In this assignment, you will implement memory management with linked lists as described in Section 3.2.3 of the textbook. Your implementation must use a linked list, not a bitmap. You must use the C++ list class, which implements a doubly linked list, for your list. For this assignment, you may work individually or in a group with one other student.

Setup

In CodeLite on your virtual machine:

- Create a C++ project names Project7.
- Create a class named MemoryChunk. Objects of this class will be used to represent either a single process or a single hole.
- Create a class named MemoryManager.

Submitting Your Assignment

Submit the source code (.cpp and .h files) from your project in a zip file. Name the zip file YourLastNameYourFirstNameProject7.zip. Upload the zip file to Canvas before the due date. Remember that late assignments are not accepted in this course. See the syllabus for details.

Important submission notes for students working in a group:

If you worked with another student, name the zip file using the last names of both students: FirstPersonsLastNameSecondPersonsLastNameProject7.zip. One person should upload the zip files to Canvas. The other person should upload a single document stating who their team member was.

Command Line

Your program must accept a single command-line argument, the name of the input file.

Input File Format

All fields on a line are separated by whitespace.

The first line of the input file contains a string and a single integer. The string is the algorithm and the number is the total amount of memory in the system. Following that, the file contains one action per line. Each line consists of a command string. If the command is "load" the rest of the line contains a process name (a string with no whitespace) and the space requested by the process. Note that there may not be enough free space to fulfill this request without compaction of memory. We will not compact memory in this project. Thus, if you are not able to load a process, you must simply print that information to the console window and move to the next command. If the command is "unload" the rest of the line contains only a process name to remove from memory. You may assume the process already exists in memory. Here is an example file:

```
firstFit 50
load A 10
load B 10
```

```
load C 2
load D 1
load E 2
load F 5
load G 20
load X 30
unload A
unload G
unload C
unload E
unload D
load H 5
unload B
unload H
unload F
```

Assignment

You are to read in the input file and create a list to manage memory. You must have a single list of type std::list that holds all the memory chunks (memory units). Each chunk must either contain the process name loaded into it or something to indicate it is a hole. Each chunk must also contain its starting address and size. When the simulation starts, the list must contain a single hole that fills the entire memory.

When you perform a load, you will find the correct hole for the new process based on the algorithm indicated in the first line of the file. You need to be able to handle the firstFit, bestFit, and worstFit algorithms. In the example output below, these 3 algorithms all produce the exact same results until the point where H is loaded. At that point they differ. Be sure you try all three algorithms in your testing.

If a process is requesting more space that any hole, then your program must indicate that the process cannot be loaded. Normally, this is when a memory compaction algorithm would be run to combine all the free space together. We will not be implementing a compaction algorithm for this lab.

When a process is unloaded adjacent holes need to be combined to create a single large hole. That is, you must never have two holes next to each other on the hole list. This is shown in Figure 3-7 of your textbook and is also demonstrated in the results below. You must combine two adjacent holes by removing one memory chunk from your linked-list and updating the length (and possibly the starting address) of the other memory chunk. Your program must not run through the entire list to combine two adjacent wholes because with a large memory this is very inefficient.

Testing Hint: You may want to create smaller test files so you can test individual cases of your code, for example removing a process at the beginning of the list, the end of the list, the middle of the list and combining holes in the various situations shown in Figure 3-7.

Implementation Notes

- Your project must use the C++ list class (http://www.cplusplus.com/reference/list/list/ to store your memory chunks. std::list is the C++ implementation of a doubly linked list.

MemoryChunk Class

- The constructor must have three parameters: the name of the process or a name to indicate a hole, the starting address of the memory chunk and the size of the chunk.
- You should create accessor and mutator methods as needed.

MemoryManager Class

- The constructor must have one parameter, the name of the input file.
- The MemoryManager Class must have one public member function named run.

Your main method must:

- Check to see if the correct number of command line arguments is given.
- Create an object of type MemoryManager.
- Call the MemoryManager object's run method to start the simulation.

