第六章 动态内存申请

1.1 动态分配内存的概述

在数组一章中，介绍过数组的长度是预先定义好的，在整个程序中**固定不变**，但是在实际的编程中，往往会发生这种情况，即所需的**内存空间取决于实际输入的数据**，而无法预先确定 。为了解决上述问题，C 语言提供了一些**内存管理函数**，这些内存管理函数可以按需要**动态的分配**内存空间，也可把不再使用的空间回收再次利用。

1.2 静态分配、动态分配

静态分配

1、 在程序编译或运行过程中，按事先规定大小分配内存空间的分配方式。int a [10]
2、 必须事先知道所需空间的大小。
3、 分配在栈区或全局变量区，一般以数组的形式。
4、 按计划分配。

动态分配

1、在程序运行过程中，根据需要大小自由分配所需空间。
2、按需分配。
3、分配在堆区，一般使用特定的函数进行分配。

1.3 动态分配函数

1、malloc 函数

头文件：#include<stdlib.h>
函数原型： void *malloc(unsigned int size);
功能说明：
在内存的动态存储区(堆区)中分配一块长度为 size 字节的连续区域，用来存放类型说明符指定的类型。函数原型返回 void*指针，使用时必须做相应的强制类型转换 ，分配的内存空间内容不确定，一般使用 memset 初始化。
返回值：
分配空间的起始地址 (分配成功)
NULL (分配失败)
注意
1、在调用 malloc 之后，一定要判断一下，是否申请内存成功。
2、如果多次 malloc 申请的内存，第 1 次和第 2 次申请的内存不一定是连续的

例 1：

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

```
int main()
{
    int count,*array,n;
    printf("请输入您要申请的数组元素个数\n");
    scanf("%d",&n);
    array=(int *)malloc(n*sizeof(int));
    if(array==NULL)
    {
        printf("申请内存失败\n");
        return 0;
    }
    memset(array,0,n*sizeof(int));
    for(count=0;count<n;count++)
    {
        array[count]=count;
    }
    for(count=0;count<n;count++)
    {
        printf("%d\n",array[count]);
    }
    free(array);//释放 array 指向的内存
    return 0;
}
```

2、 free 函数（释放内存函数）

头文件：#include<stdlib.h>

函数定义:void free(void *ptr)

函数说明：free 函数释放 ptr 指向的内存。

注意：ptr 指向的内存必须是 malloc calloc relloc 动态申请的内存

例 2：

```
char *p=(char *)malloc(100);
free(p);//
```

注意

(1)、free 后，因为没有给 p 赋值，所以 p 还是指向原先动态申请的内存。但是内存已经不能再用了，p 变成野指针了。

(2)、一块动态申请的内存只能 free 一次，不能多次 free

3、 calloc 函数

头文件：#include<stdlib.h>

函数定义：void * calloc(size_t　nmemb,size_t　size);

size_t 实际是无符号整型，它是在头文件中，用 typedef 定义出来的。

函数的功能：在内存的堆中，申请 nmemb 块，每块的大小为 size 个字节的连续区域
函数的返回值：
返回 申请的内存的首地址（申请成功）
返回 NULL（申请失败）

注意：
malloc 和 calloc 函数都是用来申请内存的。
区别：

1) 函数的名字不一样
2) 参数的个数不一样
3) malloc 申请的内存，内存中存放的内容是随机的，不确定的，而 calloc 函数申请的内存中的内容为 0

