CS 4400: Computer Systems
Problem Set 20

1. Problem 9.16 from the textbook.

2. Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes, has a four-byte header, and has a four-byte footer. Only the 29 higher-order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining three lower-order bits is as follows:

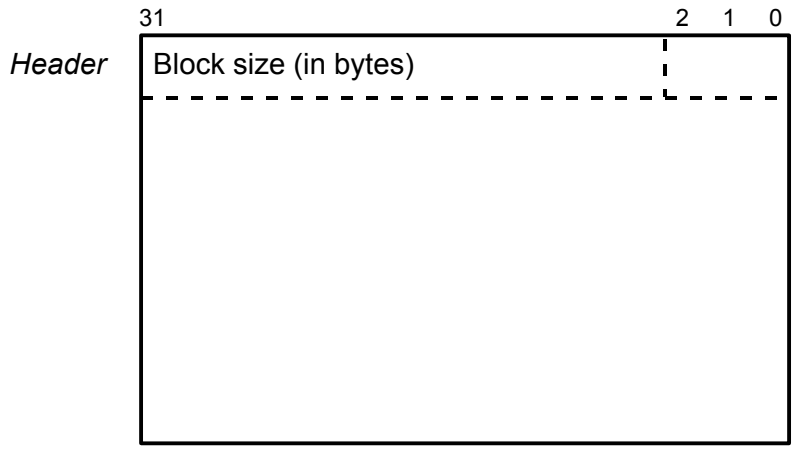
   - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
   - `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
   - `bit 2` is unused and is always set to 0.

   Five helper routines are defined to facilitate the implementation of `free(void* p)`. The functionality of each routine is explained in the comment above the function definition. Choose the code fragment, that implements the corresponding functionality correctly.

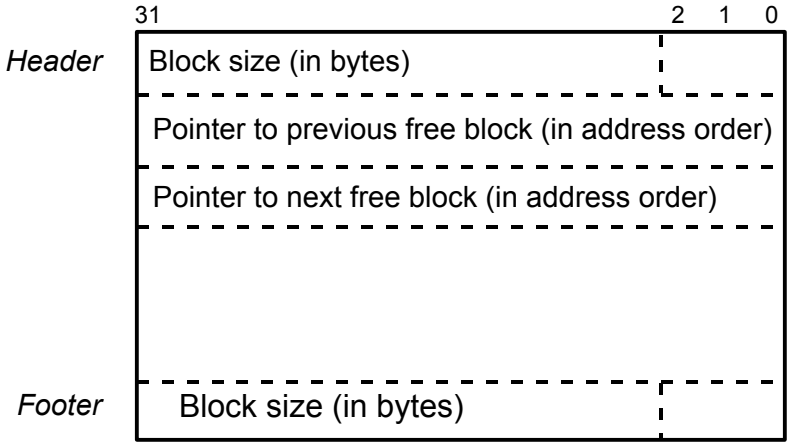
```
/* Given a pointer p to an allocated block, i.e., p is a pointer returned by some
   previous malloc()/realloc() call, returns the pointer to the header of the block.
*/
void* header(void* p) {
  void* ptr;

  _______;
  return ptr;
}

A. ptr = p - 1
B. ptr = (void*)((int*)p - 1)
C. ptr = (void*)((int*)p - 4)


/* Given a pointer to a valid block header or footer, returns the size of the block.
*/
int size(void* hp) {
  int result;

  _______;
  return result;
}

A. result = (*hp) & (~7)
B. result = ((*(char*)hp) & (~5)) << 2
C. result = (*(int*)hp) & (~7)
/* Given a pointer p to an allocated block, i.e., p is a pointer returned by some
   previous malloc()/realloc() call, returns the pointer to the footer of the block.
*/
```

```
void* footer(void* p) {
  void* ptr;

  _______;
  return ptr;
}

A. ptr = (char*)p + size(header(p)) - 8
B. ptr = (char*)p + size(header(p)) - 4
C. ptr = (int*)p + size(header(p)) - 2


/* Given a pointer to a valid block header or footer, returns the usage of the
   current block, 1 for allocated, 0 for free.
*/
int allocated(void* hp) {
  int result;

  ______;
  return result;
}

A. result = (*(int*)hp) & 1
B. result = (*(int*)hp) & 0
C. result = (*(int*)hp) | 1


/* Given a pointer to a valid block header, returns the pointer to the header of
   previous block in memory.
*/
void* prev(void* hp) {
  void* ptr;

  ______;
  return ptr;
}

A. ptr = (char*)hp - size(hp)
B. ptr = (char*)hp - size(hp - 4)
C. ptr = (char*)hp - size(hp - 4) + 4
```

3. Consider an allocator that uses an explicit free list. The layout of each allocated memory block is as follows.



The layout of each free memory block is as follows.



Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header (and footer, if a free block) are needed to record block size, which includes the header (and footer, if a free block). The usage of the remaining three lower order bits is as follows.

- `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
- `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
- `bit 2` indicates the use of the next adjacent block: 1 for allocated, 0 for free.

Given the contents of the heap shown on the left, what are the new contents of the heap (in the right table) after a call to `free(0x400b010)` is executed? *Your answer should be the contents of each cell in the table on the right, expressed as hex values.* E.g., "After a call to `free(0x400b010)`, the value `0x`_____ is stored at address `0x400b028`."

Note that the addresses grow from bottom up. Assume that the allocator uses immediate coalescing, that is, adjacent free blocks are merged immediately each time a block is freed. Also assume that any blocks not shown here are allocated.

Address	
0x400b028	0x00000016
0x400b024	0x400c000
0x400b020	0x400b000
0x400b01c	0x00000016
0x400b018	0xffffffff
0x400b014	0xffffffff
0x400b010	0xffffffff
0x400b00c	0x00000011
0x400b008	0x00000016
0x400b004	0x400b020
0x400b000	0x400af98
0x400affc	0x00000016

Address	
0x400b028	
0x400b024	
0x400b020	
0x400b01c	
0x400b018	
0x400b014	
0x400b010	
0x400b00c	
0x400b008	
0x400b004	
0x400b000	
0x400affc	