隐式空闲链表：

static void *find_fit(size_t asize)：

```
static void *find_fit(size_t asize)
{

    /* 待补全 */
        char *bp;
        for (bp = heap_listp; GET_SIZE(HDRP(bp)) > 0; bp = NEXT_BLKP(bp)) {
                if (!GET_ALLOC(HDRP(bp)) && (asize <= GET_SIZE(HDRP(bp)))) {
                        return bp;
                }
        }
        return NULL;
}
```

此函数用于寻找合适的空闲块，从头开始找，所找的块应该是未分配，且块大小需要大于等于所需要的大小。块的大小及是否已被分配存在头部、脚部，所以利用 GET_SIZE(HDRP(bp))就可以获得块的大小。用 GET_ALLOC(HDRP(bp))就可以知道块是否已被分配。利用 NEXT_BLKP(bp)即可获得 bp 所指向块的相邻下一块的地址。如果当前存在的空闲块无合适的，就返回 NULL。(无合适块就会扩展空间)

static void place (void *bp, size_t asize)：

```
static void place(void *bp, size_t asize)
{
    const size_t total_size = GET_SIZE(HDRP(bp));
    size_t rest = total_size - asize;

    if (rest >= MIN_BLK_SIZE)
    /* need split */
    {
        /* 待补全 */
                PUT(HDRP(bp), PACK(asize, 1));
                PUT(FTRP(bp), PACK(asize, 1));
                PUT(HDRP(NEXT_BLKP(bp)), PACK(rest, 0));
                PUT(FTRP(NEXT_BLKP(bp)), PACK(rest, 0));
    }
    else
    {
        /* 待补全 */
                PUT(HDRP(bp), PACK(total_size, 1));
                PUT(FTRP(bp), PACK(total_size, 1));
    }
}
```

此函数用于判断找到的空闲块大小是否大于请求的内存大小，。如

果找到的块的大小减去所需块的大小大于等于 4 个字节的大小（独立空闲块所需最小的大小），就进行切割，否则不切割（即使有内部碎片）。

若要切割，将块切割成所需要的块和空闲块。在所需要的块头部、脚步记录相关信息。在空闲块头部、脚部也记录相关信息。

若不切断，在当前块头部、脚部记录相关信息即可。

static void *coalesce(void *bp)：

```
else
{
    /* 待补全 */
            size += GET_SIZE(FTRP(PREV_BLKP(bp))) + GET_SIZE(HDRP(NEXT_BLKP(bp)));
            PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
            PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
            bp = PREV_BLKP(bp);
}
```

此函数用于合并空闲块。前三种情况已给出。只需要完善最后一种情况。(前后相邻块均为空闲块，三块合为一块)

大小等于三块大小相加。然后在前一块的头部录入合并的新块的相关信息，在后一块脚部也录入相关信息。

然后将指针指向新块，即前一块即可。

显式空闲链表：

显示空闲链表与隐式最大的区别就是，专门给出了空闲链表，在寻找空闲块式的效率大大提高。

find_fit()：

```
static void *find_fit(size_t asize)
{
     char *bp = free_listp;
    if (free_listp == NULL)
        return NULL;

    while (bp != NULL) /*not end block;*/
    {  if(GET_SIZE(HDRP(bp))>=asize)
            return bp;
            bp=(void *)GET_SUCC(bp);
        /* ??? */
    }
    return (bp != NULL ? ((void *)bp) : NULL);
}
```

此函数与隐式时同理，用于寻找合适的空闲块。此时只需要从空闲链表里寻找合适的即可。将 bp 指向空闲链表第一个。如果空闲链表为空（无空闲块）或无合适块返回 NULL。

利用 GET_SUCC()找到当前空闲块的后继空闲块。

place()：

```
static void place(void *bp, size_t asize)
{
    size_t total_size = 0;
    size_t rest = 0;
    size_t prev=GET_PREV_ALLOC(HDRP(bp));

    delete_from_free_list(bp);
    /*remember notify next_blk, i am alloced*/
    total_size = GET_SIZE(HDRP(bp));
    rest = total_size - asize;

    if (rest >= MIN_BLK_SIZE) /*need split*/
    {
            PUT(HDRP(bp),PACK(asize,prev,1));
            PUT(HDRP(NEXT_BLKP(bp)),PACK(rest,1,0));
            PUT(FTRP(NEXT_BLKP(bp)),PACK(rest,1,0));
            add_to_free_list(NEXT_BLKP(bp));
        /* ??? */
    }
    else
    {    PUT(HDRP(bp),PACK(total_size,prev,1));
         PUT(HDRP(NEXT_BLKP(bp)),PACK(GET_SIZE(HDRP(NEXT_BLKP(bp))),1,1));
        /* ??? */
    }
}
```

