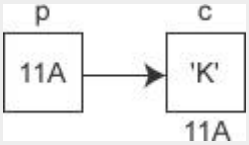
大话 C 语言指针变量

数据在内存中的地址也称为指针，如果一个变量存储了一份数据的指针，我们就称它为指针变量。

在 C 语言中，允许用一个变量来存放指针，这种变量称为指针变量。指针变量的值就是某份数据的地址，这样的一份数据可以是数组、字符串、函数，也可以是另外的一个普通变量或指针变量。

现在假设有一个 char 类型的变量 c，它存储了字符 'K'（ASCII 码为十进制数 75），并占用了地址为 0X11A 的内存（地址通常用十六进制表示）。另外有一个指针变量 p，它的值为 0X11A，正好等于变量 c 的地址，这种情况我们就称 p 指向了 c，或者说 p 是指向变量 c 的指针。



定义指针变量

定义指针变量与定义普通变量非常类似，不过要在变量名前面加星号`*`，格式为：

```
datatype *name;
```

或者

```
datatype *name = value;
```

`*`表示这是一个指针变量，`datatype`表示该指针变量所指向的数据的类型 。例如：

```
1.  int *p1;
```

p1 是一个指向 int 类型数据的指针变量，至于 p1 究竟指向哪一份数据，应该由赋予它的值决定。再如：

```
int a = 100;
int *p_a = &a;
```

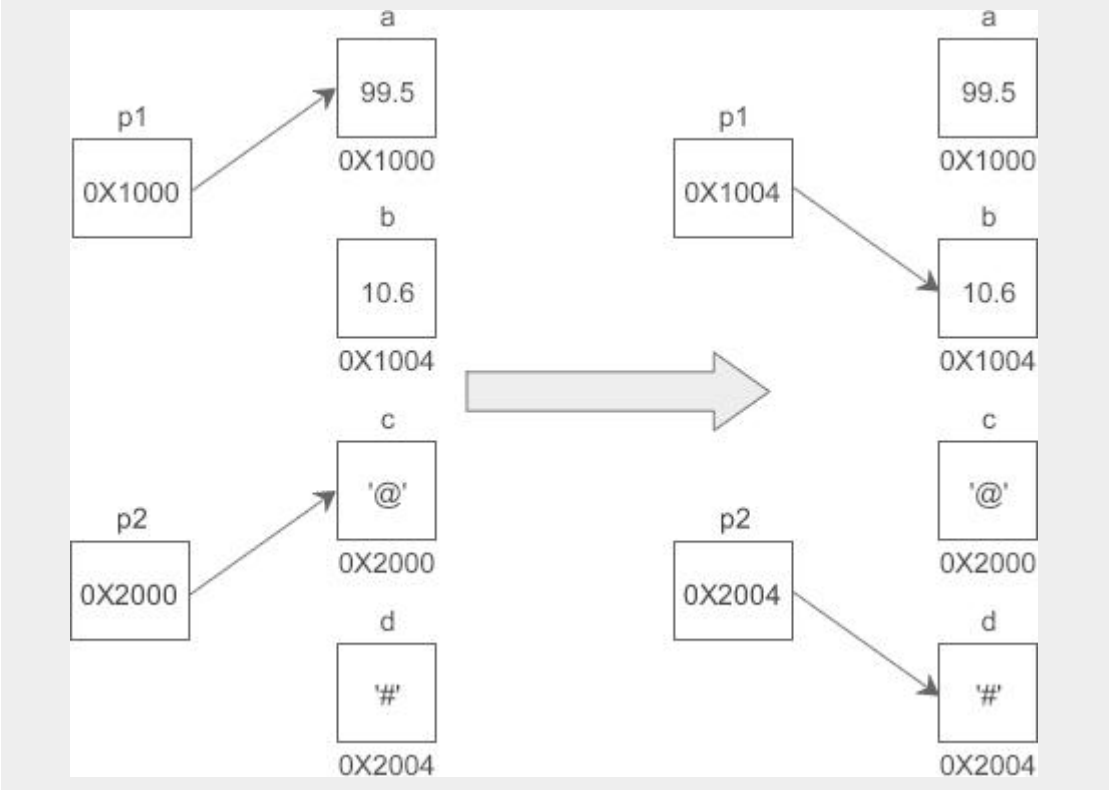
在定义指针变量 p_a 的同时对它进行初始化，并将变量 a 的地址赋予它，此时 p_a 就指向了 a。值得注意的是，p_a 需要的一个地址，a 前面必须要加取地址符&，否则是不对的。

和普通变量一样，指针变量也可以被多次写入，只要你想，随时都能够改变指针变量的值，请看下面的代码：

```
//定义普通变量
float a = 99.5, b = 10.6;
char c = '@', d = '#';
//定义指针变量
float *p1 = &a;
char *p2 = &c;
//修改指针变量的值
p1 = &b;
p2 = &d;
```

是一个特殊符号，表明一个变量是指针变量，定义 p1、p2 时必须带。而给 p1、p2 赋值时，因为已经知道了它是一个指针变量，就没必要多此一举再带上*，后边可以像使用普通变量一样来使用指针变量。也就是说，定义指针变量时必须带*，给指针变量赋值时不能带*。

