CNIC　杜川

C语言那些事

CONTENT

- 一些关键字解析
- 符号优先级
- 指针与数组

关键字

- 定义与声明
- Register
- Static
- Sizeof
- Signed/unsigned
- Const
- Struct
- union

关键字——从定义和声明说起

- 何为定义，何为声明？
  - 定义：编译器创建一个对象，为对象分配内存并给它去一个名字。名字和内存相绑定，内存位置也不能被改变。
  - 声明：告诉编译器名字已经匹配到一块内存上了。以后用到这个名字都是在别处已经定义得。
- 两者区别：定义分配了内存，声明没有分配内存。
- 两者联系：定义一定是声明，声明不一定是定义。

番外篇——定义一个变量背后的故事

- 作用域
- 链接属性
- 存储类型

作用域

- 变量在被声明时，它只在程序的一定区域才能被访问。这个区域由变量的作用域决定。
- 编译器可以确定的作用域类型:
  - 文件作用域
  - 函数作用域
  - 代码块作用域
  - 原型作用域

作用域

- 原型作用域：只适用于函数原型中声明的参数名。
- 不必与函数定义中参数名一致，也不必与函数实际调用时实参匹配。

```
1
2 extern int a(int b, char b);
3 int main()
4 {
5 }
```

```
duchuan@testbed01:~$ gcc -o test test.c
test.c:2: error: conflicting types for ‘b’
test.c:2: note: previous definition of ‘b’ was here
```

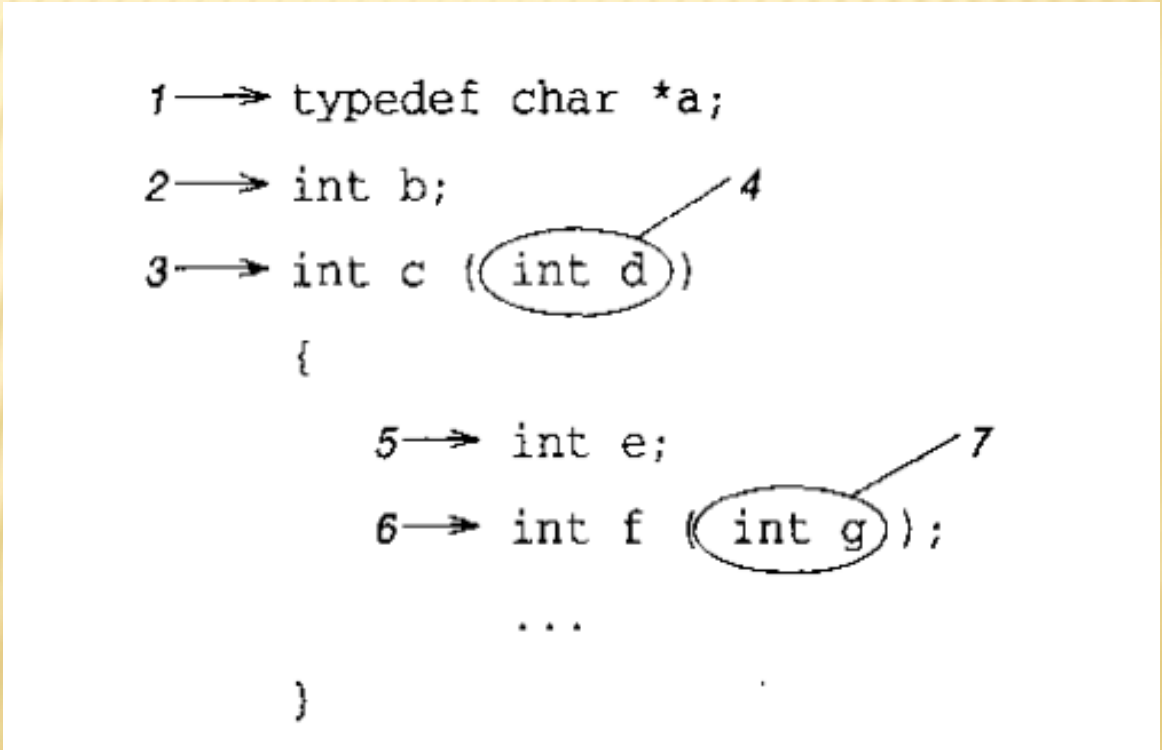
链接属性

- 标识符的连接属性决定如何处理不同文件中出现的标识。

- external, internal, none。
  - none: 标示符的多个声明被当做是多个独立不同的个体。
  - Internal: 在同一个源文件的所有声明当做是同一个实体，但是不同原文件的多个生命则属于不同的实体。
  - External: 无论声明多少次，都属于同一个实体。

