# 嵌入式C语言之-

# 指针与动态内存

讲师：叶大鹏



助力你成为优秀的电子工程师！

# 为什么要使用动态内存?

```
typedef struct
{
    uint8_t paramType;
    uint8_t value[100];
    uint32_t size;
} CfgParam;

static CfgParam g_cfgParam;

g_cfgParam.paramType = 1;
g_cfgParam.value[0] = 0x9A;
g_cfgParam.value[1] = 0x99;
g_cfgParam.value[2] = 0x99;
g_cfgParam.value[3] = 0x3F;
g_cfgParam.size = 4;
```

- 在这个例子中，结构体用来管理接收到的网络数据，比如温度校准系数、PM2.5阈值等等，实际数据保存在value数组里；
- 如果只有极少数场景，接收到的网络数据量很大，比如接近100字节；但是绝大部分场景，只有几个字节，此时使用大数组是不合理的，浪费内存空间；
- C语言提供了一种灵活的方案，在程序运行期间可以动态的申请和使用内存。

# 如何申请动态内存?

- **申请动态内存:**

  ```
  #include <stdlib.h>
  void *malloc(unsigned int size)
  ```

- malloc是动态内存分配函数，用于在堆上申请一块指定大小的连续内存区域，**单位是字节**；
- 申请成功返回分配区域的起始地址，分配失败返回NULL（数值为0）；
- 返回结果是void *，需要强制类型转换为自己需要的类型。

| 没有使用 |
| --- |
| 栈，保存函数的参数和局部变量 |
| 堆，使用malloc等分配的动态内存 |
| 未初始化、初始化为0的全局变量、静态全局变量、静态局部变量 |
| 已初始化，并且值为非0的全局变量、静态全局变量、静态局部变量 |

# 如何使用动态内存?

- 使用动态内存:

```
uint8_t *p = (uint8_t *)malloc(4);
```

- 通过malloc申请4个字节的内存空间，使用uint8_t *强制类型转换，表示可以使用uint8_t类型去解释和管理这段内存空间;

```
p[0] = 0x01;
p[1] = 0x02;
p[2] = 0x03;
p[3] = 0x04;
```

# 注意事项

- **申请动态内存存在失败的概率：**

```
uint8_t *p = (uint8_t *)malloc(5000);
```

| p | 0x00000000 |
|---|---|

- **如果堆上可用的内存空间不足时，会申请失败，返回NULL，但是在编译阶段是发现不了这种问题的；**

startup_ARMCM4.s

```
;<h> Heap Configuration
;   <o> Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>
;</h>

Heap_Size       EQU     0x00000C00
```

# 注意事项

- 申请完动态内存，一定要判断是否申请成功：

```
uint8_t *p = (uint8_t *)malloc(5000);
if (p == NULL)
{
    printf("malloc for p failed!\n");
    return -1;
}
```

申请失败如何处理，
取决于程序的业务逻辑

# 如何释放动态内存?

- 释放动态内存:

```
uint8_t *p = (uint8_t *)malloc(4);
...
free(p);
```

- 需要通过free()主动释放申请的内存空间，free原型:

```
void free(void  *ptr);
```

# 如何释放动态内存?

- 不释放有什么后果:

```
uint8_t *p = (uint8_t *)malloc(3000);
if (p == NULL)
{
        printf("malloc for p failed!\n");
        return -1;
}
p[0] = 0x01;
p[1] = 0x02;
uint8_t *m = (uint8_t *)malloc(3000);
if (m == NULL)
{
        printf("malloc for m failed!\n");
        return -1;
}
```

```
malloc for m failed!
```

# 如何释放动态内存?

- 使用free()释放动态内存，在底层库会给这段内存空间打上标签，表示可以被再次申请和分配：

```
uint8_t *p = (uint8_t *)malloc(3000);
if (p == NULL)
{
        printf("malloc for p failed!\n");
        return -1;
}
p[0] = 0x01;
p[1] = 0x02;
p[2] = 0x03;
p[3] = 0x04;
free(p);
uint8_t *m = (uint8_t *)malloc(3000);
```

| | |
|---|---|
| p | 0x20000088 |
| m | 0x20000088 |

# 如何释放动态内存?

- 只使用free()就够了吗?

```
uint8_t *p = (uint8_t *)malloc(4);
if (p == NULL)
{
    printf("malloc for p failed!\n");
    return -1;
}
p[0] = 0x01;
p[1] = 0x02;
p[2] = 0x03;
p[3] = 0x04;
free(p);
```

```
uint8_t *m = (uint8_t *)malloc(4);
if (m == NULL)
{
    printf("malloc for m failed!\n");
    return -1;
}
m[0] = 0x05;
m[1] = 0x06;
m[2] = 0x07;
m[3] = 0x08;
p[0] = 0x01;
p[1] = 0x02;
p[2] = 0x03;
p[3] = 0x04;
```

# 如何释放动态内存?

- 只使用free()就够了吗?

```
uint8_t *m = (uint8_t *)malloc(4);
if (m == NULL)
{
    printf("malloc for m failed!\n");
    return -1;
}
m[0] = 0x05;
m[1] = 0x06;
m[2] = 0x07;
m[3] = 0x08;
p[0] = 0x01;
p[1] = 0x02;
p[2] = 0x03;
p[3] = 0x04;
```

| | |
|---|---|
| p | 0x20000088 |
| m | 0x20000088 |

```
0x20000088: 01 02 03 04
```

# 如何释放动态内存?

- free()后，还需要将指针变量赋值为NULL;

```
uint8_t *p = (uint8_t *)malloc(4);
if (p == NULL)
{
    printf("malloc for p failed!\n");
    return -1;
}
p[0] = 0x01;
p[1] = 0x02;
p[2] = 0x03;
p[3] = 0x04;
free(p);
p = NULL;
```

- ✓ free(p)针对的是内存空间，将其打标签表示可以再分配;
- ✓ p = NULL针对的是p本身，将它保存的地址值设置为0，否则它被称为“野指针”。

# 野指针

- 野指针，表示指针变量保存的地址值是非法的:

```
void Test(void)
{
    uint8_t *p ;
    if (p == NULL)
    {
        printf("malloc for p failed!\n");
        return;
    }
    p[0] = 0x01;
    p[1] = 0x02;
    p[2] = 0x03;
    p[3] = 0x04;
    free(p);
    p = NULL;
}
```

# 如何避免野指针

- 定义指针变量时，如果不能马上赋值，初始化为NULL；
- 针对动态内存，free()后，需要将指针变量赋值为NULL；

# 动态内存的生命周期

```
uint8_t *MemTest(void)
{
    uint8_t *p = (uint8_t *)malloc(4);
    if (p == NULL)
    {
        printf("malloc for p failed!\n");
        return NULL;
    }
    return p;
}
```

```
int main(void)
{
    uint8_t *m = MemTest();
    if (m == NULL)
    {
        printf("malloc for m failed!\n");
        return -1;
    }
    m[0] = 0x05;
    m[1] = 0x06;
    free(m);
    m = NULL;
    return 0;
}
```

- 申请的动态内存空间不会随着函数结束而被主动释放掉，所以在函数外部可以使用，同样使用完需要free()和设置为NULL。

# THANK YOU!