malloc

原型：extern void *malloc(unsigned int num_bytes);

用法：#include <alloc.h>

功能：分配长度为 num_bytes 字节的内存块

说明：如果分配成功则返回指向被分配内存的指针，否则返回空指针 NULL。
当内存不再使用时，应使用 free()函数将内存块释放。

举例：

```
// malloc.c

#include <syslib.h>
#include <alloc.h>

main()
{
  char *p;

  clrscr();        // clear screen

  p=(char *)malloc(100);
  if(p)
    printf("Memory Allocated at: %x",p);
  else
    printf("Not Enough Memory!\n");

  free(p);

  getchar();
  return 0;
}
```

malloc 函数的返回值是一个 void 类型的指针，参数为 int 类型数据，即申请分配的内存大小，单位是 byte。内存分配成功之后，malloc 函数返回这块内存的首地址。你需要一个指针来接收这个地址。但是由于函数的返回值是 void *类型的，所以必须强制转换成你所接收的类型。也就是说，这块内存将要用来存储什么类型的数据。比如：

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

在堆上分配了 100 个字节内存，返回这块内存的首地址，把地址强制转换成 char *类型后赋给 char *类型的指针变量 p。同时告诉我们这块内存将用来存储 char 类型的数据。也就是说你只能通过指针变量 p 来操作这块内存。这块内存本身并没有名字，对它的访问是匿名访问。

上面就是使用 malloc 函数成功分配一块内存的过程。但是，每次你都能分配成功吗？不一定。上面的对话，皇帝让户部侍郎查询是否还有足够的良田未被分配出去。使用 malloc 函数同样要注意这点：如果所申请的内存块大于目前堆上剩余内存块（整块），则内存分配会失败，函数返回 NULL。注意这里说的“堆上剩余内存块”不是所有剩余内存块之和，因为 malloc 函数申请的是连续的一块内存。

既然 malloc 函数申请内存有不成功的可能，那我们在使用指向这块内存的指针时，必须用 if（NULL ！=p）语句来验证内存确实分配成功了。

free

原型：extern void free(void *p);

用法：#include <alloc.h>

功能：释放指针 p 所指向的的内存空间。

说明：p 所指向的内存空间必须是用 calloc,malloc,realloc 所分配的内存。

如果 p 为 NULL 或指向不存在的内存块则不做任何操作。

举例：

```
// free.c

#include <syslib.h>
#include <alloc.h>

main()
{
  char *p;

  clrscr();        // clear screen
  textmode(0x00);

  p=(char *)malloc(100);
  if(p)
    printf("Memory Allocated at: %x",p);
  else
    printf("Not Enough Memory!\n");

  getchar();
  free(p);         // release memory to reuse it

  p=(char *)calloc(100,1);
  if(p)
    printf("Memory Reallocated at: %x",p);
  else
    printf("Not Enough Memory!\n");
```

```
    free(p);          // release memory at program end

    getchar();
    return 0;
}
```

既然有分配，那就必须有释放。不然的话，有限的内存总会用光，而没有释放的内存却在空闲。与 malloc 对应的就是 free 函数了。free 函数只有一个参数，就是所要释放的内存块的首地址。比如上例：

```
free(p);
```

free 函数看上去挺狠的，但它到底作了什么呢？其实它就做了一件事：斩断指针变量与这块内存的关系。比如上面的例子，我们可以说 malloc 函数分配的内存块是属于 p 的，因为我们对这块内存的访问都需要通过 p 来进行。free 函数就是把这块内存和 p 之间的所有关系斩断。从此 p 和那块内存之间再无瓜葛。至于指针变量 p 本身保存的地址并没有改变，但是它对这个地址处的那块内存却已经没有所有权了。那块被释放的内存里面保存的值也没有改变，只是再也没有办法使用了。

这就是 free 函数的功能。按照上面的分析，如果对 p 连续两次以上使用 free 函数，肯定会发生错误。因为第一使用 free 函数时，p 所属的内存已经被释放，第二次使用时已经无内存可释放了。关于这点，我上课时让学生记住的是：**一定要一夫一妻制，不然肯定出错**。

malloc 两次只 free 一次会内存泄漏；malloc 一次 free 两次肯定会出错。也就是说，在程序中 malloc 的使用次数一定要和 free 相等，否则必有错误。这种错误主要发生在循环使用 malloc 函数时，往往把 malloc 和 free 次数弄错了。这里留个练习：

写两个函数，一个生成链表，一个释放链表。两个函数的参数都只使用一个表头指针。

既然使用 free 函数之后指针变量 p 本身保存的地址并没有改变，那我们就需要重新把 p 的值变为 NULL：

```
p = NULL;
```

这个 NULL 就是我们前面所说的“栓野狗的链子”。如果你不栓起来迟早会出问题的。比如：在 free（p）之后，你用 if（NULL ！= p）这样的校验语句还能起作用吗？

例如：

```
char *p = (char *) malloc(100);
strcpy(p, “hello”);
free(p);        /* p 所指的内存被释放，但是 p 所指的地址仍然不变 */
...
if (NULL != p)
{
    /* 没有起到防错作用 */
    strcpy(p, “world”);    /* 出错 */
}
```

释放完块内存之后，没有把指针置 NULL，这个指针就成为了“野指针”，也有书叫“悬垂指针”。这是很危险的，而且也是经常出错的地方。所以一定要记住一条：free 完之后，一定要给指针置 NULL。

同时留一个问题：对 NULL 指针连续 free 多次会出错吗?为什么？如果让你来设计 free 函数，你会怎么处理这个问题？

calloc

原型：extern void *calloc(int num_elems, int elem_size);

用法：#include <alloc.h>

功能：为具有 num_elems 个长度为 elem_size 元素的数组分配内存

说明：如果分配成功则返回指向被分配内存的指针，否则返回空指针 NULL。
当内存不再使用时，应使用 free()函数将内存块释放。

举例：

```
// calloc.c

#include <syslib.h>
#include <alloc.h>

main()
{
  char *p;

  clrscr();        // clear screen

  p=(char *)calloc(100,sizeof(char));
  if(p)
    printf("Memory Allocated at: %x",p);
  else
    printf("Not Enough Memory!\n");

  free(p);

  getchar();
  return 0;
}
```

calloc()函数还有一个特性：它把块中的所有位都设置为0（注意，在某些硬件系统中，不是把所有位都设置为0来表示浮点值0）。

free()函数也可用于释放calloc()分配的内存。

realloc

原型：extern void *realloc(void *mem_address, unsigned int newsize);

用法：#include <alloc.h>

功能：改变 mem_address 所指内存区域的大小为 newsize 长度。

说明：如果重新分配成功则返回指向被分配内存的指针，否则返回空指针 NULL。

当内存不再使用时，应使用 free()函数将内存块释放。

举例：

```
// realloc.c

#include <syslib.h>
#include <alloc.h>
main()
{
  char *p;

  clrscr();        // clear screen

  p=(char *)malloc(100);
  if(p)
    printf("Memory Allocated at: %x",p);
  else
    printf("Not Enough Memory!\n");

  getchar();

  p=(char *)realloc(p,256);
  if(p)
    printf("Memory Reallocated at: %x",p);
  else
    printf("Not Enough Memory!\n");
  free(p);

  getchar();
  return 0;
}
```