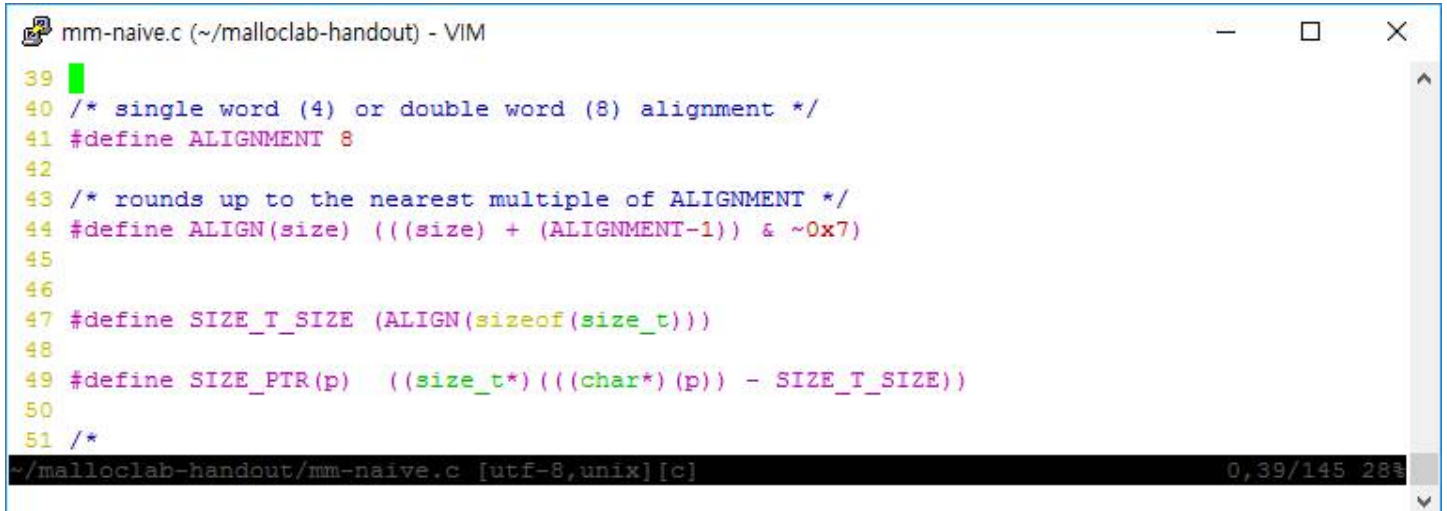


2016 시스템 프로그래밍
- malloc lab -

제출일자	2016.12.06
분 반	02
이 름	신종욱
학 번	2013024223

1. 매크로 설명



```
39
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
51 /*
~/malloclab-handout/mm-naive.c [utf-8,unix][c] 0,39/145 28%
```

`#define ALIGNMENT 8`

: 블록을 8의 배수로 할당하므로 자주 연산에 사용되서 정의하였다

`#define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7)`

: 블록이 8의 배수로 할당되도록 연산한다. size값이 무엇이 들어오든 8의 배수를 반환한다.

`#define SIZE_T_SIZE (ALIGN(sizeof(size_t)))`

: sizeof 명령은 안의 데이터 타입의 크기를 반환한다. size_t 는 int형과 같이 4를 반환한다.

따라서 블록이 최소 단위로 8을 리턴해야 하기 때문에 결과적으로 값은 8을 나타낸다.

`#define SIZE_PTR(p) ((size_t*)((char*)(p) - SIZE_T_SIZE))`

: 블록의 사이즈를 가르키고 있는 포인터를 옮긴다

2. 사용하는 함수설명 (다 구현되어있어서 사진은 생략하겠습니다)

-malloc

단순하게 heap을 늘리기만하며 할당한다

계속 늘리기만 하기 때문에 중간에 발생하는 빈 공간이 많아져서 효율이 안좋다.

-realloc

realloc의 본래 기능은 메모리를 재할당하는 함수이다.

naive에서는 size == 0, 경우 oldptr을 바로 free하나, free는 구현하지 않았기에 실질적으로 아무것도 하지 않는다.

양수의 size가 입력이 되면 해당 size 만큼을 malloc을 통해 할당한 뒤, size 만큼 복사하여 그대로 붙여넣는다.

