System Programming Project 3

담당 교수 : 김영재 교수님

이름 : 박성환

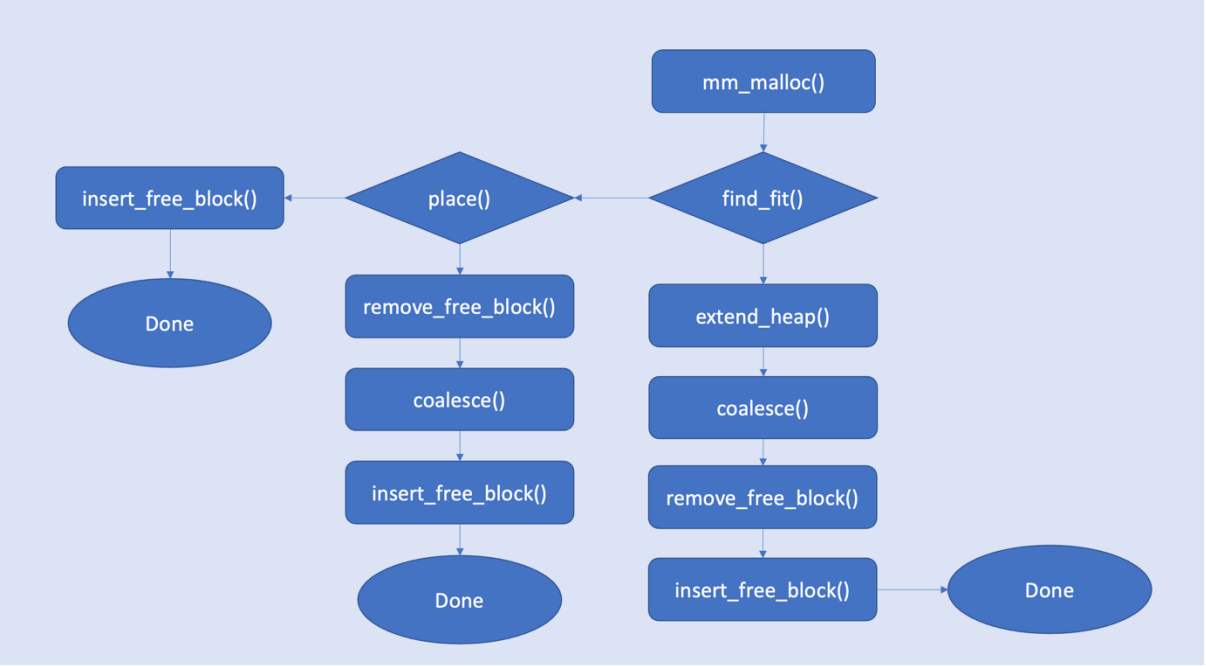
학번 : 20160169

1. 개발 목표

A. Dynamic Memory Allocator를 구현한다. mm_init(), mm_malloc(), mm_realloc(), mm_free() 함수를 중점적으로 구현하고 segregated list를 사용하여 성능을 높일 수 있도록 한다.

2. Flow chart

A. mm_malloc



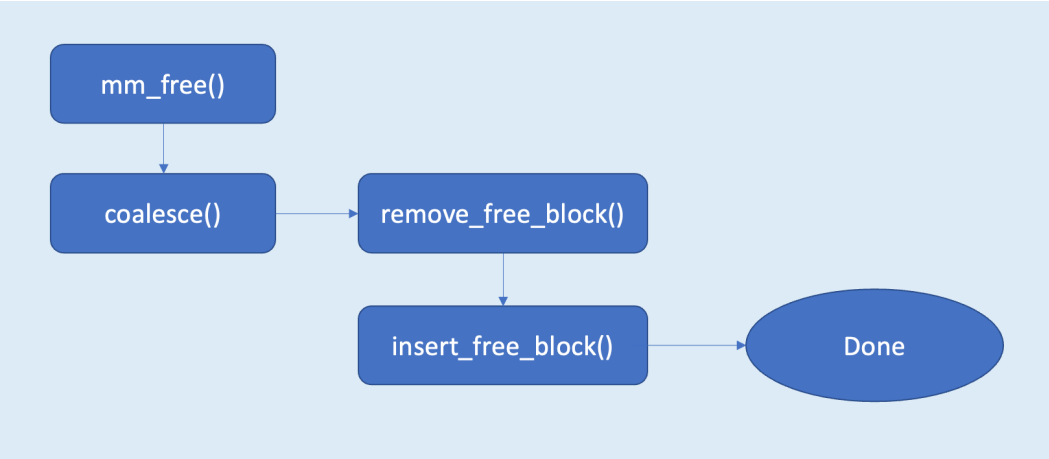
i. mm_malloc()를 호출하면 우선 find_fit() 함수로 사용 가능한 free block을 탐색한다.

