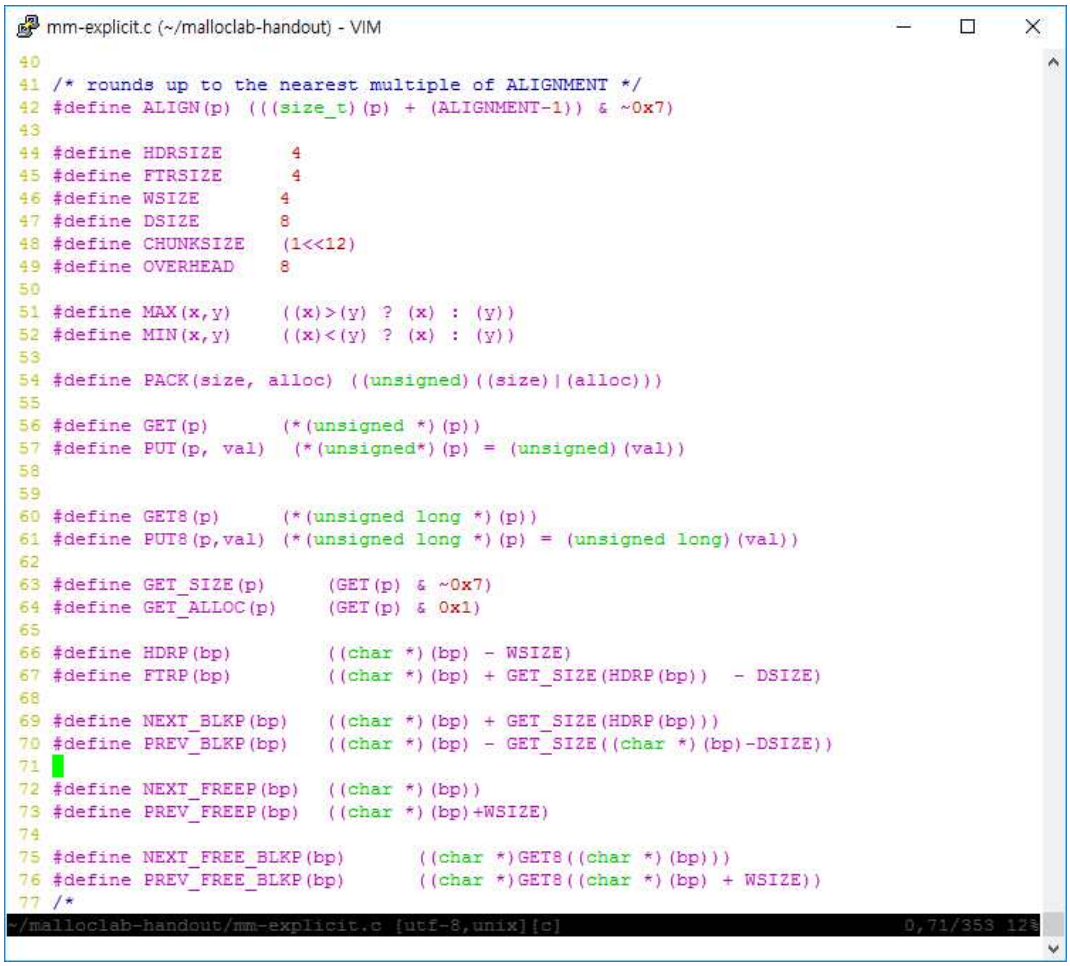
2016 시스템 프로그래밍

- malloc lab2 -

제출일자	2016.12.12
분 반	02
이 름	신종욱
학 번	2013024223



1. 매크로 설명

#define PACK(size, alloc) ((unsigned)((size)|(alloc)))

size만큼 alloc(1,0)시킨다 숫자에따라 free가될 수도 사용블럭이 될수도있따

#define GET(p) (*(unsigned *)(p))

p의 주소의 값을 구한다

#define PUT(p, val) (*(unsigned*)(p) = (unsigned)(val))

p에 val을 저장한다

#define GET8(p) (*(unsigned long *)(p))

#define PUT8(p,val) (*(unsigned long *)(p) = (unsigned long)(val))

