

BIT-2-指针的进阶

本章重点

1. 字符指针
2. 数组指针
3. 指针数组
4. 数组传参和指针传参
5. 函数指针
6. 函数指针数组
7. 指向函数指针数组的指针
8. 回调函数
9. 指针和数组面试题的解析

正文开始©比特就业课

指针的主题，我们在初级阶段的《指针》章节已经接触过了，我们知道了指针的概念：

1. 指针就是个变量，用来存放地址，地址唯一标识一块内存空间。
2. 指针的大小是固定的4/8个字节（32位平台/64位平台）。
3. 指针是有类型，指针的类型决定了指针的+-整数的步长，指针解引用操作的时候的权限。
4. 指针的运算。

这个章节，我们继续探讨指针的高级主题。

1. 字符指针

在指针的类型中我们知道有一种指针类型为字符指针 `char*`；

一般使用：

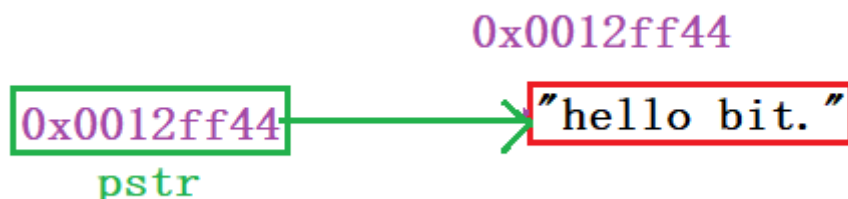
```
int main()
{
    char ch = 'w';
    char *pc = &ch;
    *pc = 'w';
    return 0;
}
```

还有一种使用方式如下：

```
int main()
{
    const char* pstr = "hello bit.");//这里是把一个字符串放到pstr指针变量里了吗？
    printf("%s\n", pstr);
    return 0;
}
```

代码 `const char* pstr = "hello bit.;"`

特别容易让同学以为是把字符串 `hello bit` 放到字符指针 `pstr` 里了，但是/本质是把字符串 `hello bit` 首字符的地址放到了 `pstr` 中。



上面代码的意思是把一个常量字符串的首字符 `h` 的地址存放到指针变量 `pstr` 中。

那就有可这样的面试题：

```
#include <stdio.h>

int main()
{
    char str1[] = "hello bit.";
    char str2[] = "hello bit.";
    const char *str3 = "hello bit.";
    const char *str4 = "hello bit.";

    if(str1 == str2)
        printf("str1 and str2 are same\n");
    else
        printf("str1 and str2 are not same\n");

    if(str3 == str4)
        printf("str3 and str4 are same\n");
    else
        printf("str3 and str4 are not same\n");

    return 0;
}
```

这里最终输出的是：

The screenshot shows a Windows command prompt window with the title bar `C:\WINDOWS\system32\cmd.exe`. The output of the program is displayed in a monospaced font: `str1 and str2 are not same`, `str3 and str4 are same`, and `请按任意键继续. . .`. A mouse cursor is visible at the bottom left of the window.

这里 `str3` 和 `str4` 指向的是一个同一个常量字符串。C/C++ 会把常量字符串存储到单独的一个内存区域，当几个指针。指向同一个字符串的时候，他们实际会指向同一块内存。但是用相同的常量字符串去初始化不同的数组的时候就会开辟出不同的内存块。所以 `str1` 和 `str2` 不同，`str3` 和 `str4` 不同。

2. 指针数组

在《指针》章节我们也学了指针数组，指针数组是一个存放指针的数组。

这里我们再复习一下，下面指针数组是什么意思？

```
int* arr1[10]; //整形指针的数组
char *arr2[4]; //一级字符指针的数组
char **arr3[5]; //二级字符指针的数组
```

3. 数组指针

3.1 数组指针的定义

数组指针是指针？还是数组？

答案是：指针。

我们已经熟悉：

整形指针： `int * pint`; 能够指向整形数据的指针。

浮点型指针： `float * pf`; 能够指向浮点型数据的指针。

那数组指针应该是：能够指向数组的指针。

下面代码哪个是数组指针？

```
int *p1[10];
int (*p2)[10];
//p1, p2分别是什么？
```

解释：

```
int (*p)[10];
//解释：p先和*结合，说明p是一个指针变量，然后指着指向的是一个大小为10个整型的数组。所以p是一个指针，指向一个数组，叫数组指针。

//这里要注意：[]的优先级要高于*号的，所以必须加上（）来保证p先和*结合。
```

