CS2106: Operating Systems
Lab 4 – Buddy System Simulation

Important:
- The deadline of submission on IVLE is **31st October 5pm**
  - Note that the deadline is a Monday, unlike the previous 3 lab submission deadlines. Reason will be explained in lecture.
- The total weightage is 6%:
  - Exercise 1: 1% [**Demo Exercise**]
  - Exercise 2: 2%
  - Exercise 3: 3%

Section 1. Overview

Since memory management is handled by the Operating System, it is not easy to understand the actual process from outside. Hence, we will write a **simulation** of memory allocation to have a better understanding of memory management in this lab.

There are three exercises in this lab, which are all based on **Buddy System** dynamic memory allocation scheme. The exercises essentially break the buddy system down to the following portions:

- Exercise 1: Setting up the array of linked lists based on the initial memory size.
- Exercise 2: Handling allocation and splitting.
- Exercise 3: Handling deallocation and merging.

Note that the exercises are **not independent**. The solution from the earlier exercises should be reused and built upon for later exercises.

Section 2. Exercises in Lab 4

Since we use notation directly from the lecture notes, please refer to the lecture slides and tutorial for the algorithm description of the buddy system memory management. A couple of implementation tips are suggested in the respective sections.

2.1 Exercise 1 – Setup and Debug [Demo Exercise]

In this exercise, add to **ex1.c** to do the following:

1. Ask the user for the following parameters:
   a. **`MS`**: Initial free size. **`0 < MS <= 4096`**.
   b. **`SS`**: Smallest allocatable size in $\log_2$. i.e. 2^{SS} is the actual size.
   c. **`LS`**: Largest allocatable size in $\log_2$, i.e. 2^{LS} is the actual size.

   Then, setup the initial array `A[]` and **print the amount of free memory wasted** due to the allocatable unit restrictions. The amount may be **0** if the initial free memory can be utilized fully.
   Notes:
   - Choose the largest possible free partition size whenever possible.
   - There may be left over (unusable) free memory due to the smallest alloctable size.
   - **`0 ≤ SS < LS ≤ 12`**. **$2^{SS} \leq$ `MS`**.

2. Implement a printing function that print out the array **`A[]`** to support debugging in exercise 2 and 3.

Sample Session (user input in **bold**):	
`1400`	*//initial free memory size is 1400*
`6`	*//smallest allocatable size is $2^6 = 64$*
`9`	*//largest allocatable size is $2^9 = 512$*
`56`	//Print wasted memory space (1400-512-512-256-64)
`A[9]: [0] [512]`	*//2 free blocks of 2^9 (512) at address 0 and 512*
`A[8]: [1024]`	*//1 free block of 2^8 (256) at address 1024*
`A[7]: EMPTY`	*//no free blocks on any other levels*
`A[6]: [1280]`	*//1 free block of 2^6 (64) at address 1024*
`A[5]: EMPTY`	//Should be empty from this point onward due
`A[4]: EMPTY`	// to the smallest alloctable unit
`A[3]: EMPTY`	
`A[2]: EMPTY`	
`A[1]: EMPTY`	
`A[0]: EMPTY`	

The debug printing function prints the linked list at each level of the array. For non-empty linked list, the starting address of each free block is printed as **`[StartAddr]`**. If there are multiple free blocks, a space is printed between the blocks information.

Note: This exercise is really just for you to code the basic structure and debugging function. Your lab TA will briefly inspect your code to ensure you have done something useful (rather than just "faking" the printout ☺).

2.2 Exercise 2 – Allocation and Splitting

In this exercise, you will add handling for memory allocation requests. Modify your solution from **ex1** to accommodate the following changes (in **bold**):

1. Ask for initial memory size, smallest and largest allocatable unit in $\log_2$.
2. **Ask for total number of request, T.**
3. **Read T requests, each request has the format:**
   `"1 request_size"`
   Note: the "**`1`**" is to distinguish between the allocation and deallocation request. It is only useful in ex3.
4. **Allocate a block suitable for *request_size* using buddy system. Note that when choosing a block for any action (e.g. for splitting, for allocation etc), choose the block with the smallest starting address (if there are multiple options).**
   a. **Report the allocated block starting address if successful.**
   b. **Print a "`-1`" if the block cannot be allocated.**