-calloc

calloc의 본래 기능은 nmemb*size 만큼의 메모리를 확보하고,초기화 한 후, 시작주소를 반환해주는 함수이다.

naive에서는 size_t 타입의 변수를 만들어 nmemb*size 의 값을 넣고, 이 변수를 이용하여 malloc() 호출해 할당한뒤, memset을 통해 0으로 초기화 시킨다.

3.naive 결과

```
c201302423@host-192-168-0-5: ~/malloclab-handout
c201302423@host-192-168-0-5:~/malloclab-handout$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 2500.0 MHz

Results for mm malloc:
  valid  util   ops   secs   Kops  trace
  yes    94%    10   0.000000153374 ./traces/malloc.rep
  yes    77%    17   0.000000169322 ./traces/malloc-free.rep
  yes   100%    15   0.000000131579 ./traces/corners.rep
* yes    71%   1494   0.000009175583 ./traces/perl.rep
* yes    68%    118   0.000001178355 ./traces/hostname.rep
* yes    65%  11913   0.000067178348 ./traces/xterm.rep
* yes    23%   5694   0.000054106087 ./traces/amptjp-bal.rep
* yes    19%   5848   0.000057101776 ./traces/cccp-bal.rep
* yes    30%   6648   0.000069 95700 ./traces/cp-decl-bal.rep
* yes    40%   5380   0.000049108805 ./traces/expr-bal.rep
* yes     0%  14400   0.000146 98391 ./traces/coalescing-bal.rep
* yes    38%   4800   0.000041116713 ./traces/random-bal.rep
* yes    55%   6000   0.000056107087 ./traces/binary-bal.rep
10      41%  62295   0.000550113362

Perf index = 26 (util) + 40 (thru) = 66/100
c201302423@host-192-168-0-5:~/malloclab-handout$
```

일단 저는 기본코드들은 모두 책을 기준으로했습니다.

1. 매크로 정의 설명

```
mm-implicit.c (~/.malloclab-handout) - VIM
39
40 /* rounds up to the nearest multiple of ALIGNMENT */
41 #define ALIGN(p) (((size_t)(p) + (ALIGNMENT-1)) & ~0x7)
42
43 #define WSIZE 4
44 #define DSIZE 8
45 #define CHUNKSIZE (1<<12)
46 #define OVERHEAD 8
47
48 #define MAX(x,y) ((x)>(y)?(x):(y))
49 #define PACK(size,alloc) ((size)|(alloc))
50
51 #define GET(p) (*(unsigned int *) (p))
52 #define PUT(p,val) (*(unsigned int *) (p)=(val))
53
54 #define GET_SIZE(p) (GET(p) & ~0x7)
55 #define GET_ALLOC(p) (GET(p) & 0x1)
56 #define HDRP(bp) ((char *) (bp) - WSIZE)
57 #define FTRP(bp) ((char *) (bp) + GET_SIZE(HDRP(bp)) - DSIZE)
58 #define NEXT_BLKP(bp) ((char *) (bp) + GET_SIZE(((char *) (bp) - WSIZE)))
59 #define PREV_BLKP(bp) ((char *) (bp) - GET_SIZE(((char *) (bp) - DSIZE)))
60
61
62 int mm_init(void);
~/malloclab-handout/mm-implicit.c [utf-8,dos][c] 0,39/278 14%
```

#define WSIZE 4 : word 크기를 지정.

#define DSIZE 8 : wouble word의 크기 지정.

#define CHUNKSIZE (1<<12) : 초기 Heap의 크기를 설정해 준다.(4096)

#define OVERHEAD 8 : header + footer의 사이즈.

#define MAX(x, y) ((x) > (y)? (x) : (y))
: x와 y를 비교하여 더 큰 값을 반환한다.

#define PACK(size, alloc) ((size) | (alloc))
: size와 alloc(a)의 값을 한 word로 묶는다. header와 footer에 저장하기 위함.

#define GET(p) (*(unsigned int *) (p))
: 포인터 p가 가리키는 곳의 한 word의 값을 읽어온다.

#define PUT(p, val) (*(unsigned int *) (p) = (val))
: 포인터 p가 가르키는 곳의 한 word의 값에 val을 저장한다.