链接属性

- 默认全局变量和函数的链接属性是external, 局部变量的链接属性是none。
- 没有标识符的默认链接属性是internal。



改变链接属性

- Static关键字把标识符的链接属性改为internal。
- Extern关键字把标识符的链接属性改为external。

存储类型

- 变量的缺省存储类型取决于声明位置。
  - 静态变量:　static
  - 自动变量:　auto
  - 寄存器变量:　register
- Static可以改变一个变量的存储类型为静态变量。

关键字—— REGISTER

- 请求编译器尽可能将变量存在CPU内部的寄存器中。
- 是请求，而不是绝对的定义（有可能失败）。
- Register类型的变量类型受到CPU寄存器类型的限制。
- Register类型的变量可能并不存在于内存中，所以不能用“&”取其地址。

关键字—— REGISTER

```c
#include<stdio.h>

typedef struct
{
        int a[10];
        double b;
}xxx;

int main()
{
        register double a;
        register xxx ff;
        xxx *q = &ff;
        double *t = &a;
        printf("%d\n",sizeof(double));
}
```

```
duchuan@testbed01:~$ gcc -o test a.c
a.c: In function ‘main’:
a.c:13: error: address of register variable ‘ff’ requested
a.c:14: error: address of register variable ‘a’ requested
```

关键字——STATIC

- Static关键字的用处比较复杂，也容易混淆。
- 涉及一个标识符的链接属性和存储属性。

- 对于一个链接属性为external的标识符，static关键字将其链接属性变为internal。
- 对于一个链接属性为none的标识符，static关键字将其存储属性变为静态变量。

关键字—— STATIC

- 对于一个链接属性为external的标识符，static关键字将其链接属性变为internal。此时不改变标识符的存储属性。

```
1
2 static int a = 1;
3 static int func()
4 {
5         printf("hello,world\n");
6 }
```

```
1
2 extern int func();
3 extern int a;
4
5 int main()
6 {
7         func();
8         printf("%d\n",a);
9 }
```

```
duchuan@testbed01:~$ gcc -c main.c
main.c: In function ‘main’:
main.c:8: warning: incompatible implicit declaration of built-in function ‘printf’
duchuan@testbed01:~$ gcc -c test.c
test.c: In function ‘func’:
test.c:5: warning: incompatible implicit declaration of built-in function ‘printf’
duchuan@testbed01:~$ ls
a.c  main.c  main.c~  main.o  sandbox  test.c  test.c~  test.o  vimrc  vimrc_tmp
duchuan@testbed01:~$ gcc -o main main.o test.o
main.o: In function `main':
main.c:(.text+0xa): undefined reference to `func'
main.c:(.text+0x10): undefined reference to `a'
collect2: ld returned 1 exit status
duchuan@testbed01:~$
```

关键字——STATIC

- 对于一个链接属性为none的标识符，static关键字将其存储属性变为静态变量。此时不改变标识符的链接属性。

```
1
2 int main(int argc, char* argv[])
3 {
4         int a = 1;
5 }
```

```
duchuan@testbed01:~/tmp$ size test.o
   text    data     bss     dec     hex filename
     76       0       0      76      4c test.o
```

```
1
2 int main(int argc, char* argv[])
3 {
4         static int a = 1;
5 }
```

```
duchuan@testbed01:~/tmp$ size test.o
   text    data     bss     dec     hex filename
     69       4       0      73      49 test.o
```

关键字——SIZEOF

- Sizeof 是关键字，不是函数。
  - int i;
  - sizeof(int) = ?
  - sizeof(i) = ?
  - sizeof int = ?
  - sizeof i = ?