ii. find_fit() 함수로 탐색에 성공하면 place() 함수를 통해 탐색한 free block에 메모리를 할당한다. place() 함수에서 free block을 전부 사용할지 또는 split 할지 정한다.

iii. split을 한 경우에는 해당 block을 segregated list에서 제거한 후 coalesce() 함수로 합친 free block을 segregated list에 다시 삽입한다.

iv. find_fit() 함수로 탐색에 실패하면 extend_heap() 함수로 heap 공간을 늘린다. 새로 할당 받은 free block을 coalesce하고 segregated list에 삽입한다.

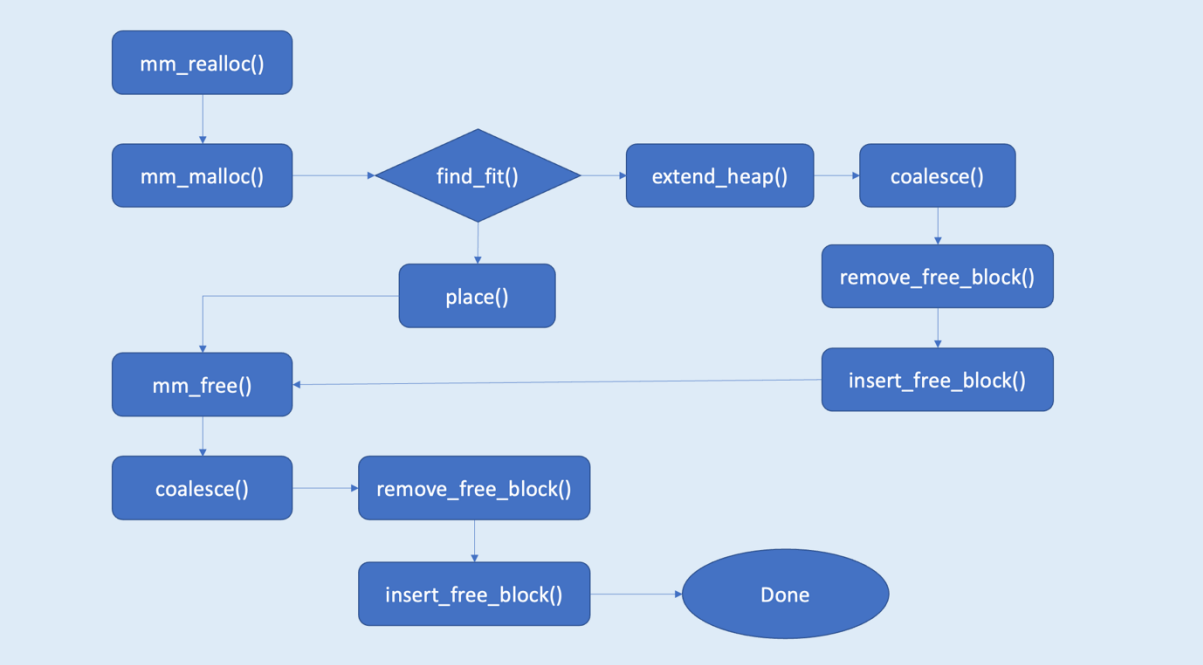
B. mm_free



i. mm_free() 함수를 호출하면 mm_free 함수 내에서 header와 footer를 해제한다.

ii. coalesce() 함수로 합친 free block을 segregated list에 삽입한다.

C. mm_realloc



i. mm_realloc()을 호출하면 우선 mm_malloc()이 호출된다. 위에서 설명한 mm_malloc()의 과정이 그대로 실행되고 mm_free()가 호출된다.

ii. mm_free()를 호출하면 위에서 설명한 mm_free()의 과정이 실행된다.

3. 함수, 매크로 및 전역 변수

A.
```
static char *seg_listp = NULL;
```

i. Segregated list에 대한 포인터이다.

ii. mem_init 함수에서 mem_sbrk() 함수를 통해 block 사이즈 별로 첫 번째 노드를 할당한다.