#define GET_SIZE(p) (GET(p) & ~0x7)
: 포인터 p가 가리키는 곳에서 한 word를 읽고 하위 3bit를 버린다.
왜냐하면 Header에서 block size를 읽기위함이다

#define GET_ALLOC(p) (GET(p) & 0x1)
:포인터 p가 가리키는 곳에서 한 word를 읽고 최하위 1bit를 가져온다.
block의 할당여부 체크하기 위함이고 할당된 블록이라면 1, 아니라면 0.

#define HDRP(bp) ((char *) (bp) - WSIZE)
:주어진 포인터 bp의 header의 주소를 계산한다.

#define FTRP(bp) ((char *) (bp) + GET_SIZE(HDRP(bp)) - DSIZE)
:주어진 포인터 bp의 footer의 주소를 계산한다.

```
#define NEXT_BLKP(bp) ((char *)(bp) + GET_SIZE( ((char *) (bp) - WSIZE)) )
```

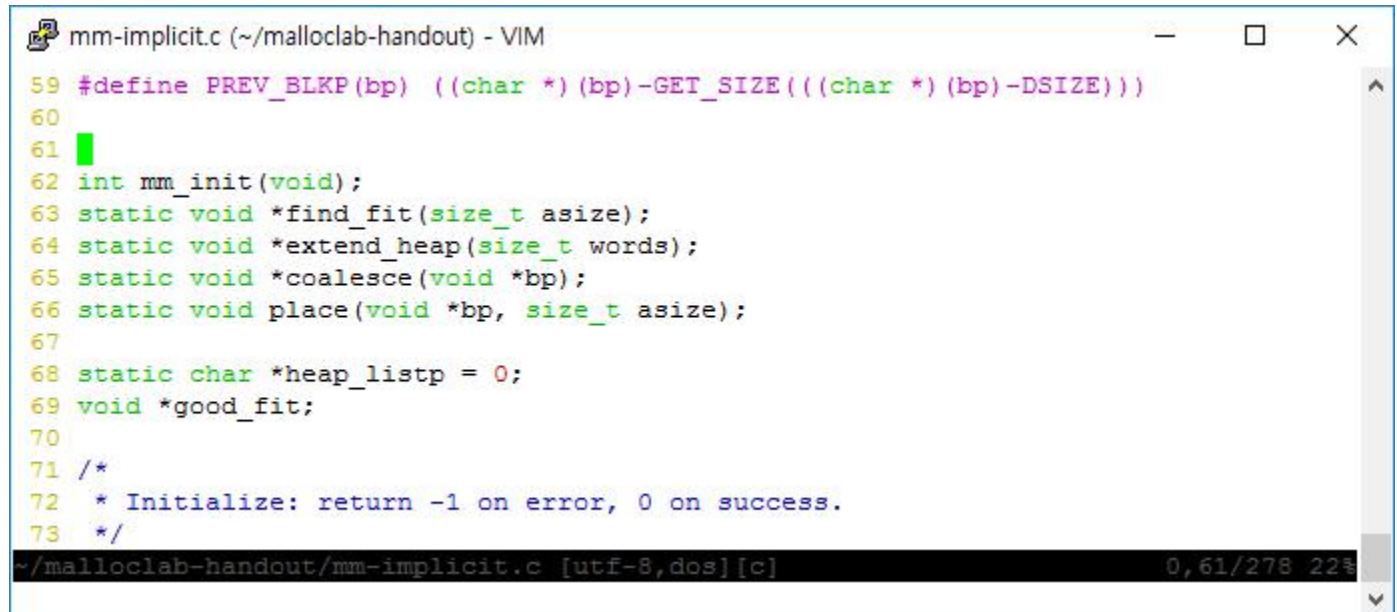
:주어진 포인터 bp를 이용하여 다음 블록의 주소를 얻어 온다.

```
#define PREV_BLKP(bp) ((char *) (bp) - GET_SIZE( ((char *) (bp) - DSIZE)) )
```

:주어진 포인터 bp를 이용하여 이전 블록의 주소를 얻어 온다.

