千锋嵌入式学院C语言培训

指针1

源自清华 值得信赖

什么是地址？

- 在程序执行过程中，所有的数据对象都存储在计算机内存储器里。任何一个数据对象在它被执行的那段期间内都有一个确定的存储位置，占据着确定数目的存储单元。且每个存储单元有一个唯一的编号，即地址。
- 地址在计算机内部是用二进制编码表示的，可作为程序中能够被处理的数据。

指针的概念

- 指针是一种特殊的变量，本质是变量，但存放的是某个变量的地址。指针变量同样具有变量名、变量数据类型和变量值。
- 注意内存单元的地址和内存单元的内容是两个不同的概念。

指针概念和作用

指针变量的定义

- 指针变量定义的一般形式:

  类型　*标识符;

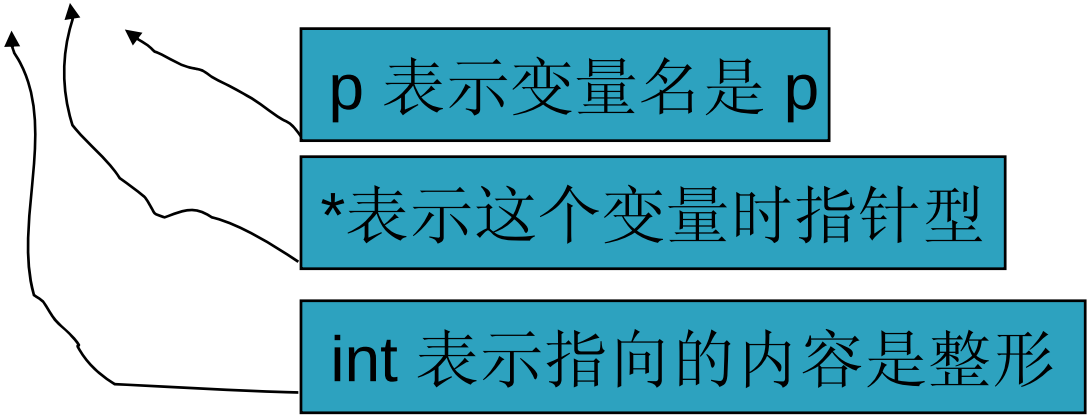
- 说明:

  “标识符”是指针变量的名字;“类型”表明该指针变量所指向的变量类型。

指针基本用法

- 指针声明和赋值操作

```
int *p;          // 声明指向int型的指针变量p
```



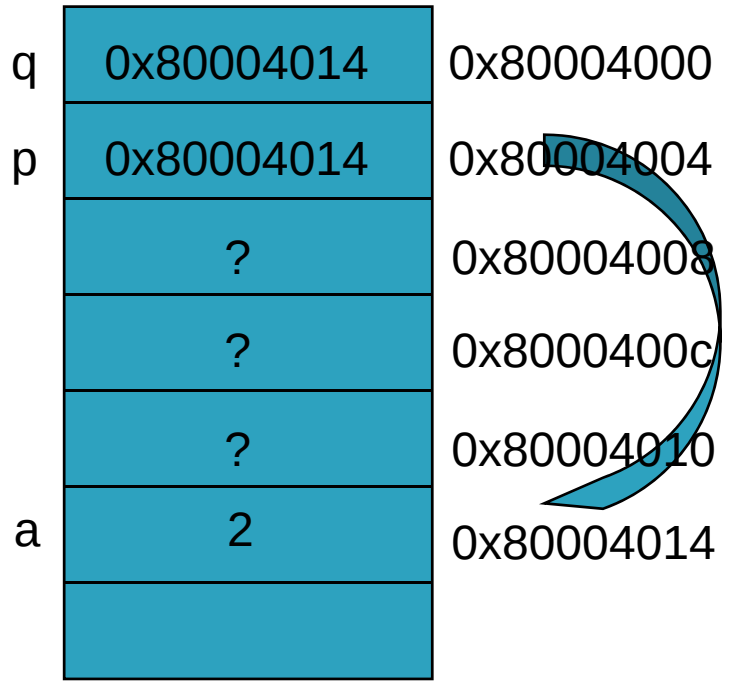
指针变量的引用

- 指针变量中只能存放地址，因此不能对一个指针变量赋值。
- 例如:　　int a,*p1,*p2;
  p1=&a;(正确写法)
  p2=3;(错误写法！)
- 指针变量的相关运算符:
  &(取地址运算符)和*(指针运算符)

