

**2016 시스템 프로그래밍**  
**- malloc lab2 -**

제출일자	2016.12.12
분 반	02
이 름	신종욱
학 번	2013024223

```
mm-explicit.c (~/.malloclab-handout) - VIM
40
41 /* rounds up to the nearest multiple of ALIGNMENT */
42 #define ALIGN(p) (((size_t)(p) + (ALIGNMENT-1)) & ~0x7)
43
44 #define HDRSIZE 4
45 #define FTRSIZE 4
46 #define WSIZE 4
47 #define DSIZE 8
48 #define CHUNKSIZE (1<<12)
49 #define OVERHEAD 8
50
51 #define MAX(x,y) ((x)>(y) ? (x) : (y))
52 #define MIN(x,y) ((x)<(y) ? (x) : (y))
53
54 #define PACK(size, alloc) ((unsigned)((size)|(alloc)))
55
56 #define GET(p) (*(unsigned *) (p))
57 #define PUT(p, val) (*(unsigned*) (p) = (unsigned) (val))
58
59
60 #define GET8(p) (*(unsigned long *) (p))
61 #define PUT8(p, val) (*(unsigned long *) (p) = (unsigned long) (val))
62
63 #define GET_SIZE(p) (GET(p) & ~0x7)
64 #define GET_ALLOC(p) (GET(p) & 0x1)
65
66 #define HDRP(bp) ((char *) (bp) - WSIZE)
67 #define FTRP(bp) ((char *) (bp) + GET_SIZE(HDRP(bp)) - DSIZE)
68
69 #define NEXT_BLKP(bp) ((char *) (bp) + GET_SIZE(HDRP(bp)))
70 #define PREV_BLKP(bp) ((char *) (bp) - GET_SIZE((char *) (bp)-DSIZE))
71
72 #define NEXT_FREEP(bp) ((char *) (bp))
73 #define PREV_FREEP(bp) ((char *) (bp)+WSIZE)
74
75 #define NEXT_FREE_BLKP(bp) ((char *)GET8((char *) (bp)))
76 #define PREV_FREE_BLKP(bp) ((char *)GET8((char *) (bp) + WSIZE))
77 /*
~/malloclab-handout/mm-explicit.c [utf-8,unix][c] 0,71/353 12%
```

## 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 unsigned 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 \*)GET8((char \*) (bp) + WSIZE))//이전 free한 블록을 리턴

## -init함수

```
mm-explicit.c (~/.malloclab-handout) - VIM
87 int mm_init(void) {
88     if((h_ptr = mem_sbrk(DSIZE + 4*HDRSIZE)) == NULL)
89         return -1;
90     heap_start = h_ptr;
91     PUT(h_ptr, NULL); //다음 블록을 가리키는 주소
92     PUT(h_ptr+WSIZE, NULL); //이전 블록을 가리키는 주소
93     PUT(h_ptr+DSIZE, 0);
94     PUT(h_ptr + DSIZE+HDRSIZE, PACK(OVERHEAD, 1));
95     PUT(h_ptr + DSIZE+HDRSIZE+FTRSIZE, PACK(OVERHEAD, 1));
96     PUT(h_ptr + DSIZE+2*HDRSIZE+FTRSIZE, PACK(0, 1));
97
98     h_ptr += DSIZE+DSIZE;
99     epilogue=h_ptr+HDRSIZE;
100
101     if(extend_heap(CHUNKSIZE/WSIZE) == NULL)
102         return -1;
103
104     return 0;
105 }
```

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

그 외 헤더와 푸터를 만드는 과정은 implicit와 같다

## -malloc함수

```
mm-explicit.c + (~/.malloclab-handout) - VIM
110 void *malloc (size_t size) {
111     char *bp;
112     unsigned asize;
113     unsigned extendsize;
114     if(size<=0) return NULL;
115     if(size<=DSIZE) {asize=2*DSIZE;}
116     else asize=DSIZE*((size+(DSIZE)+(DSIZE-1))/DSIZE);
117
118     if((bp = find_fit(asize)) != NULL)
119     {
120         place(bp, asize);
121         return bp;
122     }
123     extendsize = MAX(asize, CHUNKSIZE);
124     if((bp = extend_heap(extendsize/WSIZE)) == NULL)
125         return NULL;
126     place(bp, asize);
127     return bp;
128 }
```