(pdf에 오타있습니다. 중간에 -가 아니라 +로 되어있네요 한참 해맸습니다 ㅋㅋ)

2.사용된 함수들



```
mm-implicit.c (~/.malloclab-handout) - VIM
59 #define PREV_BLKP(bp) ((char *) (bp) - GET_SIZE( ((char *) (bp) - DSIZE)) )
60
61
62 int mm_init(void);
63 static void *find_fit(size_t asize);
64 static void *extend_heap(size_t words);
65 static void *coalesce(void *bp);
66 static void place(void *bp, size_t asize);
67
68 static char *heap_listp = 0;
69 void *good_fit;
70
71 /*
72  * Initialize: return -1 on error, 0 on success.
73  */
~/malloclab-handout/mm-implicit.c [utf-8,dos][c] 0,61/278 22%
```

static char *heap_listp = 0; :처음 first block의 포인터로 사용하였습니다

void *good_fit : 현재 block의 위치를 가리키는 포인터로 사용하였습니다.

best_fit으로 구현하는데 사용

나머지 함수들은 코드를 보면서 설명하겠습니다.

3. 함수들 설명

-init 함수

```
mm-implicit.c + (~/.malloclab-handout) - VIM
74 int mm_init(void) {
75     //초기 empty heap 생성
76     if ((heap_listp = mem_sbrk(4*WSIZE)) == NULL)
77         return -1;
78
79     PUT(heap_listp, 0); //정렬을 위한 무의미한 값
80     PUT(heap_listp + (1*WSIZE), PACK(DSIZE, 1)); // Prologue header
81     PUT(heap_listp + (2*WSIZE), PACK(DSIZE, 1)); // Prologue footer
82     PUT(heap_listp + (3*WSIZE), PACK(0, 1)); // Epilogue header
83     heap_listp += (2*WSIZE);
84     good_fit=heap_listp;
85     //empty heap free block으로 확장
86     if (extend_heap(CHUNKSIZE/WSIZE) == NULL)
87         return -1;
88     return 0;
89 }
90
~/malloclab-handout/mm-implicit.c [utf-8,dos][+][c] 20,74/265 29%
```

init함수의 경우 힙을 초기화 시키는 함수이다

heap_listp 포인터 변수를 이용하여 첫블록을 구성한다 블록 구조에 맞게 heap을 한다
그리고 다 끝나면 현재를 가르키는 good_fit에 heap_listp의 값을 같도록한다
만약 wsize로 조정되어있지않다면 에러이다

-malloc

```
mm-implicit.c + (~/.malloclab-handout) - VIM
109 */
110 void *malloc (size_t size) { //블록을 할당하는 함수
111
112     size_t asize;
113     size_t extendsize;
114     char *bp;
115
116     if (size == 0) return NULL;
117
118     if (size <= DSIZE) asize = 2*DSIZE;
119     else asize = DSIZE * ((size + (DSIZE) + (DSIZE-1)) / DSIZE);
120
121     if ((bp = find_fit(asize)) != NULL) {
122         place(bp, asize); //해당 사이즈만큼 포인터를 이동
123         return bp;
124     }
125
126     extendsize = MAX(asize, CHUNKSIZE);
127     if ((bp = extend_heap(extendsize/WSIZE)) == NULL) return NULL;
128
129     place(bp, asize);
130     good_fit = NEXT_BLKP(bp); //현재의 다음 힙의 위치를 저장한다
131     return bp;
132 }
133
~/malloclab-handout/mm-implicit.c [utf-8,dos][+][c] 4,109/265 44%
```

블록을 할당해주는 함수로 블록은 8의 배수로 bit 할당되어야하는데 그에 맞도록 바꿔주는 과정이다.

