시스템 프로그래밍

- Malloc Lab -

2014. 12. 04.

박시형

sihyeong@cnu.ac.kr

Embedded System Lab. Computer Engineering Dept.

Chungnam National University

개 요

❖ 실습 명

- Malloc Lab

❖ 목표

- Dynamic Allocator를 구현

❖ 구현사항

- malloc, realloc, free를 구현
- 다양한 알고리즘을 적용하여 높은 성능을 갖도록 함

Malloc Lab

- Malloc Lab에서는 C언어로 dynamic storage allocator를 구현하게 된다.
- 이를 통해 각자의 malloc, free, realloc 알고리즘을 구현해야 한다.
- 또한, 메모리 공간의 생성의 디자인을 고려하면서 정확하고 효율이 좋은 빠른 allocator를 구현해야 한다.

- Malloc Lab은 총 4가지의 방식으로 진행된다.
  - Naive
  - Implicit
  - Explicit
  - Segregated (보너스 구현)

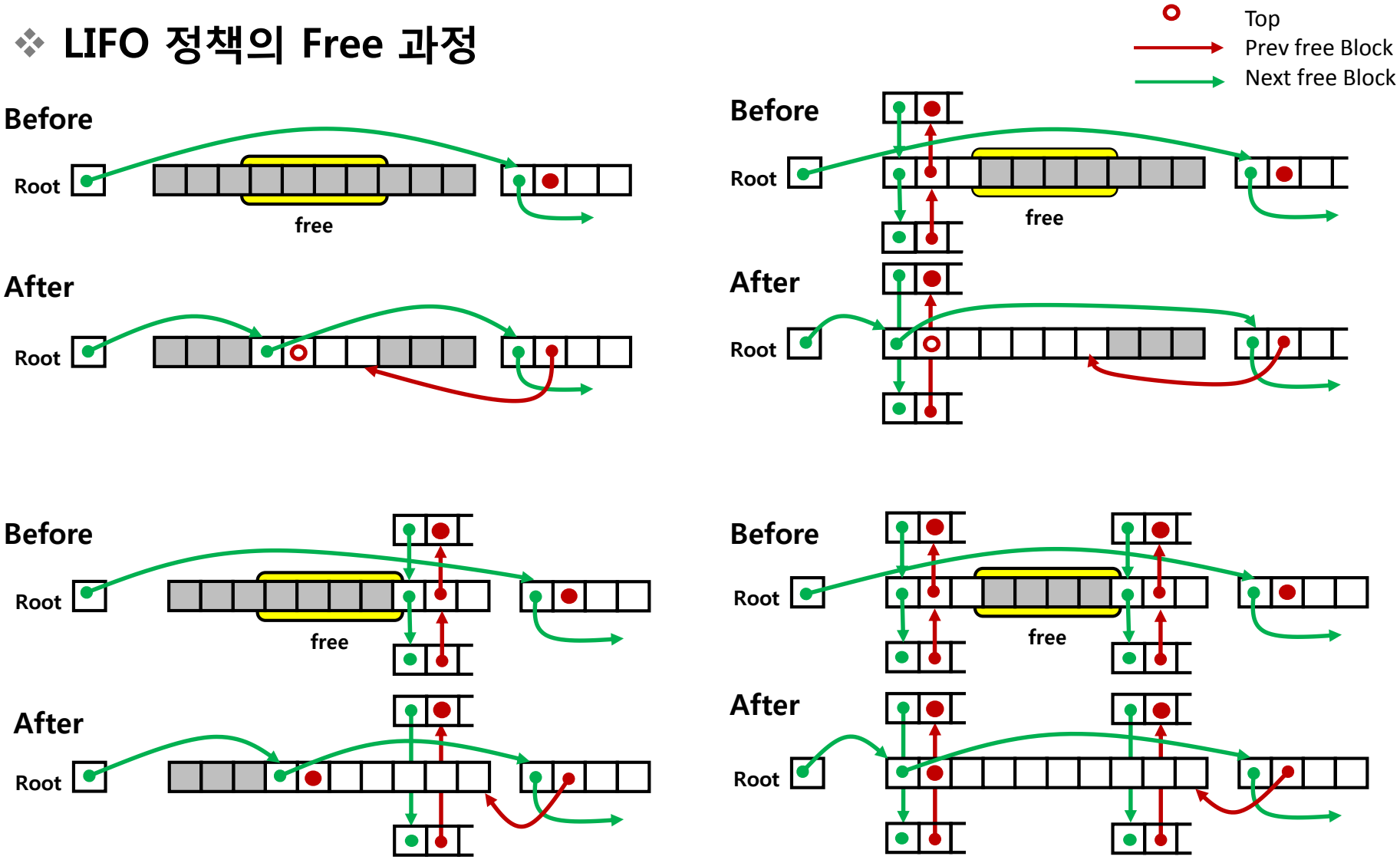
Malloc Lab – Free Block 추가 알고리즘

- Explicit 방식은 Free block 들을 list 형식으로 관리하는 방법이다. 따라서 Free block들이 새로 추가되는 경우, 이 block을 어느 위치에 추가하느냐에 따라 성능이 좌우된다.

