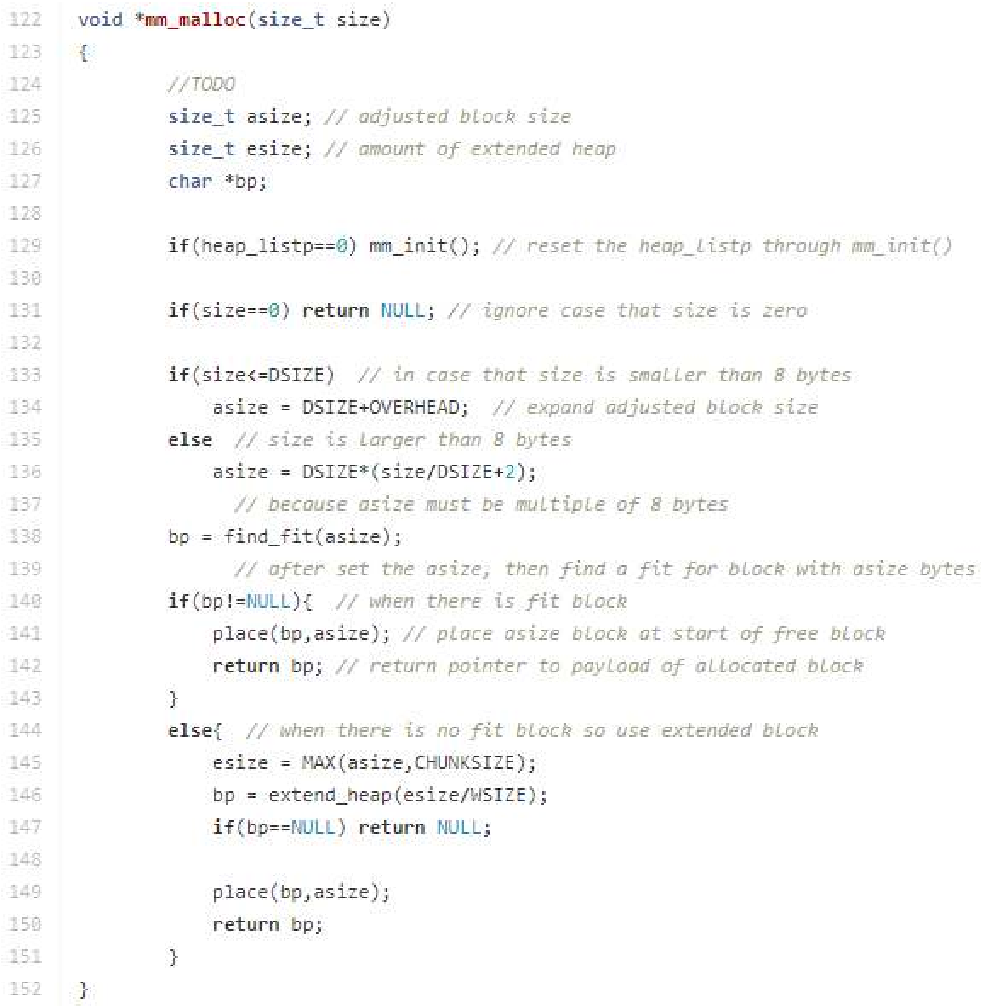
SP_Assignment5 Report 20131218 박기훈

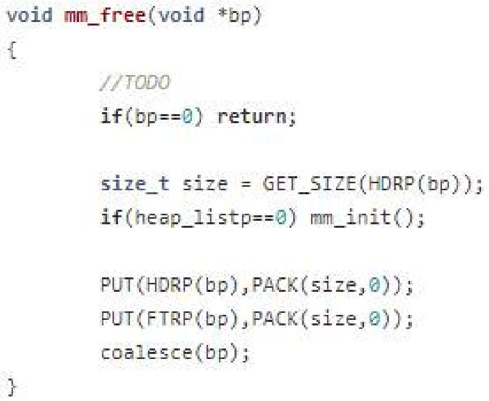
1. void *mm_malloc(size_t size)



```
122 void *mm_malloc(size_t size)
123 {
124         //TODO
125         size_t asize; // adjusted block size
126         size_t esize; // amount of extended heap
127         char *bp;
128 
129         if(heap_listp==0) mm_init(); // reset the heap_listp through mm_init()
130 
131         if(size==0) return NULL; // ignore case that size is zero
132 
133         if(size<=DSIZE)  // in case that size is smaller than 8 bytes
134             asize = DSIZE+OVERHEAD;  // expand adjusted block size
135         else  // size is larger than 8 bytes
136             asize = DSIZE*(size/DSIZE+2);
137               // because asize must be multiple of 8 bytes
138         bp = find_fit(asize);
139               // after set the asize, then find a fit for block with asize bytes
140         if(bp!=NULL){  // when there is fit block
141             place(bp,asize); // place asize block at start of free block
142             return bp; // return pointer to payload of allocated block
143         }
144         else{  // when there is no fit block so use extended block
145             esize = MAX(asize,CHUNKSIZE);
146             bp = extend_heap(esize/WSIZE);
147             if(bp==NULL) return NULL;
148 
149             place(bp,asize);
150             return bp;
151         }
152 }
```

The mm_malloc function is key function for dynamic memory allocation. In main function, we call the malloc function always with size. So we can allocate memory dynamically, suitable for parameter ‘size.’ At line 125, ‘asize’ means the adjusted size which will be the heap block size. So at line 129, 131, check an exception(heap_listp points to first block), then adjust the heap block size(asize). The reason of extend by 8bytes more is that block size includes the headers, payload, and any padding. After adjusting the asize, get the pointer to certain block with asize bytes and store to ‘bp’ pointer. The role of find_fit function is searching the free blocks which is suitable for asize, then selecting the first free block and returns its address. At line 140 and 144, we should divide 2 cases whether find_fit function search the suitable free block successfully. If success, then by using place function, places asize block at start of free block and returns pointer to the beginning of the payload. Otherwise, if fail, then we should extend heap to make memory affordable to contain asize block. Then also places asize block, and returns pointer. That is, malloc function gets the size from main function, adjusts size into asize, searches suitable heap block, then returns the pointer to the beginning of payload of that heap block.