B.
```
#define SEG_FIND(p, idx)          *((char **)p + idx)
#define SEG_INIT_NULL(p, idx)     (SEG_FIND(p, idx) = NULL)
#define NEXT_SEG(bp)              *((char **)bp + 1)
#define PREV_SEG(bp)              *((char **)bp)
#define SEG_NEXT_OF_PREV(bp)      (NEXT_SEG(PREV_SEG(bp)))
#define SEG_PREV_OF_NEXT(bp)      (PREV_SEG(NEXT_SEG(bp)))
```

i. 위는 segregated list에 관련된 매크로이다.

ii. SEG_FIND로 각각의 segregated list의 첫 번째 block으로 이동한다.

iii. SEG_INIT_NULL 매크로는 mem_init() 함수에서 segregated list의 첫 번째 block을 NULL로 초기화한다.

iv. NEXT_SEG와 PREV_SEG를 이용해 segregated list 내에서 앞뒤 노드로 이동한다.

v. SEG_NEXT_OF_PREV는 현재 포인터가 가리키는 block 이전 block의 다음 block을 가리킨다. SEG_PREV_OF_NEXT는 현재 포인터가 가리키는 block 다음 block의 이전 block을 가리킨다.

C.

```
103  int mm_init(void) {
104
105      int i;
106      if ((seg_listp = mem_sbrk(SEG_LIST_NUM * WSIZE)) == (void *) - 1){
107          return -1;
108      }
109      for (i = 0; i < SEG_LIST_NUM; i++){
110          SEG_INIT_NULL(seg_listp, i);
111      }
112
113      if ((heap_listp = mem_sbrk(4 * WSIZE)) == (void *) - 1) {
114          return -1;
115      }
116
117      PUT(heap_listp, 0);                          /* Alignment padding */
118      PUT(heap_listp + (1*WSIZE), PACK(DSIZE, 1));  /* Prologue header */
119      PUT(heap_listp + (2*WSIZE), PACK(DSIZE, 1));  /* Prologue footer */
120      PUT(heap_listp + (3*WSIZE), PACK(0, 1));      /* Epilogue header */
121      heap_listp += (2*WSIZE);
122
123      /* Extend the empty heap by CHUNKSIZE bytes and creates the initial free block */
124      if (extend_heap(CHUNKSIZE/WSIZE) == NULL) {
125          return -1;
126      }
127      return 0;
128  }
```

i. mm_init 함수에서는 교재의 코드에서 segregated list를 초기화하는 부분(106 ~ 115 line)을 추가했다.

D.

```
170  void mm_free(void *bp){
171
172      size_t size = GET_SIZE(HDRP(bp));
173
174      FREE_HEADER(bp, size);
175      FREE_FOOTER(bp, size);
176
177      bp = coalesce(bp);
178      insert_free_block(bp);
179  }
```

i. mm_free 함수에서는 우선 해당 block의 header와 footer를 해제한다.

ii. 해당 block을 coalesce() 함수로 합친 후 insert_free_block() 함수로 segregated list에 삽입한다.

E.

```
134  void *mm_malloc(size_t size) {
135
136      size_t asize;           /* Adjusted block size */
137      size_t extendsize;      /* Amount to extend heap if no fit */
138      char *bp;
139
140      /* Ignore spurious requests */
141      if (size == 0)
142          return NULL;
143
144      /* Adjust block size to include overhead and alignment requests. */
145      if (size <= DSIZE) {
146          // enforce the minimum block size of 16 bytes
147          // 8 bytes for alignment and 8 bytes for the header and footer
148          asize = 2*DSIZE;
149      } else {
150          asize = DSIZE * ((size + (DSIZE) + (DSIZE-1)) / DSIZE);
151      }
152
153      /* Search the free list for a fit */
154      if ((bp = find_fit(asize)) != NULL) {
155          place(bp, asize);   // split
156          return bp;
157      }
158
159      extendsize = MAX(asize, CHUNKSIZE);
160      if ((bp = extend_heap(extendsize/WSIZE)) == NULL) {
161          return NULL;
162      }
163      place(bp, asize);
164      return bp;
165  }
```

i. mm_malloc 함수는 교재의 코드를 사용하였다.

F.

```
184  void *mm_realloc(void *ptr, size_t size) {
185
186      void *oldptr = ptr;
187      void *newptr;
188      size_t copySize;
189
190      newptr = mm_malloc(size);
191      if (newptr == NULL)
192          return NULL;
193
194      copySize = GET_SIZE(HDRP(oldptr));
195      copySize = size < copySize ? size : copySize;
196
197      memcpy(newptr, oldptr, copySize);
198      mm_free(oldptr);
199      return newptr;
200  }
```

i. mm_realloc() 함수는 제공된 함수를 사용했으며 194번 줄에 copySize의 값을 reallocation하기 전의 block 사이즈로 바꾸는 코드를 추가하였다.