假设变量 a、b、c、d 的地址分别为 0X1000、0X1004、0X2000、0X2004，下面的示意图很好地反映了 p1、p2 指向的变化：



需要强调的是，p1、p2 的类型分别是 float*和 char*，而不是 float 和 char，它们是完全不同的数据类型，读者要引起注意。

指针变量也可以连续定义，例如：

```
int *a, *b, *c;   //a、b、c 的类型都是 int*
```

注意每个变量前面都要带*。如果写成下面的形式，那么只有 a 是指针变量，b、c 都是类型为 int 的普通变量：

```
int *a, b, c;
```

通过指针变量取得数据

指针变量存储了数据的地址，通过指针变量能够获得该地址上的数据，格式为：

```
*pointer;
```

这里的*称为指针运算符，用来取得某个地址上的数据，请看下面的例子：

```
#include <stdio.h>
```

```
2.
3.          int main(){
4.              int a = 15;
5.              int *p = &a;
6.              printf("%d, %d\n", a, *p);    //两种方式都可以输出 a 的值
7.              return 0;
8.          }
```

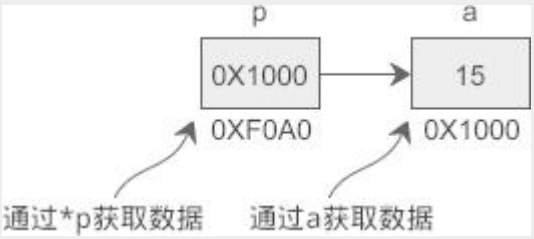
运行结果：

15, 15

假设 a 的地址是 0X1000，p 指向 a 后，p 本身的值也会变为 0X1000，*p 表示获取地址 0X1000 上的数据，也即变量 a 的值。从运行结果看，*p 和 a 是等价的。

上节我们说过，CPU 读写数据必须要知道数据在内存中的地址，普通变量和指针变量都是地址的助记符，虽然通过 *p 和 a 获取到的数据一样，但它们的运行过程稍有不同：a 只需要一次运算就能够取得数据，而 *p 要经过两次运算，多了一层“间接”。

假设变量 a、p 的地址分别为 0X1000、0XF0A0，它们的指向关系如下图所示：



程序被编译和链接后，a、p 被替换成相应的地址。使用 *p 的话，要先通过地址 0XF0A0 取得变量 p 本身的值，这个值是变量 a 的地址，然后再通过这个值取得变量 a 的数据，前后共有两次运算；而使用 a 的话，可以通过地址 0X1000 直接取得它的数据，只需要一步运算。

也就是说，使用指针是间接获取数据，使用变量名是直接获取数据，前者比后者的代价

要高。

指针除了可以获取内存上的数据，也可以修改内存上的数据，例如：

```
#include <stdio.h>

int main(){
    int a = 15, b = 99, c = 222;
    int *p = &a;   //定义指针变量
    *p = b;   //通过指针变量修改内存上的数据
    c = *p;   //通过指针变量获取内存上的数据
    printf("%d, %d, %d, %d\n", a, b, c, *p);
    return 0;
}
```

运行结果：

99, 99, 99, 99

*p 代表的是 a 中的数据，它等价于 a，可以将另外的一份数据赋值给它，也可以将它赋值给另外的一个变量。

*在不同的场景下有不同的作用：*可以用在指针变量的定义中，表明这是一个指针变量，以和普通变量区分开；使用指针变量时在前面加*表示获取指针指向的数据，或者说表示的是指针指向的数据本身。

也就是说，定义指针变量时的*和使用指针变量时的*意义完全不同。以下面的语句为例：

```
int *p = &a;
*p = 100;
```

第 1 行代码中*用来指明 p 是一个指针变量，第 2 行代码中*用来获取指针指向的数据。

需要注意的是，给指针变量本身赋值时不能加*。修改上面的语句：

```
int *p;
p = &a;
*p = 100;
```

第 2 行代码中的 p 前面就不能加*。

指针变量也可以出现在普通变量能出现的任何表达式中，例如：

```
int x, y, *px = &x, *py = &y;
y = *px + 5;   //表示把 x 的内容加 5 并赋给 y，*px+5 相当于(*px)+5
y = ++*px;   //px 的内容加上 1 之后赋给 y，++*px 相当于++(*px)
y = *px++;   //相当于 y=(*px)++
py = px;   //把一个指针的值赋给另一个指针
```

【示例】通过指针交换两个变量的值。

```
#include <stdio.h>

int main(){
    int a = 100, b = 999, temp;
    int *pa = &a, *pb = &b;
    printf("a=%d, b=%d\n", a, b);
    /*****开始交换*****/
    temp = *pa;   //将 a 的值先保存起来
    *pa = *pb;   //将 b 的值交给 a
    *pb = temp;   //再将保存起来的 a 的值交给 b
    /*****结束交换*****/
    printf("a=%d, b=%d\n", a, b);
    return 0;
}
```

运行结果：

a=100, b=999

a=999, b=100

从运行结果可以看出，a、b 的值已经发生了交换。需要注意的是临时变量 temp，它的作用特别重要，因为执行*pa = *pb;语句后 a 的值会被 b 的值覆盖，如果不先将 a 的值保存起来以后就找不到了。

关于 * 和 & 的谜题

假设有一个 int 类型的变量 a，pa 是指向它的指针，那么*&a 和&*pa 分别是什么意思呢？

&a 可以理解为(&a)，&a 表示取变量 a 的地址（等价于 pa），*(&a)表示取这个地址上的数据（等价于 *pa），绕来绕去，又回到了原点，*&a 仍然等价于 a。

&*pa 可以理解为&(*pa)，*pa 表示取得 pa 指向的数据（等价于 a），&(*pa)表示数据的地址（等价于 &a），所以&*pa 等价于 pa。

对星号*的总结

在我们目前所学到的语法中，星号*主要有三种用途：

- 表示乘法，例如 int a = 3, b = 5, c; c = a * b;，这是最容易理解的。
- 表示定义一个指针变量，以和普通变量区分开，例如 int a = 100; int *p = &a;。
- 表示获取指针指向的数据，是一种间接操作，例如 int a, b, *p = &a; *p = 100; b = *p;。