关键字——SIZEOF

- int a[100];
  - sizeof(a) = ?
  - sizeof(a[1]) = ?
  - sizeof(&a) = ?
  - sizeof(&a[0]) = ?
  - void fun(int b[100])

```
{
        sizeof(b)   //   sizeof(b) = ?
}
```

```
duchuan@testbed01:~/tmp$ ./a.out
sizeof(a) = 400
sizeof(a[1]) = 4
sizeof(&a) = 8
sizeof(&a[0]) = 8
sizeof(b) = 8
duchuan@testbed01:~/tmp$ cat /proc/sys/kernel/osrelease
2.6.32-5-amd64
duchuan@testbed01:~/tmp$
```

关键字——SIGNED, UNSIGNED

- Unsigned/singed关键字决定了计算机如何看待符号位。
- 整数的补码表示：
  - Unsigned:

$$B2U_w(\vec{x}) \doteq \sum_{i=0}^{w-1} x_i 2^i$$

  - Signed:

$$B2T_w(\vec{x}) \doteq -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

关键字——SIGNED, UNSIGNED

- Signed 和 unsigned 对符号位不同的处理导致同一数据用signed 和 unsigned 类型处理会出现问题。

关键字——SIGNED, UNSIGNED

```
 1 #include<stdio.h>
 2 #include<string.h>
 3
 4 int main( int argc, char *argv[])
 5 {
 6     char a[1000];
 7     int i;
 8     for(i = 0; i < 1000; i++)
 9     {
10             a[i] = -1 - i;
11     }
12     printf("%d\n",strlen(a));
13     return 0;
14 }
```

关键字——SIGNED, UNSIGNED

```
1
2 unsigned int i;
3 for(i = 0; i>=0; i==)
4 {
5         printf("i = %d\n",i);
6 }
~
```

关键字—CONST

- Const 关键字修饰指针的时候容易产生迷惑。
  - Const int *p;
  - Int const *p;
  - Int *const p;
  - Const int* p;

关键词——CONST

- Const 修饰typedef定义得类型，更容易产生迷惑。

- Typdefe int* int_ptr;
  - Int_ptr const a;
  - const int_ptr a;

关键字——STRUCT与系统对齐。

```
 1 #include<stdio.h>
 2
 3 typedef struct
 4 {
 5         int a;
 6         char c;
 7         double d;
 8 }test;
 9
10 int main(int argc, char* argv[])
11 {
12         printf("sizeof test is %d\n",sizeof(test));
13 }
```

关键字—UNION与系统字节序

```
 1 #include<stdio.h>
 2
 3 typedef union
 4 {
 5         char a[4];
 6         unsigned int b;
 7 }xx;
 8
 9 int main( int argc, char *argv[])
10 {
11     xx f ;
12     f.b = 0x12345678;
13     printf("a[0]%x\n",f.a[0]);
14     printf("a[1]%x\n",f.a[1]);
15     printf("a[2]%x\n",f.a[2]);
16     printf("a[3]%x\n",f.a[3]);
17 }
18
~
~
```

关键字—UNION与系统字节序

```
[2]+  Stopped                 vi test.c
duchuan@testbed01:~/tmp$ gcc test.c
duchuan@testbed01:~/tmp$ ./a.out
a[0]78
a[1]56
a[2]34
a[3]12
```

符号

- 关于注释
- 符号顺序

关于注释

- 注释可以出现在C语言代码的任何地方。
- 编译器在编译代码的时候会剔除掉注释，用空格代替。
  - int/*comment*/i;
  - Char *s = “adbc //fadfasdf”;
  - // is this a valid \
    Comment?
  - In/*dfasdf*/t i;