- Free block을 추가하기 위한 알고리즘은 다음과 같다.
  - LIFO(Last In First Out) 정책
    - Free Block을 List의 맨 앞에 끼워 넣는 방법
    - 장점 : 간단하고, 상수시간 소요
    - 단점 : 단편화가 Address-ordered 정책보다 나쁨
  - 주소정렬(Address-ordered) 정책
    - Free Block List의 block 순서를 유지하면서 삽입
      - ex) addr(pred) < addr(curr) < addr(succ)
    - 장점 : LIFO 정책보다 단편화 성능이 우수
    - 단점 : List 탐색 후 삽입해야 함

Malloc Lab – Free Block 추가 알고리즘

❖ LIFO 정책의 Free 과정



Malloc Lab – 설치 및 빌드

- Malloc Lab은 각 allocation 방식에 맞게 설정 후 빌드를 해주어야 한다.
- 아래는 각 allocation 방식 설정 방법이다.

Allocation 방식	make 명령어
Naive	make naive
Implicit	make implicit
Explicit	make explicit
Segregated	make seglist

- 설정 후에, 'make' 명령을 입력하여 컴파일 한다.

- make 명령을 통해 각 방식 별로 컴파일 하기 이전에 'make clean' 명령을 통해 이전에 컴파일 된 파일들을 제거하고 수행해준다.

Malloc Lab – 설치 및 빌드

❖ Allocation 방식 설정 및 빌드

```
Makefile  config.h  fsecs.h        mdriver.c      mm-implicit.c  mm.c
README    fcyc.c    ftimer.c       memlib.c       mm-naive.c     mm.h
clock.c   fcyc.h    ftimer.h       memlib.h       mm-orig.c      traces
clock.h   fsecs.c   malloclab.pdf  mm-explicit.c  mm-seglist.c
[c0000000000@eslab malloclab-handout]$ make explicit
rm -f mm.c mm.o; ln -s mm-explicit.c mm.c
[c0000000000@eslab malloclab-handout]$ make
gcc -Wall -g -DDRIVER   -c -o mdriver.o mdriver.c
```

❖ 빌드 결과

```
Makefile  fcyc.c   ftimer.c       mdriver.o      mm-naive.c     traces
README    fcyc.h   ftimer.h       memlib.c       mm-orig.c
clock.c   fcyc.o   ftimer.o       memlib.h       mm-seglist.c
clock.h   fsecs.c  malloclab.pdf  memlib.o       mm.c
clock.o   fsecs.h  mdriver        mm-explicit.c  mm.h
config.h  fsecs.o  mdriver.c      mm-implicit.c  mm.o
```

Malloc Lab – 필수 사항

- 각 방식 별, 소스파일 상단에 학번과 이름을 입력한다.
  - mm-explicit.c
  - mm-seglist.c

```
/*
 * mm-implicit.c - an empty malloc package
 *
 * NOTE TO STUDENTS: Replace this header comment with your own header
 * comment that gives a high level description of your solution.
 *
 * @id : 201300000
 * @name : Han Donggeon
 */
```

- 해당 파일 외에는 절대 수정을 하지 않는다. (mm.h 등)
- 외부 메모리 관리 라이브러리 및 시스템 콜 사용 불가.
- 배열, 구조체, 트리, 리스트 같은 전역 자료 구조체의 선언 금지.
  - 공간이 많이 필요한 경우 mem_sbrk 활용

