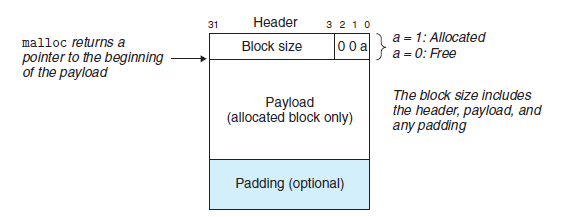
1. (10 points)(similar to Practice Problem 9.6)

Determine the block sizes and header values that would result from the following

sequence of malloc requests. Assumptions: (1) The allocator maintains doubleword (i.e., 8 bytes) alignment, and uses an implicit free list with the block format from Figure

9.35 (below). (2) Block sizes are rounded up to the nearest multiple of 8 bytes.



|  |  |  |
| --- | --- | --- |
| Request | Block size (decimal bytes) | Block header (hex) |
| malloc(1) | 2 (8 with buffer) | **0x9** |
| malloc(12) | 13 (16 with buffer) | **0x11** |
| malloc(8) | 9 (16 with buffer) | **0x11** |
| malloc(2) | 3 (8 with buffer) | **0x9** |
| malloc(16) | 17 (24 with buffer) | **0x19** |

2. (12 points) You are given groups of statements relating to memory management and garbage collection below. Your task is to indicate if each statement is true or false

**Bold is TRUE**, *italic is FALSE*

**(a) The first-fit memory allocation algorithm is faster than the best-fit algorithm (on average).** (It takes less time to find a slot, results in less segmentation.)

*(b) Deallocation using boundary tags is fast only when the list of free blocks is ordered according to increasing memory addresses.* (If boundary tags include footers, there should be no problem going backwards.)

*(c) Using the first-fit algorithm on a free list that is ordered according to decreasing block sizes results in low performance for allocations, but avoids external fragmentation.*

*(d) For the best-fit method, the list of free blocks should be ordered according to increasing memory addresses.* (It honestly shouldn’t matter, as best-fit scans the entire list anyway.)

*(e) The best-fit method chooses the largest free block into which the requested segment fits.* (It chooses the smallest free block into which the requested segment fits.)

**(f) Using the first-fit algorithm on a free list that is ordered according to increasing block sizes is equivalent to using the best-fit algorithm.**

3. (10 points) In one or two sentences, describe each of the following terms?

1. Internal fragmentation

Internal fragmentation is space in memory that’s wasted due to allocated block being bigger than strictly necessary (or bigger than the payload + the space for the header and footer). A good example of this is when your block has additional padding to match the addressing pattern, such as a byte sized payload in a section of memory using double-word addressing.

1. External fragmentation

External fragmentation has to do with the way that blocks are allocated and freed. If there are a lot of tiny free spaces, but all that are left are large payloads of data, than you end up wasting a lot of space.

4. (10 points) What is the problem with each of the following code snippets?

(A)

int \* x ; int \* y ;

x = (int \*)malloc(N\*sizeof(int));

/\* <manipulate x> \*/

**free(x);**

/ \* some other codes …\*/

y = (int \*) malloc(M\*sizeof(int));

for (i=0; i<M; i++)

y[i] = **x[i]++**;

**You can’t free x and then try to use it later in function. (Well, you can maybe in some cases, but may get some type of error depending on the size of M relative to N and how the system allocates space in the heap.)**

(B)

int \*BinheapDelete(int \*\*binheap, int \*size) {

int packet;

\*packet = binheap[0];

binheap[0] = binheap[\*size - 1];

\*size--;

Heapify(binheap, \*size, 0);

return(packet);

}

**size-- is accidentally referencing the pointer (size) as oppossed to the dereferenced value (\*size). Should do something like (\*size)--.**