Sample Output for Example Input File

```
hole:  start 0, size 50

load A 10
A:  start 0, size 10
hole:  start 10, size 40

load B 10
A:  start 0, size 10
B:  start 10, size 10
hole:  start 20, size 30

load C 2
A:  start 0, size 10
B:  start 10, size 10
C:  start 20, size 2
hole:  start 22, size 28

load D 1
A:  start 0, size 10
B:  start 10, size 10
C:  start 20, size 2
D:  start 22, size 1
hole:  start 23, size 27
load E 2
A:  start 0, size 10
B:  start 10, size 10
C:  start 20, size 2
```

D: start 22, size 1
E: start 23, size 2
hole: start 25, size 25

load F 5
A: start 0, size 10
B: start 10, size 10
C: start 20, size 2
D: start 22, size 1
E: start 23, size 2
F: start 25, size 5
hole: start 30, size 20

load G 20
A: start 0, size 10
B: start 10, size 10
C: start 20, size 2
D: start 22, size 1
E: start 23, size 2
F: start 25, size 5
G: start 30, size 20

load X 30
Unable to load process X
A: start 0, size 10
B: start 10, size 10
C: start 20, size 2
D: start 22, size 1
E: start 23, size 2
F: start 25, size 5
G: start 30, size 20

unload A
hole: start 0, size 10
B: start 10, size 10
C: start 20, size 2
D: start 22, size 1
E: start 23, size 2
F: start 25, size 5
G: start 30, size 20
unload G
hole: start 0, size 10
B: start 10, size 10
C: start 20, size 2

D: start 22, size 1
E: start 23, size 2
F: start 25, size 5
hole: start 30, size 20

unload C
hole: start 0, size 10
B: start 10, size 10
hole: start 20, size 2
D: start 22, size 1
E: start 23, size 2
F: start 25, size 5
hole: start 30, size 20

unload E
hole: start 0, size 10
B: start 10, size 10
hole: start 20, size 2
D: start 22, size 1
hole: start 23, size 2
F: start 25, size 5
hole: start 30, size 20

unload D
hole: start 0, size 10
B: start 10, size 10
hole: start 20, size 5
F: start 25, size 5
hole: start 30, size 20

load H 5
H: start 0, size 5
hole: start 5, size 5
B: start 10, size 10
hole: start 20, size 5
F: start 25, size 5
hole: start 30, size 20

unload B
H: start 0, size 5
hole: start 5, size 20
F: start 25, size 5
hole: start 30, size 20

unload H
hole: start 0, size 25
F: start 25, size 5
hole: start 30, size 20

unload F
hole: start 0, size 50

Grading Criteria (20 points possible)
Note: Your program must compile to receive credit for this assignment!

Points	Criteria
0-1 point	**Main Function:** Is the main function implemented as described in the assignment? Does it accept one command line argument, the name of the input file? Is a "usage" error message printed if the number of command lines arguments is not correct? Does the main function create an instance of the MemoryManager class and call the run method?
0-2 points	**MemoryChunk Class** Does the program implement a MemoryChunk class? Are function prototypes and member variables declared in the header (.h) file? Are all member variables private? Are all member functions implemented in the .cpp file of the corresponding class? Does the constructor have three parameters as described above? Are accessor and mutator methods used to access member variables?
0-2 points	**MemoryManager Class Structure:** Does the MemoryManager class constructor have one parameter as described above? Does it contain one public member function named "run"? Are private member functions used to divide up the code into logical segments? Are all memory chunks stored in an instance of the C++ list class?
0-6 points	**MemoryManager Correctness:** Does the MemoryManager class load processes into the correct hole for the first fit, best fit, and worst fit algorithms?
0-6 points	**MemoryManager Correctness:** Does the MemoryManager class correctly unload processes? When unloading a process results in two adjacent holes, are these holes combined as described in the assignment?
0-2 points	**MemoryManager Output:** Is the output easy to read? Is the initial hole printed to the console? Is the current command printed followed by the current state of the memory? Are the processes and holes of the current state clearly identified? Is the starting address of the process or hole and the length indicated?
0-1 point	**Style:** Is the code easy to read? Does it follow class style guidelines (See Program Style page in Canvas.)