@TOC

在C语言里，我们对于一块内存空间的使用往往并不关心其内存是否有浪费或不足，但是对于空间的需求，不仅仅是上述的情况。有时候我们需要的空间大小在程序运行的时候才能知道，那数组的编译时开辟空间的方式就不能满足了。这时候就只能动态存开辟内存空间。

C语言主要提供了如下几种动态内存开辟的函数：

- malloc
- calloc
- realloc
- free

其中free对于内存空间的释放起着关键性作用，这对于内存控制极为关键。

`动态内存开辟实在堆区开辟的。`

 malloc

函数定义如下：

```
void* malloc (size_t size);
```

这个函数主要功能就是向内存申请一块连续的内存空间，并返回指向该空间起始位置的指针。

1. 如果开辟成功，则返回一个指向开辟好空间的指针。
2. 如果开辟失败，则返回一个NULL指针，因此malloc的返回值一定要做检查。
3. 返回值的类型是 void* ，malloc函数并不知道开辟空间的类型，在使用时使用者自己来决定。
4. 如果参数 size 为 0，malloc的行为是标准是未定义的，取决于编译器。

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>

void* malloc (size_t size);
int main() {
    //返回值指向该分配的内存的起始地址
    int* p = (int*)malloc(40);  //INT_MAX
    if (p == NULL) {
        printf("%s\n", strerror(errno));
        return 1;
    }

    //释放
    //void free (void* ptr);
    free(p);
    p = NULL;
    return 0;
}
```

显然在这里要注意不再使用这块内存时要使用free函数，C语言提供了函数free，专门是用来做动态内存的释放和回收的，函数原型如下：

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

- free函数用来释放动态开辟的内存。
- 如果参数 ptr 指向的空间不是动态开辟的，那free函数的行为是未定义的。
- 如果参数 ptr 是NULL指针，则函数什么事都不做。

`malloc和free都声明在 stdlib.h 头文件中`

---

calloc

calloc 函数也用来动态内存分配

```
void* calloc (size_t num, size_t size);
```

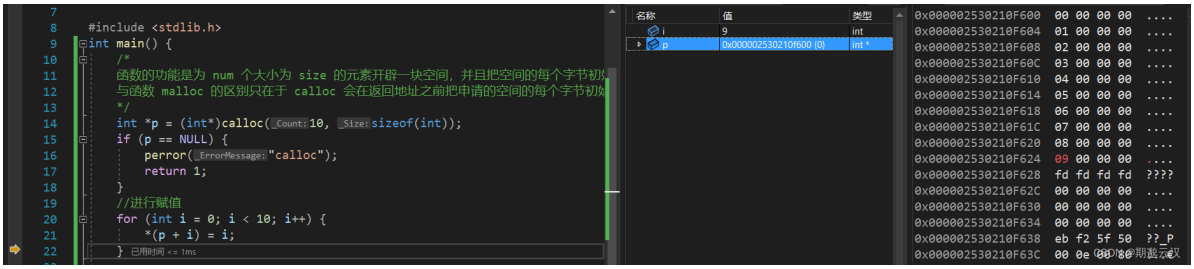
- 函数的功能是为 num 个大小为 size 的元素开辟一块空间，并且把空间的每个字节初始化为0。
- 与函数 malloc 的区别只在于 calloc 会在返回地址之前把申请的空间的每个字节初始化为全0。

```
#include <stdlib.h>
int main() {
    /*
    函数的功能是为 num 个大小为 size 的元素开辟一块空间，并且把空间的每个字节初始化为0。
    与函数 malloc 的区别只在于 calloc 会在返回地址之前把申请的空间的每个字节初始化为全0。
    */
    int *p = (int*)calloc(10, sizeof(int));
    if (p == NULL) {
        perror("calloc");
        return 1;
    }
    for (int i = 0; i < 10; i++) {
        *(p + i) = i;
    }

    free(p);
    p = NULL;
    return 0;
}
```

我们在调试窗口试着看一下程序：

显然我们看到，在14行对内存空间初始化后，在内存监视窗口确实有40个字节的空间被初始化为0，接着我们走下面的for循环进行赋值



显然，随着 i 的变化，内存空间里10个数也成功赋值。

---

realloc

`realloc`函数的出现让动态内存管理更加灵活

```
void* realloc (void* ptr, size_t size);
```

有时会我们发现过去申请的空间太小了，有时候我们又会觉得申请的空间过大了，那为了合理的时候内存，我们一定会对内存的大小做灵活的调整。

那 **realloc 函数就可以做到对动态开辟内存大小的调整。**

- ptr 是要调整的内存地址
- size 表示调整之后新大小
- 返回值为调整之后的内存起始位置
- 这个函数调整原内存空间大小的基础上，还会将原来内存中的数据移动到新的空间

那么这里问题就来了，在虚拟内存地址中，假设ptr指针指向的原始空间有20个字节的大小，现在我们需要对其扩容，`realloc（ptr，40）`，那么在原始20个字节空间之后的内存是否足够再开辟20个字节，也就是原有空间之后没有足够多的空间时应当怎样？这里就有两种情况：

- 情况1

  当原有空间之后有足够大的空间 时候，要扩展内存就直接原有内存之后直接追加空间，原来空间的数据不发生变化。

- 情况2

  当原有空间之后没有足够大的空间的时候，扩展的方法是：**在堆空间上另找一个合适大小的连续空间来使用。这样函数返回的是一个新的内存地址。**

```
# define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#include <stdlib.h>

/*
    对内存的大小做灵活的调整
    void* realloc(void*ptr, size_t size)

    ptr 是要调整的内存地址
    size 调整之后新大小
    返回值为调整之后的内存起始位置
*/

int main()
{
    int* ptr = (int*)malloc(100);
```

```
    if (ptr != NULL)
    {
        //业务处理
    }
    else
    {
        exit(EXIT_FAILURE);
    }
    //扩展容量

    ptr = (int*)realloc(ptr, 1000);

    int* p = NULL;
    p = realloc(ptr, 1000);
    if (p != NULL)
    {
        ptr = p;
    }

    free(ptr);
    return 0;
}
```