

**2018 시스템 프로그래밍**  
**- Lab 09 -**

제출일자	2018.12.03
분 반	00
이 름	김민기
학 번	201502023

## 1. mm-naive

```
a201502023@2018-sp: ~/malloclab-handout
a201502023@2018-sp:~/malloclab-handout$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 2097.6 MHz

Results for mm malloc:
  valid  util   ops   secs    Kops  trace
  yes    94%    10   0.000000  42895  ./traces/malloc.rep
  yes    77%    17   0.000000  63904  ./traces/malloc-free.rep
  yes   100%    15   0.000000  48555  ./traces/corners.rep
* yes    71%   1494  0.000022  66858  ./traces/perl.rep
* yes    68%   118   0.000002  64457  ./traces/hostname.rep
* yes    65%  11913  0.000165  72027  ./traces/xterm.rep
* yes    23%   5694  0.000094  60643  ./traces/amptjp-bal.rep
* yes    19%   5848  0.000097  60207  ./traces/cccp-bal.rep
* yes    30%   6648  0.000117  57003  ./traces/cp-decl-bal.rep
* yes    40%   5380  0.000087  61992  ./traces/expr-bal.rep
* yes     0%  14400  0.000232  62102  ./traces/coalescing-bal.rep
* yes    38%   4800  0.000086  55781  ./traces/random-bal.rep
* yes    55%   6000  0.000094  64169  ./traces/binary-bal.rep
10      41%  62295  0.000995  62581

Perf index = 26 (util) + 40 (thru) = 66/100
a201502023@2018-sp:~/malloclab-handout$
```

## 소스 코드

```
40 /* single word (4) or double word (8) alignment */
41 #define ALIGNMENT 8
42
43 /* rounds up to the nearest multiple of ALIGNMENT */
44 #define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7)
45
46
47 #define SIZE_T_SIZE (ALIGN(sizeof(size_t)))
48
49 #define SIZE_PTR(p) ((size_t*)((char*)(p)) - SIZE_T_SIZE))
50
```

매크로 설명

- #define ALIGNMENT 8 : 사용하는 word를 상수로 정의 (double-word alignment을 유지해야 하므로 8로 정의)
- #define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7) : 주어진 size에 맞는 블록의 크기를 계산
- #define SIZE\_T\_SIZE (ALIGN(sizeof(size\_t))) : size\_t에 맞는 블록 size를 반환
- #define SIZE\_PTR(p) ((size\_t\*)((char\*)(p)) - SIZE\_T\_SIZE) : p주소의 값에서 SIZE\_T\_SIZE 만큼 뺀 위치의 포인터를 반환

## 구현 방법

### 1. mm\_init 함수

- heap의 구조를 생성하는 함수

### 2. malloc함수

```
mm-naive.c (~/.malloclab-handout) - VIM
61  /* Always allocate a block whose size is a multiple of the alignment.
62  */
63  void *malloc(size_t size)
64  {
65      int newsize = ALIGN(size + SIZE_T_SIZE);
66      unsigned char *p = mem_sbrk(newsize);
67      //dbg_printf("malloc %u => %p\n", size, p);
68
69      if ((long)p < 0)
70          return NULL;
71      else {
72          p += SIZE_T_SIZE;
73          *SIZE_PTR(p) = size;
74          return p;
75      }
76  }
77
78  /*
~/malloclab-handout/mm-naive.c [utf-8,unix] [c] 2,61/145 47%
```

- 주어진 size의 크기를 가지는 블록 포인터를 리턴하는 함수
- newsize는 ALIGN을 호출해 할당해야 할 블록 size와 header를 합한 영역을 계산한 값
- mem\_sbrk를 이용해 (newsize + SIZE\_T\_SIZE)의 시작 포인터를 할당 받음
- 할당 받은 결과값이 0보다 작으면 할당이 실패했으므로 0을 반환하고, 0 이상이면 할당을 제대로 받은 것이므로 포인터를 SIZE\_T\_SIZE만큼 증가시키고 p가 위치하는 부분에 size를 넣은후 포인터를 리턴

### 3. free 함수

- naive에서는 구현되어있지 않다.