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

```
mm-explicit.c + (~/.malloclab-handout) - VIM
132 void free (void *bp) {
133     if(!bp) return;
134     size_t size = GET_SIZE(HDRP(bp));
135     PUT(HDRP(bp),PACK(size,0));
136     PUT(FTRP(bp),PACK(size,0));
137     coalesce(bp);
138 }
139
140
141
~/malloclab-handout/mm-explicit.c [utf-8,unix][+][c] 23,132/351 38%
```

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

## -realloc 함수

```
mm-explicit.c + (~/.malloclab-handout) - VIM
145 void *realloc(void *oldptr, size_t size) {
146
147     size_t oldsize;
148     void *newptr;
149     if(size == 0)
150     { free(oldptr);
151       return 0;}
152     if(oldptr == NULL)
153     { return malloc(size);}
154     newptr = malloc(size);
155     if(!newptr){ return 0;}
156     oldsize = GET_SIZE(oldptr);
157     if(size < oldsize)
158     {memcpy(newptr,oldptr, size);}
159     else {memcpy(newptr,oldptr,oldsize);}
160
161     free(oldptr);
162     return newptr;}
163
164 /*
~/malloclab-handout/mm-explicit.c [utf-8,unix][+][c] 32,145/344 44%
```

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

```
mm-explicit.c + (~/.malloclab-handout) - VIM
169 void *calloc (size_t nmemb, size_t size) {
170     size_t bytes = nmemb * size;
171     void *newptr;
172
173     newptr = malloc(bytes);
174     memset(newptr, 0, bytes);
175
~/malloclab-handout/mm-explicit.c [utf-8,unix][+][c] 32,169/344 49%
```

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 }
```

pdf파일 그대로 사용하였습니다

size를의 빈블록을 만들어주는 함수로서 만약 malloc을 할 때 기존공간을 활용하여서 블록에 넣을수 없을 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; }//free리스트를 보면서 알맞은 크기를 찾는다
201     }
202     return NULL;
203 }
204 }
```

heap\_start는 free를 가리키는 루트이다 free리스트를 뒤지면서 사이즈가 asize보다 큰 free한 블록을 찾을때까지 찾는다



## -place 함수

```
mm-explicit.c (~/.malloclab-handout) - VIM
204 static void *place(void *bp, size_t asize){
205     size_t csize = GET_SIZE(HDRP(bp));
206
207     if((csize-asize)>=(2*DSIZE)){
208         //블록의 사이즈와 입력받은 사이즈 차가 16이하라면
209         PUT(HDRP(bp),PACK(asize,1));
210         PUT(FTRP(bp),PACK(asize,1));
211         bp=NEXT_BLK(bp);
212         //블록의 헤더와 풋터에 입력 사이즈 만큼 할당됐다고 한 뒤 다음 블록으로 간다
213
214         PUT(bp,GET(PREV_BLKP(bp)));
215         PUT(bp+WSIZE,GET(PREV_BLKP(bp)+WSIZE));
216         //할당 받고 남은 블록의 다음,이전 블록을 가리키는 주소에 원래의 다음,이전 블록 주소로 바꾼다
217         PUT(HDRP(bp),PACK(csize-asize,0)); //할당 받고 남은 사이즈를 저장
218         PUT(FTRP(bp),PACK(csize-asize,0)); //할당 받고 남은 사이즈를 풋터에 저장
219         PUT(GET(bp+WSIZE),bp);
220         if(GET(bp)!=NULL)
221             {PUT(GET(bp)+WSIZE,bp);}
222         //남은 블록의 다음이 있다면 다음 블록의 이전을 bp로 저장한다
223
224     }else{//자가 16보다 작으면 블록을 나누지 않고 모두 할당한다
225         PUT(HDRP(bp),PACK(csize,1));
226         PUT(FTRP(bp),PACK(csize,1));
227         //해당 블록의 헤더와 풋터에 할당 여부를 저장한다
228         PUT(GET(bp+WSIZE),GET(bp));
229
230         if(GET(bp)!=NULL){ PUT(GET(bp)+WSIZE,GET(bp+WSIZE));}
231         //블록의 다음 블록의 이전을 가리키는 주소로 할당받은 블록의 이전을 저장
232     }
233 }
234 }
235
~/malloclab-handout/mm-explicit.c [utf-8,unix][c] 1,204/344 65%
```