名称	取地址运算符&	指针运算符*
功能	取地址	指针运算(间接访问)
操作对象	变量(除寄存器变量)或数组元素	指针或指针表达式
操作数	单目	
优先级	第二优先级，仅次于括号	
结合方向	从右到左	

表 10.1 比较&与*

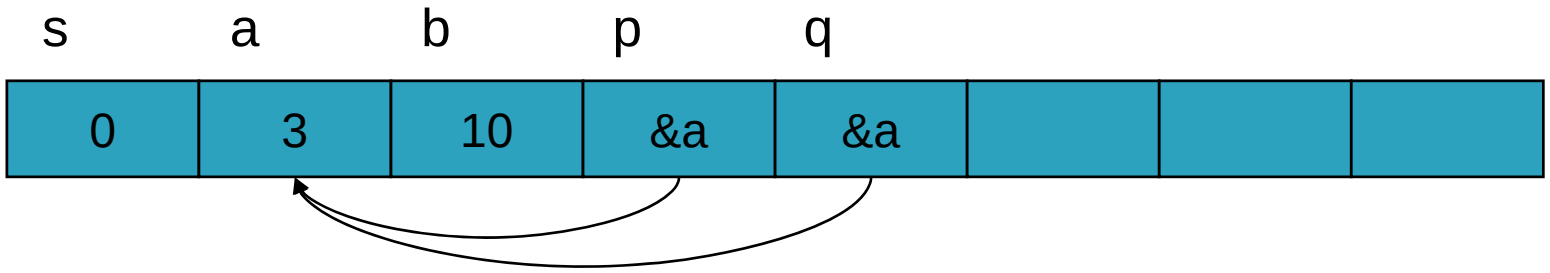
```
int a = 2;
int *p = &a;   //p=0x80004000
int *q = p;    //q=0x80004000
```



指针基本用法

- 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
```



指针概念和定义

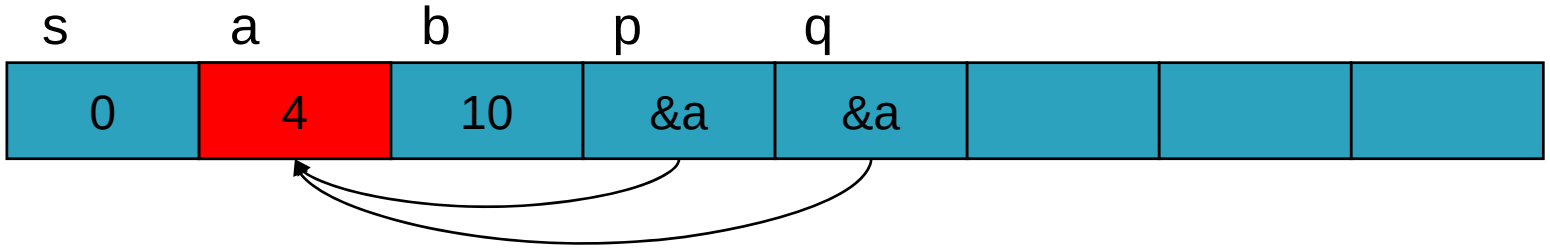
```c
#include <stdio.h>

int main()
{
        int a = 2;
        int *p = &a;
        printf("p = %x\n", p);
        printf("*p = %x\n", *p);
        return 0;
}
```