#### 4. realloc 함수

```
mm-implicit.c (~/.malloclab-handout) - VIM
305  */
306 void *realloc(void *oldptr, size_t size) {
307     size_t oldsize;
308     void *newptr;
309
310     //size가 0이면 free시키고 null을 리턴
311     if(size == 0) {
312         free(oldptr);
313         return 0;
314     }
315
316     // oldptr이 null이면, 새로 할당
317     if(oldptr == NULL)
318         return malloc(size);
319
320     //새로운 공간 할당
321     newptr = malloc(size);
322
323     //realloc()이 실패할 경우 원래 블록은 그대로
324     if(!newptr)
325         return 0;
326
327     //이전 데이터를 복사
328     oldsize = GET_SIZE(HDRP(oldptr));
329
330     if(size < oldsize)
331         oldsize = size;
332
333     memcpy(newptr, oldptr, oldsize);
334
335     //이전 공간 free
336     free(oldptr);
337
338     return newptr;
339 }
```

- 이미 할당되어 있는 메모리의 크기를 재할당하는 함수

#### 5. calloc 함수

```
mm-naive.c (~/.malloclab-handout) - VIM
124
125 /*
126  * calloc - Allocate the block and set it to zero.
127  */
128 void *calloc (size_t nmemb, size_t size)
129 {
130     size_t bytes = nmemb * size;
131     void *newptr;
132
133     newptr = malloc(bytes);
134     memset(newptr, 0, bytes);
135
136     return newptr;
137 }
```

- 블록을 리셋시킨 블록을 리턴하는 함수
- 할당되는 메모리의 크기는 (nmemb \* size) byte

- 새로 생성된 메모리 블록의 주소를 리턴

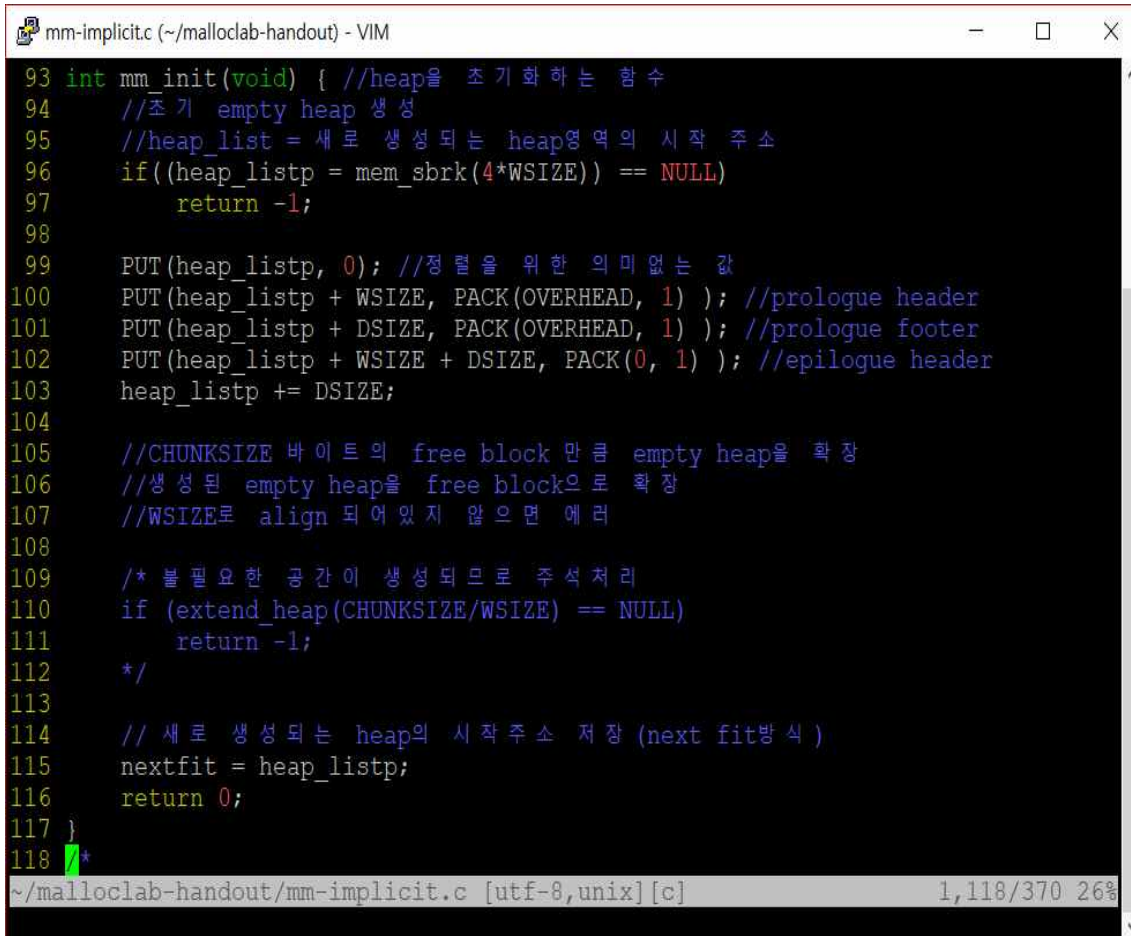
## 2. mm-implicit

```
a201502023@2018-sp: ~/malloclab-handout
a201502023@2018-sp:~/malloclab-handout$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 2097.6 MHz

Results for mm malloc:
  valid  util   ops    secs    Kops  trace
  yes    34%    10    0.000001 14689 ./traces/malloc.rep
  yes    28%    17    0.000001 27968 ./traces/malloc-free.rep
  yes    96%    15    0.000001 24814 ./traces/corners.rep
* yes    81%   1494    0.000377  3965 ./traces/perl.rep
* yes    75%    118    0.000006 20064 ./traces/hostname.rep
* yes    89%  11913    0.002956  4030 ./traces/xterm.rep
* yes    90%   5694    0.000479 11876 ./traces/amptjp-bal.rep
* yes    92%   5848    0.000577 10141 ./traces/cccp-bal.rep
* yes    95%   6648    0.000608 10927 ./traces/cp-decl-bal.rep
* yes    97%   5380    0.000477 11271 ./traces/expr-bal.rep
* yes   100%  14400    0.000401 35889 ./traces/coalescing-bal.rep
* yes    85%   4800    0.001577  3043 ./traces/random-bal.rep
* yes    55%   6000    0.000337 17795 ./traces/binary-bal.rep
10      86%  62295    0.007796  7990

Perf index = 55 (util) + 40 (thru) = 95/100
a201502023@2018-sp:~/malloclab-handout$
```