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

```
mm-explicit.c + (~/.malloclab-handout) - VIM
236 static void *coalesce(void *bp) {
237     size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKPP(bp)));
238     size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKPP(bp)));
239     size_t size = GET_SIZE(HDRP(bp));
240
241     if(prev_alloc && next_alloc){//앞 뒤 블록이 모두 할당될 경우
242         if(GET(heap_start)==NULL)//만약 아직 freelist가 비었다면
243         { PUT(heap_start, bp); //root를 새롭게 지정하고
244           PUT(bp, NULL); //bp의 next는 NULL로 지정
245           PUT(bp+WSIZE, heap_start); //bp의 이전 블록은 root를 저장
246         }else{
247             PUT(bp, GET(heap_start)); //bp의 next는 루트의 next를 받고
248             PUT(bp+WSIZE, heap_start); //bp의 이전은 루트를 받는다
249             if (GET(bp) != NULL)
250                 {PUT(GET(bp)+WSIZE, bp); } //다음 블록의 이전 블록은 bp로 지정한다
251             PUT(heap_start, bp); //루트의 next는 bp를 지정한다
252         }
253     }else if (prev_alloc && !next_alloc){//뒷 블록이 free일 경우
254         PUT(GET(NEXT_BLKPP(bp))+WSIZE, GET(NEXT_BLKPP(bp)));
255         //프리된 블록의 다음다음블록의 이전은 다음블록으로 지정한다
256         if (GET(NEXT_BLKPP(bp)) != NULL)
257             {PUT(GET(NEXT_BLKPP(bp))+WSIZE, GET(NEXT_BLKPP(bp)+WSIZE)); }
258         //다음블록이 존재한다면 다음다음블록의 이전블록을 다음블록의 이전블록으로 한다
259         PUT(bp, GET(heap_start)); //bp의 다음블록은 원래 root가 가리키던걸 받는다
260         PUT(bp+WSIZE, heap_start); //bp의 이전블록은 root를 가리킨다
261         if (GET(bp) != NULL) {PUT(GET(bp)+WSIZE, bp); }
262         //다음블록이 존재하면 다음블록의 이전주소에 프리된블록을 지정
263         PUT(heap_start, bp); //root의 다음블록에는 bp를 지정한다
264         size+= GET_SIZE(HDRP(NEXT_BLKPP(bp)));
265         PUT(HDRP(bp), PACK(size, 0));
266     }
```

free할 때 합칠 때 앞뒤를 보고 합쳐야하는데 그때 사용되는 함수이다

-case 1 : 첫 번째 경우에는 앞 뒤블록이 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 뒷블록이 free일 때이다 뒷블록이 free이면 뒷뒷블록까지 포인터를 생각해줘야한다

