<https://www.cnblogs.com/xslbk/p/7363897.html>

什么是动态内存的申请和释放？

     当程序运行到需要一个动态分配的变量时，必须向系统申请取得堆中的一块所需大小的存储空间，用于存储该变量。当不再使用该变量时，也就是它的生命结束时，要显式释放它所占用的存储空间，这样系统就能对该堆空间进行再次分配，做到重复使用有限的资源。

下面将介绍动态内存申请和释放的函数

**1.malloc函数**

在C语言中，使用malloc函数来申请内存。函数原型如下：

#include<stdlib.h>

void \*malloc(size\_t size);

参数size代表需要动态申请的内存的字节数，若内存申请成功，函数返回申请到的内存的起始地址，若申请失败，返回NULL， 在使用该函数时应注意以下几点

1.只关心申请内存的大小，该函数的参数很简单，只有申请内存的大小，单位是字节

2.申请的是一块连续的内存，该函数一定是申请一块连续的区间，可能申请到内存比实际申请的大，但也有可能申请不到，若申请失败，则返回NULL

3.返回值类型是void\*，函数的返回值是void\*，不是某种具体类型的指针，可以理解成该函数只是申请内存，对在内存中存储什么类型的数据，没有要求，因此，返回值是void\*，实际编程中，根据实际情况将void\*转换成需要的指针类型

4.显示初始化，注意：堆区是不会自动在分配时做初始化的（包括清零），所以程序中需要显示的初始化

**2.free 函数**

在堆区上分配的内存，需要用free函数显示释放。函数原型如下：

#include <stdlib.h>

void free(void \*ptr);

函数的参数ptr，指的是需要释放的内存的起始地址。该函数没有返回值。使用该函数，也有下面几点需要注意：

（1）必须提供内存的起始地址。调用该函数时，必须提供内存的起始地址，不能提供部分地址，释放内存中的一部分是不允许的。因此，必须保存好malloc返回的指针值，若丢失，则所分配的堆空间无法回收，称内存泄漏。

（2）malloc和free配对使用。编译器不负责动态内存的释放，需要程序员显示释放。因此，malloc与free是配对使用的，避免内存泄漏。

示例程序如下：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int \*get\_memory(int n){

    int \*p, i;

    if ((p = (int \*)malloc(n \* sizeof(int))) == NULL)    {

        printf("malloc error\n");

        return p;

    }

    memset(p, 0, n \* sizeof(int));

    for (i = 0; i < n; i++)

        p[i] = i+1;

       return p;

}

int main(){

    int n, \*p, i;

    printf("input n:");

    scanf("%d", &n);

    if ((p = get\_memory(n)) == NULL){

        return 0;

    }

    for (i = 0; i < n; i++){

           printf("%d ", p[i]);

    }

    printf("\n");

    free(p);

    p = NULL;

    return 0;

}

程序执行结果如下：

linux@ubuntu:~/book/ch10$ cc malloc.c -Wall

linux@ubuntu:~/book/ch10$./a.out

input n:10

1 2 3 4 5 6 7 8 9 10

该程序演示了动态内存的标准用法。动态内存的申请，通过一个指针函数来完成。内存申请时，判断是否申请成功，成功后，对内存初始化。在主调函数中，动态内存依然可以访问，不再访问内存时，用free函数释放。

（3）不允许重复释放。同一空间的重复释放也是危险的，因为该空间可能已另分配。在上面程序中，如果释放堆空间两次（连续调用两次free(p)），会出现下面的结果。

linux@ubuntu:~/book/ch10$ cc malloc.c –Wall

linux@ubuntu:~/book/ch10$./a.out

input n:1

1

\*\*\* glibc detected \*\*\* ./a.out: double free or corruption (fasttop): 0x08f1a008 \*\*\*

======= Backtrace: =========

/lib/libc.so.6(+0x6c501)[0x687501]

/lib/libc.so.6(+0x6dd70)[0x688d70]

/lib/libc.so.6(cfree+0x6d)[0x68be5d]

./a.out[0x804861e]

/lib/libc.so.6(\_\_libc\_start\_main+0xe7)[0x631ce7]

./a.out[0x8048471]

======= Memory map: ========

0061b000-00772000 r-xp 00000000 08:01 1048623    /lib/libc-2.12.1.so

00772000-00773000 ---p 00157000 08:01 1048623   /lib/libc-2.12.1.so

00773000-00775000 r--p 00157000 08:01 1048623    /lib/libc-2.12.1.so

00775000-00776000 rw-p 00159000 08:01 1048623   /lib/libc-2.12.1.so

00776000-00779000 rw-p 00000000 00:00 0

008e1000-008fb000 r-xp 00000000 08:01 1048657   /lib/libgcc\_s.so.1

008fb000-008fc000 r--p 00019000 08:01 1048657    /lib/libgcc\_s.so.1

008fc000-008fd000 rw-p 0001a000 08:01 1048657   /lib/libgcc\_s.so.1

00a8f000-00aab000 r-xp 00000000 08:01 1048599   /lib/ld-2.12.1.so

00aab000-00aac000 r--p 0001b000 08:01 1048599 /lib/ld-2.12.1.so

00aac000-00aad000 rw-p 0001c000 08:01 1048599   /lib/ld-2.12.1.so

00b6c000-00b6d000 r-xp 00000000 00:00 0          [vdso]