두 개다 usigned long이다 아마 사이즈를 8단위로 하여 주소를 관리하기 위함이다

#define GET_SIZE(p) (GET(p) & ~0x7) : 사이즈를 리턴

#define GET_ALLOC(p) (GET(p) & 0x1) : 할당유무를 리턴

#define HDRP(bp) ((char *)(bp) - WSIZE) : 헤더를 리턴

#define FTRP(bp) ((char *)(bp) + GET_SIZE(HDRP(bp)) - DSIZE) : 풋터를 리턴

#define NEXT_BLKP(bp) ((char *)(bp) + GET_SIZE(HDRP(bp))) : 다음 블록을 리턴

#define PREV_BLKP(bp) ((char *)(bp) - GET_SIZE((char *)(bp)-DSIZE)) : 이전 블록을 리턴

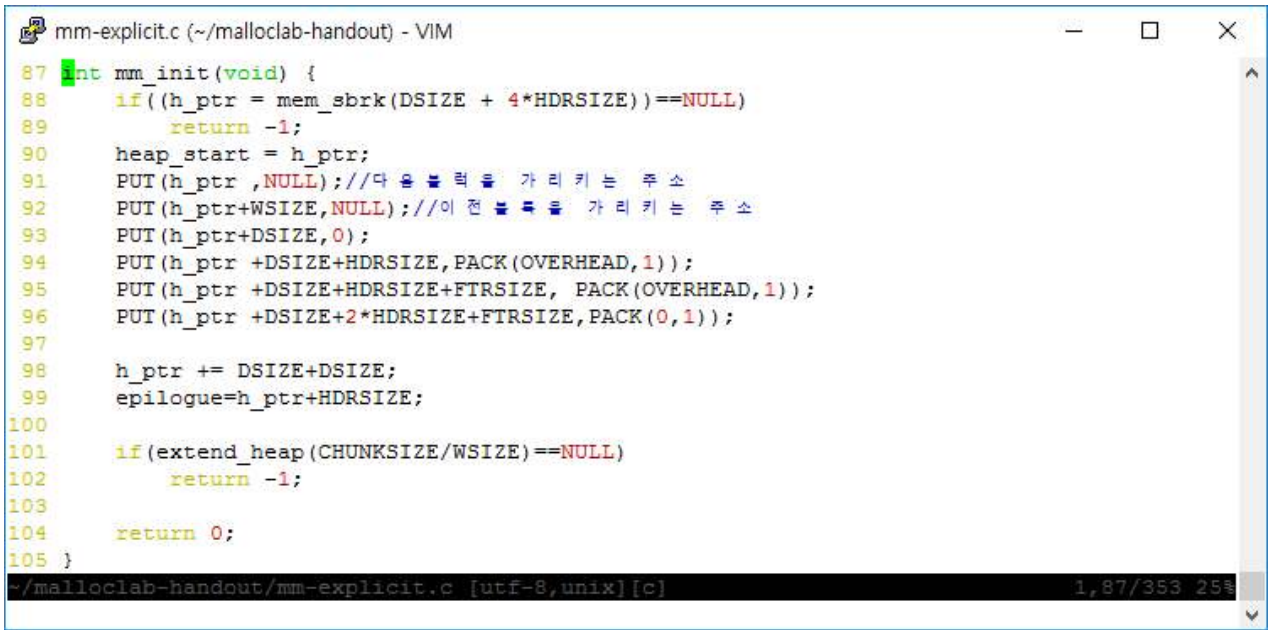
#define NEXT_FREEP(bp) ((char *)(bp))//다음 free한 블록을 리턴