Malloc Lab – 테스트

- Malloc Lab 테스트
  - './mdriver'를 통해 구현 내용을 테스트 한다.
  - 여러 개의 trace들을 실행시켜 정확성(correctness), 공간 활용도(utilization), 처리량(throughput)을 계산한다.
    - 평가 방식은 malloclab.pdf의 '8 Evaluation' 을 참고

```
[c0000000000@eslab malloclab-handout]$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 3392.3 MHz

Results for mm malloc:
   valid  util   ops    secs     Kops  trace
   yes     94%      10  0.000000 68394 ./traces/malloc.rep
   yes     77%      17  0.000000 91250 ./traces/malloc-free.rep
   yes    100%      15  0.000000 51924 ./traces/corners.rep
 * yes     71%    1494  0.000016 94386 ./traces/perl.rep
 * yes     68%     118  0.000001 93179 ./traces/hostname.rep
 * yes     65%   11913  0.000103115651 ./traces/xterm.rep
 * yes     23%    5694  0.000064 88723 ./traces/amptjp-bal.rep
 * yes     19%    5848  0.000069 84858 ./traces/cccp-bal.rep
 * yes     30%    6648  0.000076 87679 ./traces/cp-decl-bal.rep
 * yes     40%    5380  0.000063 85902 ./traces/expr-bal.rep
 * yes      0%   14400  0.000143100760 ./traces/coalescing-bal.rep
 * yes     38%    4800  0.000064 75161 ./traces/random-bal.rep
 * yes     55%    6000  0.000056107450 ./traces/binary-bal.rep
10         41%   62295  0.000654 95214

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

naive 를 테스트 한 결과

Malloc Lab – 테스트

- Malloc Lab 특정 파일 테스트
  - 테스트에 이용되는 trace 파일을 이용해서 개별적으로 테스트가 가능하다.
    - 각각의 테스트 방식이 다르기 때문에 내용을 이해하고, 해당 테스트의 동작이 정상적으로 이루어 지는지 검토할 필요가 있음. (gdb 활용)
  - ./mdriver -f [파일경로/파일명]

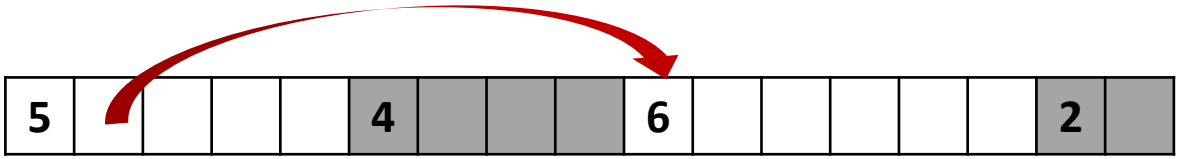
```
[cOOOOOOOOO@eslab malloclab-handout]$ ./mdriver -f ./traces/coalescing-bal.rep
Measuring performance with a cycle counter.
Processor clock rate ~= 3392.3 MHz

Results for mm malloc:
   valid  util   ops    secs     Kops  trace
 * yes     0%   14400  0.000148 97587 ././traces/coalescing-bal.rep
 1         0%   14400  0.000148 97587