此函数用于切割找到的空闲块。先将找到块从空闲链表删除，然后判断其大小减去所需大小是否大于等于独立空闲块的最小大小。

若需要切割，先在块头部录入相关信息。此时额外需要录入前一相邻块是否被分配，GET_PREV_ALLOC()得到即可（分配块无脚部，不需要录入）。然后在切割得到的空闲块头部、脚部录入相关信息，并将其加入空闲链表。

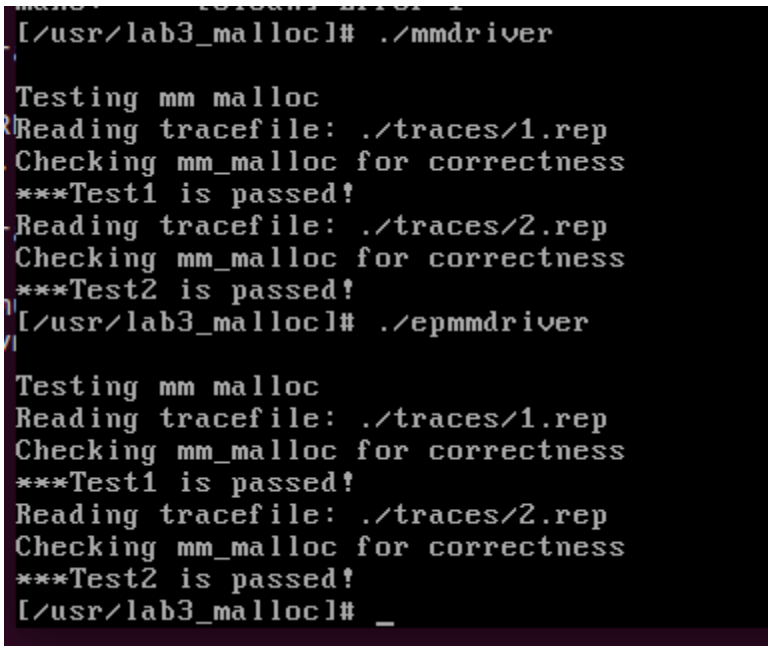
若不需要切割，录入头部，并告知相邻的下一块，前块已被分配。

Makefile:

```
#
# Students' Makefile for the Malloc Lab
#

CC = gcc -g
CFLAGS = -Wall

# 待补充
OBJS1 = mmdriver.o mm.o memlib.o
OBJS2 = mmdriver.o ep_mm.o memlib.o
all: mmdriver epmmdriver

mmdriver: $(OBJS1)
	$(CC) $(CFLAGS) -o mmdriver $(OBJS1)

epmmdriver: $(OBJS2)
	$(CC) $(CFLAGS) -o epmmdriver $(OBJS2)

#待补充
mmdriver.o: memlib.h mm.h
memlib.o: memlib.h config.h
mm.o: mm.h memlib.h
ep_mm.o: mm.h memlib.h

clean:
	rm -f *~ *.o mmdriver epmmdriver
```

根据相关文件的依赖关系补全 makefile。

编译测试结果：


[/usr/lab3_malloc]# ./mmdriver
Testing mm malloc
Reading tracefile: ./traces/1.rep
Checking mm_malloc for correctness
***Test1 is passed!
Reading tracefile: ./traces/2.rep
Checking mm_malloc for correctness
***Test2 is passed!
[/usr/lab3_malloc]# ./epmmdriver
Testing mm malloc
Reading tracefile: ./traces/1.rep
Checking mm_malloc for correctness
***Test1 is passed!
Reading tracefile: ./traces/2.rep
Checking mm_malloc for correctness
***Test2 is passed!
[/usr/lab3_malloc]# _