G.

```
202  static void *extend_heap(size_t words) {
203
204      char *bp;
205      size_t size;
206      /*
207          Allocate an even number of words to maintain alignment
208          Rounds up the requested size to the nearest multiple of 2 words (8 bytes)
209      */
210      size = (words % 2) ? (words + 1) * WSIZE : words * WSIZE;
211      if ((long)(bp = mem_sbrk(size)) == -1) {    // request the additional heap space from the memory
212          return NULL;
213      }
214
215      /* Initialize free block header/footer and the epilogue header */
216      FREE_HEADER(bp, size);                  /* Free block header */
217      FREE_FOOTER(bp, size);                  /* Free block footer */
218      SET_HEADER(NEXT_BLKP(bp), 0);           /* New epilogue header */
219
220      /* Coalesce if the previous block was free */
221      bp = coalesce(bp);
222      insert_free_block(bp);
223      return bp;
224  }
```

i. extend_heap 함수는 교재의 코드에서 221, 222번 줄만 변경하였다. Coalesce() 함수로 앞뒤의 free block과 합쳐진 새 free block을 반환 받고 이를 insert_free_block() 함수를 통해 segregated list에 삽입한다.

H.

```
226  static void *coalesce(void *bp) {
227
228      size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
229      size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
230      size_t size = GET_SIZE(HDRP(bp));
231
232      if (prev_alloc && !next_alloc) {
233          size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
234          remove_free_block(NEXT_BLKP(bp));
235          FREE_HEADER(bp, size);
236          FREE_FOOTER(bp, size);
237      }
238
239      else if (!prev_alloc && next_alloc) {
240          size += GET_SIZE(HDRP(PREV_BLKP(bp)));
241          remove_free_block(PREV_BLKP(bp));
242          FREE_HEADER(PREV_BLKP(bp), size);
243          FREE_FOOTER(bp, size);
244          bp = PREV_BLKP(bp);
245      }
246
247      else if (!prev_alloc && !next_alloc) {
248          size += GET_SIZE(HDRP(PREV_BLKP(bp))) + GET_SIZE(FTRP(NEXT_BLKP(bp)));
249          remove_free_block(PREV_BLKP(bp));
250          remove_free_block(NEXT_BLKP(bp));
251          FREE_HEADER(PREV_BLKP(bp), size);
252          FREE_FOOTER(NEXT_BLKP(bp), size);
253          bp = PREV_BLKP(bp);
254      }
255
256      return bp;  /* This include the case when prev and next are all allocated */
257  }
```

i. coalesce() 함수는 free한 block의 앞뒤에 allocated 되어 있지 않은 block이 있을 경우 block들을 합친다.

ii. 첫 번째 if문은 해당 block 뒤의 block이 free block인 경우이고 234번 줄에 그 block을 segregated list에서 제거하는 코드를 추가하였다.

iii. 두 번째 if문은 해당 block 앞의 block이 free block인 경우이고 해당 block을 앞의 block과 합쳐야 하기 때문에 241번 줄에 해당 block을 segregated list에서 제거하는 코드를 추가하였다.

iv. 마지막 if문은 해당 block의 앞뒤 block이 모두 free block인 경우이고 그 block들을 모두 segregated list에서 제거하는 코드를 추가하였다.

I.

```
259  static void *find_fit(size_t asize) {
260
261      /* First-fit search */
262      unsigned int idx = find_seg_idx(asize);
263
264      while (idx < SEG_LIST_NUM) {
265          void* bp = SEG_FIND(seg_listp, idx);
266          while (bp) {
267              if (asize <= GET_SIZE(HDRP(bp))) {
268                  return bp;
269              }
270              bp = NEXT_SEG(bp);
271          }
272          idx++;
273      }
274      return NULL; /* NO fit */
275  }
```

i. find_fit 함수는 block의 사이즈에 맞는 segregated list를 찾는 함수이다. find_seg_idx() 함수를 통해 asize 보다 작은 segregated list 중에서 가장 큰 리스트에 대한 인덱스를 반환 받고 그 리스트부터 시작하여 asize보다 큰 free block을 탐색하여 반환한다. 이를 만족하는 Free block이 없는 경우에는 NULL을 반환한다.

