数组名的含义：

数组名在不同的场合下的含义有两种：

表示整个数组的首地址：

- 在数组定义的时候
- 在使用sizeof 运算符中 数组名表示整个数组的首地址（求得整个数组的大小）
- 在取地址符 中 &arr ，arr 表示整个数组

表示整个数组首元素的首地址：

- 其它情况

```
int arr[10] ; // arr 表示整个数组的首地址
int len = sizeof arr ; // arr 表示整个数组的首地址 ， sizeof 运算符后的括号可以省略
int (* p) [10] = &arr; //arr 表示整个数组的首地址

int *p1 = arr ; // arr 表示数组的首元素的首地址
```

数组下标：

数组的下标实际上只是编译器提供一种简写，实际上如下：

```
int a [100] ;
a[10] = 250 ; ====> *(a+10) = 250 ;
```

通过加法交换律，有如下结果：

```
a[10] = 250 ;
*(a+10) = 250 ;
*(10+a) = 250 ;
10[a] = 250 ;
```

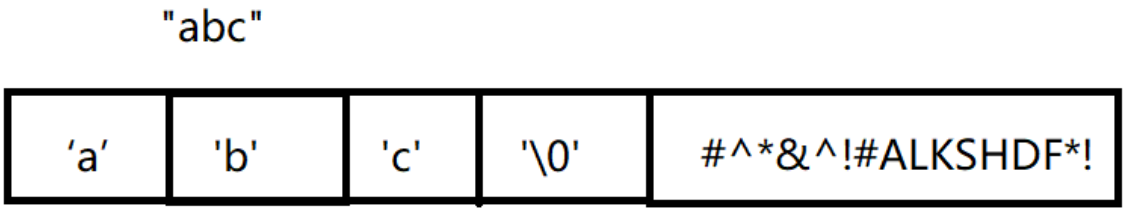
字符串常量：

字符串常量是一个被存放在常量区的字符串，实际上也可称为一个匿名数组。

匿名数组，同样满足数组名的含义。

```
char * msg2 = "Hello Even" ;// "Hello Even" 字符串常量首元素的首地址
char * msg1 = "Hello Even"+1 ;
```

```
3
4
5  printf("%s\n", "Hello Even" ) ;// "Hello Even" 字符串常量首元素的首地址
6  printf("%s\n", &"Hello Even" ) ; // "Hello Even" 字符串常量的整个数组的地址
7
8  printf("%c\n", "Hello Even"[6] ) ; // "Hello Even" 字符串常量首元素的首地址
[6]
9   // [6] 相当于+6个单位（char） 得到 ‘E’
10
```



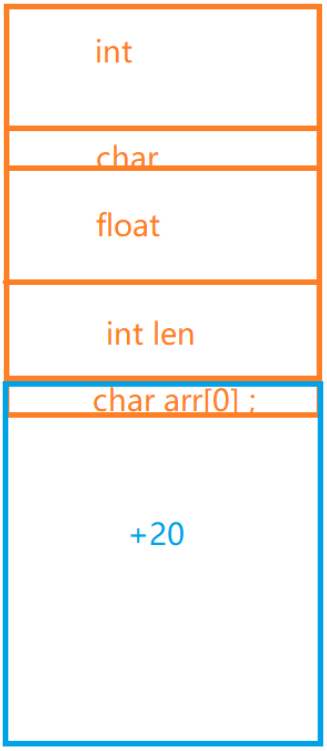
字符串在内存种的存储方式

零长数组：（预习结构体）

概念： 数组的长度为0， char arr [0] ;

用途：一般会放在结构体的末尾， 作为可变长度的入口。（数组是唯一一个允许越界访问的载体）

```
1  struct node
2  {
3   int a;
4   char b ;
5   float c ;
6   ..
7   ...
8   int len ;
9   char arr[0] ;
10  }
11
12  struct node *p = malloc(sizeof(struct node) + 20 ); // + 20 就是在原有的基
础上增加20字节
13  p->len = 20 ; // 设置额外增长的长度为 20
14
15
16
```



变长数组：

概念： 定义是， 使用一个变量作为数组的长度（元素的个数）。

重点： 变长数组并不是说在任意时候他的长度可以随意变化， 实际上只是在定义之前数组的长度是未知的有一个变量来决定， 但是定义语句过后变长数组的长度由定义那一刻变量的大小来决定。

```
int a = 200 ; // a 作为一个普通的变量 ， 200 则可以作为arr 的长度
a = 99 ; // 99 可以作为 arr 的长度

int arr[a]; // a 当前是 99 ， 因此数组arr 的长度已经确定为 99
 //从此以后该数组的长度已经确定为99 不会再变换

a = 10 ;  // a = 10 并不会影响数组的长度
```

注意：

1. 因为数组的长度未确定， 因此它不允许初始化。
2. 在使用的时候可以通过该变长数组来有限的节省内存空间。

多维数组：

概念： 数组内部的成员也是数组

```
int a [2][3] ;
```

定义与初始化：

```
int arr[2][3] = { {1,2,3} , { 4,5,6} };
int arr1[2][3] = { 1,2,3,4,5,6};
```

如何引用：

```
arr[0] [0] = 100 ; // 数组：（通过下标来访问）
*(*(arr+0)+0) = 100 ; // 通过指针偏移来访问
```

实例：

```
int arr[2][3] = { {1,2,3} , { 4,5,6} };
    int arr1[2][3] = { 1,2,3,4,5,6};

    int *p = arr ; // p指向数组arr 的首元素

    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            printf("arr[%d][%d]:%d\t" ,i ,j , arr[i][j] );
         }
     }

     printf("\n");

     for (int i = 0; i < 6 ; i++)
     {          //          *arr   得到元素1 的地址  + 1 则是加一个 int 类型
         printf("*(*(arr+%d)):%d\t" , i,*(*(arr)+i) );
     }

     printf("\n");

     for (int i = 0; i < 6 ; i++)
     {       //       arr  指的是首元素的首地址  {1,2,3} 的首地址  + 1则 + 3个整型
         printf("*(*(arr+%d)):%d\t" , i,*(*(arr+i)) );
     }


     for (int i = 0; i < 6 ; i++)
     {   //                    p 只是一个普通的整型指针，  与二维数组没有任何的关系
```

```
31          printf("*(p+%d): %d\n" , i ,*(p+i));
32      }
33
```

作业：

1. 把数组与指针理解代码搞透
2. 用变量 a 给出下面的定义
   (1) 一个整型数（An integer）
   (2) 一个指向整型数的指针（A pointer to an integer）
   (3) 一个指向指针的的指针，它指向的指针是指向一个整型数（A pointer to a pointer to aninteger）
   (4) 一个有 10 个整型数的数组（An array of 10 integers）
   (5) 一个有 10 个指针的数组，该指针是指向一个整型数的（An array of 10 pointers to integers）
   (6) 一个指向有 10 个整型数数组的指针（A pointer to an array of 10 integers）
   (7) 一个指向函数的指针，该函数有一个整型参数并返回一个整型数（A pointer to a functionthat takes an integer as an argument and returns an integer）
   (8) 一个有 10 个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数（An array of ten pointers to functions that take an integer argument and return an integer)

预习：

进程内存布局

堆/栈