#define PREV_FREEP(bp) ((char *)(bp)+WSIZE)//이전 free한 블록을 리턴

#define NEXT_FREE_BLKP(bp) ((char *)GET8((char *)(bp)))//다음 free한 블록을 리턴

#define PREV_FREE_BLKP(bp) ((char *)GET((char *)(bp) + WSIZE))//이전 free한 블록을 리턴

-init함수



pdf와 동일하게 작성하였다 그전과 다른점은 heap에 이전과 다음블럭을 가리키는 주소를 가진채 초기화를 하였다

그 외 헤더와 푸터를 만드는 과정은 implicit와 같다

-malloc함수

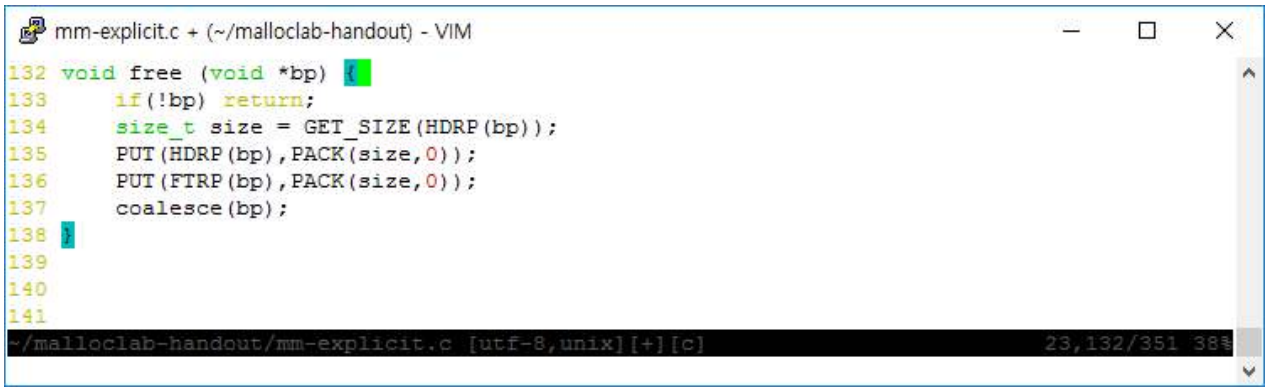
일단 요청 size가 0이하일 경우와 size가 8보다 작을경우와 클경우로 나누어서 판단하였다

