C 语言重点问题回顾

变量

- 左值和右值

字符串

- 字符串的常见操作
- C 字符串的缺陷
- 自行实现字符串函数的误区

指针和内存

- 指针传递的信息
- 如何正确交换两个变量？--值拷贝问题
- 二维数组传参问题
- malloc 与动态数组、动态二维数组
- 函数指针
- C 语言内存没有属性
- 大小端问题
- 结构体的对齐问题
- 什么是内存泄漏？

线程安全

- 同步、互斥
- 生产者消费者问题　使用条件变量的准则
- C 语言实现线程池

左值和右值

左值是指可以位于赋值语句的左边或者右边。

右值，只可以出现在赋值语句的右边。

例如：

```
int i = 5;
const int j = 12;
int &x = i;
```

在上面的代码中，i、x 为左值，j 和 5、12 为右值。注意这里的第二条不是赋值语句！

左值和右值本质区别在哪里？**左值一般用来表明变量的身份，右值则侧重于值本身**。

字符串

字符串常见的操作有 strcat、strlen、strcmp 等，C 风格字符串的缺陷就在于需要自己把握内存的大小。

```
char *p;
strcpy(p, "hello");

char str[3];
strcpy(str, "welcome");
```

上面就是两种错误的做法，第一个没有给 p 分配相应的内存空间，第二个则是内存空间不够。

因为以上的原因，很多同学在自己实现字符串的一系列函数中，就出现了以下错误的做法：

```
char *strcat(char *s1, const char *s2){
    s1 = (char *)malloc(strlen(s1) + strlen(s2));
    //

}

char *strcpy(char *s1, const char *s2){

    s1 = (char *)malloc(strlen(s2));
    //
}
```

我们来说明上述代码的错误，首先**对 s1 进行重新分配内存是无效的**，如果我们这样调用 strcat(p, “hello”)，那么 s1 的任何改动均和 p 无关，因为 C 语言传参数采用的是值拷贝，这里实际上是造成了内存泄露。

其次，**就算这里能够改变** p 指向的位置，这样做更加不允许，因为这样会使得其他地方的字符串失效。当然这里不太可能，不做额外讨论。

指针和内存

指针本身的含义：

1.内存的基地址

2.数据的类型

例如 int *p = malloc(100);

char * s = malloc(100);

很显然，p 和 s 本身的值就是内存基地址的数值，但是 p[3]和 s[3]

的值是否相同呢？显然不是，因为 p 和 s 的类型不同，p 是 int 类型指针，所以 p[3]是把后面这段内存当做 int 数组，s[3]则是看做 char 数组。所以 p++一次增加的数值为 4(32bit)，而 s++增加的为 1.

void*是个例外，它只有基地址，没有类型信息，所以无法解引用。

如何正确交换两个变量？--值拷贝问题

```
void swap(int a, int b){
    int temp = a;
    a = b;
    b = temp;
}
```

这段代码究竟错在哪里？

原因在于 C 语言的参数传递方式为 value 拷贝。函数形参拷贝实参的值后，与实参再无关联。

正确的方式是采用指针。这里记住交换变量的原则，**交换 T 类型的变量，那么 swap 函数的形参就要用 T*类型的参数**。也就是说，交换两个 int *，就要使用 int **才可以达到目的。

二维数组传参问题

有一个 4*3 的二维数组，我想通过参数传递，在一个函数中打印这个数组，应该怎么办？

新手想到数组可以用 int*传递，二维数组是否可以用 int **传递，于是有下列的代码：

```
void print_array(int **a, int m, int n){
    int i, j;
    for(i = 0; i != m; ++i){
        for(j = 0; j != n; ++j){
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}
```

我们尝试着编译，使用的是 a[4][3]，得到下列的错误：

error: cannot convert ‘int (*)[3]’ to ‘int**’ for argument ‘1’ to ‘void print_array(int**, int, int)’

这个错误我们可以这样去理解，一维数组可以看做指针 int *，二维数组也是指针，但是不是 int **，而是一维数组的指针。类型为 int(*)[3]，所以引发了错误。

