2016년 시스템 프로그래밍

-MallocLab-

제출일자	2016.12.12.
이 름	정윤수
학 번	201302482
분 반	00

mm-navie의 대한 설명

(1)naive에 사용된 매크로

```
#define ALIGNMENT 8

/* rounds up to the nearest multiple of ALIGNMENT */
#define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7)


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

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

ALIGNMENT : Alignment를 double word로 설정을 함으로 8을 할당을 한다.
ALIGN(size) : 입력을 한 사이즈를 가장 rkRKdns ALIGNMENT의 배수로 변경을 시킨다.
SIZE_T_SIZE : size_t형 변수의 크기를 ALIGN 한 값을 반환,gdb로 확인시 8의 값을 가진다.
SIZE_PTR(p) : 현재가리키고있는 포인터에서 -8만큼 떨어진 곳을 반환을 해준다.

(2)함수

malloc() :

```
void *malloc(size_t size)
{
  int newsize = ALIGN(size + SIZE_T_SIZE);
  unsigned char *p = mem_sbrk(newsize);
  //dbg_printf("malloc %u => %p\n", size, p);

  if ((long)p < 0)
    return NULL;
  else {
    p += SIZE_T_SIZE;
    *SIZE_PTR(p) = size;
    return p;
  }
}
```

naive에서의 malloc 함수는 newsize의 값으로 size+SIZE_TSIZE의 값을 배정을 시켜주고 p의 값으로는 mem_sbrk()함수를 호출을 하여 newsize만큼 메모리를 할당을 한다. p에는 메모리가 할당되기 전 주소가 지정이 된다. p가 0보다 작으면 메모리를 할당을할 공간이 부족하여 할당을 못했다는 것 임으로 NULL을 반환한다 p값이 제대로 할당이 되었다면 p를 double word만큼 한 칸 옮기고 SIZE_PTR()매크로를 사용을하여 p가 가리키고 있는곳에 -8만큼 떨어진 포인터에 size의 값을 저장을 한다. 그 후 p를 반환을 해준다.

free() :

할당을 한 메모리를 반환을 시켜주는 함수이다. naive에서는 구현이 안되어있어 성능이 매우 떨어진다.

init() :

naive에서 init()함수는 구현이 되어있지 않다.

mem_sbrk() :

heap 영역을 확장을 시켜주는 함수로 반환값으로 새로 확장이 되어진 공간의 첫 주소를 반환을 해준다. 이 함수를 사용하여 확장을 한 heap의 크기를 감소를 시킬수는 없다.

realloc() :

```
void *realloc(void *oldptr, size_t size)
{
  size_t oldsize;
  void *newptr;

  /* If size == 0 then this is just free, and we return NULL. */
  if(size == 0) {
    free(oldptr);
    return 0;
  }

  /* If oldptr is NULL, then this is just malloc. */
  if(oldptr == NULL) {
    return malloc(size);
  }

  newptr = malloc(size);

  /* If realloc() fails the original block is left untouched  */
  if(!newptr) {
    return 0;
  }

  /* Copy the old data. */
  oldsize = *SIZE_PTR(oldptr);
  if(size < oldsize) oldsize = size;
  memcpy(newptr, oldptr, oldsize);

  /* Free the old block. */
  free(oldptr);

  return newptr;
}
```

할당이 되어진 공간을 다른 사이즈로 같은 내용을 가진 채로 할당을 해주는 함수이다.
자신이 요청을 한 size값이 0이면 free를 해주고 할당을 요청을 하는 변수가 NULL이면 malloc함수를 이용을 하여 메모리를 할당을 해준다. size의 크기를 가진 메모리를 할당을 하여 memcpy()함수를 이용을하여 oldsize 만큼 값을 복사를 한다. 원래 할당된 메모리를 가지고 있던 공간은 free()함수를 이용을하여 반환을 해주고 새로 할당을 한 변수를 반환을 시켜준다.

(3)결과