Perf index = 0 (util) + 40 (thru) = 40/100
```

Test case인 coalescing를 naive 에서 테스트 한 결과

  - 그 외의 옵션
    - -h 옵션을 사용하여 직접 확인해본다.

Malloc Lab – Explicit

- **Explicit list** 방식은 Free block에 내재되어 있는 pointer를 이용하여 Free block들을 연결하고, 연결된 모든 Free block들을 탐색한다.

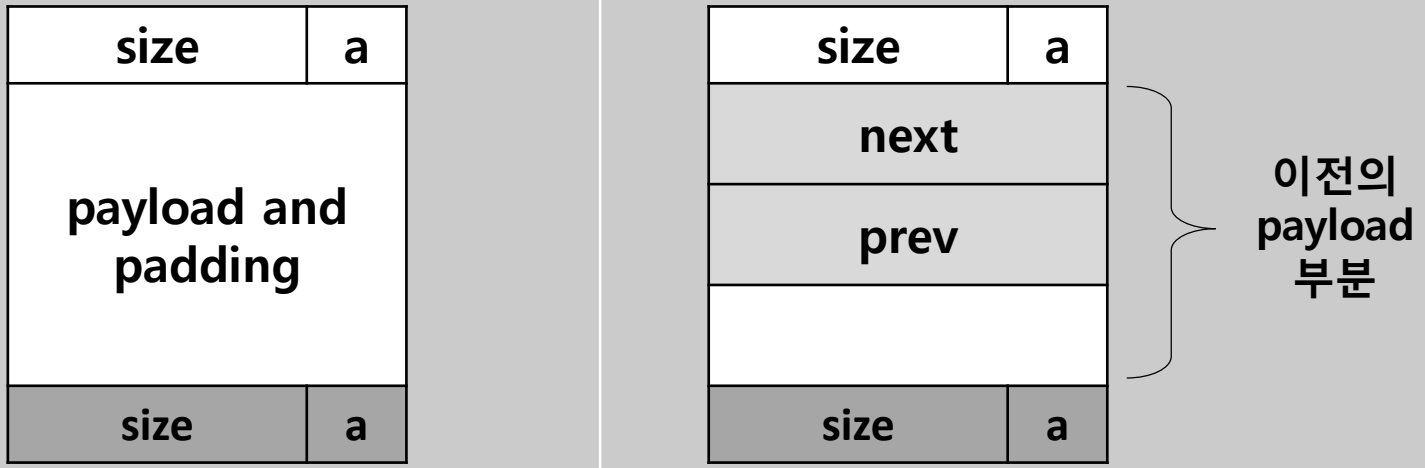


- Free block만 탐색하므로 **높은 throughput**을 보인다.

Malloc Lab – Explicit

❖ 자료구조

할당된 블록의 구조 (Allocated block)	할당되지 않은 가용 블록의 구조 (Free block)

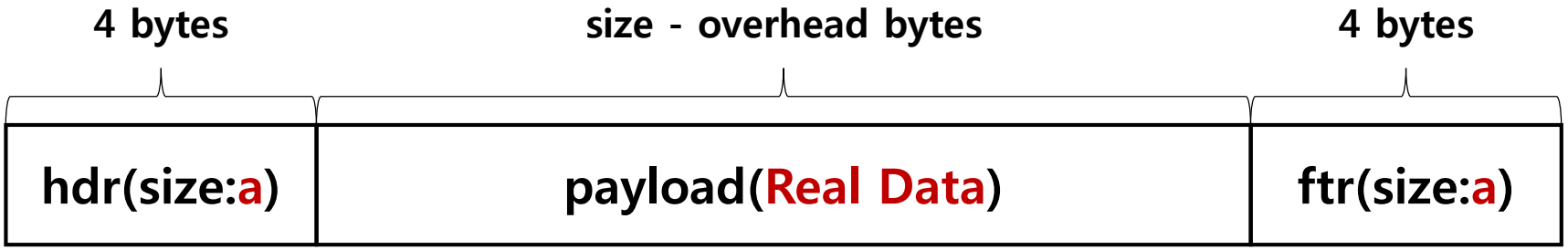


- 할당된 블록의 경우, implicit에서 사용했던 구조를 그대로 이용
- Free 블록의 경우, 다음 Free 블록을 가리키는 next와 이전 Free 블록을 가리키는 prev를 이전 payload 영역에 가짐

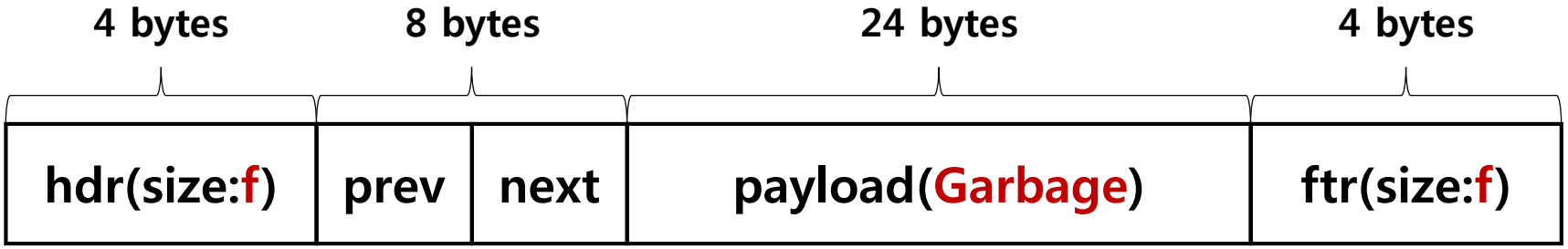
Malloc Lab – Explicit

❖ 자료구조

- <Allocated block>



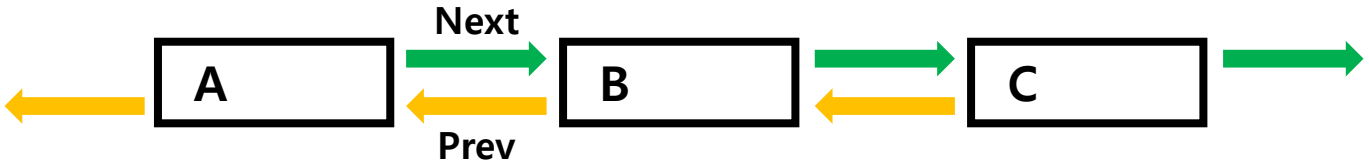
- <Free block>



Malloc Lab – Explicit

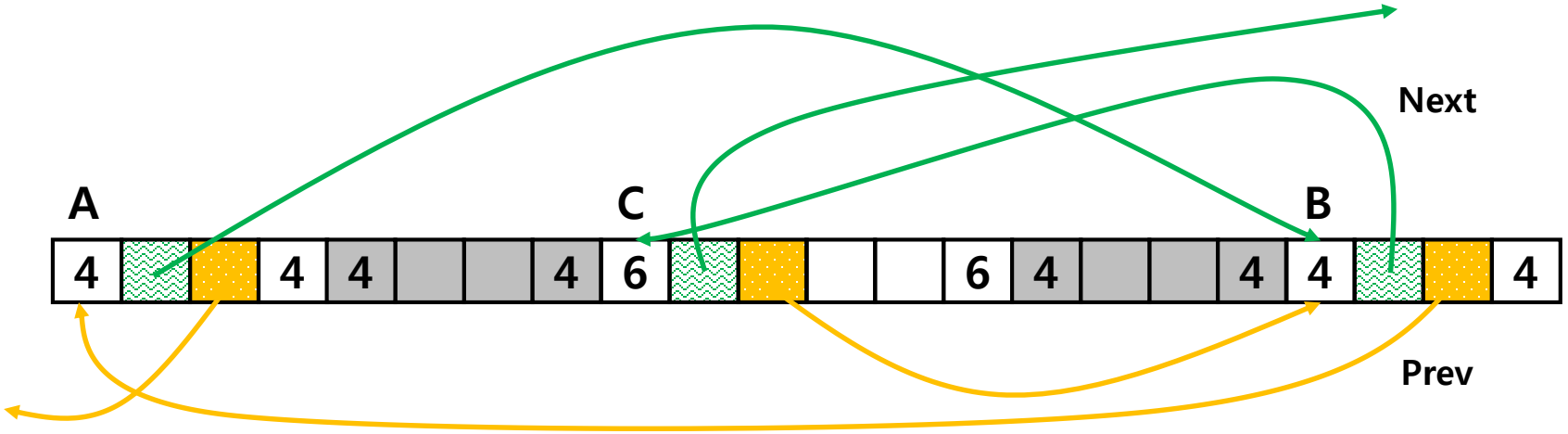
❖ 논리구조

- 논리상으로는 각 Free 블록들이 순서대로 연결된 형태로 되어있다.



❖ 실제구조

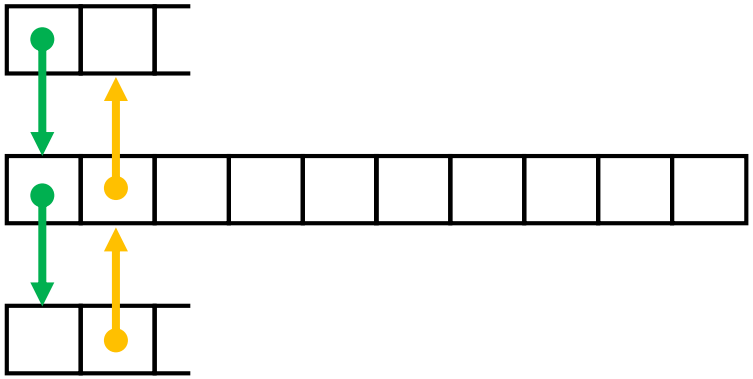
- 실제 연결 링크는 메모리 블록의 순서와 무관하다.



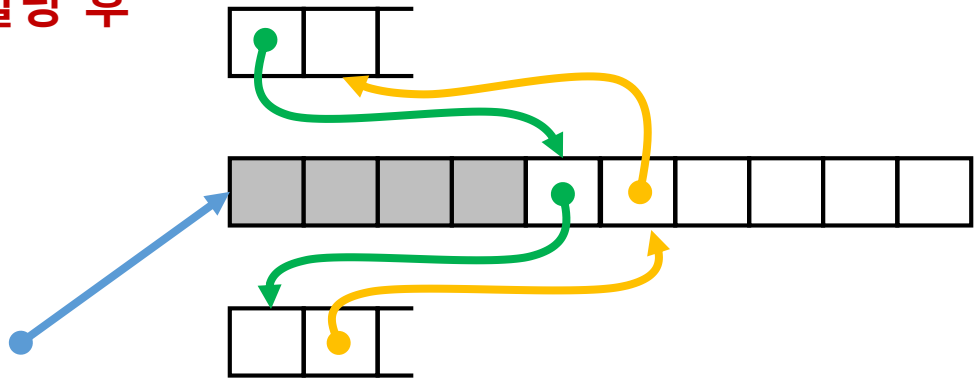
Malloc Lab – Explicit

Explicit의 메모리 할당

- 할당 전



- 할당 후



Malloc Lab – Explicit

❖ Explicit에서 사용되는 매크로(1)

