基本概念：

逻辑： 一次性定义多个相同类型的变量，并且给他分配一片连续的内存

语法：

```
int arr [5] ;
```

1. 向系统申请一片连续的内存，名字为 arr

2. [5] 告诉系统这一片连续的内存需要或分成5等份

3. 每一个等份中都用来存放一个int 数据， 4字节

初始化：

只有在定义的时候赋值， 才可以称为初始化。数组只有在初始化的时候才可以统一赋值。

```
int arr [5] = {1,2,3,4,5} ; // 定义并初始化数组

int arr [5] = {1,2,3} ; //可以， 不完全初始化
int arr [5] = {1,2,3,4,5,6,7,8,9} ; // 错误（但是可以用） ， 越界初始化， 越界部分将会被编译器舍弃
int arr [ ] = {1,2,3,4,5,6,7,8,9} ; // 可以， 用户没有直接给定数组大小，
 // 但是有初始化， 因此数组的大小会在初始化时确定， 大小为 9
int arr [ ] ; // 错误的， 没有给定大小也没有初始化， 因此数组的内存大小无法确定系统无法分配
```

```
25
26      // 初始化越界
27      int arr1 [5]  = {1,2,3,4,5,6,7,8,9} ;
28      for (size_t i = 0; i < 9 ; i++)
29      {
30          printf("arr1[%ld]:%d\n" , i , arr1[i]);
31      }
32
```

问题 输出 调试控制台 终端 1: wsl

```
arr1[0]:1
arr1[1]:2
arr1[2]:3
arr1[3]:4
arr1[4]:5
arr1[5]:32767
arr1[6]:1718362880
arr1[7]:-1178838953
arr1[8]:1115686928
```

正常的数据

越界访问到内存中的未知数据

注意：

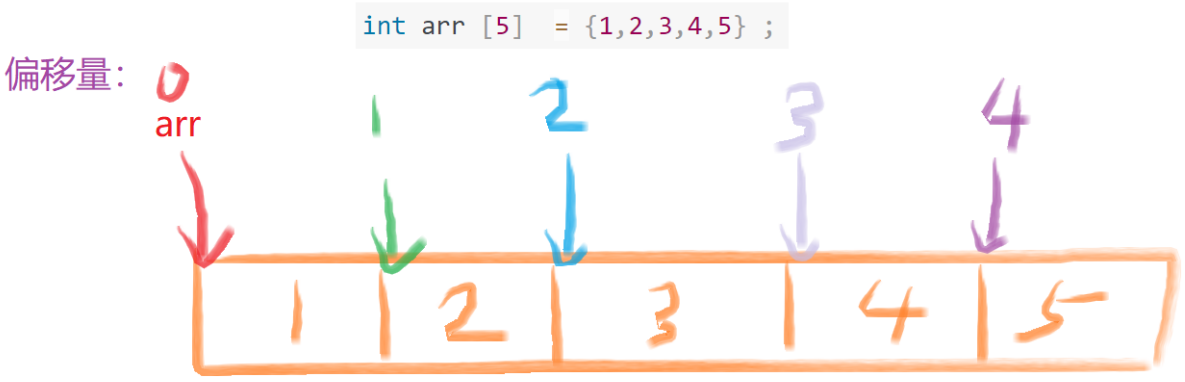
数组在定义的时候必须确定他的大小。

说白了就是中括号中[] 必须有数组的大小，如果没有就必须初始化

数组元素引用：

存储模式：一片连续的内存，按照数据的类型进行分割成若干个大小相同的格子

元素的下标与偏移量：以数组开头为基础的偏移的量（数据类型大小）




```
1  int arr [5]  = {1,2,3,4,5} ; // 定义并初始化数组
2
3  printf("arr[0]:%d\n" , arr[0]);
4
5  arr[0] = 99 ;  //  把数组的第1个元素（ 偏移量为0 ）修改为 99
6
```

```c
printf("arr[0]:%d\n" , arr[0]);

arr[5] = 250 ; //"错误" 越界访问， 并很有可能造成非法访问
 printf("arr[5]:%d\n" , arr[5]);

 arr = {9,8,7,6,5,4}; // 整体赋值只允许在初始化中
 arr = 100 ; // 错误


 printf("sizeof(arr):%ld\n" , sizeof(arr));
 int len = sizeof(arr) / sizeof(int) ; //  求数组元素的个数

 for (size_t i = 0; i < len ; i++)
 {
     printf("arr[%ld]:%d\n" , i , arr[i]);
 }
```

字符数组：

概念： 专门用来存放字符类型数据的数组， 称为字符数组

初始化+引用：

```c
char ch1 [5] = {'H','e','l','l','o'} ; // 定义一个字符类型的数组并把'H','e','l','l','o' 一个一个存进去
char ch2 [6] = {"Hello"} ; // 定义一个字符型的数组， 并把 "Hello" 字符串存放到数组中 ，因此该数组为字符串数组

char ch3 [6] = "Hello" ; // 与ch2 一样， 大括号可以省略

ch3[1] = 'E' ; // 可以， 把数组中第二个元素‘e’修改为‘E’
ch3 = "Even" ; // 不可以， 只有在初始化的时候才能整体赋值

printf("%s\n" , ch1); // 在访问ch1的时候并没有发现结束符，因此很有可能会把ch2的内容一并输出
```

注意：

ch1 为字符数组， 它没有结束符，因此在打印输出的时候应该避免使用 %s 进行输出， 有可能会造成越界访问。

思考：

定义数组时没有初始化会怎么样?

```c
int main(int argc, char const *argv[])
{
    int arr [5] ; // 定义并没有初始化数组

    for (size_t i = 0; i < 5 ; i++)
    {
        printf("ar1[%ld]:%d\n" , i , arr[i]);
    }
```

```
问题  输出  调试控制台  终端   1: wsl
even@PC-20210112EPXS:/mnt/d/GZ2123/01 C语言基础/08 数组$ gcc test.c
even@PC-20210112EPXS:/mnt/d/GZ2123/01 C语言基础/08 数组$ ./a.out
ar1[0]:-1153431776
ar1[1]:32662
ar1[2]:-1153432160
ar1[3]:32662
ar1[4]:-598840800
even@PC-20210112EPXS:/mnt/d/GZ2123/01 C语言基础/08 数组$
```

结论： 没有初始化的情况看内存的能容是随机值（内存中原有的内容）

不完全初始化又会怎么样?

结论：不完全初始化， 已经初始化部分是已知值， 其它则是0

注意:

把1 .2.3.4.5 当作字符的ASCII值存放到数组中

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

存放到内存中的是字符1.2.3.4.5所对应的ASCII值

```c
    char  buf [5] = {'1','2','3','4','5'};
    for (size_t i = 0; i < 5 ; i++)
    {
        printf("buf[%ld]:%d\n" , i , buf[i]);
        printf("buf[%ld]:%c\n" , i , buf[i]);
    }
```