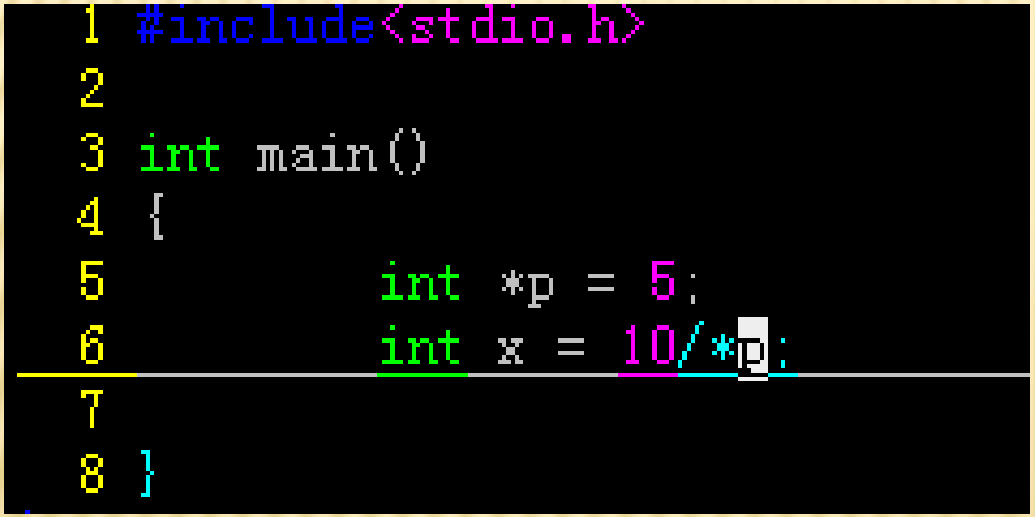
关于注释

```
 1 #include<stdio.h>
 2
 3 int main()
 4 {
 5         int *p = 5;
 6         int x = 10;
 7         int/*hello*/world;
 8         char* s= "asdfadsf //hfasdfasd";
 9         //is this \
10             a valid comment?
11         in/*asdfasdf*/t i;
12
13 }
```

关于注释

int *p = 5;

int y = 10/*p?



符号顺序

- 注意运算符的操作顺序。一个表达式看起来的意思也许和实际上相差很远。

- 基本顺序： 组合运算符>单目运算符>双目运算符>三目运算符>赋值运算符>逗号运算符

符号顺序引发的血案

- int i = 3;　　(++i)+(++i)+(++i);
- int i = 3;　　(i++)+(i++)+(i++);
- *p.f;
- Int *ap[];
- (val & mask != 0)
- Msb << 4 + lsb
- i = 1,2