2. void *mm_free(void *bp)



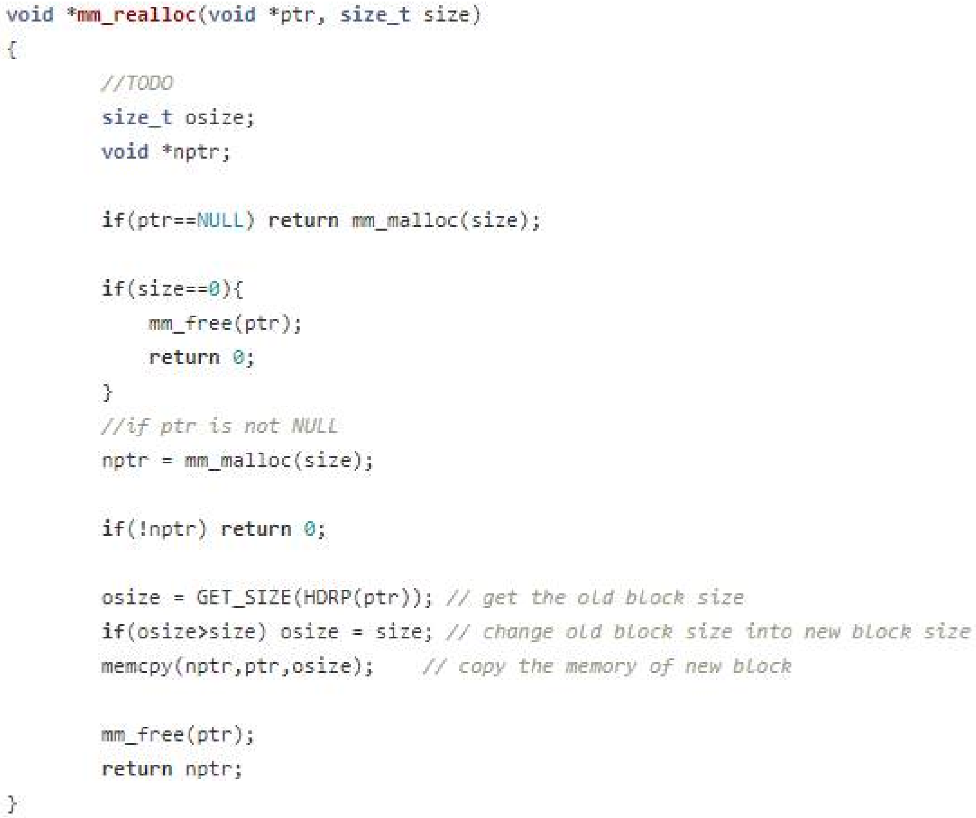
```
void mm_free(void *bp)
{
        //TODO
        if(bp==0) return;

        size_t size = GET_SIZE(HDRP(bp));
        if(heap_listp==0) mm_init();

        PUT(HDRP(bp),PACK(size,0));
        PUT(FTRP(bp),PACK(size,0));
        coalesce(bp);
}
```

The mm_free function is very important in dynamic memory allocation. We should free the block pointed to by ‘bp’ whenever call the malloc function, because os doesn’t manage the heap memory automatically. By using GET_SIZE function, get the size of block which pointed by parameter bp. Then free the block pointers, HDRP(bp) and FTRP(bp). (HDRP is header ptr, FTRP is footer ptr.) At last, call the coalesce function, whose role is just recombining splitting free blocks and returns the pointer to coalesced block.

3. void *mm_malloc(size_t size)



```
void *mm_realloc(void *ptr, size_t size)
{
        //TODO
        size_t osize;
        void *nptr;

        if(ptr==NULL) return mm_malloc(size);

        if(size==0){
            mm_free(ptr);
            return 0;
        }
        //if ptr is not NULL
        nptr = mm_malloc(size);

        if(!nptr) return 0;

        osize = GET_SIZE(HDRP(ptr)); // get the old block size
        if(osize>size) osize = size; // change old block size into new block size
        memcpy(nptr,ptr,osize);    // copy the memory of new block

        mm_free(ptr);
        return nptr;
}
```

The role of mm_realloc function is changing the allocated memory. So it takes two parameters, *ptr and size. Parameter ‘ptr’ points to old block, and ‘size’ is size of new block. Before change, we check several things. At first, if ptr is NULL, then just returns mm_malloc(size). Because there is no old block, we just make new block by using malloc function. Next, if size is zero, then call the free(ptr) to free the old block and return 0. If belongs to above both cases, then we change the size of old block memory. So by using GET_SIZE function, store the old block size into ‘osize.’ Then compare the osize with size (new block size), because new block size is min(osize,size). After setting the size, then copy the memory of newblock through memcpy. At last, frees the old blocks and returns the pointer to new block.