만약 8보다 클경우에는 8의배수로 올림하여서 asize를 바꾸면된다

find_fit으로 적절한 블록이있는지 찾은후 place를 이용해 할당한다

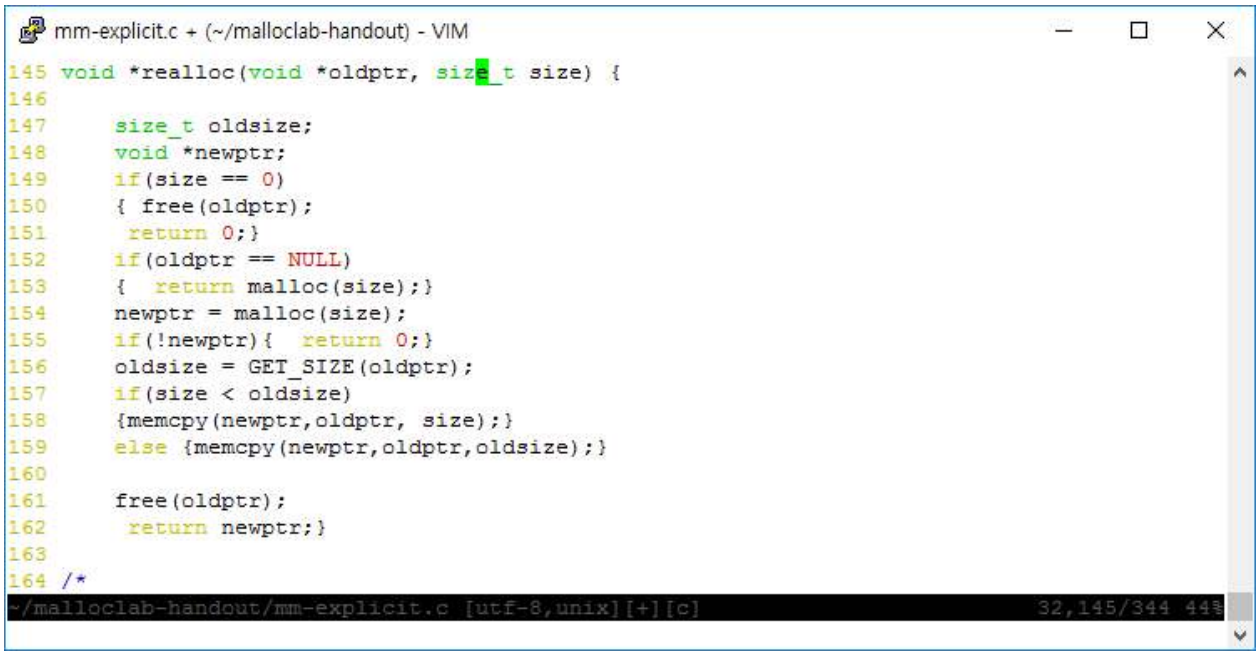
만약 없을경우에는 힙을 늘린다

-free 함수



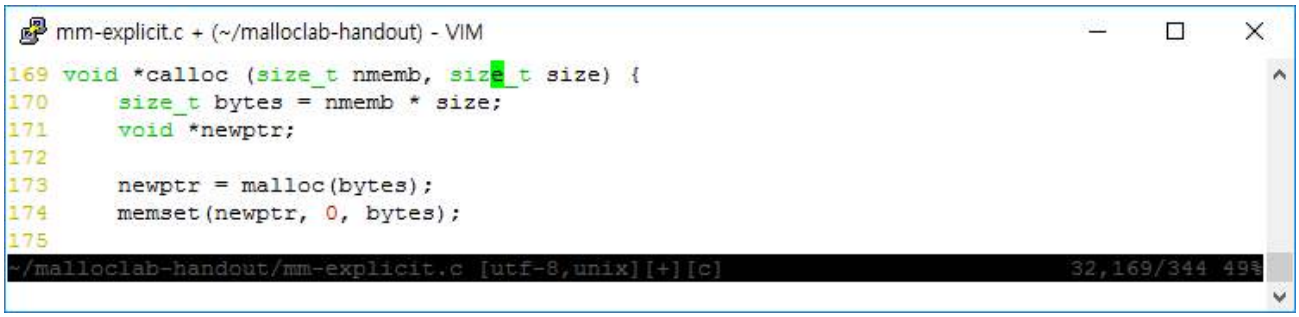
free는 이전과같이 현재 사이즈를 구한후 pack 매크로를 이용하여 할당을 0으로 바꾼후 해당 bp를 coalesce하여 정리해준다

-realloc 함수



재 할당할 때 쓰는 함수로서 새로 할당받은 사이즈에 따라 새로운 블록을 만들고 사이즈가 원래의 사이즈보다 작으면 원래의 사이즈 보다 큰부분에 있는 데이터는 복사가되어도 쓸모가없으므로 새로운 사이즈의 크기만큼만 복사한다 더 크다면 다복사한다

-calloc



calloc는 nmemb*size 만큼의 메모리를 확보하고,초기화 한 후, 시작주소를 반환해주는 함수이다.
size_t 타입의 변수를 만들어 nmemb*size 의 값을 넣고, 이 변수를 이용하여 malloc() 호출해 할당한뒤, memset을 통해 0으로 초기화 시킨다.

-extend_heap 함수

mm-explicit.c (~/malloclab-handout) - VIM

```
178 inline  void *extend_heap(size_t words){
179     unsigned *old_epilogue;
180     char *bp;
181     unsigned size;
182
183     size = (words%2) ? (words+1)*WSIZE : words*WSIZE;
184     if((long)(bp = mem_sbrk(size))<0)
185     {return NULL;}
186
187     old_epilogue = epilogue;
188     epilogue = bp+size-HDRSIZE;
189
190     PUT(HDRP(bp), PACK(size,0));
191     PUT(FTRP(bp), PACK(size, 0));
192     PUT(epilogue, PACK(0,1));
193     return coalesce(bp);
194 }
~/malloclab-handout/mm-explicit.c [utf-8,unix][c]                    1,194/344 54%
```

pdf파일 그대로 사용하였습니다

size를의 빈블록을 만들어주는 함수로서 만약 malloc을 할 때 기존공간을 활용하여서 블록에 넣을수없을 때 extend_heap을 이용해 블록을 만드는 것이다.