```
/* single word (4) or double word (8) alignment */
#define ALIGNMENT 8

/* Basic constants and macros */
#define HDRSIZE         4       // header size (bytes)
#define FTRSIZE         4       // footer size (bytes)
#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))

/* Pack a size and allocated bit into a word */
#define PACK(size, alloc)       ((unsigned) ((size) | (alloc)))

/* Read and write a word at address p */
#define GET(p)          (*(unsigned *)(p))
#define PUT(p,val)      (*(unsigned *)(p) = (unsigned)(val))
#define GET8(p)         (*(unsigned long *)(p))
#define PUT8(p,val)     (*(unsigned long *)(p) = (unsigned long)(val))
```

Malloc Lab – Explicit

❖ Explicit에서 사용되는 매크로(2)

```
/* Read the size and allocated fields from address p */
#define GET_SIZE(p)     (GET(p) & ~0x7)
#define GET_ALLOC(p)    (GET(p) & 0x1)

/* Given block ptr bp, compute address of its header and footer */
#define HDRP(bp)        ((char *)(bp) - WSIZE)
#define FTRP(bp)        ((char *)(bp) + GET_SIZE(HDRP(bp)) - DSIZE)

/* Given block ptr bp, compute address of next and previous blocks */
#define NEXT_BLKP(bp)   ((char *)(bp) + GET_SIZE(HDRP(bp)))
#define PREV_BLKP(bp)   ((char *)(bp) - GET_SIZE((char*)(bp) - DSIZE))

#define NEXT_FREEP(bp)  ((char *)(bp))
#define PREV_FREEP(bp)  ((char *)(bp) + WSIZE)

#define NEXT_FREE_BLKP(bp)      ((char *)GET8((char *)(bp)))
#define PREV_FREE_BLKP(bp)      ((char *)GET8((char *)(bp) + WSIZE))