## 1. mm\_init함수



```

93 int mm_init(void) { //heap을 초기화 하는 함수
94     //초기 empty heap 생성
95     //heap_listp = 새로 생성되는 heap영역의 시작 주소
96     if((heap_listp = mem_sbrk(4*WSIZE)) == NULL)
97         return -1;
98
99     PUT(heap_listp, 0); //정렬을 위한 의미 없는 값
100    PUT(heap_listp + WSIZE, PACK(OVERHEAD, 1)); //prologue header
101    PUT(heap_listp + DSIZE, PACK(OVERHEAD, 1)); //prologue footer
102    PUT(heap_listp + WSIZE + DSIZE, PACK(0, 1)); //epilogue header
103    heap_listp += DSIZE;
104
105    //CHUNKSIZE 바이트의 free block 만큼 empty heap을 확장
106    //생성된 empty heap을 free block으로 확장
107    //WSIZE로 align 되어 있지 않으면 에러
108
109    /* 불필요한 공간이 생성되므로 주석처리
110    if (extend_heap(CHUNKSIZE/WSIZE) == NULL)
111        return -1;
112    */
113
114    // 새로 생성되는 heap의 시작주소 저장 (next fit방식)
115    nextfit = heap_listp;
116    return 0;
117 }
118 */

```

- heap을 초기화 하는 함수이다.
- mem\_sbrk함수를 이용해 heap의 크기를 16바이트만큼 증가시킨다. 그리고 4번의 PUT을 통해 heap의 정보저장을 한다.
- 이렇게 heap을 생성한 후, 다음 heap의 크기를 extend\_heap함수를 이용해 CHUNKSIZE바이트(4096)의 free블록만큼 확장한다. 이 값이 NULL이면 확장을 할 수 없으므로 -1을 리턴한다. 하지만 이 부분은 불필요한 공간을 만들어 성능이 떨어질 수도 있기 때문에 주석처리를 하였다.
- 그리고 next\_fit으로 구현할 것이기 때문에 next\_fit 포인터를 heap\_listp가 있는 곳에 넣어준다.

