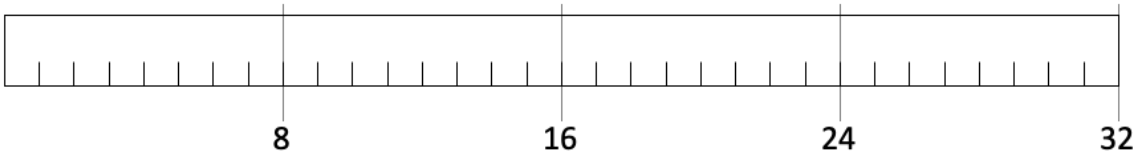
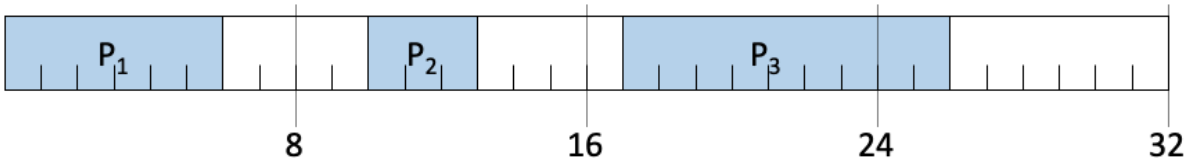
Assignment 3

Deadline in lab on Dec 1

For this assignment, you will be working in a group of maximum 4. The groups can be formed across different sets. We will be working on a real-life application of *Linked Lists*. Older operating systems used to manage memory for processes by creating dynamic partitions. Think about memory (heap) as *one giant block at first*. The *starting index* is 0 and the *size of the memory* is *M* bytes. The initial memory looks like this (Let's say *M* = 32):

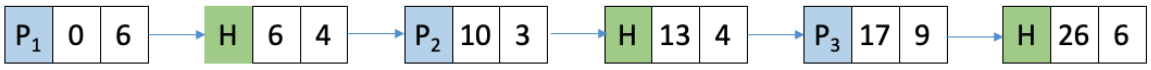


Let's say eventually three processes are loaded in memory by the memory management unit (MMU): The state of the memory can look like this:



- Here, process P_1 starts from index 0 and takes 6 bytes.
- There is a hole (not allocated memory) that starts from index 6 and is 4 bytes.
- Process P_2, starts from index 10 and takes 3 bytes.
- There is a hole that starts from index 13 and is 4 bytes.
- Process P_3, starts from index 17 and takes 9 bytes.
- And finally, there is a hole that starts from index 26 and takes 6 bytes.

Operating systems, use linked lists to represent the above memory:



In this linked list, each node is a block of memory. It has either "H" to indicate there is a hole, or a process identifier, for example "P2". Each node has a start index. We call this **base**. Each node also has the size of the block of memory. This is called **limit**.

For this assignment, you would be creating data structure (linked list) to represent the memory similar to above. You would also implement three operations in memory management unit:

1) **Merging holes**: When we represent memory like above, we want to avoid fragmentation as much as possible. So, if you have two or more blocks (nodes) which are all free memory (holes), you should merge them all into only one block of free memory (hole). The following is an example:

   Example: Imagine the nodes in your memory linked lists are like this:

   Node 1: P1, base = 0, limit = 6
   Node 2: P17, base = 6, limit = 1
   Node 3: H (Hole), base = 7, limit = 4
   Node 4: H (Hole), base = 11, limit = 4
   Node 5: H (Hole), base = 15, limit = 1
   Node 6: P3, base = 16, limit = 10
   Node 7, H (Hole), base = 26, limit = 6

   You can see that Nodes 3, 4 and 5 are all holes. So, we should be merging them into one node. After merging, your linked list should look like this:

   Node 1: P1, base = 0, limit = 6
   Node 2: P17, base = 6, limit = 1
   Node 3: H (Hole), base = 7, limit = 9
   Node 4: P3, base = 16, limit = 10
   Node 5, H (Hole), base = 26, limit = 6

   Your task for the first part of the assignment is to implement the *mergeFreeBlocks* function.

2) **Compaction**: If there are multiple holes in our memory (this is called *external fragmentation*), the operating system might not be able to allocate memory for a new process. At times like this, the operating system might run an operation called ***compaction***. When we run compaction, the operating system moves all allocated blocks of memory in a way that they are all next to each other. This would leave (at most) one big hole at the end of the list. Let's use the example from before.

   Example: Suppose the current state of memory is like this:

   Node 1: P1, base = 0, limit = 6
   Node 2: P2, base = 6, limit = 1
   Node 3: H (Hole), base = 7, limit = 9
   Node 4: P3, base = 16, limit = 10

Node 5, H (Hole), base = 26, limit = 6

After compaction, it should look like this:
Node 1: P1, base = 0, limit = 6
Node 2: P2, base = 6, limit = 1
Node 3: P3, base = 7, limit = 10
Node 4: H (Hole), base = 17, limit = 15

You can see that process P3 is moved up and all the holes are merged into one big hole at the end of the list. Your second task for this assignment is to implement *compaction* function.

3) Finally, you should implement a ***printMemory*** function. The memory should print each block of memory (one line at a time) on the console like this:

   Node 1: P1, base = 0, limit = 6
   Node 2: P2, base = 6, limit = 1
   Node 3: P3, base = 7, limit = 10
   Node 4: H (Hole), base = 17, limit = 15

4) Your program will be user interactive, so you will ask for user inputs. When the program is run by ./a.out, your program will prompt 5 menus exactly as the following:
   1. load an input file
   2. merge holes
   3. compact memory
   4. print memory view
   5. Exit the program
5) Except Option 1, everything else should be pretty self explanatory on what each is supposed to do. Except Option 4, no output is needed. Option 1, 2, and 3 just need to print "operation successful" after a required job is done. Every option except Option 5 should look back to this main menu after an operation is done.
6) If any unexpected user behavior is observed such as selecting Option 2, the program treats this as a corner case (print "ERROR: XX" and exit the program)
7) When Option 4 is selected, it must print the memory view exactly as in the example above.
8) The initial memory will be loaded into your program via the user input. When Option 1 is selected, your program will prompt the following:
   1. Type the file name:

   The user will type the file name and your program must load the file contents into your program. After that point on, you should create a linked list.
9) Here is the sample input file format where Process 1 occupies memory location from 2 (inclusive) and it occupies 4 bytes, so it occupies from byte 2-5.

   P1 2 4

   Here is another sample input file.

P2 3 1
P1 0 1
H 1 2
H 4 6
P3 10 8

Notes:

- The format of the output must exactly match the above example
- Assume that memory starts from address 0 and your OS must be able to handle an infinite amount of memory
- Assume that your program will not load the file twice. The file will be loaded only once
- The processes are not sorted based on its memory address in the input file
- Any corner or not valid cases must print “Error: XX” with XX being a brief description of the error.

Grading

This assignment will be marked out of 12. For full marks this week, you must:

- (2 points) Good coding styles including commenting, division of functions, division of work into different files
- (2 points) Optimization techniques in terms of time and space complexity
- (6 points) Correctly implementing required functions
- (2 points) Handle corner cases

Submissions

This assignment will not be submitted in a traditional github. It will be presented in one of the assigned time slots on the day of the lab. I will not be running the code. I will direct you to perform operations on your machine and you will run the code in front of me. I will also ask you to open files, so I can do a visual inspection for coding styles, comments etc. I will provide some input files as well.