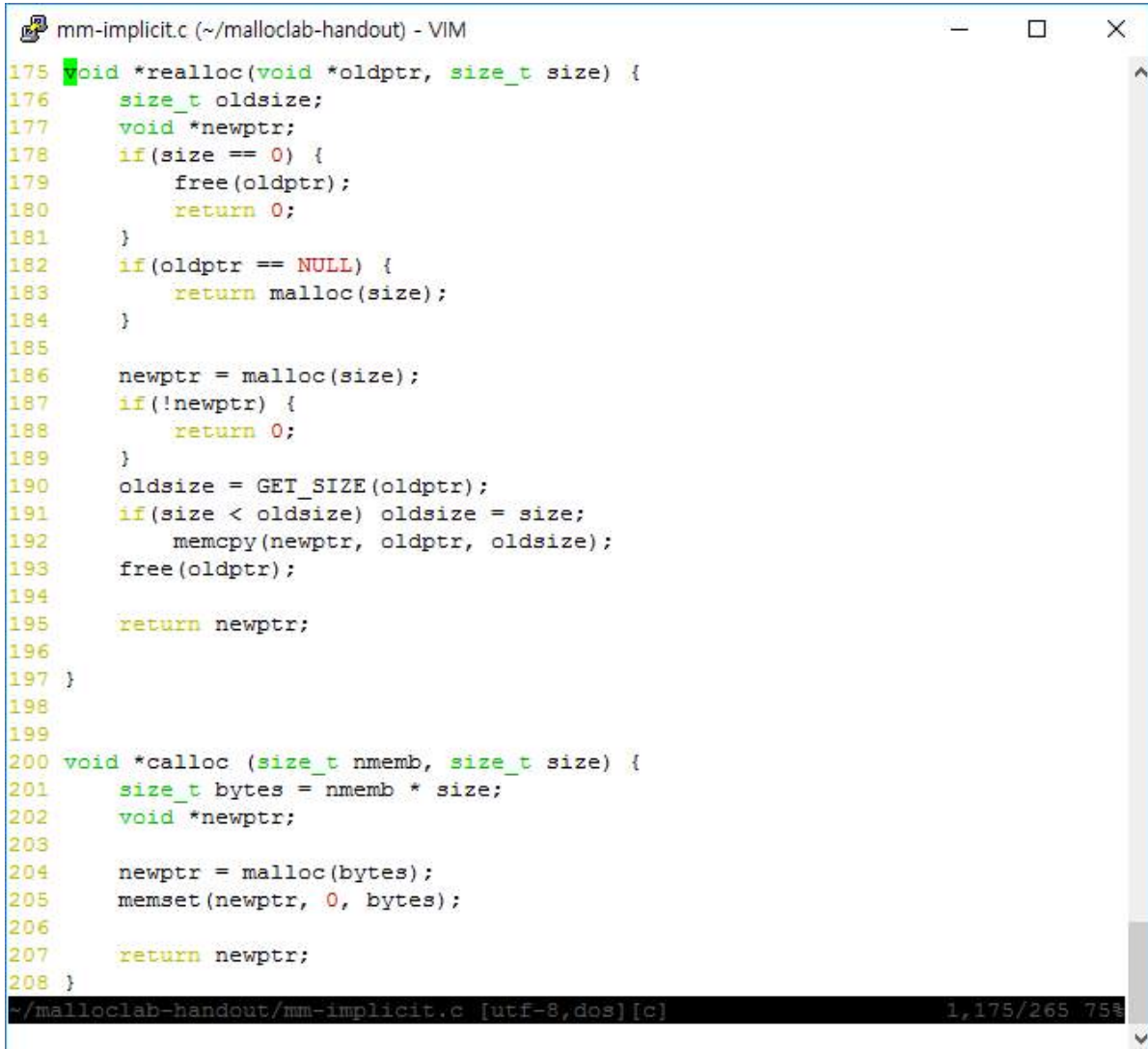
size가 8bit보다 작을경우는 바뀔사이즈가 8이되면되고 클경우에는 그에맞는 사이즈를 구해야한다

size+7/8하여 나온 몫에 8을 곱하면 해당 사이즈가 나온다

사이즈를 정했으면 find_fit을 써서 freeblock중에 좋은 곳을 찾는다 못찾을경우에는 종료.

밑에도 마찬가지로 에러가있는지 확인한후 할당을 마쳤으니 good_fit의 위치를 옮긴다

-realloc과 calloc



```
mm-implicit.c (~/.malloclab-handout) - VIM
175 void *realloc(void *oldptr, size_t size) {
176     size_t oldsize;
177     void *newptr;
178     if(size == 0) {
179         free(oldptr);
180         return 0;
181     }
182     if(oldptr == NULL) {
183         return malloc(size);
184     }
185
186     newptr = malloc(size);
187     if(!newptr) {
188         return 0;
189     }
190     oldsize = GET_SIZE(oldptr);
191     if(size < oldsize) oldsize = size;
192     memcpy(newptr, oldptr, oldsize);
193     free(oldptr);
194
195     return newptr;
196 }
197
198
199
200 void *calloc (size_t nmemb, size_t size) {
201     size_t bytes = nmemb * size;
202     void *newptr;
203
204     newptr = malloc(bytes);
205     memset(newptr, 0, bytes);
206
207     return newptr;
208 }
```

naive와 똑같이 구현하였는데

-realloc함수

realloc함수는 메모리를 재할당하는 함수이다.

size == 0, 경우 oldptr을 바로 free하고 들어온 포인터가 NULL일 경우 바로 size만큼 할당한다
그 외는 size가 입력이 되면 해당 size 만큼을 malloc을 통해 할당한 뒤, size 만큼 복사하여 그대로 붙여넣는다. 그리고 그전포인터는 을 free해주고 새로만든 포인터를 리턴한다

-calloc함수

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.