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

Malloc Lab – Explicit

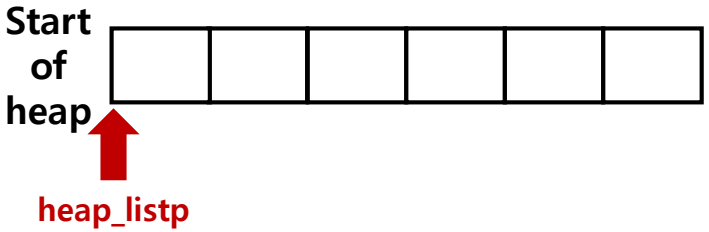
❖ Explicit의 mm_init 함수 (참고용)

```
/*
 * Initialize: return -1 on error, 0 on success.
 */
int mm_init(void) {
    /* Request memory for the initial empty heap */
    if((h_ptr = mem_sbrk(DSIZE + 4 * HDRSIZE)) == NULL)
        return -1;
    heap_start = h_ptr;

    PUT(h_ptr, NULL);
    PUT(h_ptr + WSIZE, NULL);
    PUT(h_ptr + DSIZE, 0);                                  // alignment padding
    PUT(h_ptr + DSIZE + HDRSIZE, PACK(OVERHEAD,1));         // prologue header
    PUT(h_ptr + DSIZE + HDRSIZE + FTRSIZE, PACK(OVERHEAD,1));   // prologue footer
    PUT(h_ptr + DSIZE + 2 * HDRSIZE + FTRSIZE, PACK(0,1));      // epilogue header

    /* Move heap pointer over to footer */
    h_ptr += DSIZE + DSIZE;

    /* Leave room for the previous and next pointers, place epilogue 3 words down */
    epilogue = h_ptr + HDRSIZE;

    /* Extend the empty heap with a free block of CHUNKSIZE bytes */
    if(extend_heap(CHUNKSIZE/WSIZE) == NULL)
        return -1;

    return 0;
}
```

Malloc Lab – Explicit

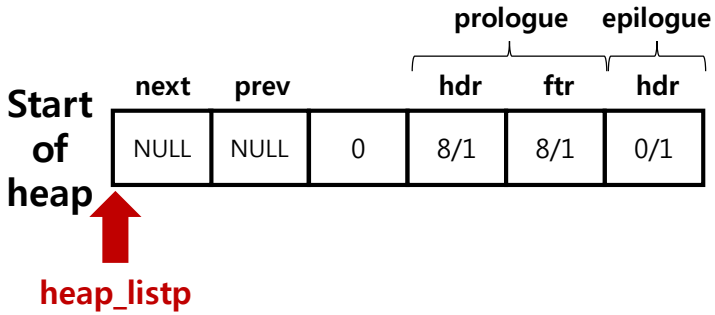
- Explicit의 mm_init 함수 (참고용)
  - 1) 초기 empty heap을 생성한다.

```
/* Request memory for the initial empty heap */
if((h_ptr = mem_sbrk(DSIZE + 4 * HDRSIZE)) == NULL)
    return -1;
heap_start = h_ptr;
```



  - 2) Root free block의 next와 prev를 생성하고, padding block과 prologue, epilogue를 할당한다.

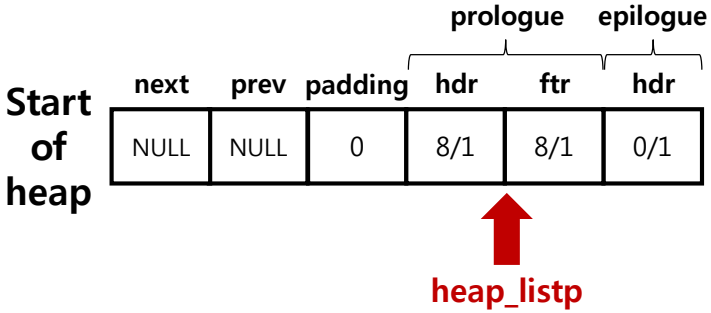
```
PUT(h_ptr, NULL);
PUT(h_ptr + WSIZE, NULL);
PUT(h_ptr + DSIZE, 0);
PUT(h_ptr + DSIZE + HDRSIZE, PACK(OVERHEAD,1));
PUT(h_ptr + DSIZE + HDRSIZE + FTRSIZE, PACK(OVERHEAD,1));
PUT(h_ptr + DSIZE + 2 * HDRSIZE + FTRSIZE, PACK(0,1));
```