## 2. malloc 함수

```
mm-implicit.c + (~/.malloclab-handout) - VIM
121 void *malloc (size_t size) { //size크기의 블록을 heap에 할당하는 함수
122     size_t asize;
123     size_t extendsize;
124     char *bp;
125
126     //size가 0이면 할당할 필요가 없으므로 null을 리턴
127     if(size == 0)
128         return NULL;
129
130     //할당하려는 사이즈가 8보다 작거나 같은 경우
131     //asize = DSIZE + OVERHEAD(header, footer)
132     if(size <= DSIZE)
133         asize = DSIZE + OVERHEAD;
134
135     //할당하려는 사이즈가 8보다 클 경우
136     //8byte 단위블록으로 할당하기 위한 size로 블록을 할당
137     else
138         //8의 배수 중 가장 근접한 숫자만큼 할당
139         asize = DSIZE * ((size + OVERHEAD + (DSIZE-1)) / DSIZE);
140
141     //할당하려는 size와 맞는 free블록이 있으면 할당
142     if((bp = find_fit(asize)) != NULL){
143         place(bp, asize);
144         return bp;
145     }
146
147     //asize와 CHUNKSIZE 중 큰 값으로 확장
148     extendsize = MAX(asize, CHUNKSIZE);
149
150     //extendsize 만큼 확장
151     if((bp = extend_heap(extendsize/WSIZE)) == NULL )
152         return NULL;
153
154     place(bp, asize);
155     return bp;
156 }
157 /*
158 * coalesce
159 */
161 static void *coalesce(void *bp) { //인접한 free상태의 블록을 합쳐주는 함수
    ~/.malloclab-handout/mm-implicit.c [utf-8,unix][+][c] 25,150/370 36%
    :wq
```

- size크기의 블록을 heap에 할당하는 함수이다.
- 할당할 size가 0이면 null 리턴하고, size가 8보다 작거나 같을 때는 원하는 크기를 허용하는 블록 size는 DSIZE(8)가 최소로 적합하므로 블록의 정보를 저장하는 공간만큼의 크기를 할당하고, 8보다 클 때는 8의 배수의 가장 근접한 숫자만큼 할당한다.
- 할당한 블록의 크기를 정한 find\_fit함수를 이용해 할당하려는 size와 맞는 free블록이 있으면 place함수를 이용해 헤더부분에 size를 저장한다. 할당하려는 size와 맞는 free블록이 없으면 extend\_heap함수를 이용해 heap을 확장한 뒤 place함수를 이용해 위치를 잡아준다.
- heap을 확장했는데도 null이 반환되면 heap을 벗어나는 영역을 침범했기 때문에 null을 리턴해준다.



