GUIDE TO WRITING YOUR BUDDY MALLOC

Note that this is only a rough guide, you will have to add all the necessary checks to guard against the possibility of errors.

///////////////////////////////////////////////////////////////////////////////////////

PSEUDOCODE: BUDDY MALLOC using only a list of FREE BLOCKS

///////////////////////////////////////////////////////////////////////////////////////

Buddy\_malloc(int memory\_request\_of\_n\_bytes)

///////////////////////////////////////////////////////////////////////////////////////

INPUT: memory\_request\_of\_n\_bytes

OUTPUT: pointer to the block allocated (void\*)

///////////////////////////////////////////////////////////////////////////////////////

1) As memory blocks are represented as Nodes in a linked list, we need to take into account

the amount of memory consumed by a Node structure.

Given a memory request of n bytes, the Node that can satisfy this request must include the sizeof(Node).

Therefore, incorporate the sizeof(Node) as part of the memory request.

memory\_request = n bytes + sizeof(Node); // make sure the data fits within a Node

2) Find the smallest bin (k) that could accommodate the memory request.

We want to find the appropriate value for index k, in the expression 2^k (the size of the block). This is our desired bin.

\*3)Check if the desired bin (k) found is greater than the MAXIMUM possible bin size,

if yes, then there's not enough memory to satisfy the request. return NULL.

4)If (there's a free block in the FreeList whose index is equal to the desiredBin (k) ),

then allocate that block to satisfy the request. This is the block perfectly matching the request.

This also means that you need to remove that particular block from the FreeList -> that block is now allocated.

set the properties for that block (e.g. size)

return the base address of the data section of the block found to satisfy the request.

This data address is equal to the base address of the free block + sizeof(Node)

5) Else if (the desired block size is not available), then find an alternative bin size (the next larger bin available):

...

remove a free block from the alternative bin.

LOOP: (objective: Find the MINIMAL 2^k block that is larger or equal to that of the requested memory)

Split the free block removed.

Add the smaller blocks to the appropriate bin in the Free List (this is the new alternative bin). Make sure that the connections in the linked list are handled correctly.

Pick one of the smaller blocks from the new alternative bin (remove this block from the Free List).

Execute statements in the loop until the block removed is the smallest bin that could satisfy the memory request.

set the properties for the block (e.g. size) found to satisfy the request.

return the base address of the data section of the block found to satisfy the request.

This data address is equal to the base address of the free block + sizeof(Node)

...

End Else If