J.

```
277  static void place(void *bp, size_t asize) {
278
279      size_t csize = GET_SIZE(HDRP(bp));
280
281      if ((csize - asize) >= (2 * DSIZE)) {
282          remove_free_block(bp);
283          SET_HEADER(bp, asize);
284          SET_FOOTER(bp, asize);
285          bp = NEXT_BLKP(bp);
286          FREE_HEADER(bp, csize - asize);
287          FREE_FOOTER(bp, csize - asize);
288          bp = coalesce(bp);
289          insert_free_block(bp);
290      }
291      else {
292          remove_free_block(bp);
293          SET_HEADER(bp, csize);
294          SET_FOOTER(bp, csize);
295      }
296  }
```

i. place 함수는 malloc한 block을 free block에 할당하는 함수이다.

ii. 우선 281 ~ 289번 줄은 split 후 남은 block이 최소 block 크기인 16byte 보다 크거나 같은 경우이다. 일단 현재 bp가 가리키는 block은 할당된 block 이기에 segregated list에서 제거하고 footer와 header를 allocated로 설정한다. 그리고 bp를 split한 후 남은 block을 가리키게 하고 해당 block의 footer와 header를 해제한다. 마지막으로 해당 block을 segregated list에 삽입한다.

iii. Split 후 남은 block이 최소 block 크기보다 작을 경우엔 해당 block을 segregated list에서 제거하면 된다.

K.

```
298  static void insert_free_block(void *bp) {
299
300      size_t csize = GET_SIZE(HDRP(bp));
301      unsigned int idx = find_seg_idx(csize);
302      char** targ = &(SEG_FIND(seg_listp, idx));
303
304      PREV_SEG(bp) = NEXT_SEG(bp) = NULL;
305
306      if (*targ) {
307          NEXT_SEG(bp) = *targ;
308          PREV_SEG(*targ) = bp;
309      }
310      *targ = bp;
311  }
```

i. insert_free_block 함수는 파라미터로 받은 block을 segregated list에 삽입하는 함수이다.

L.

```
313  static void remove_free_block(void *bp) {
314
315      size_t csize = GET_SIZE(HDRP(bp));
316      unsigned int idx = find_seg_idx(csize);
317
318      if (bp == SEG_FIND(seg_listp, idx)) {
319          SEG_FIND(seg_listp, idx) = NEXT_SEG(bp);
320      }
321      else if (PREV_SEG(bp) && NEXT_SEG(bp)) {
322          SEG_NEXT_OF_PREV(bp) = NEXT_SEG(bp);
323          SEG_PREV_OF_NEXT(bp) = PREV_SEG(bp);
324      }
325      else if (PREV_SEG(bp) && !NEXT_SEG(bp)) {
326          SEG_NEXT_OF_PREV(bp) = NULL;
327      }
328      else if (NEXT_SEG(bp) && !PREV_SEG(bp)) {
329          SEG_PREV_OF_NEXT(bp) = NULL;
330      }
331  }
```

i. remove_free_block 함수는 파라미터로 받은 block을 segregated list에 제거하는 함수이다.

M.

```
333  static unsigned int find_seg_idx(size_t asize) {
334
335      size_t target_size = asize / DSIZE;
336      int idx = 1;
337
338      while (idx < SEG_LIST_NUM) {
339          if (target_size < (1<<idx)) {
340              return idx - 1;
341          }
342          idx++;
343      }
344
345      return idx - 1;
346  }
```

i. find_seg_idx 함수는 파라미터로 받은 block size 값에 해당하는 segregated list의 번호를 반환한다. 해당 번호는 SEG_FIND 매크로에서 사용된다.

4. mm_check

A.

```
348  static int mm_check() {     // return 0 if error, else return 1
349
350      int result = 1, tmp;
351
352      /* Check if all free blocks are in valid heap */
353      result = traverse(check_blocks_in_heap);
354
355      /* Check if all free blocks are in segregated list */
356      tmp = check_free_blocks_in_seg();
357      if (result && !tmp) result = tmp;
358
359      /* Check if all free blocks in segregated list are marked free */
360      tmp = traverse(check_free_block_mark);
361      if (result && !tmp) result = tmp;
362
363      /* Check if coalesce working */
364      tmp = traverse(check_coalesce);
365      if (result && !tmp) result = tmp;
366
367      return result;
368  }
```

i. mm_check 함수는 위와 같다. 위 사진의 주석과 같이 다음 사항들을 확인한다.