3.2 &数组名VS数组名

对于下面的数组：

```
int arr[10];
```

`arr` 和 `&arr` 分别是啥？

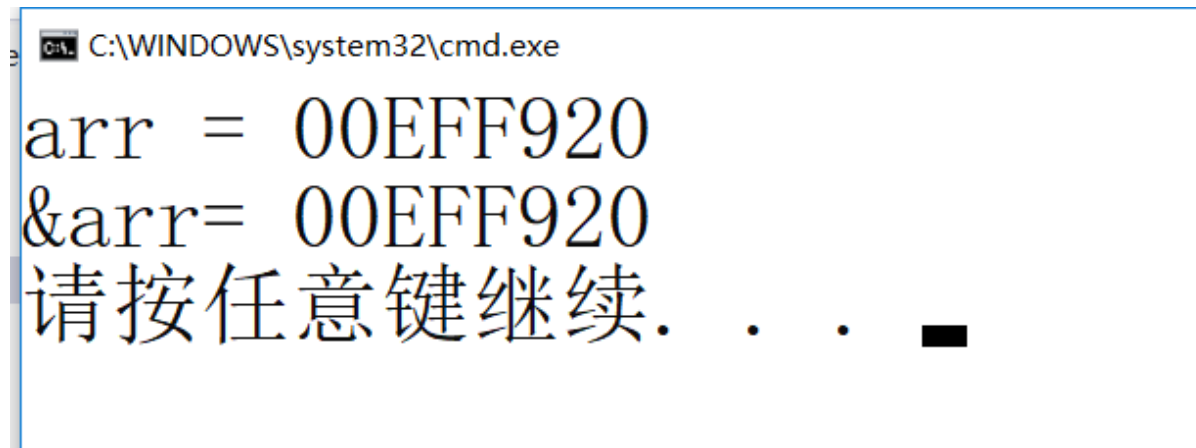
我们知道`arr`是数组名，数组名表示数组首元素的地址。

那`&arr`数组名到底是啥？

我们看一段代码：

```
#include <stdio.h>
int main()
{
    int arr[10] = {0};
    printf("%p\n", arr);
    printf("%p\n", &arr);
    return 0;
}
```

运行结果如下：



```
C:\WINDOWS\system32\cmd.exe
arr = 00EFF920
&arr= 00EFF920
请按任意键继续. . .
```

可见数组名和&数组名打印的地址是一样的。

难道两个是一样的吗？

我们再看一段代码：

```
#include <stdio.h>
int main()
{
    int arr[10] = { 0 };
    printf("arr = %p\n", arr);
    printf("&arr= %p\n", &arr);

    printf("arr+1 = %p\n", arr+1);
    printf("&arr+1= %p\n", &arr+1);
    return 0;
}
```

```
templateTest - Microsoft Visual Studio
C:\WINDOWS\system32\cmd.exe

arr = 0133FBB0
&arr= 0133FBB0
arr+1 = 0133FBB4
&arr+1= 0133FBD8
请按任意键继续. . .
```

根据上面的代码我们发现，其实`&arr`和`arr`，虽然值是一样的，但是意义应该不一样的。

实际上：`&arr` 表示的是**数组的地址**，而不是数组首元素的地址。（细细体会一下）

本例中`&arr` 的类型是：`int(*)[10]`，是一种数组指针类型

数组的地址+1，跳过整个数组的大小，所以`&arr+1` 相对于 `&arr` 的差值是40。

3.3 数组指针的使用

那数组指针是怎么使用的呢？

既然数组指针指向的是数组，那数组指针中存放的应该是数组的地址。

看代码：

```
#include <stdio.h>
int main()
{
    int arr[10] = {1,2,3,4,5,6,7,8,9,0};
    int (*p)[10] = &arr; //把数组arr的地址赋值给数组指针变量p
    //但是我们一般很少这样写代码
    return 0;
}
```