epilogue의 계산과정이 추가되었다

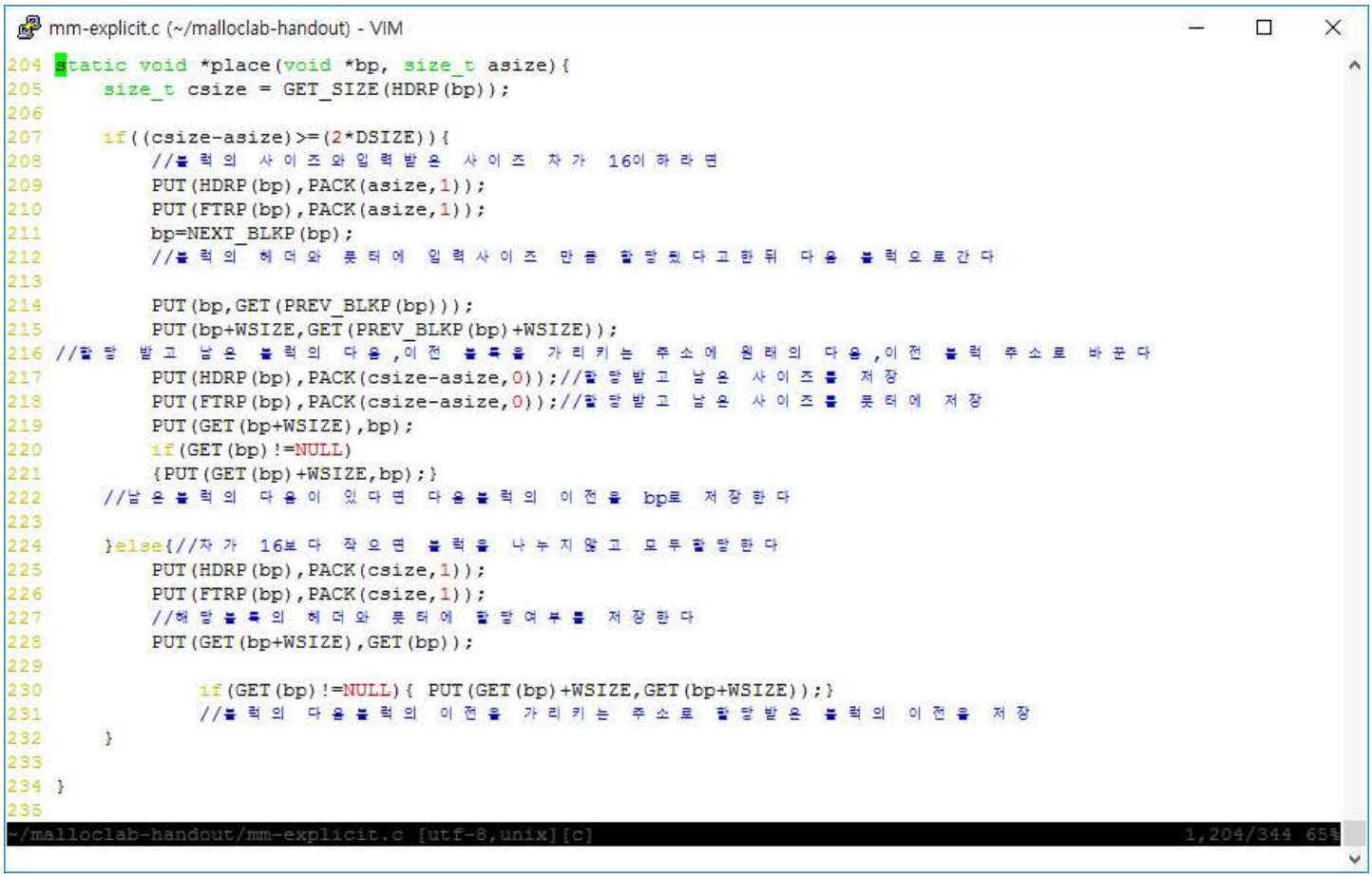
-find_fit

mm-explicit.c + (~/malloclab-handout) - VIM

```
194 }
195 static void *find_fit(size_t asize){
196     char *bp;
197     for(bp=heap_start;bp!=NULL;bp=((char *)GET(NEXT_FREEP(bp)))){
198
199         if(asize <= GET_SIZE(HDRP(bp)))
200         {return bp; }//free리 스 트 를   보 면 서 알 맞 은  크 기 를  찾 는 다
201 }
202     return NULL;
203 }
204
~/malloclab-handout/mm-explicit.c [utf-8,unix][+][c]                 2,194/345 57%
```

heap_Start는 free를 가리키는 루트이다 free리스트를 뒤지면서 사이즈가 asize보다 큰 free한 블록을 찾을때까지 찾는다

-place 함수



malloc에서 요청된 사이즈만큼의 크기를 가진 적당한 블록을 찾아 place 함수로 보내는데
여기서 블록과 사이즈에 따라 낭비되는 부분이 많아 나누어 할당할지 그냥할지 사이즈에 맞게 블록을 할당한다 단 explicit에서는 서로 앞뒤로 가리키는 포인터가있음으로 생각하고 구현하면된다