```
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  0.000000150602 ./traces/malloc.rep
   yes     77%      17  0.000000171371 ./traces/malloc-free.rep
   yes    100%      15  0.000000137867 ./traces/corners.rep
 * yes     71%    1494  0.000009171165 ./traces/perl.rep
 * yes     68%     118  0.000001169637 ./traces/hostname.rep
 * yes     65%   11913  0.000069172998 ./traces/xterm.rep
 * yes     23%    5694  0.000055103671 ./traces/amptjp-bal.rep
 * yes     19%    5848  0.000057101954 ./traces/cccp-bal.rep
 * yes     30%    6648  0.000070 94837 ./traces/cp-decl-bal.rep
 * yes     40%    5380  0.000051105475 ./traces/expr-bal.rep
 * yes      0%   14400  0.000165 87301 ./traces/coalescing-bal.rep
 * yes     38%    4800  0.000042114732 ./traces/random-bal.rep
 * yes     55%    6000  0.000058103726 ./traces/binary-bal.rep
10         41%   62295  0.000576108094

Perf index = 26 (util) + 40 (thru) = 66/100
```

mm-implicit 의 대한 설명

(1)implicit에 사용된 매크로

```
static char *heap_listp=0;
static char *rover;
```

heap_listp : firstblock을 가리키기 위해서 선언

rover : nextfit을 구현을 하기 위해서 선언

```
#define ALIGNMENT 8
#define WSIZE 4
#define DSIZE 8
#define CHUNKSIZE (1<<12)
#define OVERHEAD 8
#define MAX(x,y) ((x) >(y) ? (x):(y))
#define PACK(size,alloc) ((size) | (alloc))
#define GET(p) (*(unsigned int *)(p))
#define PUT(p,val) (*(unsigned int *)(p) =(val))
#define GET_SIZE(p) (GET(p) & ~0x7)
#define GET_ALLOC(p) (GET(p) & 0x1)
#define HDRP(bp) ((char*)(bp) - WSIZE)
#define FTRP(bp) ((char*)(bp) + GET_SIZE(HDRP(bp))-DSIZE)
#define NEXT_BLKP(bp) ((char*)(bp) + GET_SIZE((char*)(bp)-WSIZE))
#define PREV_BLKP(bp) ((char*)(bp) - GET_SIZE((char*)(bp)-DSIZE))
/* rounds up to the nearest multiple of ALIGNMENT */
#define ALIGN(p) (((size_t)(p) + (ALIGNMENT-1)) & ~0x7)
```

ALIGNMENT : Alignment를 double word로 설정을 함으로 8을 할당을 한다.

ALIGN(size) : 입력을 한 사이즈를 ALIGNMENT의 배수로 변경을 시킨다.

WSIZE : word size 4

DSIZE : double word size 8

CHUNKSIZE : 2^{12}=4096 heap영역을 확장을 할 때 사용이 되어진다.

MAX(x,y) : x와 y중 큰 값을 반환

GET(p) : p의 주소값을 반환을 한다.

PUT(p,val) : p가 가리키고 있는곳에 val값을 저장을 한다.

GET_SIZE(p) : p의 끝에있는 1비트를 제외한 비트들과 and연산을 하여 크기를 반환을 해준다.

GET_ALLOC(p) : p의 끝에있는 1비트와 and연산을 하여 alloc값을 반환을 해준다.

HDRP(bp) : bp가 가리키고 있는 공간에 header주소를 반환을 해준다.

FTRP(bp) : bp가 가리키고 있는 공간이 footer주소를 반환 해준다.

NEXT_BLKP(bp) : bp가 가리키고 있는 공간의 다음 block의 주소를 반환을 해준다.

PREV_BLKP(bp) : bp가 가리키고 있는 공간의 그 전 block의 주소를 반환을 해준다.

(2)함수

init() :