一个数组指针的使用：

```
#include <stdio.h>
void print_arr1(int arr[3][5], int row, int col)
{
    int i = 0;
    for(i=0; i<row; i++)
    {
        for(j=0; j<col; j++)
        {
            printf("%d ", arr[i][j]);
        }
    }
}
```

```

        printf("\n");
    }
}
void print_arr2(int (*arr)[5], int row, int col)
{
    int i = 0;
    for(i=0; i<row; i++)
    {
        for(j=0; j<col; j++)
        {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
int main()
{
    int arr[3][5] = {1,2,3,4,5,6,7,8,9,10};
    print_arr1(arr, 3, 5);
    //数组名arr, 表示首元素的地址
    //但是二维数组的首元素是二维数组的第一行
    //所以这里传递的arr, 其实相当于第一行的地址, 是一维数组的地址
    //可以数组指针来接收
    print_arr2(arr, 3, 5);
    return 0;
}

```

学了指针数组和数组指针我们来一起回顾并看看下面代码的意思：

```

int arr[5];      arr是一个5个元素的整形数组
int *parr1[10];  parr1是一个指针数组, 数组里有10个元素, 每个元素存放的都是int类型的
int (*parr2)[10]; parr2是一个数组指针, 指向一个类型为int的元素个数为10的数组。
int (*parr3[10])[5]; parr3先与方块[10]结合, 是一个指针数组, 数组里有10个元素, 每个
                    元素都是一个数组指针, 每个数组指针指向的数组有5个元素, 每个元
                    素是int类型的

```

4. 数组参数、指针参数

在写代码的时候难免要把【数组】或者【指针】传给函数，那函数的参数该如何设计呢？

4.1 一维数组传参

```

#include <stdio.h>
void test(int arr[])//ok? ok
{}
void test(int arr[10])//ok? ok
{}
void test(int *arr)//ok? ok
{}
void test2(int *arr[20])//ok? ok 传入arr2 指针类型的数组, 接受的参数
也是指针类型为int的指针数组, 完全ok, [20]可以省略
{}
void test2(int **arr)//ok? 传入arr2指针类型的数组, arr2表示一级指针
的地址, 接受的参数是指针数组的指针, 也就是二级指针, 一级指针的地址, 正好是
二级指针。
{}
int main()
{
    int arr[10] = {0};
}

```

```

int *arr2[20] = {0};
test(arr);
test2(arr2);
}

```

一维数组进行传参时，~~参数~~部分可以写成数组，也可以写成指针，写成数组的时候，数组大小可以省略，写成指针的时候，要找好合理的指针类型。

4.2 二维数组传参

```

void test(int arr[3][5])//标准写法，传入的二维数组，接受的也是二维数组
{}
void test(int arr[][5])//不可以，只能省略行不能省略列
{}
void test(int arr[][5])//可以，接受的二维数组下表只能省略行，不能省略列
{}

```

//总结：二维数组传参，函数形参的设计只能省略第一个[]的数字。也就是行 //因为对一个二维数组，可以不知道有多少行，但是必须知道一行多少元素。 //这样才方便运算。

```

void test(int *arr)//如果arr是一维数组可以，二维数组就不可以了，二维数组的第一个元素是第一行的数组{}
void test(int* arr[5])//ok?
{}
void test(int (*arr)[5])//ok?
{}
void test(int **arr)//不可以

int main(){
    int arr[3][5] = { 0 };
    return 0;
}

```

4.3 一级指针传参

```

#include <stdio.h>
void print(int *p, int sz)
{
    int i = 0;
    for(i=0; i<sz; i++)
    {
        printf("%d\n", *(p+i));
    }
}
int main()
{
    int arr[10] = {1,2,3,4,5,6,7,8,9};
    int *p = arr;
    int sz = sizeof(arr)/sizeof(arr[0]);
    //一级指针p，传给函数
    print(p, sz);
    return 0;
}

```

思考：

当一个函数的参数部分为一级指针的时候，函数能接收什么参数？

比如：

```
void test1(int *p) //传入参数可以是整形类型的地址(变量取地址值，或者是存放地址的指针变量){}
//test1函数能接收什么参数?
void test2(char* p) //传入参数可以是字符类型的地址
{}
//test2函数能接收什么参数?
```

4.4 二级指针传参

```
#include <stdio.h>
void test(int** ptr)
{
    printf("num = %d\n", **ptr);
}
int main()
{
    int n = 10;
    int*p = &n;
    int **pp = &p;
    test(pp);
    test(&p);
    return 0;
}
```

思考：

当函数的参数为二级指针的时候，可以接收什么参数？

```
void test(char **p)
{
}
int main()
{
    char c = 'b';
    char*pc = &c;
    char**ppc = &pc;
    char* arr[10];
    test(&pc);
    test(ppc);
    test(arr); //ok?
    return 0;
}
```