-coalesce함수


```
static void *coalesce(void *bp){
    size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
    size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
    size_t size = GET_SIZE(HDRP(bp));

    if(prev_alloc && next_alloc){//앞뒤 블럭이 모두 할당될경우
    if(GET(heap_start)==NULL)//만약 아직 freelist가 비었다면
    { PUT(heap_start,bp);//root를 새롭게 지정하고
      PUT(bp,NULL);//bp의 next는 NULL로 지정
      PUT(bp+WSIZE,heap_start);//bp의 이전블록은 root를 저장
    }else{
        PUT(bp,GET(heap_start));//bp의 next는 루트의 next를 받고
        PUT(bp+WSIZE,heap_start);//bp의 이전은 루트를 받는다
        if(GET(bp)!=NULL)
            {PUT(GET(bp)+WSIZE,bp);}//다음블록의 이전블록은 bp로 지정한다
        PUT(heap_start,bp);//루트의 next는 bp를 지정한다
    }}
    else if (prev_alloc && !next_alloc){//뒤블럭이 free일경우
        PUT(GET(NEXT_BLKP(bp)+WSIZE),GET(NEXT_BLKP(bp)));
        //프리된 블럭의 다음다음블럭의 이전은 다음블럭으로 지정한다
        if(GET(NEXT_BLKP(bp))!=NULL)
        {PUT(GET(NEXT_BLKP(bp))+WSIZE,GET(NEXT_BLKP(bp)+WSIZE));}
        //다음블럭이 존재한다면 다음다음블러의 이전블럭을 다음블록의 이전블록으로 한다
        PUT(bp,GET(heap_start));//bp의 다음블록은 원래 root가 가리키던걸 받는다
        PUT(bp+WSIZE,heap_start);//bp의 이전블럭은 root를 가리킨다
        if(GET(bp)!=NULL) {PUT(GET(bp)+WSIZE,bp);}
        //다음블럭이 존재하면 다음블럭의 이전주소에 프리된블럭을 지정
        PUT(heap_start,bp);//root의 다음블록에는 bp를 지정한다
        size+= GET_SIZE(HDRP(NEXT_BLKP(bp)));
        PUT(HDRP(bp),PACK(size,0));
```

free할 때 합칠 때 앞뒤를 보고 합쳐야하는데 그때 사용되는 함수이다

-case 1 : 첫 번째 경우에는 앞 뒤블럭이 모두 사용중 일 때이다.