### 3. coalesce 함수

```
mm-implicit.c (~/.malloclab-handout) - VIM
162 static void *coalesce(void *bp) { //인접한 free상태의 블록을 합쳐주는 함수
163     //이전 블록의 할당 여부 0 = No, 1 = Yes
164     size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKPTR(bp)));
165     //다음 블록의 할당 여부 0 = No, 1 = Yes
166     size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKPTR(bp)));
167     //현재 블록의 크기
168     size_t size = GET_SIZE(HDRP(bp));
169
170     //case1 : 이전 블록, 다음 블록 최하위 bit가 둘다 1인 경우 (할당)
171     if(prev_alloc && next_alloc){
172         //모두 할당되어 있으므로 bp 리턴
173         return bp;
174     }
175
176     //case2 : 이전 블록 최하위 bit가 1, 다음 블록 최하위 bit가 0인 경우 (>
    비할당) 다음 블록과 병합한 뒤 bp return
177     else if(prev_alloc && !next_alloc){
178         //이전 블록만 할당되어 있으므로 현재 size에 다음 블록 size 더함
179         size += GET_SIZE(HDRP(NEXT_BLKPTR(bp)));
180         //bp의 header에 블록 size와 alloc = 0을 저장
181         PUT(HDRP(bp), PACK(size, 0));
182         //bp의 footer에 블록 size와 alloc = 0을 저장
183         PUT(FTRP(bp), PACK(size, 0));
184     }
185
186     //case3 : 이전 블록 최하위 bit가 0이고 (비할당), 다음 블록 최하위 bit>
    가 1인 경우 (할당) 이전 블록과 병합한 뒤 새로운 bp return
187     else if(!prev_alloc && next_alloc){
188         //다음 블록만 할당되어 있으므로 현재 size에 이전 블록 size 더함
189         size += GET_SIZE(HDRP(PREV_BLKPTR(bp)));
190         //bp의 footer에 블록 size와 alloc = 0을 저장
191         PUT(FTRP(bp), PACK(size, 0));
192         //bp의 header에 블록 size와 alloc = 0을 저장
193         PUT(HDRP(PREV_BLKPTR(bp)), PACK(size, 0));
194         //이전 블록의 포인터를 저장
195         bp = PREV_BLKPTR(bp);
196     }
197
198     //case4 : 이전 블록 최하위 bit가 0이고 (비할당), 다음 블록 최하위 bit>
    가 0인 경우 (비할당) 이전 블록, 현재 블록, 다음 블록을 모두 병합한 뒤 새로운
    bp return
    ~/.malloclab-handout/mm-implicit.c [utf-8,unix][c] 0,197/370 48%
```



```
mm-implicit.c (~/.malloclab-handout) - VIM
191     PUT(FTRP(bp), PACK(size, 0));
192     //bp의 header에 블록 size의 alloc = 0을 저장
193     PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
194     //이전 블록의 포인터를 저장
195     bp = PREV_BLKP(bp);
196 }
197
198 //case4 : 이전블록 최하위 bir가 0이고 (비할당), 다음 블록 최하위 bir가
0인 경우 (비할당) 이전블록, 현재블록, 다음블록을 모두 병합한 뒤 새로운 bp
return
199 else {
200     //현재블록, 이전블록, 다음블록의 size를 더함
201     size += GET_SIZE(HDRP(PREV_BLKP(bp))) + GET_SIZE(FTRP(NEXT_BLKP(b
p))) );
202     //이전 블록 bp의 header에 더해진 size와 alloc = 0을 저장
203     PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
204     //다음 블록 bp의 footer에 더해진 size와 alloc = 0을 저장
205     PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
206     //이전 블록의 포인터를 저장
207     bp = PREV_BLKP(bp);
208 }
209 //병합된 블록의 주소 bp return
210 return bp;
211 }
212 /*
213 *place
214 */
@
~/malloclab-handout/mm-implicit.c [utf-8,unix][c] 2,212/370 54%
```

- 인접한 free상태의 블록을 합쳐주는 함수이다.
- free 시킨 블록의 앞, 뒤 공간의 alloc 유무를 확인하여 4가지 경우로 나눔.
- case 1 : free한 블록의 앞뒤가 모두 할당된 경우, 인접한 free 블록이 없기 때문에 블록 병합 없이 bp를 리턴
- case 2 : free한 블록의 다음 블록이 free 블록인 경우, 다음 블록과 병합 후 bp를 리턴
- case 3 : free한 블록의 이전 블록이 free 블록인 경우, 이전 블록과 병합 후 새로운 bp 리턴
- case 4 : free한 블록의 이전과 다음 블록이 모두 free 블록인 경우, 이전 블록, 현재 블록, 다음 블록을 모두 병합 후 새로운 bp 리턴