- ✓ Segregated list의 모든 free block들이 유효한 heap 영역에 있는지 확인.
- ✓ 모든 free block들이 segregated list에 있는지 확인
- ✓ Segregated list의 모든 block들의 헤더에 free 표시가 되어 있는지 확인
- ✓ Coalescing에 잘 작동하는지 확인

B.

```
401  static int traverse(block_action_func *func) {
402      int i, result = 1;
403
404      for (i = 0; i < SEG_LIST_NUM; i++){
405          void* seg_p = SEG_FIND(seg_listp, i);
406          while (seg_p) {
407              int val = func(seg_p);
408              if (result && !val) result = val;
409
410              seg_p = NEXT_SEG(seg_p);
411          }
412      }
413      return result;
414  }
```

i. 우선 traverse 함수는 위와 같다. Segregated list를 순회하며 파라미터로 받은 함수 포인터에 대한 함수를 실행하여 확인하고자 하는 사항을 수행한다.

C.

```
416  static int check_blocks_in_heap(void *seg_p) {
417      if (seg_p < mem_heap_lo() || seg_p > mem_heap_hi()) {
418          printf("%p: Error: free block not in valid heap address. \n", seg_p);
419          return 0;
420      }
421      return 1;
422  }
```

i. check_blocks_in_heap 함수는 traverse 함수에서 작동하는 함수로 seg_p 파라미터는 traverse 함수로부터 받은 segregated list의 각각의 free block들을 의미한다. 해당 free block이 유효한 heap 영역에 있는지 검사한다.

D.

```
424  static int check_free_block_mark(void *seg_p) {
425      if (GET_ALLOC(HDRP(seg_p)) == 1) {
426          printf("%p: Error: block in segregated list but not marked free. \n", seg_p);
427          return 0;
428      }
429      return 1;
430  }
```

i. check_free_block_mark 함수는 traverse 함수에서 작동하는 함수로 seg_p 파라미터는 traverse 함수로부터 받은 segregated list의 각각의 free block들을 의미한다. 해당 free block의 header가 allocated로 비트 마스킹이 되어 있으면 오류를 발생시킨다.

E.

```
434  static int check_coalesce (void *seg_p) {
435      void *prev = PREV_BLKP(seg_p);
436      void *next = NEXT_BLKP(seg_p);
437      if (prev && !GET_ALLOC(HDRP(prev))) {
438          printf("%p, %p: Error: Contiguous free blocks but not coalesced. \n", prev, seg_p);
439          return 0;
440      }
441      if (next && !GET_ALLOC(HDRP(next))) {
442          printf("%p, %p: Error: Contiguous free blocks but not coalesced. \n", seg_p, next);
443          return 0;
444      }
445      /*
446      if (next)
447          printf("%p: Coalescing done. Next_seg_alloc : %d \n", seg_p, GET_ALLOC(HDRP(next)));
448      */
449      return 1;
450  }
```

i. Check_coalesce 함수는 traverse 함수에서 작동하는 함수로 seg_p 파라미터는 traverse 함수로부터 받은 segregated list의 각각의 free block들을 의미한

다. 해당 free block의 앞 뒤로 인접한 block이 free인 경우 coalescing이 제대로 이루어지지 않은 것이므로 오류를 발생시킨다.

F.

```
370  static int check_free_blocks_in_seg() {
371
372      int i;
373      void *hp;
374
375      for (hp = heap_listp; GET_SIZE(HDRP(hp)); hp = NEXT_BLKP(hp)) {
376          if (!GET_ALLOC(HDRP(hp))) {
377
378              int found = 0;
379              for (i = 0; i < SEG_LIST_NUM && !found; i++) {
380                  void* seg_p = SEG_FIND(seg_listp, i);
381                  while (seg_p) {
382                      if (seg_p == hp) {
383                          found = 1;
384                          break;
385                      }
386                      seg_p = NEXT_SEG(seg_p);
387                  }
388              }
389              if (!found) {
390                  printf("%p: Error: free block but not in segregated list. \n", hp);
391              }
392              else {
393                  //printf("%p: FOUND \n", hp);
394                  return 1;
395              }
396          }
397      }
398      return 0;
399  }
```

i. check_free_blocks_in_seg 함수는 heap_listp로부터 시작하여 모든 block들을 순회하며 해당 block이 free block인 경우에 그 block을 segregated list에서 탐색한다. 만약 탐색에 실패하면 free block과 segregated list의 consistency가 위반된 것이므로 오류를 발생시킨다.