Linked Lists in C++ April 21, 2014
Courtesy of Mike Wilder **CS 120**

Due: April 28, 2014

Assignment 11: Linked Lists in C++

Exercise 11.1

In this lab we will explore using C++ lists to create a simulator that will aid in the study of tracking robots deployed to mine the Martian soil. The deployed robots will be of two distinct classes: miners and scouts. The role of scout robots is to scour the surface of the planet and take readings of the soil to find locations where miner robots should be deployed. The role of miner robots is, of course, to mine the soil for material of interest.

All robots will be transported to Mars via a "mother ship" called Argus. Argus contains a control center from which all robots (or *Argonauts*) will be dispatched to the planet surface. Argus contains two distinct holding areas for robots: one for miners and one for scouts.

Your task is to simulate the movement of robots to and from the Argus, and to keep track of the location and status of all robots. Your simulation will begin by reading a robot manifest from a file specified by the user. Each line of the manifest file contains information pertaining to a single robot. The information for a single robot is presented in the following format:

```
RobotNumber RobotType RobotManufacturer
```

where `RobotNumber` is a unique number greater than zero and less than 2,000,000,000 that is assigned to each robot, `RobotType` is one of 'M' for miner or 'S' for scout, and `RobotManufacturer` is a string of no more than 64 characters identifying the manufacturer of a given robot.

Once your simulation has read all robots from the manifest file, it should begin interactively accepting commands from the user. Your simulation should accept the following commands:

1. `D`: Dispatch a scout robot from the Argus. This command takes a single argument, which is the quadrant number (1-4) where the scout will be deployed. Your simulator can deploy any scout robot that is on the Argus. If there exist no robots to fulfill the request, your simulator should emit a message indicating such. If successful, your simulator should emit a message of the form "Robot X deployed to quadrant Y".

2. `d`: Dispatch a miner robot from the Argus. This command takes a single argument, which is the quadrant number (1-4) where the miner will be deployed. Your simulator can deploy any miner robot that is on the Argus. If there exist no robots to fulfill the request, your simulator should emit a message indicating such. If successful, your simulator should emit a message of the form "Robot X deployed to quadrant Y".

3. `R`: Retrieve a scout robot from deployment and return it to the Argus. This command takes a single argument, which is the `RobotNumber` of the scout to be retrieved. Your simulator should verify that the robot is indeed a scout and is indeed deployed. If the

request is invalid, your simulator should emit a message indicating such. If successful, your simulator should emit a message of the form “Robot X retrieved from quadrant Y”.

4. `r`: Retrieve a miner robot from deployment and return it to the Argus. This command takes a single argument, which is the `RobotNumber` of the miner to be retrieved. Your simulator should verify that the robot is indeed a miner and is indeed deployed. If the request is invalid, your simulator should emit a message indicating such. If successful, your simulator should emit a message of the form “Robot X retrieved from quadrant Y”.

5. `?`: The user requests information regarding a specific robot. This command takes a single argument, which is the `RobotNumber` of the robot for which information is requested. Your simulator should verify that the `RobotNumber` is valid. If the `RobotNumber` does not specify a valid robot on this Martian excursion, your simulation should emit a message indicating such. If the `RobotNumber` is valid, your simulator should emit an information message of the form

   ```
   Robot <RobotNumber>
       type: <RobotType>
       status: <Status>
       num-deployments: X
       manufacturer: <ManufacturerName>
   ```

   Where <`Status`> is one of

   ```
       "Deployed in quadrant X" or
       "In miner holding bay" or
       "In scout holding bay"
   ```

6. `s`: The user requests to know the number of scout robots that are deployed on the surface. This command takes no arguments. Your simulator should print a message of the form: “Scouts deployed: X”.

7. `S`: The user requests to know the number of scout robots that are in the scout holding bay. This command takes no arguments. Your simulator should print a message of the form: “Scouts in holding bay: X”.

8. `m`: The user requests to know the number of miner robots that are deployed on the surface. This command takes no arguments. Your simulator should print a message of the form: “Miners deployed: X”.

9. `M`: The user requests to know the number of miner robots that are in the miner holding bay. This command takes no arguments. Your simulator should print a message of the form: “Miners in holding bay: X”.

10. `q`: The user requests to quit the simulation. Your simulator should use the `delete` operator to free all dynamically allocated memory and terminate.

Your simulator must employ at least two C++ lists: one list containing scouts, and another containing miners.

HINTS

1. Do not allow yourself to become confused by the details of this problem. Sit back and carefully consider what functionality you do and don't need to create your simulator. Spend your time *thinking* so that you don't end up spending your time *debugging*.
2. Notice that the narrative does not provide an upper bound on the number of robots that will be contained in a robot manifest. You can assume that a robot manifest will contain at least 2 robots because it takes a lot of fuel to get to Mars.
3. The information that is required to keep track of a robot goes beyond the information that is explicitly specified in the robot manifest file. You should re-read this assignment carefully and determine what information that you will need to keep track of the robots. You are free to encode the information within a robot in any way that you choose, as long as you can grant valid requests and deny invalid ones.
4. Although the narrative specifies that each `RobotNumber` will be unique, notice that it does not specify any restrictions regarding the assignment of unique numbers to specific robots. For instance, the first robot contained within the manifest file could be assigned a `RobotNumber` of 1 or 4321. The assignment of `RobotNumber` is arbitrary and your simulator has no control over the assignment of this number.

When you have completed your program, print and turn in all your source code files and a copy of test output of your program (using the script command).