#### 4. place 함수

```
mm-implicit.c (~/.malloclab-handout) - VIM
215 static void place(void *bp, size_t asize) //bp 위치에 asize 크기의 메모리
    리를 위치시키는 함수
216
217     //bp가 가리키는 header에서 size를 가져와서 저장
218     size_t newsize = GET_SIZE(HDRP(bp));
219
220     //header에서 가져온 size와 할당하려는 size의 차가
221     //블록의 최소 크기인 16보다 크거나 같을 때
222     //블록을 분할
223     if((newsize - asize) >= (2*DSIZE)){
224         //header에 asize와 1을 or연산한 것을 넣음
225         PUT(HDRP(bp), PACK(asize, 1));
226         //footer에 asize와 1을 or연산한 것을 넣음
227         PUT(FTRP(bp), PACK(asize, 1));
228         //다음 블록으로 이동
229         bp = NEXT_BLKP(bp);
230         //header에 newsize-asize한 size를 넣고 비할당 상태로 만들
231         PUT(HDRP(bp), PACK(newsize-asize, 0));
232         //footer에 newsize-asize를 size를 넣고 비할당 상태로 만들
233         PUT(FTRP(bp), PACK(newsize-asize, 0));
234     }
235     //블록의 최소 크기보다 작을 때
236     //블록을 분할하지 않음
237     else{
238         PUT(HDRP(bp), PACK(newsize, 1));
239         PUT(FTRP(bp), PACK(newsize, 1));
240     }
241 }
```

- bp 위치에 asize 크기의 메모리를 위치시키는 함수이다.
- 각 block포인터의 size를 newsize에 저장하고 할당하고 싶은 size의 차이와 비교해서 16byte보다 크거나 같으면 padding이 생겨 블록을 분할한다. 16byte보다 작으면 더 이상 분할 할 수 없으므로 header와 footer의 크기정보를 변환한다.

## 5. extend\_heap 함수

```
mm-implicit.c (~/.malloclab-handout) - VIM
244 */
245 static void *extend_heap(size_t words) { //요청 받은 크기의 빈 블록을 만들어
//줌. 이전 블록을 검사하여 case별로 free시켜주도록 구현하는 함수
246
247     char *bp;
248     size_t size;
249     //word가 짝수이면 word*WSIZE, 홀수이면 짝수로 만들어 준 뒤 WSIZE를 곱 >
함
250     size = (words % 2) ? (words + 1) * WSIZE : words * WSIZE;
251
252     //mem_sbrk를 통해 size만큼 heap을 확장
253     if((long)(bp = mem_sbrk(size)) == -1)
254         return NULL;
255
256     //확장한 블록의 header에 size를 넣고 free블록 표시
257     PUT(HDRP(bp), PACK(size, 0));
258     //확장한 블록의 footer에 size를 넣고 free블록 표시
259     PUT(FTRP(bp), PACK(size, 0));
260     //확장했으므로 epilogue를 표시하고
261     //다음 블록의 header에 alloc부분을 1로 만들어줌.
262     PUT(HDRP(NEXT_BLKP(bp)), PACK(0, 1));
263
264     //coalesce를 이용해 free블록끼리 결합
265     return coalesce(bp);
266 }
267 /*
268 *find_fit
269 */
~/malloclab-handout/mm-implicit.c [utf-8,unix] [c] 3,244/370 70%
```

- 요청 받은 크기의 빈 블록을 만들어 줌. 이전 블록을 검사하여 case별로 free시켜주도록 구현(heap을 주어진 크기만큼 확장하는 함수).
- words를 받아와 짝수와 홀수일 때 size를 다르게 계산하는데 Alignment를 지켜주기 위해 짝수로 만들어준다.
- 그런 다음 mem\_sbrk함수를 이용해 정한 size만큼 heap을 확장한다. 확장이 불가능한 상태라면 -1을 리턴하므로 null을 리턴하면 된다.
- 확장이 되면 확장 된 블록자체를 반환한다. heap구조의 마지막 부분이 이동되었으므로 에필로그 header 부분의 정보를 확장된 블록의 맨 마지막에 할당한다.
- 그런 다음 coalesce함수를 이용해 free블록끼리 합쳐주고 리턴한다.