传入的参数是二级指针本身，或者是一级指针的取地址，存放一级指针的数组名

5. 函数指针

首先看一段代码：


```
#include <stdio.h>
void test()
{
    printf("hehe\n");
}
int main()
{
    printf("%p\n", test);
    printf("%p\n", &test);
    return 0;
}
```

输出的结果：

 选择C:\WINDOWS\system32\cmd.exe

```
013211DB
013211DB
请按任意键继续. . .
```

输出的是两个地址，这两个地址是 `test` 函数的地址。

那我们的函数的地址要想保存起来，怎么保存？

下面我们看代码：

```
void test()
{
    printf("hehe\n");
}
//下面pfun1和pfun2哪个有能力存放test函数的地址？
void (*pfun1)();
void *pfun2();
```

首先，能给存储地址，就要求 `pfun1` 或者 `pfun2` 是指针，那哪个是指针？

答案是：

`pfun1` 可以存放。`pfun1` 先和 `*` 结合，说明 `pfun1` 是指针，指针指向的是一个函数，指向的函数无参数，返回值类型为 `void`。

阅读两段有趣的代码：

从0开始分析，0是数字，0前面的 `(void(*)())` 是个无返回值的函数指针类型，`(void(*p)())` 是一个无返回值的函数指针变量，就是把0强制转换为函数指针类型；就当成那个无名函数的地址，`(* (void(*p)())0)` 对地址为0的函数进行解引用，就是调用函数，后面加了个 `()`，意味着调用了那个地址为0的无参函数。该函数无参，返回类型是 `void`。

```
//代码1
(* (void (*)())0)();
//代码2
void (*signal(int, void(*) (int)))(int);
```

【注】推荐《C陷阱和缺陷》

这本书中提及这两个代码。

从 `signal` 突破，`signal` 是函数名，函数第一个参数类型是 `int`，第二个参数类型是 `void(*) (int)` 函数指针类型，把 `signal(int, void(*) (int))` 提出来，就剩下 `void(*) (int)`；是一个函数指针类型。`signal` 函数返回类型也是一个函数指针，该函数指针，指向一个参数为 `int`，返回值类型是 `void` 的函数。`signal` 是一个函数的声明。

代码2太复杂，如何简化：

```
typedef void (*pfun_t)(int);      typedef 一对类型进行重定义
pfun_t signal(int, pfun_t);
```

6. 函数指针数组

数组是一个存放相同类型数据的存储空间，那我们已经学习了指针数组，比如：

```
int *arr[10];
//数组的每个元素是int*
```

那要把函数的地址存到一个数组中，那这个数组就叫函数指针数组，那函数指针的数组如何定义呢？

```
int (*parr1[10])();
int *parr2[10]();
int (*)() parr3[10];
```

答案是：parr1

parr1 先和 [] 结合，说明 parr1 是数组，数组的内容是什么呢？是 int (*)() 类型的函数指针。

函数指针数组的用途：**转移表**

例子：（计算器）

```
#include <stdio.h>
int add(int a, int b)
{
    return a + b;
}
int sub(int a, int b)
{
    return a - b;
}
int mul(int a, int b)
{
    return a*b;
}
int div(int a, int b)
{
    return a / b;
}
int main()
{
    int x, y;
    int input = 1;
    int ret = 0;
    do
    {
        printf( "*****\n" );
        printf( "  1:add          2:sub  \n" );
        printf( "  3:mul          4:div  \n" );
        printf( "*****\n" );
        printf( "请选择: " );
        scanf( "%d", &input);
        switch (input)
        {
```

```

    case 1:
        printf( "输入操作数: " );
        scanf( "%d %d", &x, &y);
        ret = add(x, y);
        printf( "ret = %d\n", ret);
        break;
    case 2:
        printf( "输入操作数: " );
        scanf( "%d %d", &x, &y);
        ret = sub(x, y);
        printf( "ret = %d\n", ret);
        break;
    case 3:
        printf( "输入操作数: " );
        scanf( "%d %d", &x, &y);
        ret = mul(x, y);
        printf( "ret = %d\n", ret);
        break;
    case 4:
        printf( "输入操作数: " );
        scanf( "%d %d", &x, &y);
        ret = div(x, y);
        printf( "ret = %d\n", ret);
        break;
    case 0:
        printf("退出程序\n");
        break;
    default:
        printf( "选择错误\n" );
        break;
}
} while (input);

return 0;
}