-free함수

```
mm-implicit.c + (~/.malloclab-handout) - VIM
161 void free (void *bp) {
162     if (bp == 0) return;
163
164     size_t size = GET_SIZE (HDRP (bp)); //헤더에서 size를 읽음
165
166     PUT (HDRP (bp), PACK (size, 0));
167     PUT (FTRP (bp), PACK (size, 0)); //bp의 header와 footer에 size 변경
168     coalesce (bp); //주위를 살펴보고 병합한다
169
170 }
```

사용중인 메모리를 다시 반환하는 과정이다 size값을 구한뒤 header와 footer를 0으로 바꾼뒤 주변의 빈블럭을 살핀후 병합한다.

-find_fit함수

```
mm-implicit.c (~/.malloclab-handout) - VIM
132
133 static void *find_fit (size_t asize) {
134
135     char *bp;
136     for (bp=good_fit; GET_SIZE (HDRP (bp)) > 0; bp= NEXT_BLKP (bp))
137         if (!GET_ALLOC (HDRP (bp)) && (asize <= GET_SIZE (HDRP (bp))))
138             return bp;
139     //가장 최근에 생성된 block부터 끝까지 서치하여 기록할 수 있는
140     //freeblock을 찾아내어 반환한다.
141
142     return NULL;
143 }
144
```

탐색하는 방법중에 best_fit이 가장 성능이 좋아 선택하였습니다.

현재포인터를 가르키는 good_fit을 이용하여 가장최근 생성된 block부터 heap의 끝까지 서치하면서 아직 freeblock이면서 asize만큼을 기록할 수 있는 공간이 있는지 검색하여 그곳을 리턴한다 없을 경우 NULL을 리턴

-coalesce함수

```
mm-implicit.c (~/.malloclab-handout) - VIM
226
227 static void *coalesce(void *bp){// 빈공간을 살펴 합치는 함수
228     size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKPTR(bp)));
229     size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKPTR(bp)));
230     size_t size = GET_SIZE(HDRP(bp));
231
232     if(prev_alloc && next_alloc){
233         return bp;
234     }//앞 뒤 블록이 둘 다 할당되어 있을 경우에는 바로 리턴
235     else if (prev_alloc && !next_alloc){
236         size += GET_SIZE(HDRP(NEXT_BLKPTR(bp)));
237         PUT(HDRP(bp), PACK(size, 0));
238         PUT(FTRP(bp), PACK(size, 0));
239     }//다음 블록이 freeblock일 경우 같이 병합
240     else if (!prev_alloc && next_alloc) {
241         size += GET_SIZE(HDRP(PREV_BLKPTR(bp)));
242         PUT(FTRP(bp), PACK(size, 0));
243         PUT(HDRP(PREV_BLKPTR(bp)), PACK(size, 0));
244         bp = PREV_BLKPTR(bp);
245     }//이전 블록이 freeblock일 경우 같이 병합
246     else {
247         size += GET_SIZE(HDRP(PREV_BLKPTR(bp))) +
248             GET_SIZE(FTRP(NEXT_BLKPTR(bp)));
249         PUT(HDRP(PREV_BLKPTR(bp)), PACK(size, 0));
250         PUT(FTRP(NEXT_BLKPTR(bp)), PACK(size, 0));
251         bp = PREV_BLKPTR(bp);
252     }//양쪽 다 비할당일 경우 둘 다 합치고 리턴한다
253
254
255     if (good_fit > bp) good_fit = bp; //현재 를 가리키는 포인터 값이 더 클 경우
256     //이전 블록도 같이 병합했다는 것이니 가리키는 포인터를 옮긴다
257     return bp;
258 }
259
```

coalesce 함수는 free나 extend를 할 때 해당 블록의 앞뒤를 살펴서 freeblock이 있으면 같이 size를 합쳐서 리턴하는 함수이다.