```
int mm_init(void) {

    if((heap_listp = mem_sbrk(4*WSIZE)) == NULL)
        return -1;
    PUT(heap_listp,0);
    PUT(heap_listp + (1*WSIZE),PACK(OVERHEAD,1));
    PUT(heap_listp + (2*WSIZE),PACK(OVERHEAD,1));
    PUT(heap_listp + (3*WSIZE),PACK(0,1));
    heap_listp +=(2*WSIZE);
    rover = heap_listp;
    if((extend_heap(CHUNKSIZE / WSIZE)) == NULL)
        return -1;

    return 0;
}
```

초기에 빈 heap을 생성을 하여 초기화를 해주는 함수이다. mem_sbrk()함수를 이용을 하여 4워드 만큼 공간을 만든 후 그곳에 extend_heap()함수를 호출하여 heap의 사이즈를 확장을 한다.

extend_heap() :

```
static void* extend_heap(size_t words){

    char *bp;
    size_t size;

    size = (words %2) ? (words+1)*WSIZE : words*WSIZE;
    if((long)(bp=mem_sbrk(size))==-1)
        return NULL;
    PUT(HDRP(bp),PACK(size,0));
    PUT(FTRP(bp),PACK(size,0));
    PUT(HDRP(NEXT_BLKP(bp)),PACK(0,1));
    return coalesce(bp);
}
```

extend_heap은 heap이 초기화 될 때와 malloc()함수를 시행할 때 적당한 fit를 찾지 못하였을 때 호출이 된다.

mm_free() :

```
void free (void *ptr) {
    if(ptr == 0) return;
    size_t size = GET_SIZE(HDRP(ptr));
    PUT(HDRP(ptr),PACK(size,0));
    PUT(FTRP(ptr),PACK(size,0));
    coalesce(ptr);
}
```

할당이 되어진 메모리 공간을 반환을 해주는 함수이다. 메모리 공간의 header와 footer의 alloc값을 0으로 설정을 해주고 coalesce()함수를 이용을 하여 근처의 가용블록들과 합친다.

coalesece() :

```
void* coalesce(void* bp){

    size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
    size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
    size_t size = GET_SIZE(HDRP(bp));

    if(prev_alloc && next_alloc)
        return bp;
    else if(prev_alloc && !next_alloc){
        size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
        PUT(HDRP(bp),PACK(size,0));
        PUT(FTRP(bp),PACK(size,0));

    }
    else if(!prev_alloc && next_alloc){
        size +=GET_SIZE(HDRP(PREV_BLKP(bp)));
        PUT(FTRP(bp),PACK(size,0));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size,0));
        bp = PREV_BLKP(bp);
    }
    else{
        size += GET_SIZE(HDRP(PREV_BLKP(bp))) + GET_SIZE(FTRP(NEXT_BLKP(bp)));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size,0));
        PUT(FTRP(NEXT_BLKP(bp)), PACK(size,0));
        bp = PREV_BLKP(bp);
    }
    if((rover > (char *)bp)&&(rover<NEXT_BLKP(bp)))
        rover = bp;
    return bp;
}
```

가용블록의 근처의 있는 가용블록들과 합쳐주는 역할을 해주는 함수로 이전과 다음에 가용블록이 없을때는 자기 자신을 반환하지만 근처에 가용블록이 있다면 합친후 반환을 해준다.

mm_malloc() :

```
void *malloc (size_t size) {
    size_t asize;
    size_t extendsize;
    char *bp;

    if(size==0)
        return NULL;
    if(size<=DSIZE)
        asize = 2*DSIZE;
    else
        asize = DSIZE*((size+(DSIZE)+(DSIZE-1))/DSIZE);

    if((bp = find_fit(asize)) != NULL){
        place(bp,asize);
        return bp;
    }
    extendsize = MAX(asize,CHUNKSIZE);
    if((bp = extend_heap(extendsize/WSIZE))==NULL)
        return NULL;
    place(bp,asize);
    return bp;
}
```

메모리를 할당을 해주는 함수로 입력한 사이즈가 최소 사이즈를 만족을 하지못하면 최소 사이즈 만큼 키워준다 최소사이즈를 만족을 하면 alignment에 맞는 값으로 환산을 해주고 find_fit함수를 이용을하여 맞는 공간을 찾아 place()함수를 이용을 하여 찾은 자리에 size값을 지정을 해준다 find_fit()함수 찾지 못하였다면 extend_heap함수를 호출을 하여 메모리 공간을 늘리고 늘린 공간에 메모리를 할당을 한다.

find_fit() :