```

使用函数指针数组的实现：

```

#include <stdio.h>
int add(int a, int b)
{
    return a + b;
}
int sub(int a, int b)
{
    return a - b;
}
int mul(int a, int b)
{
    return a*b;
}
int div(int a, int b)
{
    return a / b;
}
int main()
{

```

```

int x, y;
int input = 1;
int ret = 0;
int(*p[5])(int x, int y) = { 0, add, sub, mul, div }; //转移表
while (input)
{
    printf( "*****\n" );
    printf( "  1:add          2:sub  \n" );
    printf( "  3:mul          4:div  \n" );
    printf( "*****\n" );
    printf( "请选择: " );
    scanf( "%d", &input);
    if ((input <= 4 && input >= 1))
    {
        printf( "输入操作数: " );
        scanf( "%d %d", &x, &y);
        ret = (*p[input])(x, y);
    }
    else
        printf( "输入有误\n" );
    printf( "ret = %d\n", ret);
}
return 0;
}

```

7. 指向函数指针数组的指针

指向函数指针数组的指针是一个 [指针](#)

指针指向一个 [数组](#)，数组的元素都是 [函数指针](#)；

如何定义？

```

void test(const char* str)
{
    printf("%s\n", str);
}

int main()
{
    //函数指针pfun
    void (*pfun)(const char*) = test;
    //函数指针的数组pfunArr
    void (*pfunArr[5])(const char* str);
    pfunArr[0] = test;
    //指向函数指针数组pfunArr的指针ppfunArr
    void ((*ppfunArr)[5])(const char*) = &pfunArr;
    return 0;
}

```

8. 回调函数

回调函数就是一个通过函数指针调用的函数。如果你把函数的指针（地址）作为参数传递给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应。

首先演示一下qsort函数的使用：

```
#include <stdio.h>

//qsort函数的使用者得实现一个比较函数
int int_cmp(const void * p1, const void * p2)
{
    return (*(int *)p1 - *(int *) p2);
}

int main()
{
    int arr[] = { 1, 3, 5, 7, 9, 2, 4, 6, 8, 0 };
    int i = 0;

    qsort(arr, sizeof(arr) / sizeof(arr[0]), sizeof(int), int_cmp);
    for (i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

使用回调函数，模拟实现qsort（采用冒泡的方式）。

注意：这里第一次使用void*的指针，讲解void*的作用。

```
#include <stdio.h>

int int_cmp(const void * p1, const void * p2)
{
    return (*(int *)p1 - *(int *) p2);
}

void _swap(void *p1, void * p2, int size)
{
    int i = 0;
    for (i = 0; i < size; i++)
    {
        char tmp = *((char *)p1 + i);
        *((char *)p1 + i) = *((char *) p2 + i);
        *((char *)p2 + i) = tmp;
    }
}

void bubble(void *base, int count, int size, int(*cmp)(void *, void *))
{
    int i = 0;
    int j = 0;
```

```

for (i = 0; i < count - 1; i++)
{
    for (j = 0; j < count - i - 1; j++)
    {
        if (cmp ((char *) base + j * size, (char *) base + (j + 1) * size) > 0)
        {
            _swap((char *) base + j * size, (char *) base + (j + 1) * size, size);
        }
    }
}

int main()
{
    int arr[] = { 1, 3, 5, 7, 9, 2, 4, 6, 8, 0 };
    //char *arr[] = {"aaaa", "dddd", "cccc", "bbbb"};
    int i = 0;
    bubble(arr, sizeof(arr) / sizeof(arr[0]), sizeof(int), int_cmp);
    for (i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)
    {
        printf(" %d ", arr[i]);
    }
    printf("\n");
    return 0;
}

```

9. 指针和数组笔试题解析

数组名是首元素地址但是有两个意外，
1. sizeof(数组名)，数组名表示整个数组
2. &数组名，数组名表示整个数组。

//一维数组 sizeof(数组名)-计算的是数组总大小-单位是字节-