```
[root@localhost tmp]# gcc -o pointer pointer.c
[root@localhost tmp]# ./pointer
p = bfd7cf7c
*p = 2
[root@localhost tmp]#
```

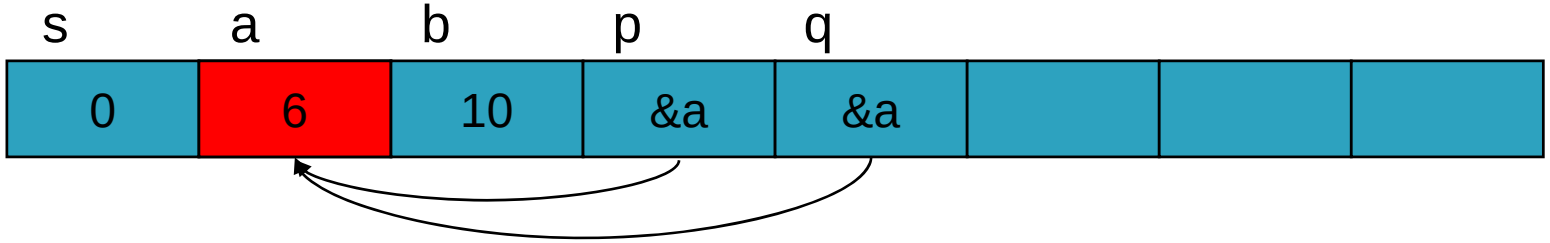
指针基本用法

- 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



指针基本用法

- 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？
```

常见的指针定义错误

- int a, b;
- int *p1, p2;
- p1=&a;
- p2=&b;

- 深入理解指针运算 p++
- 地址 += sizeof(指向的变量类型)
- int *p;
  char *p;
  double *p;

指针基本用法

比较下面两段代码意义和结果。

- int a=2, b=3, c=0;
  int *p = &a, *q = &b, *t = &c;
  *t = *p;
  *p = *q;
  *q = *t;
- // a = 3, b=2, c=2
  // *p=3, *q=2, *t =2

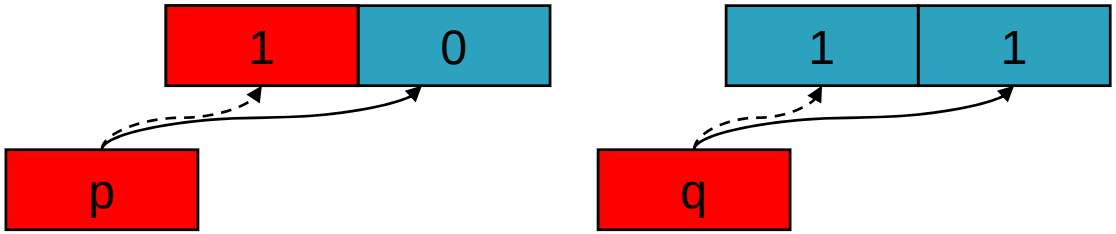
- int a=2, b=3, c=0;
  int *p = &a, *q = &b, *t = &c;
  t = p;
  p = q;
  q = t;
- // a=2,b=3,c=0
  // *p=3, *q=2,*t=2

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

运算符*和&的优先级

- 思考：*p ++ = *q ++ 怎么运算呢？



- *p = *q;
- p ++;
- q ++;

基本概念和定义

- 通配指针(void *)
  - void *p = &a;
  - 表示指针指向的内容没有类型限制
  - 可以与任何类型的指针互相赋值
  - 根据需要转成相关数据类型的指针
- 空指针(NULL)
  - 一个预先定义好的(void *)类型指针常量
  - 表示一个指针没有指向任何东西
  - 可以与任何类型指针进行比较操作
  - 可以与0进行比较操作
  - 不能进行 * 操作