```
static void* find_fit(size_t asize){
    char *oldrover = rover;
    for(;GET_SIZE(HDRP(rover)) >0;rover = NEXT_BLKP(rover))
        if(!GET_ALLOC(HDRP(rover)) && (asize <=GET_SIZE(HDRP(rover))))
            return rover;
    for(rover = heap_listp;rover < oldrover;rover = NEXT_BLKP(rover))
        if(!GET_ALLOC(HDRP(rover)) && (asize <= GET_SIZE(HDRP(rover))))
            return rover;

    return NULL;
}
```

알맞은 메모리공간을 찾는 find_fit()함수이다. find_fit함수를 next_fit방식으로 구현을 하였다. next_fit방식은 맨 처음부터 탐색을 하는 first_fit방식과는 달리 찾은 자리부터 다시 탐색을 시작을 하는 방식이다.

place() :

```
static void place(void *bp,size_t asize){
    size_t csize = GET_SIZE(HDRP(bp));

    if((csize - asize) >= (2*DSIZE)){
        PUT(HDRP(bp),PACK(asize,1));
        PUT(FTRP(bp),PACK(asize,1));
        bp = NEXT_BLKP(bp);
        PUT(HDRP(bp),PACK(csize-asize,0));
        PUT(FTRP(bp),PACK(csize-asize,0));
    }
    else{
        PUT(HDRP(bp),PACK(csize,1));
        PUT(FTRP(bp),PACK(csize,1));
    }
}
```

place()함수는 매개변수로 받은 bp 공간에 size를 할당을 해주는 함수이다. bp의 size가 자신이 할당하고자 하는 size보다 더 크다면 split를 하여 최대 메모리 이용률을 증가 시킬수 있다.

realloc :

```
void *realloc(void *oldptr, size_t size) {
    size_t oldsize;
    void *newptr;

    if(size == 0){
        free(oldptr);
        return 0;
    }
    if(oldptr == NULL)
        return malloc(size);
    newptr = malloc(size);
    if(!newptr)
        return 0;
    oldsize = +GET_SIZE(HDRP(oldptr));
    if(size<oldsize) oldsize = size;
    memcpy(newptr,oldptr,oldsize);
    free(oldptr);
    return newptr;
}
```

할당이 되어진 공간을 다른 사이즈로 같은 내용을 가진 채로 할당을 해주는 함수이다.
자신이 요청을 한 size값이 0이면 free를 해주고 할당을 요청을 하는 변수가 NULL이면 malloc함수를 이용을 하여 메모리를 할당을 해준다. size의 크기를 가진 메모리를 할당을 하여 memcpy()함수를 이용을하여 oldsize 만큼 값을 복사를 한다. 원래 할당된 메모리를 가지고 있던 공간은 free()함수를 이용을하여 반환을 해주고 새로 할당을 한 변수를 반환을 시켜준다.

(3)결과

```
a201302482@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 80906 ./traces/malloc.rep
   yes    28%      17  0.000000102409 ./traces/malloc-free.rep
   yes    96%      15  0.000000 75000 ./traces/corners.rep
 * yes    81%    1494  0.000052 28969 ./traces/perl.rep
 * yes    75%     118  0.000001105470 ./traces/hostname.rep
 * yes    91%   11913  0.000851 13994 ./traces/xterm.rep
 * yes    91%    5694  0.001943  2931 ./traces/amptjp-bal.rep
 * yes    92%    5848  0.001246  4692 ./traces/cccp-bal.rep
 * yes    95%    6648  0.003682  1806 ./traces/cp-decl-bal.rep
 * yes    97%    5380  0.003760  1431 ./traces/expr-bal.rep
 * yes    66%   14400  0.000119121075 ./traces/coalescing-bal.rep
 * yes    91%    4800  0.003075  1561 ./traces/random-bal.rep
 * yes    55%    6000  0.004701  1276 ./traces/binary-bal.rep
10         83%   62295  0.019430  3206