08048000-08049000 r-xp 00000000 08:01 1079938    /home/linux/book/ch10/a.out

08049000-0804a000 r--p 00000000 08:01 1079938    /home/linux/book/ch10/a.out

0804a000-0804b000 rw-p 00001000 08:01 1079938   /home/linux/book/ch10/a.out

08f1a000-08f3b000 rw-p 00000000 00:00 0          [heap]

b7700000-b7721000 rw-p 00000000 00:00 0

b7721000-b7800000 ---p 00000000 00:00 0

b7815000-b7816000 rw-p 00000000 00:00 0

b7823000-b7827000 rw-p 00000000 00:00 0

bf9a5000-bf9c6000 rw-p 00000000 00:00 0           [stack]

Aborted

（4）free只能释放堆空间。像代码区、全局变量与静态变量区、栈区上的变量，都不需要程序员显示释放，这些区域上的空间，不能通过free函数来释放，否则执行时，会出错。

示例程序如下：

#include <stdlib.h>

int main(){

    int a[10] = {0};

    free(a);

    return 0;

}

程序执行结果如下：

linux@ubuntu:~/book/ch10$ cc free.c –o free -Wall

free.c: In function 'main':

free.c:7: warning: attempt to free a non-heap object 'a'

可以看到有一个警告，即释放一个非堆上的空间。如果强行执行程序，会出现下面的结果：

linux@ubuntu:~/book/ch10$./a.out

Segmentation fault

**3.野指针**

野指针指的是指向“垃圾”内存的指针，不是NULL指针。出现“野指针”主要有以下原因：

（1）指针变量没有被初始化。指针变量和其它的变量一样，若没有初始化，值是不确定的。也就是说，没有初始化的指针，指向的是垃圾内存，非常危险。

示例程序如下：

#include <stdio.h>

int main(){

    int \*p;

    printf("%d\n", \*p);

    \*p = 10;

    printf("%d\n", \*p);

    return 0;

}

程序执行结果如下：

linux@ubuntu:~/book/ch10$ cc p.c –o p -Wall

linux@ubuntu:~/book/ch10$./p

1416572

Segmentation fault

（2）指针p被free之后，没有置为NULL。free函数是把指针所指向的内存释放掉，使内存成为了自由内存。但是，该函数并没有把指针本身的内容清楚。指针仍指向已经释放的动态内存，这是很危险。程序员稍有疏忽，会误以为是个合法的指针。就有可能再通过指针去访问动态内存。实际上，这时的内存已经是垃圾内存了，关于野指针会造成什么样的后果，这是很难估计的。若内存仍然是空闲的，可能程序暂时正常运行；若内存被再次分配，又通过野指针对内存进行了写操作，则原有的合法数据，会被覆盖，这时，野指针造成的影响将是无法估计的。

示例程序如下：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(){

    int n = 5, \*p, i;

    if ((p = (int \*)malloc(n \* sizeof(int))) == NULL) {

        printf("malloc error\n");

        return 0;

    }

    memset(p, 0, n \* sizeof(int));

    for (i = 0; i < n; i++) {

        p[i] = i+1;

        printf("%d ", p[i]);

    }

    printf("\n");

    printf("p=%p \*p=%d\n", p, \*p);

    free(p);

    printf("after free:p=%p \*p=%d\n", p, \*p);

    \*p = 100;

    printf("p=%p \*p=%d\n", p, \*p);

    return 0;

}

程序执行结果如下：

linux@ubuntu:~/book/ch10$cc test.c –o test -Wall

linux@ubuntu:~/book/ch10$./test

1 2 3 4 5

p=0x92cf008 \*p=1

after free:p=0x92cf008 \*p=0

p=0x92cf008 \*p=100

该程序中，故意在执行了“free(p)”之后，通过野指针p对动态内存进行了读写，程序正常执行，也在预料之中。前面已经分析过，内存释放后，若继续访问甚至修改，后果是不可预料的。

（3）指针操作超越了变量的作用范围。指针操作时，由于逻辑上的错误，导致指针访问了非法内存，这种情况让人防不胜防，只能依靠程序员好的编码风格，已及扎实的基本功。下面演示一个指针操作越界的情况：

示例程序如下：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(){

    int a[5] = {1, 9, 6, 2, 10}, \*p, i, n;

    n = sizeof(a) / sizeof(n);

    p = a;

    for (i = 0; i <= n; i++) {

       printf("%d ", \*p);

        p++;

    }

    printf("\n");

    \*p = 100;

    printf("\*p=%d\n", \*p);

    return 0;

}

程序执行结果如下：

linux@ubuntu:~/book/ch10$ cc test.c –o test -Wall

linux@ubuntu:~/book/ch10$./test

1 9 6 2 10 5

\*p=100

该程序故意出了两个错误，一是for循环的条件“i <= n”，p指针指向了数组以外的空间。二是“\*p = 100”，对非法内存进行了写操作。

（4）不要返回指向栈内存的指针。指针函数会返回一个指针。在主调函数中，往往会通过返回的指针，继续访问指向的内存。因此，指针函数不能返回栈内存的起始地址，因为栈内存在函数结束时会被释放。