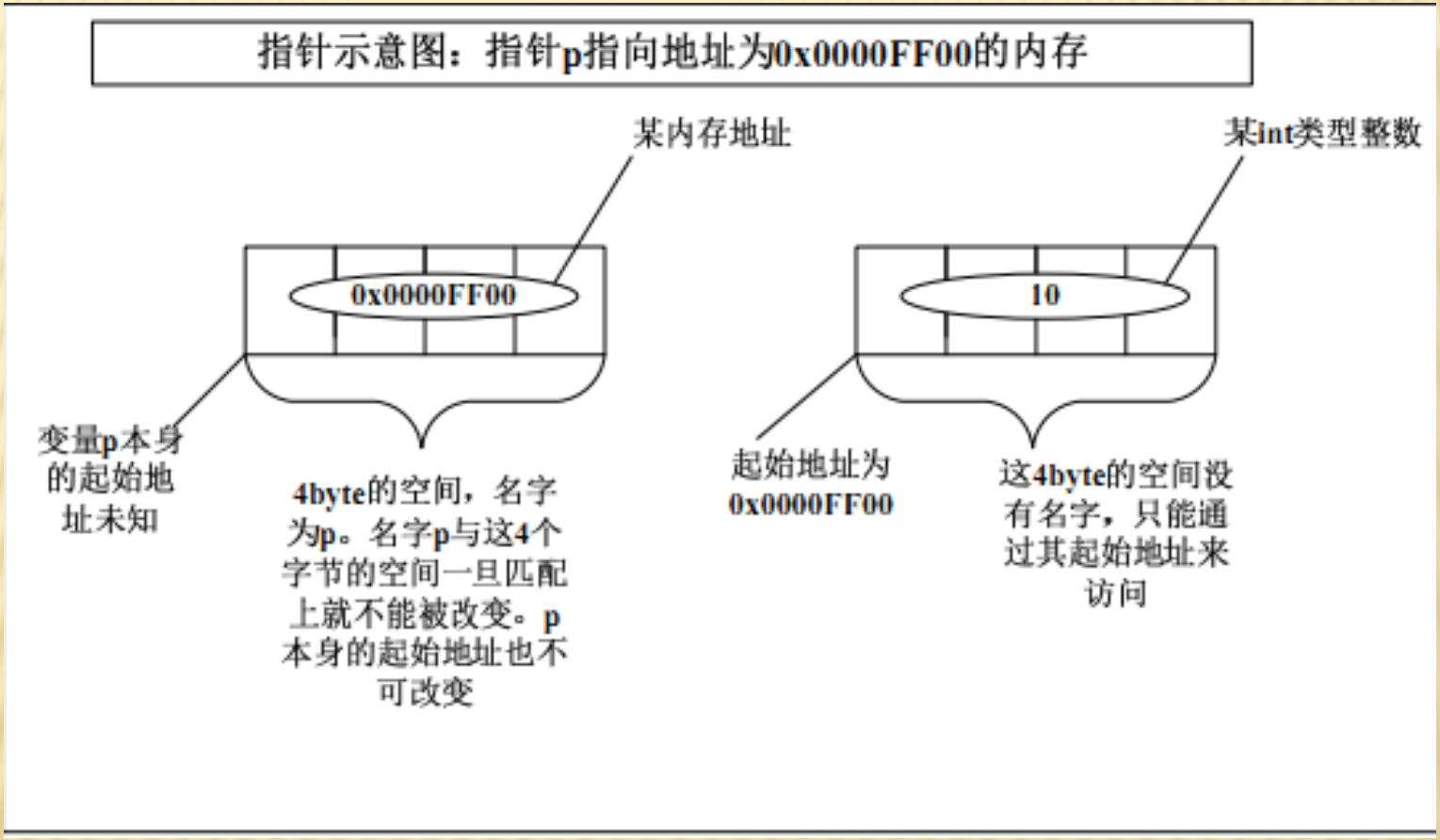
指针与数组

- 什么是指针?
- 什么是数组?
- 数组与指针之间区别与联系?

指针

- int *p 的问题。
  - 是一个定义。
  - 定义了一个名字(p)；分配了一块内存x；并且把名字和内存相绑定。(p ←→内存x)。
  - 这块内存中存储的是一个内存地址y。
  - 内存地址y存储的是一个int 类型的变量。

指针示意图


指针示意图：指针p指向地址为0x0000FF00的内存
某内存地址
某int类型整数
0x0000FF00
10
变量p本身的起始地址未知
4byte的空间，名字为p。名字p与这4个字节的空间一旦匹配上就不能被改变。p本身的起始地址也不可改变
起始地址为0x0000FF00
这4byte的空间没有名字，只能通过其起始地址来访问

指针示意图

```c
#include<stdio.h>

int main( int argc, char *argv[])
{
        int a = 2;
        int *p = &a ;
        printf("address of p is %x\n",&p);
        printf("content of p is %x\n",p);
        printf("content of address p point to is %d\n",*p);
}
```

```
duchuan@testbed01:~$ gcc test.c
duchuan@testbed01:~$ ./a.out
address of p is 7bb82640
content of p is 7bb8264c
content of address p point to is 2
duchuan@testbed01:~$
```

一个问题

```c
#include<stdio.h>

int main()
{
        char *p = 5;

        printf("%d\n",*p);
}
```

```c
#include<stdio.h>

int main()
{
        char *s = "hello, world";
        printf("%s\n",s);
}
```