Perf index = 54 (util) + 40 (thru) = 94/100
```

mm-explicit 의 대한 설명

(1)explicit에 사용된 매크로

```
static char* heap_listp=0;
static char* free_listp=0;
```

처음 블록을 가리키기 위한 heap_listp와 처음 가용 블록을 가리키기 위한 free_listp를 선언을 한다.

```
#define ALIGNMENT 8
#define HDRSIZE 4
#define FTRSIZE 4
#define WSIZE 4
#define DSIZE 8
#define CHUNKSIZE (1<<12)
#define OVERHEAD 8
#define MAX(x,y) ((x) > (y) ? (x) : (y))
#define MIN(x,y) ((x) < (y) ? (x) : (y))
#define PACK(size,alloc) ( ((size) | (alloc)))
#define GET(p) (*(unsigned *)(p))
#define PUT(p,val) (*(unsigned *)(p) = (val))
#define GET_SIZE(p) (GET(p) & ~0x7)
#define GET_ALLOC(p) (GET(p) & 0x1)
#define HDRP(bp)     ((void *)(bp) -WSIZE)
#define FTRP(bp)     ((void *)(bp) + GET_SIZE(HDRP(bp)) - DSIZE)
#define NEXT_BLKP(bp)    ((void *)(bp) + GET_SIZE(HDRP(bp)))
#define PREV_BLKP(bp)    ((void *)(bp) - GET_SIZE(HDRP(bp) - WSIZE))
#define NEXT_FREEP(bp)   (*(void **)(bp+DSIZE))
#define PREV_FREEP(bp)   (*(void **)(bp))
#define ALIGN(p) (((size_t)(p) + (ALIGNMENT-1))&~0x7)
```

ALIGNMENT : ALIGNMENT를 8byte 단위로 한다.
HDRSIZE : header의 크기는 4byte로 한다.
FTRSIZE : footer의 크기는 4byte로 한다.
ALIGN(size) : 입력을 한 사이즈를 ALIGNMENT의 배수로 변경을 시킨다.
WSIZE : word size 4
DSIZE : double word size 8
CHUNKSIZE : 2^{12}=4096 heap영역을 확장을 할 때 사용이 되어진다.
MAX(x,y) : x와 y중 큰 값을 반환
MIN(x,y) : x와 y중 작은 값을 반환한다.
GET(p) : p의 주소값을 반환을 한다.
PUT(p,val) : p가 가리키고 있는곳에 val값을 저장을 한다.
GET_SIZE(p) : p의 끝에있는 1비트를 제외한 비트들과 and연산을 하여 크기를 반환을 해준다.
GET_ALLOC(p) : p의 끝에있는 1비트와 and연산을 하여 alloc값을 반환을 해준다.
HDRP(bp) : bp가 가리키고 있는 공간에 header주소를 반환을 해준다.
FTRP(bp) : bp가 가리키고 있는 공간이 footer주소를 반환 해준다.
NEXT_BLKP(bp) : bp가 가리키고 있는 공간의 다음 block의 주소를 반환을 해준다.

PREV_BLKP(bp) : bp가 가리키고 있는 공간의 그 전 block의 주소를 반환을 해준다.
NEXT_FREEP(bp) : 해당 메모리의 다음 가용 블록을 가리켜 준다.
PREV_FREEP(bp) : 해당 메모리의 이전 가용 블록을 가리켜 준다.

(2)함수

init :