```

int a[] = {1,2,3,4};
printf("%d\n", sizeof(a));    16, int 一个元素4字节, 4个元素16字节
printf("%d\n", sizeof(a+0));  4, 首元素地址+0, 还是首元素地址
printf("%d\n", sizeof(*a));    4, 还是首元素地址
printf("%d\n", sizeof(a+1));  4, 首元素地址, a+1表示第二个元素的地址
printf("%d\n", sizeof(a[1]));  4, 第二个元素的大小
printf("%d\n", sizeof(&a));    4, 取出的数组的地址, 地址的大小也是4字节(不管是那种类型的数据类型的地址都是4个字节)
printf("%d\n", sizeof(*&a));  16, 效果与第一个printf一样, a是数组名, 取地址拿它的地址, 解引用,
printf("%d\n", sizeof(&a+1));  4, 取数组a地址4个字节, 再加一, 还是地址类型, 还是4个字节..
printf("%d\n", sizeof(&a[0]));  4, 第一个元素的地址
printf("%d\n", sizeof(&a[0]+1)); 4, 第二个元素的地址

```

//字符数组

```

char arr[] = {'a', 'b', 'c', 'd', 'e', 'f'};
printf("%d\n", sizeof(arr));    6, sizeof计算的是数组大小, 6*1=6字节
printf("%d\n", sizeof(arr+0));  4, arr表示首元素字节, arr+0表示的还是首元素地址
printf("%d\n", sizeof(*arr));    1, 解引用首元素地址就是首元素, char类型占1字节
printf("%d\n", sizeof(arr[1]));  1,
printf("%d\n", sizeof(&arr));    4, 取数组地址arr, 地址大小4/8个字节 (32位/64位)
printf("%d\n", sizeof(&arr+1));  4, 数组地址+1跑到了数组后面, 但是还是地址所以是4/8字节
printf("%d\n", sizeof(&arr[0]+1)); 4, 第一个元素地址+1, 第二个元素地址

```

strlen(): 该函数从第一个字符开始计算字符串中字符数, 直到遇到空字符即'\0'为止, 然后返回计算字符数的长度, 包括'\0'。

```

printf("%d\n", strlen(arr));    随机值,
printf("%d\n", strlen(arr+0));  与第一行完全相同,
printf("%d\n", strlen(*arr));    运行错误
printf("%d\n", strlen(arr[1]));  运行错误
printf("%d\n", strlen(&arr));    随机值, 与第一行相同

```

strlen不计
\\0，
参数可以接受
地址，
sizeof计算\\0

```
printf("%d\\n", strlen(&arr+1));    随机值，取数组地址从数组首地址开始+1到数组尾部，是上一行随机值-6
printf("%d\\n", strlen(&arr[0]+1));    随机值，上一行的随机值-1

char arr[] = "abcdef"; [abcdef\\0]
printf("%d\\n", sizeof(arr));        7, sizeof也计算\\0为一个字节
printf("%d\\n", sizeof(arr+0));      4/8, arr数组名首元素地址，首元素加0还是首元素地址
printf("%d\\n", sizeof(*arr));        1, 首元素地址解引用，就是第一个元素
printf("%d\\n", sizeof(arr[1]));      1, 第二个元素，
printf("%d\\n", sizeof(&arr));        4/8, 数组取地址，取整个数组，但是属于地址，char(*)[7]
printf("%d\\n", sizeof(&arr+1));      4/8, 数组取地址取出整个数组，加1跳过整个数组的地址，数组后面的地址，是地
printf("%d\\n", sizeof(&arr[0]+1));    址就是4/8
                                      4/8,取出第一个元素的地址，加1，为第二个元素的地址
                                      。
printf("%d\\n", strlen(arr));          6, 首元素地址
printf("%d\\n", strlen(arr+0));        6, 首元素地址
printf("%d\\n", strlen(*arr));         error, 首元素地址解引用，把a传过来了，a的ASCII码为97，不是数组，返回error
printf("%d\\n", strlen(arr[1]));       error, 第二个元素，传过来的b，b=98，返回错误，
printf("%d\\n", strlen(&arr));          6, strlen可以传地址
printf("%d\\n", strlen(&arr+1));        随机值，从数组结束开始到\\0结束
printf("%d\\n", strlen(&arr[0]+1));    5, 取数组第一个元素的地址+1，就是第二个元素的地址

char *p = "abcdef";                  把首元素a的地址放到p里
printf("%d\\n", sizeof(p));            4/8
printf("%d\\n", sizeof(p+1));          4/8,b的地址
printf("%d\\n", sizeof(*p));            1,*p拿出a的字节
printf("%d\\n", sizeof(p[0]));          1, 效果与*(p+0)相同
printf("%d\\n", sizeof(&p));            4/8
printf("%d\\n", sizeof(&p+1));          4/8
printf("%d\\n", sizeof(&p[0]+1));      1, b

printf("%d\\n", strlen(p));            6
printf("%d\\n", strlen(p+1));          5
printf("%d\\n", strlen(*p));            error, *p找到a，传过来的是a的ASCII码
printf("%d\\n", strlen(p[0]));          error, 和上一行一个道理
printf("%d\\n", strlen(&p));            未知，从p指针开头开始
printf("%d\\n", strlen(&p+1));          未知，从p指针开头加一开始，
printf("%d\\n", strlen(&p[0]+1));      5, 从b开始
```

//二维数组

