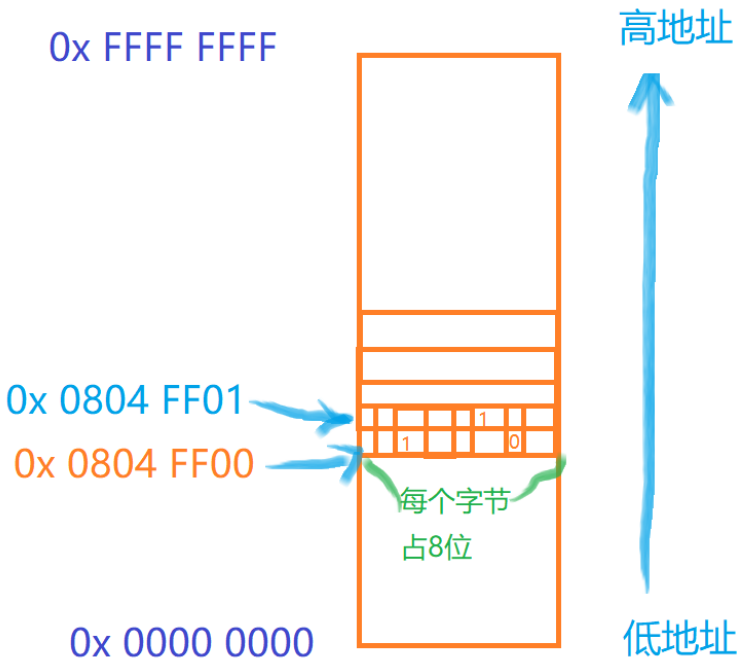
内存地址

字节： 字节是内存容量的一个单位， byte ， 一个字节byte 有 8个位 bit

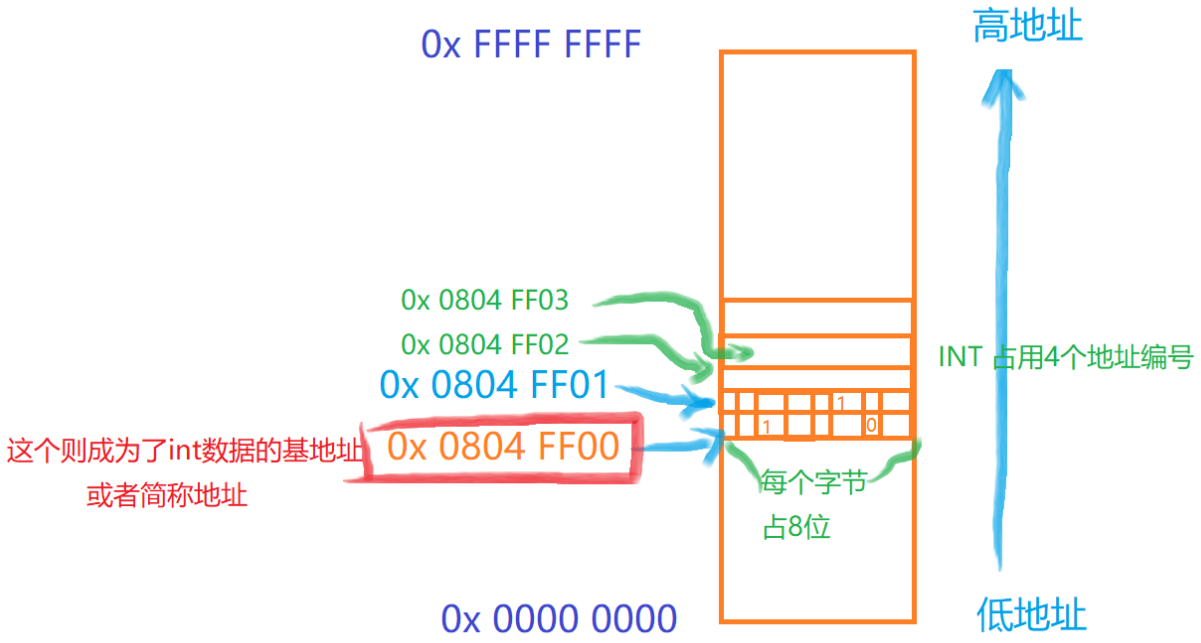
地址： 系统位了方便区分每一个字节的数据， 而对内存进行了逐一编号， 而该编号就是内存地址。