```
int mm_init(void) {
    if((heap_listp = mem_sbrk(6*DSIZE)) == NULL)
        return -1;
    PUT(heap_listp,0);
    PUT(heap_listp+WSIZE,PACK(24,1));
    PUT(heap_listp+DSIZE,0);
    PUT(heap_listp+DSIZE+WSIZE,0);
    PUT(heap_listp+24,PACK(24,1));
    PUT(heap_listp+WSIZE+24,PACK(0,1));
    free_listp = heap_listp+DSIZE;
    if(extend_heap(CHUNKSIZE/WSIZE) == NULL)
        return -1;
    return 0;
}
```

초기에 빈 heap을 생성을 하여 초기화를 해주는 함수이다. mem_sbrk()함수를 이용을 하여 최소 사이즈 만큼의 공간을 만든 후 그곳에 extend_heap()함수를 호출하여 heap의 사이즈를 확장을 한다. free_listp에는 다음 가용블록을 저장을 하는 next를 가리키게 한다.

extend_heap :

```
static void *extend_heap(size_t words){
    char *bp;
    size_t size;

    size = (words%2) ? (words+1)*WSIZE : words*WSIZE;
    if(size < 3*DSIZE)
        size = 3*DSIZE;
    if((long)(bp = mem_sbrk(size))==-1)
        return NULL;
    PUT(HDRP(bp),PACK(size,0));
    PUT(FTRP(bp),PACK(size,0));
    PUT(HDRP(NEXT_BLKP(bp)),PACK(0,1));
    return coalesce(bp);
}
```

extend_heap은 heap이 초기화 될 때와 malloc()함수를 시행할 때 적당한 fit를 찾지 못하였을 때 호출이 된다. heap에 할당하는 메모리를 확장을 시켜줄 때 사용을 하는 함수이다 반환을 할 때 근천 가용블록들과 합치기 위해서 coalesece()를 호출을 한 후 반환을 한다.

free() :

```
void free (void *ptr) {
    if(!ptr)
        return ;
    size_t size = GET_SIZE(HDRP(ptr));
    PUT(HDRP(ptr),PACK(size,0));
    PUT(FTRP(ptr),PACK(size,0));
    coalesce(ptr);
}
```

할당이 되어진 메모리 공간을 반환을 해주는 함수이다. 메모리 공간의 header와 footer의 alloc값을 0으로 설정을 해주고 coalesce()함수를 이용을 하여 근처의 가용블록들과 합친다.

coalesce() :

```
void *coalesce(void *bp){
    size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp))) || PREV_BLKP(bp)==bp;
    size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
    size_t size = GET_SIZE(HDRP(bp));
    if(prev_alloc && !next_alloc){
        size+=GET_SIZE(HDRP(NEXT_BLKP(bp)));
        removeBlock(NEXT_BLKP(bp));
        PUT(HDRP(bp),PACK(size,0));
        PUT(FTRP(bp),PACK(size,0));
    }
    else if(!prev_alloc && next_alloc){
        size +=GET_SIZE(HDRP(PREV_BLKP(bp)));
        bp = PREV_BLKP(bp);
        removeBlock(bp);
        PUT(HDRP(bp),PACK(size,0));
        PUT(FTRP(bp),PACK(size,0));
    }
    else if(!prev_alloc && !next_alloc)
    {
        size+=GET_SIZE(HDRP(PREV_BLKP(bp)))+GET_SIZE(HDRP(NEXT_BLKP(bp)));
        removeBlock(PREV_BLKP(bp));
        removeBlock(NEXT_BLKP(bp));
        bp = PREV_BLKP(bp);
        PUT(HDRP(bp),PACK(size,0));
        PUT(FTRP(bp),PACK(size,0));
    }
    insertAtFront(bp);
    return bp;
}
```

coalesce() 함수는 매개변수로 들어온 bp와 근처에 있는 가용블록들을 합치는 함수이다. 근처에 가용블록들이 없다면 insertAtFront()함수를 이용하여 자기자신을 free_listp가 가리키게 하고 반환을 한다. 근처에 가용블록이 있다면 그 가용블록의 size와 합친다. removeBlock() 함수를 이용을 하여 합쳐진 블록의 다음 가용블록을 가리키게 하고 다음 가용블록은 합쳐진 블록을 이전 가용블록을 만든다.

removeBlock() :