Sample Session	
Input	**Output**
280 *//initial memory size* **5** *//smallest size, 32* **8** *//largest size, 256* **4** *//4 requests* **1 64** *//request to allocate 64 bytes* **1 30** **1 120** **1 128**	24 //24 bytes wasted (280-256) **0 //64 bytes allocated at addr 0** **64 //32 bytes allocated at addr 64** **128 //128 bytes allocated at addr 128** **-1 //failed to allocate**

To ease your programming effort, you are given "**`testX.debug`**" in the **`ex2/`** directory which contains detailed debug information after every request. These debug files are provided as guidance only, there is no need to produce the same detail tracing information from your code.

Note: Make a copy of your solution **`ex1.c`** and rename it **`ex2.c`** for modification.
Note 2: Use the debug printing from ex1 to help with your coding. **However, make sure you turn off the debug output before submission.**
Note 3: The provided test cases **do not cover** all scenarios, design additional tests of your own to check for corner cases.

Hint 1: Recursion "may" be your friend here ☺.
Hint 2: Although it is not required for this exercise, you are strongly encouraged to keep the successful allocation information in some fashion, minimally keeping track of the **`{starting address, allocated block size, actual size in use}`** so that you'll have an easier time for **`ex3`**.

2.3 Exercise 3 – Deallocation and Merging

You are now ready to complete the entire Buddy System implementation. Modify your implementation from **`ex2`** to handle memory deallocation requests. The modified steps are in **bold**:

1. Ask for initial memory size, smallest and largest allocatable unit in $\log_2$.
2. Ask for total number of request, **T**.
3. Read **T** request, each request has the format:
   "`1` ***`request_size`***" for allocation requests OR
   "`2` *`starting_address`*****" for deallocation requests.**

4. For allocation request: Allocate a block suitable for *request_size* using buddy system.
   a. Report the allocated block starting address if successful.
   b. Print a "**`-1`**" if the block cannot be allocated.

5. **For deallocation request: Free the block at the indicated ***`starting_address.`***
   a. **Print "`ok`" if the block is freed successfully.**
   b. **Print "`failed`" if the starting address does not correspond to any currently allocated block.**

6. **After all requests have been performed. Print the following 3 statistics regarding the memory usage at that point:**
   a. **Total memory in use (allocated).**
   b. **Total free memory (i.e. a + b = initial memory size).**
   c. **Total internal fragmentation (for each allocated block, sum the allocated size – actual requested size).**

   Note: (a) + (b) does not consider the bytes wasted due to allocatable units constraints.

Sample Session **Input**	**Output**
290 *//initial memory size* **5** *//smallest size, 32* **8** *//largest size, 256* **6** *//6 requests* **1 60** **1 62** **1 64** **2 64** *//free block at address 64* **2 128** *//free block at address 128* **2 64** *//free block at address 64*	2 //2 bytes wasted (290-256-32) 0 //64 bytes allocated at address 0 64 //64 bytes allocated at address 64 128 //64 bytes allocated at address 128 **ok //64 bytes at address 64 freed** **ok //64 bytes at address 128 freed** **failed //failed to free (block not allocated)** **64 //statistic a: 64 bytes in use** **224 //statistic b: 224 bytes free** **4 //statistic c: 4 bytes internal frag.** **// Detail: Only 60 bytes of the** **// 64 bytes at address 0 is in use** **// ➔ 4 bytes internal frag.**

Note: Make a copy of your **`ex2.c`** and rename it **`ex3.c`**, then modify and add on.

Hint 1: You need to keep track of the allocated memory blocks in order to handle deallocation and the statistics calculation.
Hint 2: Similar to allocation, the free and merge operation is quite suitable to a recursive solution. Not compulsory, but definitely makes your code shorter ☺.

Section 3. Submission

Zip the following files as A0123456X.zip (**use your student id!**)**:**

- **`a. ex2.c`** (Remember to remove all debug messages)
- **`b. ex3.c`** (Remember to remove all debug messages)

Upload the zip file to the "Student Submission→Lab 4" workbin folder on IVLE. Note the deadline for the submission is **31st October, 5pm**.

Again, please ensure you follow the instructions carefully (**output format**, how to zip the files etc). Deviations will be penalized.