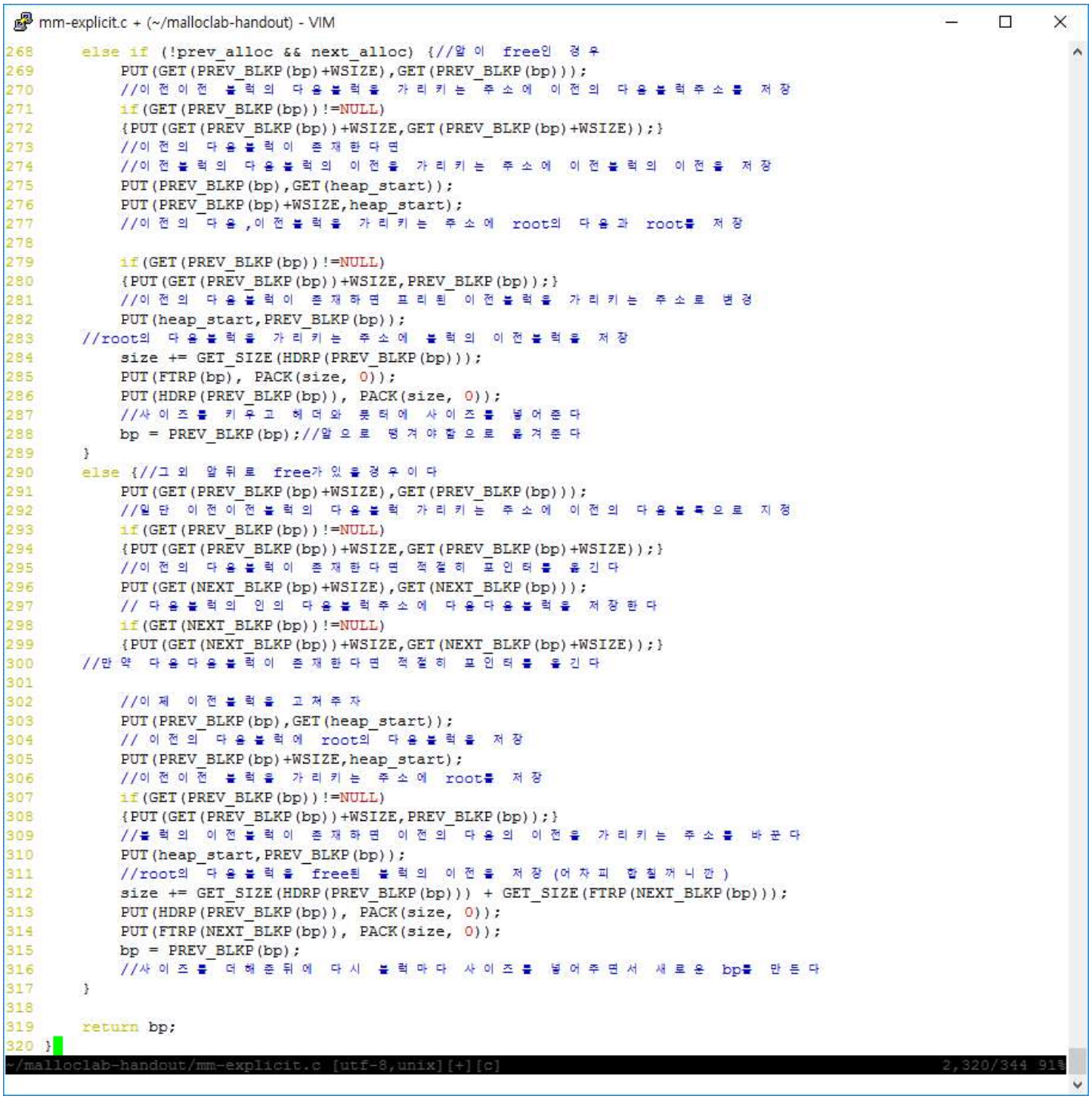
이럴때는 FIFO정책에 맞게 이전 heap_start가 가지고있는 next를 bp의 next로 이어주고