```
static void removeBlock(void *bp){
    if(PREV_FREEP(bp))
        NEXT_FREEP(PREV_FREEP(bp)) = NEXT_FREEP(bp);
    else
        free_listp = NEXT_FREEP(bp);
    PREV_FREEP(NEXT_FREEP(bp)) = PREV_FREEP(bp);


}
```

removeBlock()함수는 매개변수로 온 bp가 이전 가용블록을 가지고 있다면 그 이전 블록의 다음 블록을 bp의 다음 가용블록으로 만들고 없다면 free_listp가 bp의 이전 블록이 되게 한다. 그 후 bp의 다음 가용블록의 이전 가용블록으로 bp의 이전 가용블록가리 키고 함수를 종료를 한다.

insertAtFront() :

```
static void insertAtFront(void *bp){
    NEXT_FREEP(bp) = free_listp;
    PREV_FREEP(free_listp) = bp;
    PREV_FREEP(bp) = NULL;
    free_listp = bp;

}
```

insertAtFront() 함수는 매개변수로 온 bp를 free_listp를 이용을 하여서 가용블록의 첫 번째 블록으로 만들어 주는 함수이다. free_listp가 원래 가리키던 곳을 bp의 다음 가용블록으로 설정을 하고 free_listp가 가리키던 곳의 이전 블록으로 bp를 설정을 해준 후 bp를 free_listp로 가리킨다.

mm_malloc :

```
void *malloc (size_t size) {
    char *bp;
    unsigned asize;
    unsigned extendsize;
    if(size <= 0)
        return NULL;
    asize = MAX(ALIGN(size)+DSIZE,3*DSIZE);
    if((bp = find_fit(asize)) !=NULL){
        place(bp,asize);
        return bp;
    }
    extendsize = MAX(asize,CHUNKSIZE);
    if((bp = extend_heap(extendsize/WSIZE)) ==NULL)
        return NULL;
    place(bp,asize);
    return bp;
}
```

메모리를 할당을 해주는 함수로 입력한 사이즈가 최소 사이즈를 만족을 하지못하면 최소 사이즈 만큼 키워준다 최소사이즈를 만족을 하면 alignment에 맞는 값으로 환산을 해주고 find_fit함수를 이용을하여 맞는 공간을 찾아 place()함수를 이용을 하여 찾은 자리에 size값을 지정을 해준다 find_fit()함수 찾지 못하였다면 extend_heap함수를 호출을 하여 메모리 공간을 늘리고 늘린 공간에 메모리를 할당을 한다.

find_fit() :

```
static void *find_fit(size_t asize){

    void *bp;
    for(bp=free_listp;GET_ALLOC(HDRP(bp))==0;bp=NEXT_FREEP(bp)){

        if(asize <= (size_t)GET_SIZE(HDRP(bp)))
            return bp;
    }
    return NULL;
}
```

find_fit() 함수는 알맞은 크기의 메모리 공간을 반환을 해주는 함수이다. explicit에서는 first_fit 방식으로 구현을 하였다 first_fit에서는 처음부터 시작해서 알맞은 크기의 메모리 공간이 있으면 반환을 해준다. explicit에서는 가용블록들을 이용을하여서 first_fit를 시행을 한다.

place() :

```
static void place(void *bp,size_t asize){
    size_t csize = GET_SIZE(HDRP(bp));
    if((csize-asize)>=3*DSIZE){
        PUT(HDRP(bp),PACK(asize,1));
        PUT(FTRP(bp),PACK(asize,1));
        removeBlock(bp);
        bp = NEXT_BLKP(bp);
        PUT(HDRP(bp),PACK(csize-asize,0));
        PUT(FTRP(bp),PACK(csize-asize,0));
        coalesce(bp);
    }
    else{
        PUT(HDRP(bp),PACK(csize,1));
        PUT(FTRP(bp),PACK(csize,1));
        removeBlock(bp);
        bp=NEXT_BLKP(bp);
    }