Malloc Lab – Explicit

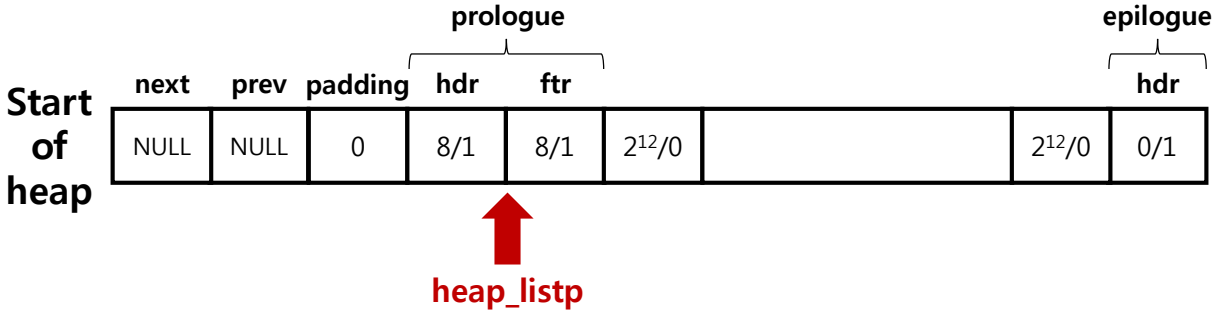
- Explicit의 mm_init 함수 (참고용)
  - 3) Heap pointer를 footer 위치로 이동시킨다.
  - 4) epilogue를 초기화 한다.

```
h_ptr += DSIZE + DSIZE;
epilogue = h_ptr + HDRSIZE;
```



  - 5) 사용할 최대 크기의 heap을 미리 할당한다.

```
if(extend_heap(CHUNKSIZE/WSIZE) == NULL)
    return -1;
```



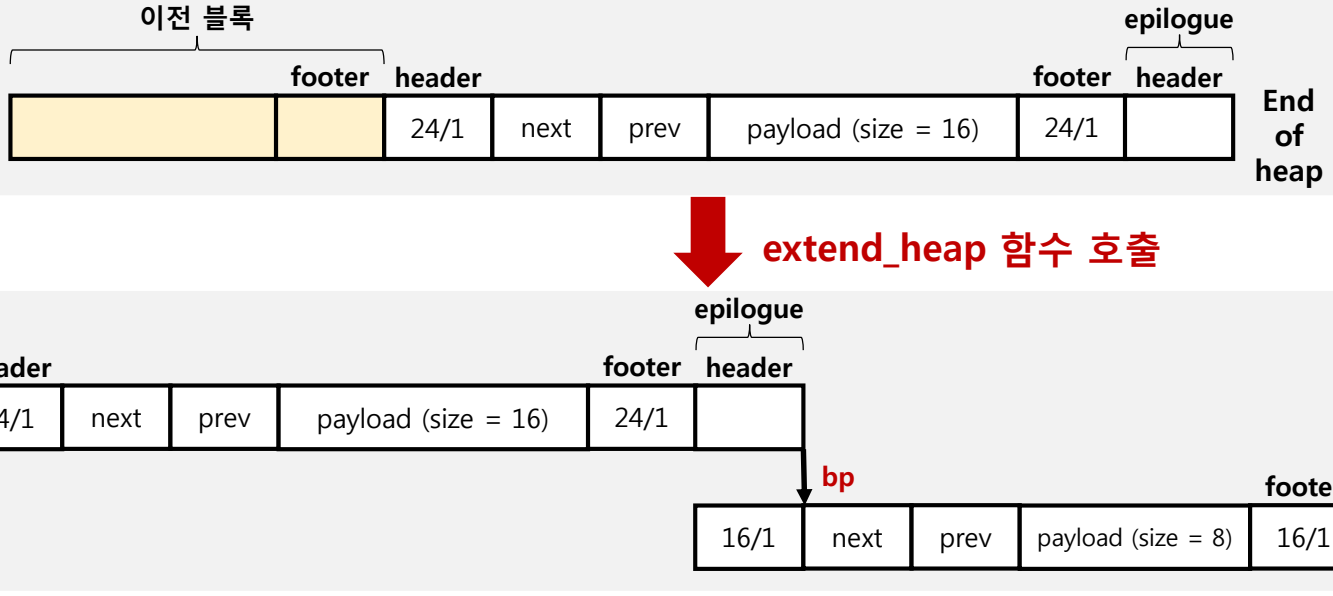
Malloc Lab – Explicit

❖ Explicit의 extend_heap 함수 (참고용)

```
/*
 * extend_heap
 */
inline void *extend_heap(size_t words) {
    unsigned *old_epilogue;     // Temp storage for current epilogue
    char *bp;                   // New block pointer after heap extension
    unsigned size;              // Request size for heap memory

    /* Allocate an even number of words to maintain alignment */
    size = (words % 2) ? (words+1)*WSIZE : words*WSIZE ;

    /* Request more memery from heap */
    if((long)(bp = mem_sbrk(size)) < 0)
        return NULL;

    /* Save the old epilogue pointer */
    old_epilogue = epilogue;
    epilogue = bp + size - HDRSIZE;

    /* Write in the header, footer, and new epilogue */
    PUT(HDRP(bp), PACK(size,0));
    PUT(FTRP(bp), PACK(size,0));
    PUT(epilogue, PACK(0,1));

    return coalesce(bp);
}
```

Malloc Lab – Explicit

- Explicit의 extend_heap 함수 (참고용)
  - extend_heap 함수가 호출 되면, 요청된 크기만큼 heap을 늘리고 이전 블록의 epilogue가 다음 블록의 header가 된다.