基本概念和定义

- 通配指针(void *)
  - int *p, char *q, void *str;
  - p = str = q;
- 空指针(NULL)
  - 一个预先定义好的(void *)类型指针常量
  - 表示一个指针没有指向任何东西
  - 可以与任何类型指针进行比较操作
  - 可以与0进行比较操作
  - 不能进行 * 操作

指针常见错误

- 错误：访问空指针 int *p = NULL;

```
(*p) ++;    // 或是p++, 运行时出段错误
```

- 良好编码习惯，检查指针是否为空

```
if( p )        //等效于p!=NULL
    (*p)++;
```

- 对于指针为空的情况，进行异常处理

```
if( !p )        //等效于NULL==p或p==NULL
    printf("null pointer.")      //指针为空
```

- 良好编码习惯：比较 NULL==p 与 p==NULL
  - 防止手误写成 NULL=p;

指针常见错误

- 错误：访问或释放野指针，结果不可预测
  - 当一个指针指向的内容非法时，这个指针就是野指针
  - 定义一个指针没有赋初值勤，这时候指针就是野指针
  - 当一个指针指向的内容被销毁后，这个指针就成为野指针
- 良好编码习惯：定义时初始化，无值赋空值，不留野指针。

```
void function{
        int *p = NULL;
        … …
```

- 良好编码习惯，释放完内存后，指针置空
  - 销毁一个指针指向的内容后，把这个指针置空(NULL)
  - free(p); p = NULL;

指针类型的参数和返回值

```
#include <stdio.h>

void inc( int ai)
{
        printf("ai = %d\n", ai);
        ai ++;
        printf("ai = %d\n", ai);
}

int main()
{
        int a = 0;
        printf("a = %d\n", a);
        inc(a);
        printf("a = %d\n", a);
        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

指针类型的参数和返回值

```c
#include <stdio.h>

void inc( int ai)
{
        printf("ai = %d\n", ai);
        ai ++;
        printf("ai = %d\n", ai);
}

int main()
{
        int a = 0;
        printf("a = %d\n", a);
        inc(a);
        printf("a = %d\n", a);
        return 0;
}
```

```c
#include <stdio.h>

void inc(int *ai)
{
        printf("ai = %d\n", *ai);
        (*ai) ++;
        printf("ai = %d\n", *ai);
}

int main()
{
        int a = 0;
        printf("a = %d\n", a);
        inc(&a);
        printf("a = %d\n", a);
        return 0;
}
```

指针类型的参数和返回值

- 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。

```
#include <stdio.h>
int *swap(int *px, int *py)
{
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
  return px;
}
int main(void)
{
  int i = 10, j = 20;
  int *p = swap(&i, &j);
  printf("now i=%d j=%d *p=%d\n", i, j, *p);
  return 0;
}
```

指针类型的参数和返回值

- 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。

指针与const限定符

- const int *a; // a可改写，a指向的内容不可改写
  - 与 int const *a; 相同
  - a++; //正确
  - (*a) ++; //错误
- int * const a =# //a不可改写，a指向的内容可改写
  - a ++; //错误
  - (*a) ++; 正确
- const int * const a; //a 和 *a都不可改写
- 区别办法: const 和 * 右结合

指针与const限定符

- const 与非 const相互转换
  - 非const转到const，意味着可写内容成为不可写内容，是安全的，可以隐式转换。
  - const到非const，意味着不可写的内容变成了可写的内容，不安全，要小心处理。程序员必须显示转换，说明很清楚这一点。
- const 的好处
  - 不需要修改的变量尽量用const, 防止内存意外被修改出错
  - 传达的意思更强，让人一看就知道这里是不可写的内容，可以放心的使用

指针与const限定符

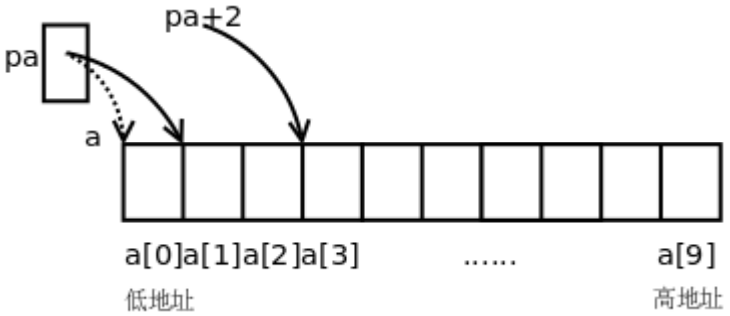
```
void add(int *a, const int *b)
{
        *a += *b;
        //*b = 3;
}

