C 语言空指针 NULL 以及 void 指针

空指针 NULL

一个指针变量可以指向计算机中的任何一块内存，不管该内存有没有被分配，也不管该内存有没有使用权限，只要把地址给它，它就可以指向，C 语言没有一种机制来保证指向的内存的正确性，程序员必须自己提高警惕。

很多初学者会在无意间对没有初始化的指针进行操作，这是非常危险的，请看下面的例子：

```
#include <stdio.h>

int main(){
    char *str;
    gets(str);
    printf("%s\n", str);
    return 0;
}
```

这段程序没有语法错误，能够通过编译和链接，但当用户输入完字符串并按下回车键时就会发生错误，在 Linux 下表现为段错误（Segment Fault），在 Windows 下程序直接崩溃。如果你足够幸运，或者输入的字符串少，也可能不报错，这都是未知的。

前面（《C 语言局部变量和全局变量》一节）我们讲过，未初始化的局部变量的值是不确定的，C 语言并没有对此作出规定，不同的编译器有不同的实现，我曾警告大家不要直接使用未初始化的局部变量。上面的代码中，str 就是一个未初始化的局部变量，它的值是不确定的，究竟指向哪块内存也是未知的，大多数情况下这块内存没有被分配或者没有读写权限，使用 gets() 函数向它里面写入数据显然是错误的。

我强烈建议对没有初始化的指针赋值为 NULL，例如：

```
char *str = NULL;
```

NULL 是“零值、等于零”的意思，在 C 语言中表示空指针。从表面上理解，空指针是不指向任何数据的指针，是无效指针，程序使用它不会产生效果。

注意区分大小写，null 没有任何特殊含义，只是一个普通的标识符。

很多库函数都对传入的指针做了判断，如果是空指针就不做任何操作，或者给出提示信息。更改上面的代码，给 str 赋值 NULL，看看会有什么效果：

```
#include <stdio.h>

int main(){
    char *str = NULL;
    gets(str);
    printf("%s\n", str);
    return 0;
}
```

运行程序后发现，还未等用户输入任何字符，printf() 就直接输出了(null)。我们有理由据此推断，gets() 和 printf() 都对空指针做了特殊处理：

- gets() 不会让用户输入字符串，也不会向指针指向的内存中写入数据；
- printf() 不会读取指针指向的内容，只是简单地给出提示，让程序员意识到使用了一个空指针。

我们在自己定义的函数中也可以进行类似的判断，例如：

```
void func(char *p){
    if(p == NULL){
        printf("(null)\n");
```

```
    }else{
        printf("%s\n", p);
    }
}
```

这样能够从很大程度上增加程序的健壮性，防止对空指针进行无意义的操作。

其实，NULL 是在 stdio.h 中定义的一个宏，它的具体内容为：

```
#define NULL ((void *)0)
```

(void *)0 表示把数值 0 强制转换为 void *类型，最外层的( )把宏定义的内容括起来，防止发生歧义。从整体上来看，NULL 指向了地址为 0 的内存，而不是前面说的不指向任何数据。

在进程的虚拟地址空间中，最低地址处有一段内存区域被称为保留区，这个区域不存储有效数据，也不能被用户程序访问，将 NULL 指向这块区域很容易检测到违规指针。

关于虚拟地址空间的概念以及程序的内存分布，我们将在《C 语言和内存》专题中深入讲解，现在读者只需要记住，在大多数操作系统中，极小的地址通常不保存数据，也不允许程序访问，NULL 可以指向这段地址区间中的任何一个地址。

注意，C 语言没有规定 NULL 的指向，只是大部分标准库约定成俗地将 NULL 指向 0，所以不要将 NULL 和 0 等同起来，例如下面的写法是不专业的：

```
int *p = 0;
```

而应该坚持写为：

```
int *p = NULL;
```

注意 NULL 和 NUL 的区别：NULL 表示空指针，是一个宏定义，可以在代码中直接使用。而 NUL 表示字符串的结束标志 '\0'，它是 ASCII 码表中的第 0 个字符。NUL 没

有在 C 语言中定义，仅仅是对 '\0' 的称呼，不能在代码中直接使用。

void 指针

对于空指针 NULL 的宏定义内容，上面只是对((void *)0)作了粗略的介绍，这里重点说一下 void *的含义。void 用在函数定义中可以表示函数没有返回值或者没有形式参数，用在这里表示指针指向的数据的类型是未知的。

也就是说，void *表示一个有效指针，它确实指向实实在在的数据，只是数据的类型尚未确定，在后续使用过程中一般要进行强制类型转换。

C 语言动态内存分配函数 malloc() 的返回值就是 void *类型，在使用时要进行强制类型转换，请看下面的例子：

```
#include <stdio.h>

int main(){
    //分配可以保存 30 个字符的内存，并把返回的指针转换为 char *
    char *str = (char *)malloc(sizeof(char) * 30);
    gets(str);
    printf("%s\n", str);
    return 0;
}
```

运行结果：

c.biancheng.net↙

c.biancheng.net

void *不是空指针的意思，而是实实在在的指针，只是指针指向的内存中不知道保存的是什么类型的数据。