基地址

单字节的数据： char 它所在地址的编号就是该数据的地址

多字节的数据：int 它拥有4个连续的地址的编号， 其中地址值最小的称为该变量的地址

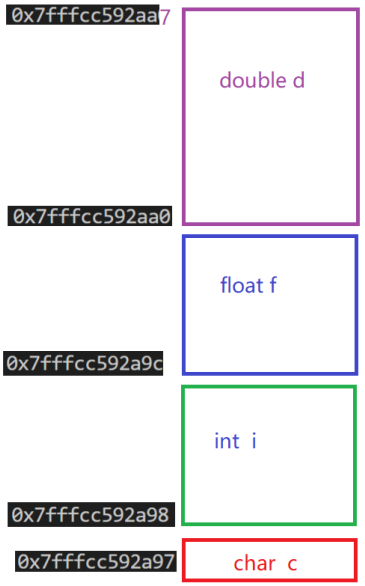


取地址符号

每一个变量其实都对应了片内存， 因此都可以通过 & 取地址符号把其地址获得。

```
int a ;
char c ;

printf("a的地址为：%p\n" , &a);
printf("c的地址为：%p\n" , &c);


运行结果：
a的地址为：0x7fffd5595d84
 c的地址为：0x7fffd5595d83
```

```
    int i ;
    double d ;
    float f ;
    char c ;

    printf("i的地址为：%p\n" , &i);
    printf("d的地址为：%p\n" , &d);
    printf("f的地址为：%p\n" , &f);
    printf("c的地址为：%p\n" , &c);

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/09 指针$ ./a.out
i的地址为：0x7fffcc592a98
d的地址为：0x7fffcc592aa0
f的地址为：0x7fffcc592a9c
c的地址为：0x7fffcc592a97
```



注意：

- 虽然不同的数据类型所占用的内存空间不同， 但是他们的地址所占用的内存空间（地址的大小= 指针的大小）是恒定的， 由系统的位数来决定 32位 / 64位
- 不同的地址他从表面上看似乎没有什么差别，但是由他们所代表的内存的尺寸是不一样的（由内存中所存放的数据类型相关），因此我们在访问这些地址的时候需要严格区分它们的逻辑关系。

指针的基础

指针的概念：

&a 就是a的地址， 实质上也可以理解为他是一个指针 指向 a的地址。

专门用来存放地址的一个变量，因此指针的大小是恒定的，由系统的位数来决定。

指针的定义语法：

```
int a ; // 定义一片内存名字叫 a ， 约定好该内存用来存放 整型数据
int * p ; // 定义一片内存名字叫 p ， 约定好该内存用来存放 整型数据的地址
char * p1 ; // 定义一片内存名字叫 p1 ， 约定好该内存用来存放 字符数据的地址
double * p2 ; // 定义一片内存名字叫 p2 ， 约定好该内存用来存放 双精度数据的地址
```

注意：

指针的类型，并不是用来决定该指针的大小,而是用来告诉编译器如果我们通过来指针来访问内存时需要访问的内存的大小尺寸。

指针的赋值以及初始化：

```
int a = 100 ;
int * p = &a ; // 定义并初始化

double d = 1024.1234 ;
double * p1 = &d ;// 定义并初始化

float f ;
float * p2 ;// 定义并没有初始化

 p2 = &f ; // 给指针赋值

 .....
 ......
```