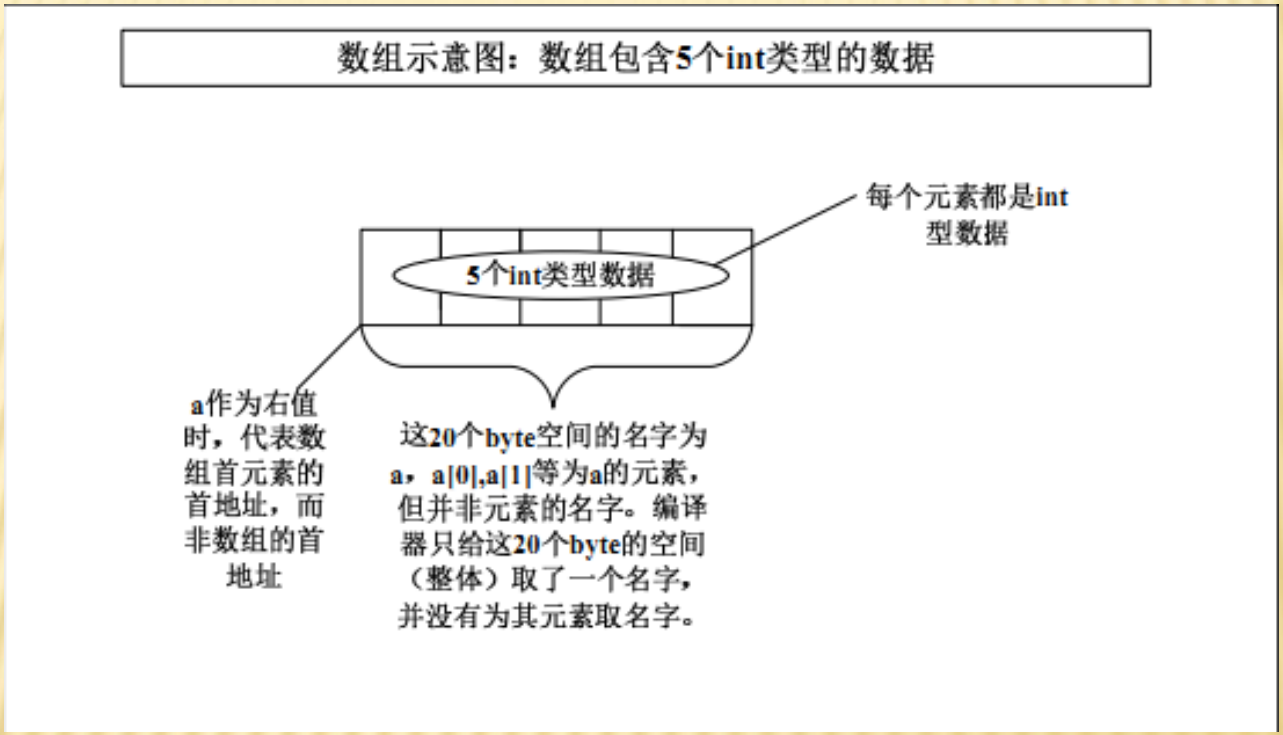
结果

```
duchuan@testbed01:~/tmp$ gcc -o test test.c
duchuan@testbed01:~/tmp$ gcc -o xx xx.c
xx.c: In function ‘main’:
xx.c:5: warning: initialization makes pointer from integer without a cast
duchuan@testbed01:~/tmp$ ./test
hello, world
duchuan@testbed01:~/tmp$ ./xx
Segmentation fault
```

数组

- int a[5];
  - 是一个定义。
  - 定义了一个名字(a)；分配了一块内存x；并且把名字和内存相绑定。(a ←→内存x)。
  - 这块内存中存储的是五个int 类型的变量。

数组示意图


数组示意图：数组包含5个int类型的数据
每个元素都是int型数据
5个int类型数据
a作为右值时，代表数组首元素的首地址，而非数组的首地址
这20个byte空间的名字为a，a[0],a[1]等为a的元素，但并非元素的名字。编译器只给这20个byte的空间（整体）取了一个名字，并没有为其元素取名字。

一个问题

- int a[5];
  - a ?
  - &a ?
  - &a[0] ?
  - a + 1 ?
  - &a + 1?
  - &a[0] + 1?