  - 새로운 free 블록이 할당되었으므로 free 블록을 통합하는 coalesce( ) 함수를 호출하도록 되어있다.
    - LIFO 정책의 free 과정과 동일하게 free 블록을 통합하도록 함수 작성
    - free( ) 함수 내부에서도 호출

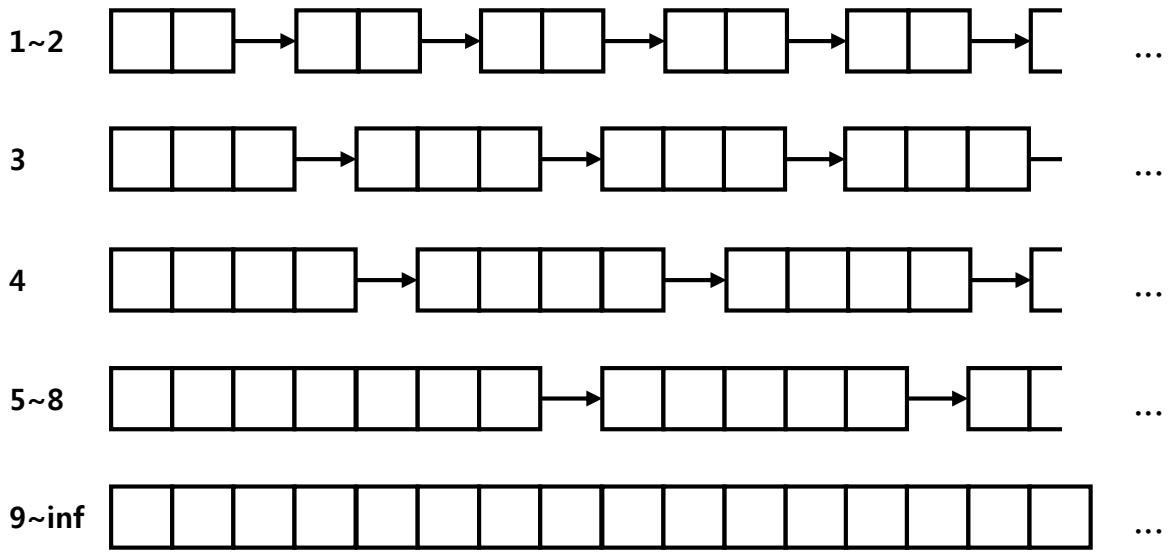
Malloc Lab – Explicit

- Explicit의 malloc 함수 (참고용)
  - 주석 부분을 채워야 한다.
  - free list를 검색하는 find_fit( ) 함수를 작성해야 한다.
  - free 블록에 할당하는 함수인 place( ) 함수를 작성한다.
  - LIFO 정책의 free 과정을 coalesce( ) 함수에 작성한다.

```
void *malloc (size_t size) {
        char *bp;               // Block pointer, points to first byte of payload
        unsigned asize;         // Block size adjusted for alignment and overhead
        unsigned extendsize;    // Amount to extend heap if no fit

        // size가 올바르지 않을 때 예외처리

        // block의 크기 결정

        // 결정한 크기에 알맞은 블록을 list에서 검색하여 해당 위치에 할당
        if((bp = find_fit(asize)) != NULL)
        {
                place(bp, asize);
                return bp;
        }

        // free list에서 적절한 블록을 찾지 못했으면 힙을 늘려서 할당
        extendsize = MAX(asize, CHUNKSIZE);
        if((bp = extend_heap(extendsize/WSIZE)) == NULL)
                return NULL;
        place(bp, asize);
        return bp;
        return NULL;
}
```

Malloc Lab – Seglist (옵션)

❖ 자료구조



- 각 크기 별로 free 블록 리스트를 관리한다.
- Allocator는 free 블록 리스트 배열을 관리한다.
- free 블록 리스트 배열은 오름차순으로 관리한다.
- 할당 받고자 하는 크기의 free 블록이 리스트에 없으면, 그 다음 크기의 리스트를 검색하고 할당한다.

제출 사항

❖ 제출 내용

- mm-explicit에 대한 설명
  - explicit에 사용된 매크로 및 구현 함수 설명
- explicit 방식에 대한 ./mdriver 결과 첨부
- malloclab-handout을 통째로 압축
  - 압축파일명 : [sys02]malloc_학번.tar.gz
- 압축파일과 보고서를 조교메일로 제출 (보고서는 서면으로도 제출)
  - sihyeong@cnu.ac.kr
  - 제목 : [sys02]HW10_학번_이름

❖ 제출 일자

- 메일 제출 : 2014년 12월 10일 23시 59분 59초
- 서면 제출 : 2014년 12월 11일 수업시간

- seglist는 구술 테스트를 통해 확인하고 점수를 부여할 예정