CPSC 457 - Assignment 5

Due date is posted on D2L.
Individual assignment. Group work is NOT allowed.
Weight: 23% of the final grade.

Q1 – Programming question (30 marks)

Write a best-fit dynamic memory allocation simulator, called `memsim.cpp`. Your program will simulate some of the functionality performed by `malloc()` and `free()` in the standard C library. The input to the program will be a page size (specified on the command line) and sequence of allocation and deallocation requests (supplied on standard input). Your simulator will process all requests and then display some statistics.

Throughout the simulation your program will maintain an ordered list of memory chunks (dynamic partitions). Some chunks (partitions) will be marked as occupied, the rest marked as free. Each chunk will also contain its size in bytes. Further, occupied chunks will have a numeric tag attached to it. Your simulator will manipulate this list of chunks as a result of processing requests. Allocation requests will be processed by finding the most appropriately sized chunk and then allocating a memory from it. Deallocation requests will free up any relevant occupied chunks, and also merging any adjacent free chunks.

Allocation request

Each allocation request will have two parameters – a tag and a size. Your program will use best-fit algorithm to find a free chunk. If more than one chunk qualifies, it will pick the first chunk it finds. If the chunk is bigger than the requested size, the chunk will be split in two – an occupied chunk and a free chunk. The tag specified with the allocation request will be stored in the occupied chunk.

The simulation will start with an empty list of chunks, or, if you prefer, a list containing one free chunk of size 0 bytes. When the simulator fails to find a suitably large free chunk, it will simulate asking the OS for more memory. The amount of memory that can be requested from OS must be a multiple of page size. Your program must simulate requesting only the minimum number of pages that will satisfy the current request. The newly obtained memory will be appended at the end of your list of chunks, and if appropriate, merged with the last free chunk.

Deallocation request

A deallocation request will have a single parameter – a tag. Your simulator will find all allocated chunks with the given tag and mark them free. Any adjacent free chunks will be merged. If there are no chunks with the given tag, your simulator will skip such deallocation request.

Command line arguments:

Your program will accept a single command line argument. This argument will represent the page size in bytes. Page size will be a positive integer in range [1..1000000].

Input:

Your program will read allocation requests from standard input, until EOF. Lines containing only white spaces will be ignored. Each non-empty line will represent one request, either allocation or deallocation.

Any line with two integers will represent an allocation request. The first integer will represent the tag of the request, and the second one will represent the size of the allocation request in bytes. Tags will be in range [0..1000] and sizes in range [1..10000000]. For example, the line `"3 100"` represents an allocation request for 100 bytes with tag 3.

A line with a single negative integer will represent a deallocation request. The absolute value of the integer will represent the tag to be deallocated. For example, the line `"-3"` will represent a deallocation request for all chunks marked with tag 3.

Output:

At the end of the simulation your simulator will output two numbers:

- total number of pages requested from the operating system
  - this could be 0, but only if there are no allocation requests in the input
- the size of the largest free chunk at the end of the simulation
  - this can be 0