}
```

place()함수는 매개변수로 들어온 bp에 asize를 값을 할당을 하는 함수이다. 만약 할당을 해주는 공간보다 가지고 있는 공간이 더 크면 split를 하여 쪼개서 할당을 한다. 할당이 되어진 메모리 공간은 removeBlock()함수를 호출하여 bp이전의 가용블록이 bp의 다음 가용블록을 가리키고 bp의 다음 가용블록이 bp 이전의 가용블록을 가리킨다. 할당을 하는 size가 적당하다면 할당을 하고 removeBlock()함수 호출 후 함수가 끝이난다.

mm_realloc() :

```
void *realloc(void *oldptr, size_t size) {
    size_t oldsize;
    void *newptr;
    size_t asize=MAX(ALIGN(size)+DSIZE,24);
    if(size<=0){
        free(oldptr);
        return 0;
    }
    if(oldptr == NULL)
        return malloc(size);
    oldsize = GET_SIZE(HDRP(oldptr));
    if(asize ==oldsize)
        return oldptr;
    if(asize <= oldsize){
        size = asize;
        if(oldsize - size <=24)
            return oldptr;
        PUT(HDRP(oldptr),PACK(size,1));
        PUT(FTRP(oldptr),PACK(size,1));
        PUT(HDRP(NEXT_BLKP(oldptr)),PACK(oldsize-size,1));
        free(NEXT_BLKP(oldptr));
        return oldptr;
    }
    newptr = malloc(size);

    if(!newptr)
        return 0;
    if(size<oldsize)
        oldsize =size;
    memcpy(newptr,oldptr,oldsize);
    free(oldptr);
    return newptr;


}
```

할당된 함수의 공간을 다른 사이즈로 할당을 해주는 함수이다. 할당을 원하는 size가 0보다 작으면 free()함수를 호출을 해준다. 할당을 원하는 공간이 비워있으면 malloc()함수를 호출을 한 후 반환, 할당을 하고자 하는 사이즈가 원래의 크기보다 작으면 할당하고자 하는 사이즈를 할당을 한 후 split을 사용을 하여 남은 공간을 free()를 시켜준다. 할당을 하고자 하는 사이즈가 더 크면 malloc()함수를 이용을하여 할당을 한 후 memcpy()함수를 이용을 하여 이전 메모리 공간의 내용을 복사 한다. 이전 메모리 공간을 free()함수를 이용을 하여 반환을 한 후 새로운 공간을 반환을 해준다.

(3)결과

```
a201302482@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 60096 ./traces/malloc.rep
   yes    28%      17  0.000000 87629 ./traces/malloc-free.rep
   yes    96%      15  0.000000 71157 ./traces/corners.rep
 * yes    81%    1494  0.000019 77970 ./traces/perl.rep
 * yes    74%     118  0.000002 69200 ./traces/hostname.rep
 * yes    89%   11913  0.000148 80418 ./traces/xterm.rep
 * yes    89%    5694  0.000202 28197 ./traces/amptjp-bal.rep
 * yes    92%    5848  0.000139 41960 ./traces/cccp-bal.rep
 * yes    94%    6648  0.000274 24262 ./traces/cp-decl-bal.rep
 * yes    96%    5380  0.000227 23735 ./traces/expr-bal.rep
 * yes    66%   14400  0.000162 88943 ./traces/coalescing-bal.rep
 * yes    87%    4800  0.000268 17906 ./traces/random-bal.rep
 * yes    55%    6000  0.000858  6994 ./traces/binary-bal.rep
10        82%   62295  0.002299 27099

Perf index = 53 (util) + 40 (thru) = 93/100
```