正确的代码是这样的：

```
void print_array(int (*a)[3], int m){
    int i, j;
    for(i = 0; i != m; ++i){
        for(j = 0; j != 3; ++j){
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}
```

这里大家也能看出这种做法的缺陷，就是我们只能传递第二位为 3 的数组。那么如果才能更通用呢？

我的解决方案是：

1.把数组用一个结构体包装起来。传参数时，传递它的指针。

2.数组用动态内存去分配，这样我们的第一种代码也就是用 int**传递数组是可行的。如何实现动态的二维数组，这就是下面我们要讲述的

内容。

malloc 与动态数组

二维动态数组的构造方式如下：

```
int **a = (int **)malloc(5 * sizeof(int*));
int i;
for(i = 0; i != 5; ++i){
    a[i] = (int *)malloc(4 * sizeof(int));
}
```

仔细考虑这个数组在内存中的模型，我们先生成一个一维数组，每个元素都是一个指针，然后我们依次为每个指针分配一段内存空间。

考虑以下的问题：

1.这个二维数组能否进行 a[3][3]这样的下标运算？

2.这个数组能否使用 memset 进行初始化？

3.这个数组如何使用 free 进行释放？

函数指针

我们过去接触的指针通常是指向变量，但是在 C 语言中，函数也是可以具有指针的。

例如 int *p = &i;

这里是做了两个工作：

1. 我们声明一个 int *类型的指针，这个指针可以指向任何类型的 int

类型变量。

2. 我们用 i 的地址去初始化（**这里是初始化，不是赋值**）p，这样 p 就指向了 i 这个变量。

函数指针也是如此。

对于这样一个函数：

```
void test(int a, int b){
    //
}
```

它的指针是什么类型？

```
void (*) (int , int);
```

这个就是它的指针的类型。

我们使用下列的语句声明一个函数指针变量：

```
void (* pfunc) (int ,int);
```

这里我们声明了一个变量 pfunc，它的类型是 void (*) (int , int)。如何让指针指向一个函数呢？

只需 pfunc = &test 即可。

函数指针的类型看起来比较繁琐，我们尝试用 typedef 进行简化如下：

```
typedef void (* func) (int ,int);
```

这里需要注意的是： **func 是个类型，不是变量**。

C 语言内存本身没有属性

看下面的代码：

```
char *s = (char *)malloc(1000);
int *p = (int *)s;
```

后面 p 是否能当做普通的 int 数组使用？这段内存是一段“char 数组”，我们把它当 int 使用，例如 p[0]、p[1]，会不会引发问题？

再看另一个例子：

```
struct test{
    int a;
    int b;
};

int main(void) {


    int *p = (int *)malloc(2 * sizeof(int));
    p[0] = 29;
    p[1] = 34;
    struct test * pt = (struct test *)p;
    printf("%d\n", pt->a);
    printf("%d\n", pt->b);
}
```

最后结果打印又是多少？ 原因是为什么？

这里我们解释 C 语言的内存，本质上就是一段 01 区域，**本身是没有任何属性的**，没有所谓的类型 int、char、float 等。我们前面提过指针的本质，提供了两个信息，一个是内存基地址，一个是变量的类型。对于下面的代码：

```
int *p1 = malloc(1000);

char *p2 = malloc(1000);
```

这两段做的内存工作是完全一样的，都是向操作系统申请一块大小为

1000 的内存空间。**并不存在说**，第一块内存是 int 类型，第二块是 char 类型。

那么为什么 p1++和 p2++指针变动的数值不一样？**原因是指针的类型不一样，这与他们指向的内存没有任何关系**。

所以我们得出下列的结论：C 语言中内存都是相同的，如何解释他们，依据的是**采用什么指针操控他们**。

再次举例，对于一段 01 交错的内存区域，用 char*指向他们，我们得到的是 0x55，用 int *指向它们，我们得到的是 0x55 55 55 55.

大小端问题

结构体的对齐问题

```
struct test{
    int a;
    char c;
};
```

这个结构体在内存中占据几个字节？

什么是内存泄漏？