Limits

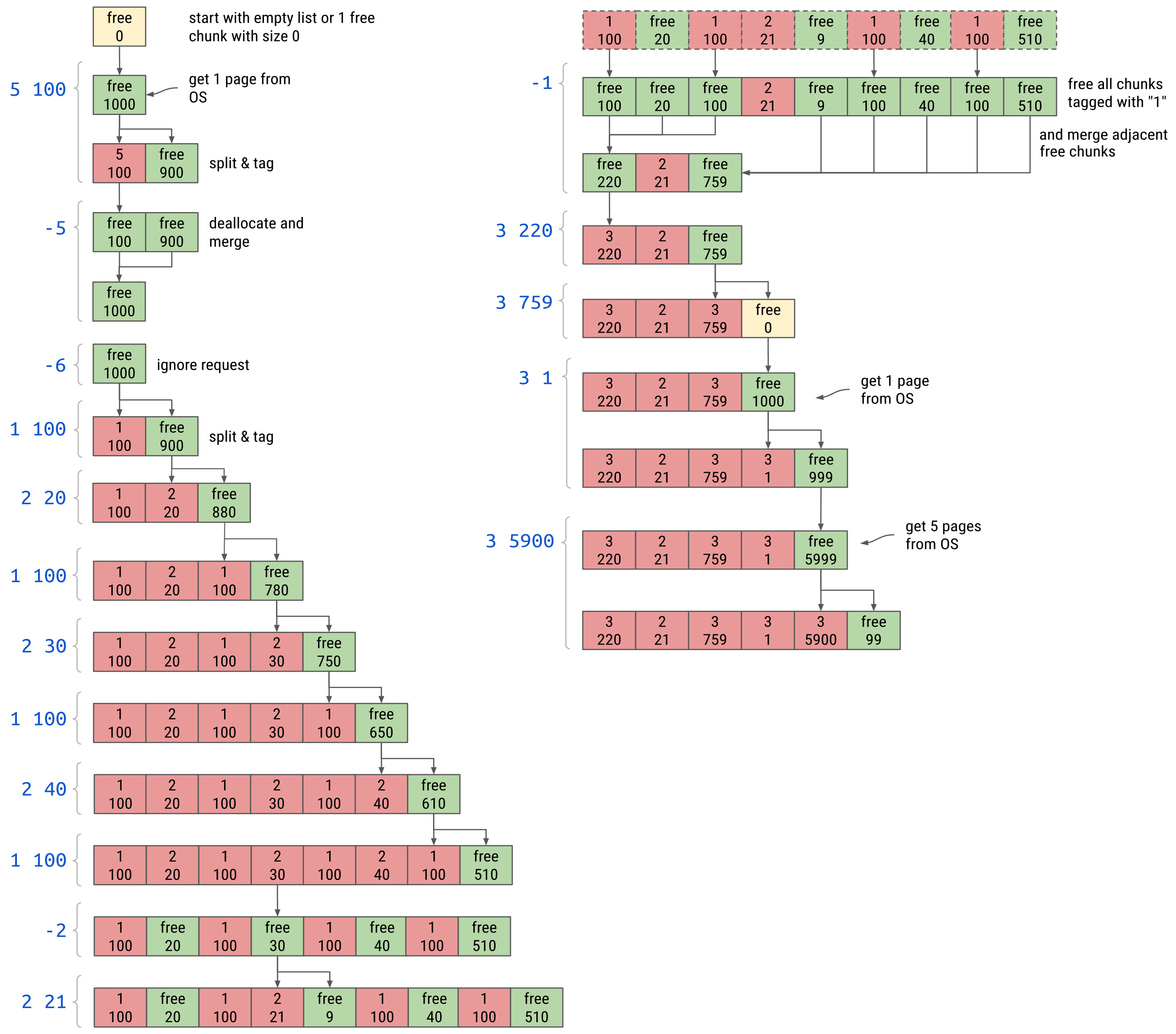
- between 0 and 1000,000 requests
- make sure your code is efficient enough to process any valid input under 10s.

Sample input / output

```
$ cat test1.txt

5 100
-5
-6
1 100
2 20
1 100
2 30
1 100
2 40
1 100
-2
2 21
-1
3 220
3 759
3 1
3 5900
```

```
$ g++ -O2 -Wall memsim.cpp -o memsim
$ ./memsim 1000 < test1.txt

pages requested: 7
largest free chunk: 99

$ ./memsim 1 < test1.txt

pages requested: 6901
largest free chunk: 0

$ ./memsim 33 < test1.txt

pages requested: 210
largest free chunk: 29
```

The illustrations below show how the simulator processes the above file for a page size of 1000.



Hints

Pseudocode for allocation request:

```
- search through the list of chunks from start to end, and
  find the smallest chunk that fits requested size
    - in case of ties, pick the first chunk found
- if no suitable chunk found:
    - get minimum number of pages from OS
    - add the new memory at the end of chunk list
    - the last chunk will be the best chunk
- split the best chunk in two
    - mark the first chunk occupied, and store the tag in it
    - mark the second chunk free
```

Pseudocode for deallocation request:

```
- for every chunk
    - if chunk is occupied and has a matching tag:
        - mark the chunk free
        - merge any adjacent free chunks
```

Starter code: https://gitlab.com/cpsc457/public/memsim . You do not have to use this code.