结果

```
 1 #include<stdio.h>
 2
 3 int main()
 4 {
 5         int a[5];
 6         printf("a  = %x\n",a );
 7         printf("&a + 1 = %x\n",&a + 1);
 8         printf("a + 1 = %x\n",a + 1);
 9         printf("&a[0] + 1 = %x\n",&a[0] + 1);
10
11 }
```

```
duchuan@testbed01:~/tmp$ ./a.out
a  = 81d10bd0
&a + 1 = 81d10be4
a + 1 = 81d10bd4
&a[0] + 1 = 81d10bd4
duchuan@testbed01:~/tmp$
```

int *p 与 int p[10] 的区别?

指针与数组混用的结果

- 定义为数组， 声明为指针。

```
1 #include<stdio.h>
2
3 extern char *a;
4
5 int main()
6 {
7     printf("a is %x\n",(int) a);
8     //printf("*a is %d\n",  *a);
9 }
10
```

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

```
duchuan@testbed01:~/tmp$ ./x
a is 4030201
duchuan@testbed01:~/tmp$
```

指针与数组混用的结果

- 定义为数组， 声明为指针。

```
 1 #include<stdio.h>
 2
 3 extern char *a;
 4
 5 int main()
 6 {
 7     //printf("a is %x\n",(int) a);
 8     printf("*a is %d\n",  *a);
 9 }
10
~
```

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

```
duchuan@testbed01:~/tmp$ gcc -c b.c
duchuan@testbed01:~/tmp$ gcc -o x a.o b.o
duchuan@testbed01:~/tmp$ ./x
Segmentation fault
```

指针与数组混用的结果

- 定义为指针， 声明为数组。

```
1 #include<stdio.h>
2
3 extern char a[20];
4
5 int main()
6 {
7     printf("a[5] is %d\n",  a[5]);
8 }
9
```

```
1
2 char *a = "hello,world";
```

```
duchuan@testbed01:~/tmp$ ls
a.c  a.c~  a.o  b.c  b.c~  b.o  x
duchuan@testbed01:~/tmp$ ./x
a[5] is 0
```

指针数组与数组指针

- Int *p[5] 与 Int (*p)[5];

指针的运算与强制类型转换

- 根据指针类型的不同，指针的运算结果会不一样。

指针的运算与强制类型转换

```
 1 #include<stdio.h>
 2
 3 typedef struct
 4 {
 5         char b;
 6         int a;
 7 }x;
 8 int main()
 9 {
10         x *a;
11         printf("a  = %x\n", a );
12         printf("a + 0x1 = %x\n", a + 0x1);
13         printf("(unsignd long)a + 0x1 = %x\n", (unsigned long)a + 0x1);
14         printf("(unsignd int*)a + 0x1 = %x\n", (unsigned int *)a + 0x1);
15 }
```

结果

```
duchuan@testbed01:~/tmp$ ./a.out
a  = 0
a + 0x1 = 8
(unsignd long)a + 0x1 = 1
(unsignd int*)a + 0x1 = 4
duchuan@testbed01:~/tmp$
```

另一个问题

```
 1 #include<stdio.h>
 2
 3 int main()
 4 {
 5         long a[4] = {1,2,3,4};
 6         long *ptr1 = (long*) (&a + 1);
 7         long *ptr2 = (long*) ((long)a + 1);
 8
 9         printf("%x, %x\n", ptr1[-1], *ptr2);
10 }
~
```

结果

```
duchuan@testbed01:~/tmp$ ./a.out
4, 0
```

```c
#include<stdio.h>

int main()
{
        int a[5][5];
        int (*p)[4];
        p = a;

        printf("&a[4][2] = %x, &p[4][2] = %x\n", &a[4][2], &p[4][2]);
        printf("&a[4][2] -  &p[4][2] = %x\n", &a[4][2] -  &p[4][2]);
}
```

结果

```
duchuan@testbed01:~/tmp$ gcc test.c
test.c: In function ‘main’:
test.c:7: warning: assignment from incompatible pointer type
duchuan@testbed01:~/tmp$ ./a.out
&a[4][2] = 13b2cfe8, &p[4][2] = 13b2cfd8
&a[4][2] -  &p[4][2] = 4
duchuan@testbed01:~/tmp$
```