注意：

不同类型的指针，应该用来指向与其相对应的类型的变量的地址。

指针的索引：

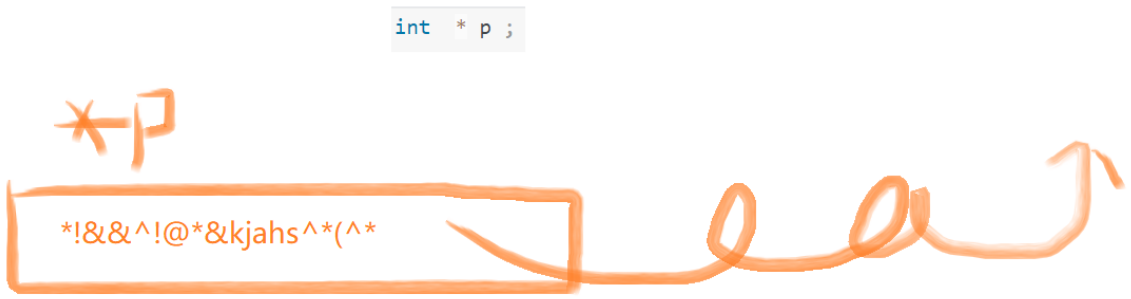
通过指针获得它所指向的数据（解引用/取目标）

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

*p = 250 ; // *p <==> a

printf("*p:%d\n" , *p) ;

```

野指针：

概念： 指向一块未知内存的指针， 被称为野指针。

```c
int * p ;
```



危害：

- 引用野指针的时候，很大概率我们会访问到一个非法内存，通常会出现段错误（Segmentation fault (core dumped)）并导致程序崩溃。
- 更加严重的后果，如果访问的时系统关键的数据，则有可能造成系统崩溃

产生原因：

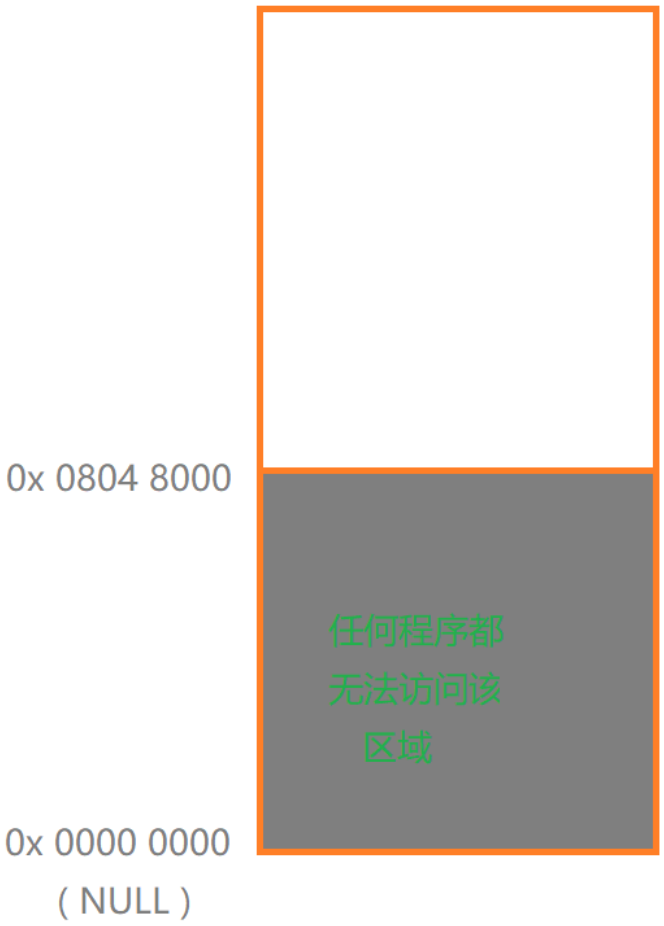
- 定义时没有对他进行初始化
- 指向的内存被释放，系统已经回收， 后该指针并没有重新初始化
- 指针越界

如何防止：

- 定义时记得对他进行初始化
- 绝对不去访问被回收的内存地址， 当我们释放之后应该重新初始化该指针。
- 确认所申请的内存的大小，谨防越界

空指针

在很多的情况下我们一开始还不确定一个指针需要指向哪里，因此可以让该指针先指向一个不会破坏系统关键数据的位置， 而这个位置一般就是NULL （空）。因此指向该地址的指针都称之为空指针。



概念：

空指针就是保存了地址值为零的一个地址， 也就零地址NULL

```
int * p1 = NULL ; // 定义一个指针， 并初始化为空指针（ 指向 NULL ）

*p1 = 250 ; // 段错误 ， 该地址不允许写入任何东西

printf("%p -- %d \n" , NULL , NULL  ); //  （nul--0）
```

指针运算

指针的加法：意味着地址向上移动若干个目标 （指针的类型）

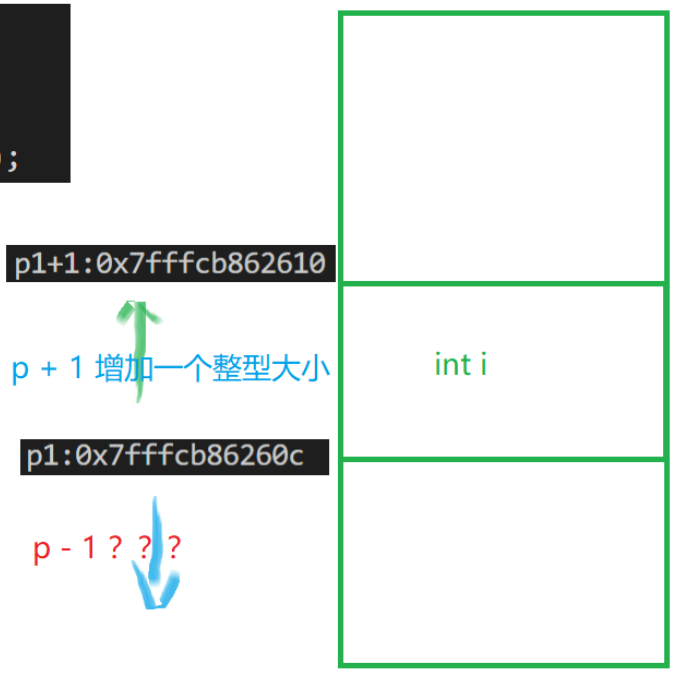
指针的减法：意味着地址向下移动若干个目标 （指针的类型）

```
long l ;
    long *p = &l ;

    int i ;
    int * p1 = &i;

    printf("p:%p\n" , p );
```

```
8      printf("p+1:%p\n" , p+1 );
9
10      printf("p1:%p\n" , p1 );
11      printf("p1+1:%p\n" , p1+1 );
12
13  运行结果：
14  p:0x7fffcb862610
15  p+1:0x7fffcb862618
16  p1:0x7fffcb86260c
17  p1+1:0x7fffcb862610
```

```
int i ;
int * p1 = &i;
printf("p1:%p\n" , p1 );
printf("p1+1:%p\n" , p1+1 );
```

```
p1:0x7fffcb86260c
p1+1:0x7fffcb862610
```



注意：

指针在加减运算的过程中， 加/减的大小取决于该指针他自己的类型，与它所执行的数据实际的类型没有关系。

练习：

使用指针来交换两个变量（a,b）的值。