int main()
{
        int a = 0, b = 3;
        add(&a, &b);
        return 0;
}
```

指针b指向的内容不可修改，*b=3 会引起编译错误。

指针与数组

- 数组类似于 int * const a;
  - pa ++;　　//正确
  - a ++;　　//错误
  - pa = a + 3; //正确
  - (*a) = 5;　//正确
- 需要在函数里面访问数组时，请使用指针
  - int find(int * const index)
  - int find(int a[10]) //与指针等价

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



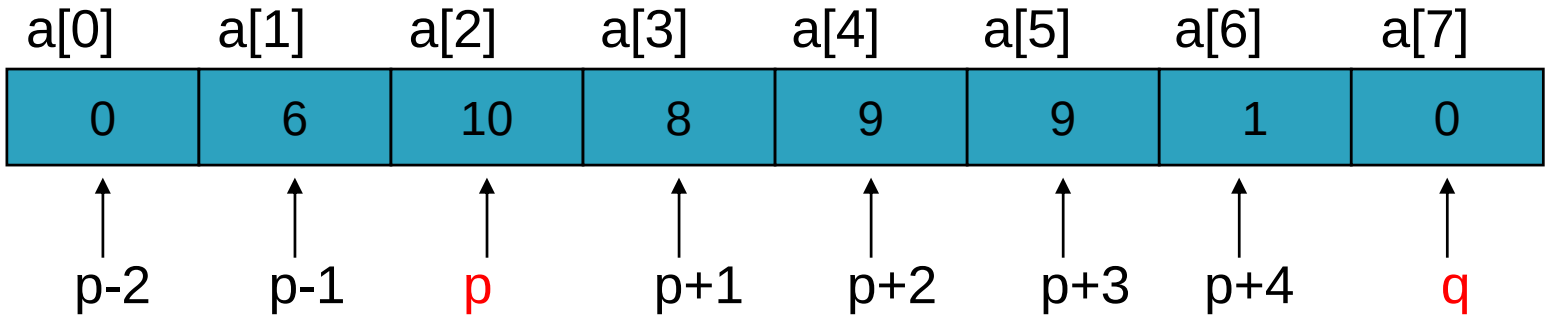
指针与数组

```
void init(int *a, int len)
{
        while(len>0){
                *a ++ = len --;
        }
}

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

```
[root@localhost dev]# gcc -g -o pfun.o pointerfunc.c
[root@localhost dev]# ./pfun.o
10 9 8 7 6 5 4 3 2 1
```

指针与数组

- 指针运算 int a[8], int *p, *q;



- p = &a[2], q = &a[7], q-p = 5
- q 与 p+5 指向相同的变量
- p-2, p-1, p …. p+4 指向连续的内存序列

练习

- 练习：编写函数 findmax, 从一个整形数组中找出来最大的一个数并返回数组下标. 函数原型如下：
  int findmax(int *array, int length)
- 练习：不要使用全局变量，编写一个对整型数组排序的函数，原型如下：
  void sort(int *array, int length)
- 练习：编写函数 replace, 在一个字符数组里面查找指定字符，并用相应的字符替代。函数原型如下：
  void replace(char *array, char old, char new, int length)
- 练习：编写函数 insert, 向一个字符数组指定位置插入一个字符，后面的字符依次向后移动。函数原型如下：
  void insert(char *array, int index, char new, int length)

其他问题？