다음다음블록의 이전을 다음블록으로 지정하고

그리고 앞에서의 case 1과 같이 FIFO정책에 맞게 Linked List수정법으로 heap\_start와 bp의 포인터들을 수정한다

```
mm-explicit.c + (~/.malloclab-handout) - VIM
268     else if (!prev_alloc && next_alloc) { //알이 free인 경우
269         PUT(GET(PREV_BLKp(bp)+WSIZE),GET(PREV_BLKp(bp)));
270         //이전이전블럭의 다음블럭을 가리키는 주소에 이전의 다음블럭주소를 저장
271         if (GET(PREV_BLKp(bp)) != NULL)
272             {PUT(GET(PREV_BLKp(bp))+WSIZE,GET(PREV_BLKp(bp)+WSIZE));}
273         //이전의 다음블럭이 존재한다면
274         //이전블럭의 다음블럭의 이전을 가리키는 주소에 이전블럭의 이전을 저장
275         PUT(PREV_BLKp(bp),GET(heap_start));
276         PUT(PREV_BLKp(bp)+WSIZE,heap_start);
277         //이전의 다음,이전블럭을 가리키는 주소에 root의 다음과 root를 저장
278
279         if (GET(PREV_BLKp(bp)) != NULL)
280             {PUT(GET(PREV_BLKp(bp))+WSIZE,PREV_BLKp(bp));}
281         //이전의 다음블럭이 존재하면 프리된 이전블럭을 가리키는 주소로 변경
282         PUT(heap_start,PREV_BLKp(bp));
283         //root의 다음블럭을 가리키는 주소에 블럭의 이전블럭을 저장
284         size += GET_SIZE(HDRP(PREV_BLKp(bp)));
285         PUT(FTRP(bp),PACK(size,0));
286         PUT(HDRP(PREV_BLKp(bp)),PACK(size,0));
287         //사이즈를 키우고 헤더와 풋터에 사이즈를 넣어준다
288         bp = PREV_BLKp(bp); //알오르 땡겨야 할오르 땡겨준다
289     }
290     else { //그외 알뒤로 free가 있을 경우이다
291         PUT(GET(PREV_BLKp(bp)+WSIZE),GET(PREV_BLKp(bp)));
292         //일단 이전이전블럭의 다음블럭 가리키는 주소에 이전의 다음블럭으로 지정
293         if (GET(PREV_BLKp(bp)) != NULL)
294             {PUT(GET(PREV_BLKp(bp))+WSIZE,GET(PREV_BLKp(bp)+WSIZE));}
295         //이전의 다음블럭이 존재한다면 적절히 포인터를 옮긴다
296         PUT(GET(NEXT_BLKp(bp)+WSIZE),GET(NEXT_BLKp(bp)));
297         //다음블럭의 인의 다음블럭주소에 다음다음블럭을 저장한다
298         if (GET(NEXT_BLKp(bp)) != NULL)
299             {PUT(GET(NEXT_BLKp(bp))+WSIZE,GET(NEXT_BLKp(bp)+WSIZE));}
300         //만약 다음다음블럭이 존재한다면 적절히 포인터를 옮긴다
301
302         //이제 이전블럭을 고쳐준다
303         PUT(PREV_BLKp(bp),GET(heap_start));
304         //이전의 다음블럭에 root의 다음블럭을 저장
305         PUT(PREV_BLKp(bp)+WSIZE,heap_start);
306         //이전이전블럭을 가리키는 주소에 root를 저장
307         if (GET(PREV_BLKp(bp)) != NULL)
308             {PUT(GET(PREV_BLKp(bp))+WSIZE,PREV_BLKp(bp));}
309         //블럭의 이전블럭이 존재하면 이전의 다음의 이전을 가리키는 주소를 바꾼다
310         PUT(heap_start,PREV_BLKp(bp));
311         //root의 다음블럭을 free된 블럭의 이전을 저장 (여차피 합칠꺼니깐)
312         size += GET_SIZE(HDRP(PREV_BLKp(bp))) + GET_SIZE(FTRP(NEXT_BLKp(bp)));
313         PUT(HDRP(PREV_BLKp(bp)),PACK(size,0));
314         PUT(FTRP(NEXT_BLKp(bp)),PACK(size,0));
315         bp = PREV_BLKp(bp);
316         //사이즈를 더해준뒤에 다시 블럭마다 사이즈를 넣어주면서 새로운 bp를 만든다
317     }
318
319     return bp;
320 }
```

-case 3 : 앞부분이 free일 경우에는 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 났을 경우이다 즉 앞 뒤가 모두 프리 블럭 일 경우이다.

이럴 때에는 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$
```