heap_start의 next를 bp로 지정해준다 그리고 만약 bp의 next가 NULL이 아니라면 bp의 next의 이전을 bp로 선언해준다

-case 2 : 두 번째 경우에는 뒷블럭이 free일 때이다 뒷블럭이 free이면 뒷뒷블럭까지 포인터를 생각해줘야한다

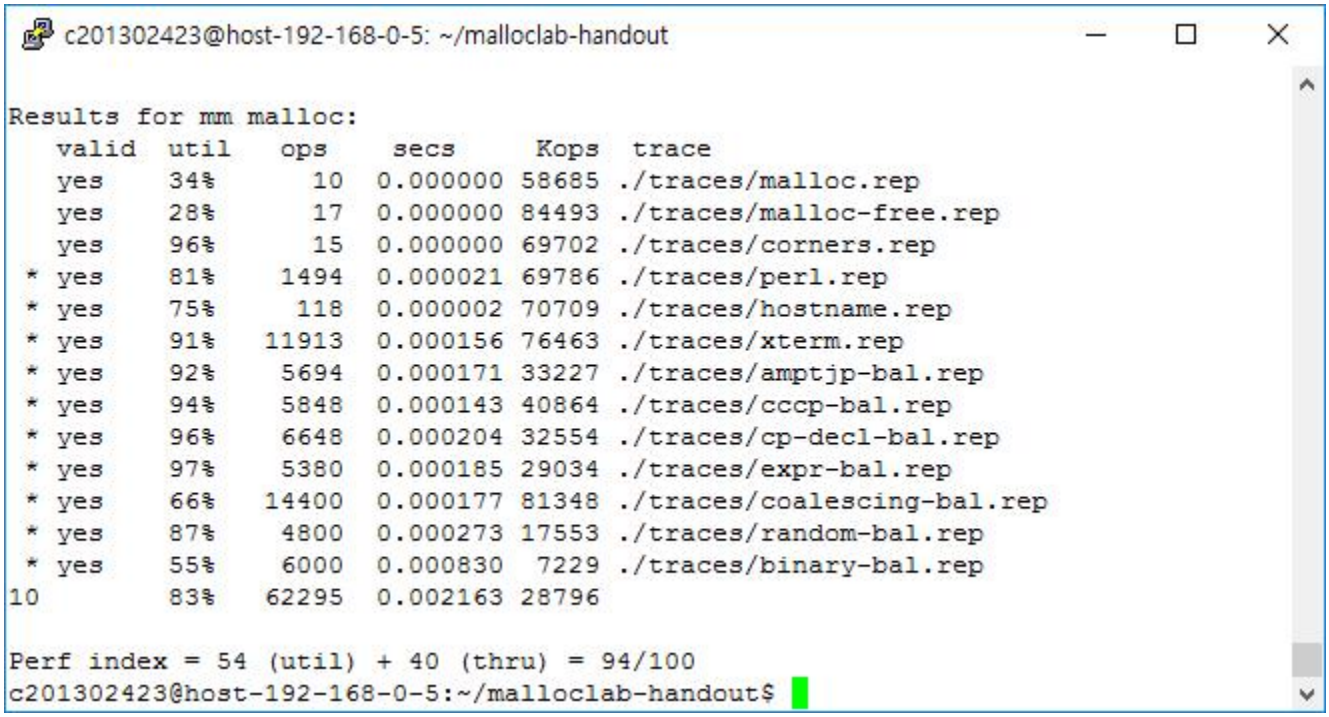
다음다음블럭의 이전을 다음블럭으로 지정하고

그리고 앞에서의 case 1과 같이 FIFO정책에 맞게 Linked List수정법으로 heap_start와 bp의 포인터들을 수정한다



-case 3 : 앞부분이 free일경우에는 앞서한 경우와 다른점이 있다
바로 bp포인터가 같이 합쳐진 프리의중 앞블럭으로 이동해야한다
그래서 PUT(HDRP(PREV_BLKP(bp)),PACK(size,0))을 이용하였고 또 해준후에
bp = PREV_BLKP(bp)를 해주었다.
-case 4: case 2와 case 3이 동시에 일어 났을경우이다 즉 앞 뒤가 모두 프리 블록 일경우이다.
이럴때에는 PUT(HDRP(PREV_BLKP(bp)),PACK(size,0)) bp = PREV_BLKP(bp)도 해주고
case2의 경우에도 첨가 하면 된다
(코드마다 설명은 주석첨가하였습니다)

실행결과


c201302423@host-192-168-0-5: ~/malloclab-handout
Results for mm malloc:
valid util ops secs Kops trace
yes 34% 10 0.000000 58685 ./traces/malloc.rep
yes 28% 17 0.000000 84493 ./traces/malloc-free.rep
yes 96% 15 0.000000 69702 ./traces/corners.rep
* yes 81% 1494 0.000021 69786 ./traces/perl.rep
* yes 75% 118 0.000002 70709 ./traces/hostname.rep
* yes 91% 11913 0.000156 76463 ./traces/xterm.rep
* yes 92% 5694 0.000171 33227 ./traces/amptjp-bal.rep
* yes 94% 5848 0.000143 40864 ./traces/cccp-bal.rep
* yes 96% 6648 0.000204 32554 ./traces/cp-decl-bal.rep
* yes 97% 5380 0.000185 29034 ./traces/expr-bal.rep
* yes 66% 14400 0.000177 81348 ./traces/coalescing-bal.rep
* yes 87% 4800 0.000273 17553 ./traces/random-bal.rep
* yes 55% 6000 0.000830 7229 ./traces/binary-bal.rep
10 83% 62295 0.002163 28796
Perf index = 54 (util) + 40 (thru) = 94/100
c201302423@host-192-168-0-5:~/malloclab-handout$