五、指针与const限定符

```
const int *a;
int const *a;
```

a是一个指向const int 类型的指针，a所指向的内存单元的内容不可改写。

```
int * const a;
```

a是一个指向int 类型的const指针，a不可改写， a所指向的内存单元可改写。

```
int const * const a;
```

a是一个指向const int 类型的const指针，a不可改写， a所指向的内存单元内容不可改写。

- 指向非const变量的指针或者非const变量的地址可以传给指向const变量的指针， 编译器可以做隐式类型转换。
- 指向const变量的指针或者const变量的地址不可以传给指向非const变量的指针。
- 即使不用const限定符也能写出功能正确的程序，但良好的编程习惯应该尽可能多地使用const。

```
int main(void)
{
   char *p = "abcd";
   ...
   *p = 'A';
   ...
}
```

编译时不出错，
运行时出错。

```
int main(void)
{
   char const  *p = "abcd";
   ...
   *p = 'A';
   ...
}
```

编译时出错。

```
1 void add(int *a, const int *b)
 2 {
 3     *a += *b;
 4     //*b=8;
 5 }

 6 int main(void)
 7 {
 8     int a=1,b=3;
 9     add(&a,&b);
10     return 0;
11 }
```

指针b指向的内容不可修改，*b=8 会引起编译错误。

六、指向指针的指针与指针数组

- 指针可以指向基本类型，也可以指向复合类型，因此也可以指向另外一个指针变量，称为指向指针的指针（指针）。

```
int i;
int *pi = &i;
int **ppi = &pi;
```

- 数组中的每个元素可以是基本类型，也可以复合类型，因此也可以是指针类型，称为指针数组。

```
int *a[10];
```

```
int *a[10];
int **pa = &a[0];
```

- main函数的标准原型应该是int main(int argc, char *argv[]);
  - argc是命令行参数的个数。
  - argv是一个指向指针的指针，为什么不是指针数组呢？

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   int i;
  for(i = 0; i < argc; i++)
    printf("argv[%d]=%s\n", i, argv[i]);
  return 0;
}
```

```
$ gcc main.c
$ ./a.out a b c
argv[0]=./a.out
argv[1]=a
argv[2]=b
argv[3]=c
```

七、数组指针

- 定义一个指向数组的指针，该数组有10个int元素：

```
int (*a)[10];
```

- 和指针数组的定义int *a[10];相比，仅仅多了一个()括号。
- 如何记住和区分这两种定义呢？
  - []比*有更高的优先级
  - 如果a先和*结合则表示a是一个指针
  - 如果a先和[]结合则表示a是一个数组。

八、函数指针

- 在C语言中，函数也是一种类型，可以定义指向函数的指针。
- 指针变量的内存单元存放一个地址值，而函数指针存放的就是函数的入口地址（位于.text段）。

```
void say_hello(const char *str)
{
    printf("Hello %s\n", str);
}
int main(void)
{
    void (*f)(const char *);
    f = say_hello;
    (*f)("Guys");
    return 0;
}
```

void (*f)(const char *)

- f首先跟*号结合在一起，因此是一个指针。
- (*f)外面是一个函数原型的格式，参数是const char *，返回值是void，所以f是指向这种函数的指针。
- 而say_hello的参数是const char *，返回值是void，正好是这种函数，因此f可以指向say_hello。

注意：say_hello是一种函数类型，做右值使用时自动转换成函数指针类型，所以可以直接赋给f。

  - 也可以写成 f= &say_hello;，

指针函数与函数指针数组

- 指针函数在调用后返回一个指针， 通过指针中存储的地址值主调函数就能访问该地址中存放的数据，并通过指针算术运算访问这个地址的前、后内存中的值。

–数据类型 *函数名(参数表)

➢数据类型是函数返回的指针所指向数据的类型。

➢*函数名声明了一个指针型的函数。

➢参数表是函数的形参列表

```
–int *fun(int a,int b);
```

函数指针数组

数据类型 (*函数指针名[常量表达式])(参数表);

与定义一般变量指针数组一样，C语言中也可以定义具有特定返回类型和特定参数类型的函数指针数组。

–int (*p[5])(int,int);

上述语句定义了一个含有5个元素的函数指针数组， 其中的每个元素都是一个指向函数的指针，且指向的函数都是返回值类型为整型、带两个整型参数的函数。

总结

- *p++与(*p)++
- void *的使用
- const与指针结合的用法
- 指针数组与数组指针
- 函数指针的作用？
- 打印内存地址 ： %p与0x%x

指针的数据类型

定义	含义
int i;	定义整型变量i
int *p;	p为指向整型数据的指针变量
int a[n];	定义含n个元素的整型数组a
int *p[n];	n个指向整型数据的指针变量组成的指针数组p
int (*p)[n];	p为指向含n个元素的一维整型数组的指针变量
int f();	f为返回整型数的函数
int *p();	p为返回指针的函数，该指针指向一个整型数据
int (*p)();	p为指向函数的指针变量，该函数返回整型数
int **p;	p为指针变量，它指向一个指向整型数据的指针变量

例　下列定义的含义

（1）int *p[3];　　指针数组

（2）int (*p)[3];　　指向一维数组的指针

（3）int *p(int);　　返回指针的函数

（4）int (*p)(int);　　指向函数的指针，函数返回int型变量

（5）int *(*p)(int);　　指向函数的指针，函数返回int 型指针

（6）int (*p[3])(int);　　函数指针数组，函数返回int型变量

（7）int *(*p[3])(int);　　函数指针数组，函数返回int型指针

- 关于“野指针”
  - “野指针”产生的3种原因：
    - 指针变量没有被初始化。任何指针变量刚被创建时不会自动成为**NULL**指针，它的默认值是随机的。
    - 指针p被free之后，没有置为NULL，让人误以为p是个合法的指针。
    - 指针操作超越了变量的作用范围。
  - 避免“野指针”产生的对策：
    - 使用指针前一定要保证它指向了有效的内存空间（或者申请，或者让指针指向一块合法的空间；不要忘记为数组和动态内存赋初值。
    - 用malloc申请内存之后，应该立即检查指针值是否为NULL。防止使用指针值为NULL的内存；用free释放了内存之后，立即将指针设置为NULL。
    - 避免数组或指针的下标越界，特别要当心发生“多1”或者“少1”操作。

Q&A

本讲结束！