이전블록과 다음블록의 최하위 1bit를 읽어봐서 판단하는데 경우의 수는 모두 4가지로서

- 1:모두 사용중임으로 현재를 리턴하여 리턴한다
- 2:이전블록은 사용중이지만 다음블록은 freeblock임으로 size를 합친후 리턴한다.
- 3:다음블록은 사용중이지만 이전블록은 freeblock임으로 size를 합친후 리턴한다.
- 4:이전,다음블록은 둘다 freeblock임으로 모두 size를 합친후 리턴한다.

-extend_heap 함수

```
mm-implicit.c (~/.malloclab-handout) - VIM
90
91 static void *extend_heap(size_t words){//요청 받은 size의 빈블록을 만든다
92     char *bp;
93     size_t size;
94
95     size = (words < 2) ? (words+1) * WSIZE : words * WSIZE;
96     if ((long)(bp = mem_sbrk(size)) == -1)
97         return NULL;
98
99
100     PUT(HDRP(bp), PACK(size, 0)); /* Free block header */
101     PUT(FTRP(bp), PACK(size, 0)); /* Free block footer */
102     PUT(HDRP(NEXT_BLKP(bp)), PACK(0, 1)); /* New epilogue header */
103
104     return coalesce(bp);
105 }
106
~/malloclab-handout/mm-implicit.c [utf-8,dos][c] 0,90/265 35%
```

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

-place 함수

```
mm-implicit.c (~/.malloclab-handout) - VIM
147 static void place(void *bp, size_t asize){
148     size_t csize = GET_SIZE(HDRP(bp)); //사이즈를 구함
149
150     if ((csize-asize)>=(2*DSIZE)){//bp의 사이즈가 더 클 경우
151         PUT(HDRP(bp),PACK(asize,1));
152         PUT(FTRP(bp),PACK(asize,1));
153         bp=NEXT_BLKP(bp);
154         PUT(HDRP(bp),PACK(csize-asize,0));
155         PUT(FTRP(bp),PACK(csize-asize,0));
156     }else{//작을 경우
157         PUT(HDRP(bp),PACK(csize,1));
158         PUT(FTRP(bp),PACK(csize,1));
159         //해당 하는 사이즈만큼 옮겨준다
160     }
161 }
162
~/malloclab-handout/mm-implicit.c [utf-8,dos][c] 1,147/265 57%
```

해당위치에서 asize만큼 위치를 옮겨주는 함수이다 현재의 bp사이즈 asize만큼이 8보다 클경우와 작을경우를 나눠서 구한다.

할당중이던 블록의 크기가 asize보다 크게 되면 뒤에 남게 되는 블록을 처리해 주어야 메모리를 효율적으로 관리 할 수 있다. asize 까지 할당을 하며, 뒤에는 asize의 뺀 값(나머지) 만큼은 비할당 상태로 전환하여준다.

4.mdriver 결과

```
c201302423@host-192-168-0-5: ~/malloclab-handout
c201302423@host-192-168-0-5:~/malloclab-handout$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 2500.0 MHz

Results for mm malloc:
  valid  util   ops    secs    Kops  trace
  yes    34%    10    0.000000  52966 ./traces/malloc.rep
  yes    28%    17    0.000000  95505 ./traces/malloc-free.rep
  yes    96%    15    0.000000  69573 ./traces/corners.rep
* yes    81%   1494    0.000233   6410 ./traces/perl.rep
* yes    50%    118    0.000012   9603 ./traces/hostname.rep
* yes    87%  11913    0.004855   2454 ./traces/xterm.rep
* yes    97%   5694    0.000750   7588 ./traces/amptjp-bal.rep
* yes    95%   5848    0.002035   2873 ./traces/cccp-bal.rep
* yes    94%   6648    0.000763   8715 ./traces/cp-decl-bal.rep
* yes    93%   5380    0.000388  13860 ./traces/expr-bal.rep
* yes    66%  14400    0.000100143296 ./traces/coalescing-bal.rep
* yes    90%   4800    0.002033   2361 ./traces/random-bal.rep
* yes    55%   6000    0.000097  61620 ./traces/binary-bal.rep
10      81%  62295    0.011267   5529

Perf index = 52 (util) + 40 (thru) = 92/100
c201302423@host-192-168-0-5:~/malloclab-handout$
```