例 3：调用方法

```
char *p=(char *)calloc(3,100);
```

在堆中申请了 3 块，每块大小为 100 个字节，即 300 个字节连续的区域。

4、 realloc 函数（重新申请内存）

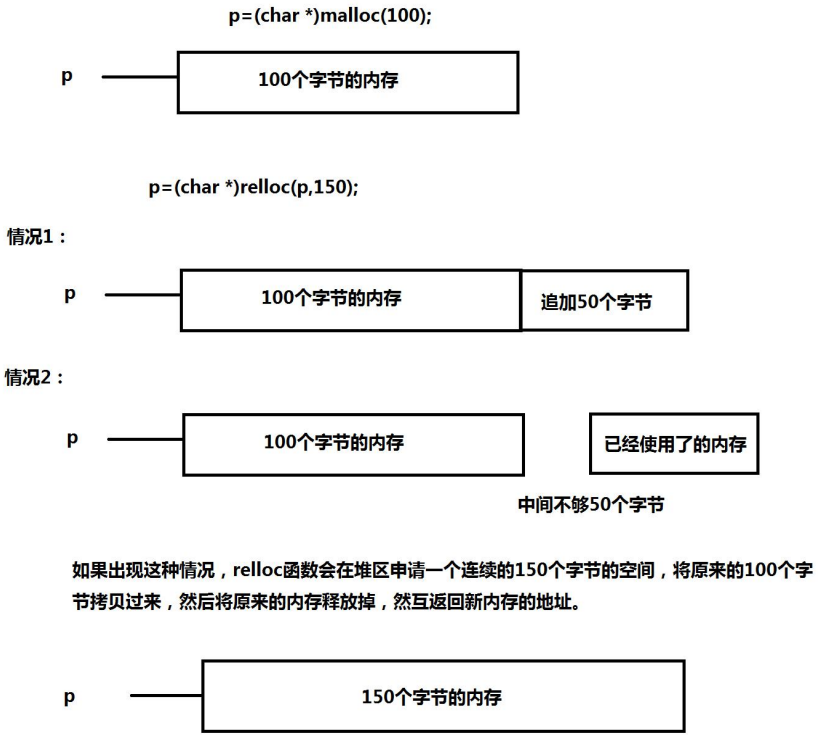
咱们调用 malloc 和 calloc 函数单次申请的内存是连续的，两次申请的两块内存不一定连续。
有些时候有这种需求，即我先用 malloc 或者 calloc 申请了一块内存，我还想在原先内存的基础上挨着申请内存。
或者我开始时候使用 malloc 或 calloc 申请了一块内存，我想释放后边的一部分内存。
为了解决这个问题，发明了 realloc 这个函数

头文件#include<stdlib.h>
函数的定义：void* realloc(void *s,unsigned int newsize);
函数的功能：
　　在原先 s 指向的内存基础上重新申请内存，新的内存的大小为 new_size 个字节，
如果原先内存后面有足够大的空间，就追加，如果后边的内存不够用，则 relloc 函数会在堆区
找一个 newsize 个字节大小的内存申请，将原先内存中的内容拷贝过来，然后释放原先的内存，最后返回
新内存的地址。

如果 newsize 比原先的内存小，则会释放原先内存的后面的存储空间，只留前面的 newsize 个字节
返回值：新申请的内存的首地址

例 4：

```
char *p;
p=(char *)malloc(100)
//咱们想在 100 个字节后面追加 50 个字节
p=(char *)realloc(p,150);//p 指向的内存的新的大小为 150 个字节
```



例 5:

```
char *p;
p=(char *)malloc(100)
//咱们想重新申请内存,新的大小为 50 个字节
p=(char *)realloc(p,50);//p 指向的内存的新的大小为 50 个字节,100 个字节的后 50 个字节的存储空间就被释放了
```

注意:malloc calloc relloc 动态申请的内存，只有在 free 或程序结束的时候才释放。

1.4 内存泄露

内存泄露的概念：

申请的内存，首地址丢了，找不了，再也没法使用了，也没法释放了，这块内存就被泄露了。

内存泄露 例 1：

```
int main()
{
    char *p;
    p=(char *)malloc(100);
    //接下来，可以用 p 指向的内存了

    p="hello world";//p 指向别的地方了
```

```
    //从此以后，再也找不到你申请的 100 个字节了。则动态申请的 100 个字节就被泄露了

    return 0;
}
```

内存泄露 例 2：

```
void fun()
{
    char *p;
    p=(char *)malloc(100);
    //接下来，可以用 p 指向的内存了
    …
}

int main()
{
    //每调用一次 fun 泄露 100 个字节
    fun();
    fun();
    return 0;
}
```

内存泄露 解决方案 1：

```
void fun()
{
    char *p;
    p=(char *)malloc(100);
    //接下来，可以用 p 指向的内存了
    ;
    ;
    free(p);
}

int main()
{
    fun();
    fun();
    return 0;
```

```
}
```

内存泄露 解决方案 2：

```
char * fun()
{
	char *p;
	p=(char *)malloc(100);
	//接下来，可以用 p 指向的内存了
	;
	;
	return p;
}

int main()
{
	char *q;
	q=fun();
	//可以通过 q 使用 ，动态申请的 100 个字节的内存了

	//记得释放
	free(q);

	return 0;
}
```

总结：申请的内存，一定不要把首地址给丢了，在不用的时候一定要释放内存。