```
int a[3][4] = {0};
printf("%d\\n", sizeof(a));
printf("%d\\n", sizeof(a[0][0]));
printf("%d\\n", sizeof(a[0]));
printf("%d\\n", sizeof(a[0]+1));
printf("%d\\n", sizeof(*(a[0]+1)));
printf("%d\\n", sizeof(a+1));
printf("%d\\n", sizeof(*(a+1)));
printf("%d\\n", sizeof(&a[0]+1));
printf("%d\\n", sizeof(&a[0]+1)));
printf("%d\\n", sizeof(*a));
printf("%d\\n", sizeof(a[3]));
```

总结:

数组名的意义:

1. sizeof(数组名)，这里的数组名表示整个数组，计算的是整个数组的大小。
2. &数组名，这里的数组名表示整个数组，取出的是整个数组的地址。
3. 除此之外所有的数组名都表示首元素的地址。

10. 指针笔试题

笔试题1:

```
int main()
{
    int a[5] = { 1, 2, 3, 4, 5 };
    int *ptr = (int *)(&a + 1);
    printf( "%d,%d", *(a + 1), *(ptr - 1));
    return 0;
}
//程序的结果是什么？
```

因为a是数组，所以&a返回的地址类型是数组指针类型，所以(&a+1)是数组后的地址，强制转换为整形指针，赋值给ptr，*(a+1)首元素地址加一解引用，返回2，*(ptr-1),数组后第一个元素的地址减一解引用

笔试题2

//由于还没学习结构体，这里告知结构体的大小是20个字节

```
struct Test
```

```
{
    int Num;
    char *pcName;
    short sDate;
    char cha[2];
    short sBa[4];
}
```

指针的类型决定了指针的运算

```
*p;
```

//假设p 的值为0x100000。 如下表表达式的值分别为多少？

//已知，结构体Test类型的变量大小是20个字节

```
int main()
{
    printf("%p\n", p + 0x1);
    printf("%p\n", (unsigned long)p + 0x1);
    printf("%p\n", (unsigned int*)p + 0x1);
    return 0;
}
```

笔试题3

```
int main()
{
    int a[4] = { 1, 2, 3, 4 };
    int *ptr1 = (int *)(&a + 1);
    int *ptr2 = (int *)((int)a + 1);
    printf( "%x,%x", ptr1[-1], *ptr2);
    return 0;
}
```

笔试题4


```
#include <stdio.h>
int main()
{
    int a[3][2] = { (0, 1), (2, 3), (4, 5) };
    int *p;
    p = a[0];
    printf( "%d", p[0]);
    return 0;
}
```

笔试题5

```
int main()
{
    int a[5][5];
    int(*p)[4];
    p = a;
    printf( "%p,%d\n", &p[4][2] - &a[4][2], &p[4][2] - &a[4][2]);
    return 0;
}
```

笔试题6

```
int main()
{
    int aa[2][5] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int *ptr1 = (int *)(&aa + 1);
    int *ptr2 = (int *)(* (aa + 1));
    printf( "%d,%d", *(ptr1 - 1), *(ptr2 - 1));
    return 0;
}
```

笔试题7

```
#include <stdio.h>

int main()
{
    char *a[] = {"work","at","alibaba"};
    char**pa = a;
    pa++;
    printf("%s\n", *pa);
    return 0;
}
```

笔试题8

```
int main()
{
    char *c[] = {"ENTER", "NEW", "POINT", "FIRST"};
    char**cp[] = {c+3, c+2, c+1, c};
    char***cpp = cp;
    printf("%s\n", **++cpp);
    printf("%s\n", *--*++cpp+3);
    printf("%s\n", *cpp[-2]+3);
    printf("%s\n", cpp[-1][-1]+1);
    return 0;
}
```

本章结束