## 6. find\_fit

```
mm-implicit.c (~/malloclab-handout) - VIM
267 /*
268 *find_fit
269 */
270 static void *find_fit(size_t asize) { //free block을 검색하는 함수
271     //nextfit으로 구현
272
273     void *bp;
274     //조기값을 nextfit으로 설정해서 검색이 끝난 위치부터 다시 검색을 하도록 함
275     for(bp = nextfit; GET_SIZE(HDRP(bp)) > 0; bp = NEXT_BLKP(bp))
276         if(!GET_ALLOC(HDRP(bp)) && (asize <= GET_SIZE(HDRP(bp))))
277             return bp;
278     return NULL;
279 }
280 /*
281 * free
~/malloclab-handout/mm-implicit.c [utf-8,unix][c] 1,280/370 74%
```

- free블록을 검색하는 함수이다.
- next fit으로 구현하여 bp를 마지막으로 free시킨 block의 포인터로 정하여 block의 위치를 증가해가며 다음블럭 순으로 찾아 free된 블록을 찾는 함수이다.

## 7. free 함수

```
mm-implicit.c (~/.malloclab-handout) - VIM
283 * free
284 */
285 void free (void *bp){
286     //잘못된 free요청일 경우 함수 종료
287     if (bp == 0)
288         return;
289
290     //bp의 header에서 block size를 읽어옴.
291     size_t size = GET_SIZE(HDRP(bp));
292
293     //bp의 header에 블록 size와 alloc=0을 저장
294     PUT(HDRP(bp), PACK(size, 0));
295     //bp의 footer에 블록 size와 alloc=0을 저장
296     PUT(FTRP(bp), PACK(size, 0));
297     //주위에 빈 블록이 있으면 병합
298     nextfit = coalesce(bp);
299 }
300
301 /*
302 * realloc - you may want to look at mm-naive.c
303 */
~/malloclab-handout/mm-implicit.c [utf-8,unix][c] 5,283/368 81%
```

- 할당된 메모리를 해제하는 함수이다.
- 잘못된 free요청일 경우 함수를 종료 시킨다.
- 정상적인 경우라면 header에서 정보를 읽어와 포인터의 size를 알아내, 받아온 블록의 header와 footer에 block size와 alloc = 0을 저장한다.
- 그 이후 coalesce함수를 호출해 주위에 빈 블록이 있으면 병합한다.

## 8. realloc 함수

```
mm-implicit.c (~/.malloclab-handout) - VIM
305 */
306 void *realloc(void *oldptr, size_t size) {
307     size_t oldsize;
308     void *newptr;
309
310     //size가 0이면 free시키고 null을 리턴
311     if(size == 0) {
312         free(oldptr);
313         return 0;
314     }
315
316     // oldptr이 null이면, 새로 할당
317     if(oldptr == NULL)
318         return malloc(size);
319
320     //새로운 공간 할당
321     newptr = malloc(size);
322
323     //realloc()이 실패할 경우 원래 블록은 그대로
324     if(!newptr)
325         return 0;
326
327     //이전 데이터를 복사
328     oldsize = GET_SIZE(HDRP(oldptr));
329
330     if(size < oldsize)
331         oldsize = size;
332
333     memcpy(newptr, oldptr, oldsize);
334
335     //이전 공간 free
336     free(oldptr);
337
338     return newptr;
339 }
```

- 이미 할당되어 있는 메모리의 크기를 다시 할당하는 함수이다.
- size가 0인 경우, 이미 할당되어 있는 메모리를 0으로 만들면 되기 때문에 free함수를 호출하고 종료한다.
- 주소가 null이라면 새로운 주소에 원하는 크기로 새로 할당을 하고 다음 주소를 리턴한다.
- 주소도 정확하고 size도 0이상인 경우, malloc함수를 호출하여 원하는 size의 메모리 포인터를 가져와 새로운 size를 갖는 주소포인터를 할당받는다.
- 새로 받아온 주소가 정확하지 않으면 0을 리턴하고, 가용한 주소이면 이전 데이터를 새로운 공간에 복사해서 이전공간을 free시켜준다. 이전시킬 주소의 크기를 GET\_SIZE로 받아오고, 이전시킬 주소의 크기가 새로 저장할 주소의 크기보다 크면 넘치게 되므로 작은 크기에 맞춰서 저장한다. 반대의 경우는 남는 공간이 필요가 없으므로 크기 조절을 하지 않아도 된다.
- 크기 조절이 끝나면 memcpy함수를 이용해 새로운 주소로 header와 이전시킬 메모리의 정보를 가져와서 저장한다. 그 후에 이전 주소를 free시켜주고 새로 할당한 주소를 리턴한다.