Q2 – FAT simulation (30 marks)

Write a program `fatsim.cpp` that examines the contents of a FAT table and then displays the following information:

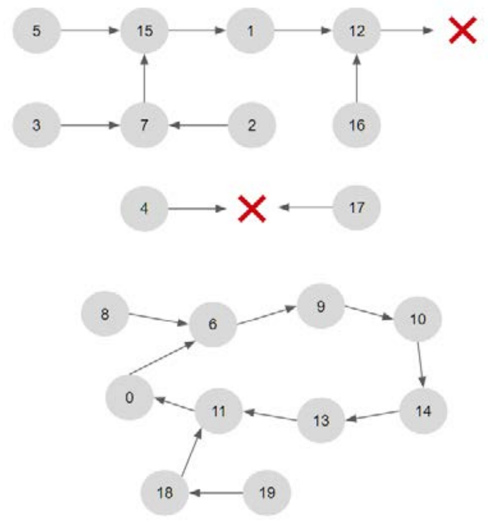
- number of blocks in the largest possible file (length of the longest terminated chain of blocks), and if no files are possible, display “0”.
- number of blocks that could not belong to any files (because they are part of a cycle).

Input:

Your program will read the FAT contents from standard input. FAT entries will be represented by N integers, all separated by at least one whitespace. Each number will be an integer in range [-1 .. N-1], where “-1” represents null pointer (or end of chain), and numbers >-1 represent pointers to blocks. The i-th integer will represent the i-th entry in the FAT.

Below is a sample test file and expected output. The output of your program should match the above output exactly.

```
$ cat test1.txt
  6 12  7  7 -1 15  9 15  6 10
 14  0 -1 11 13  1 12 -1 11 18
$ g++ fatsim.cpp && ./a.out < test1.sim
blocks in largest file: 5
blocks not in any file: 10
```



For example, the first integer ‘6’ represents the fact that block 0 is linked to block 6. The graph on the right illustrates how the blocks are linked. There are 3 files possible in the above FAT (since it contains ‘-1’ three times). The largest file could start either on block 3 or on block 2, but in both cases, it would have a length of 5 blocks. The other two files would start on blocks 4 and 17, respectively, and both would be 1 block long. Notice that no files could start on any of the 10 nodes in the bottom half of the graph, as none of them are part of a terminated chain.

Limits:

- number of entries in FAT will be in the range [1 .. 1000000]
- make sure your code is efficient enough to process any valid input under 10s.

Starter code: https://gitlab.com/cpsc457/public/fatsim . You do not have to use this code.

Submission

Submit 2 files to D2L for this assignment:

`memsim.cpp`	solution to Q1
`fatsim.cpp`	solution to Q2

General information about all assignments:

1. All assignments are due on the date listed on D2L. Late submissions will be not be marked.
2. Extensions may be granted only by the course instructor.
3. After you submit your work to D2L, verify your submission by re-downloading it.
4. You can submit many times before the due date. D2L will simply overwrite previous submissions with newer ones. It is better to submit incomplete work for a chance of getting partial marks, than not to submit anything. Please bear in mind that you cannot re-submit a single file if you have already submitted other files. Your new submission would delete the previous files you submitted. So please keep a copy of all files you intend to submit and resubmit all of them every time.
5. Assignments will be marked by your TAs. If you have questions about assignment marking, contact your TA first. If you still have questions after you have talked to your TA, you can contact your instructor.
6. All programs you submit must run on `linux.cpsc.ucalgary.ca`. If your TA is unable to run your code on the Linux machines, you will receive 0 marks for the relevant question.
7. **Assignments must reflect individual work**. For further information on plagiarism, cheating and other academic misconduct, check the information at this link: http://www.ucalgary.ca/pubs/calendar/current/k-5.html.
8. Here are some examples of what you are not allowed to do for individual assignments: you are not allowed to copy code or written answers (in part, or in whole) from anyone else; you are not allowed to collaborate with anyone; you are not allowed to share your solutions with anyone else; you are not allowed to sell or purchase a solution. This list is not exclusive.
9. Automated similarity detection software will be used to help us check for plagiarism. Your submission will be compared to other students (current and previous), as well as to any known online sources. Any cases of detected plagiarism or any other academic misconduct will be investigated and reported.