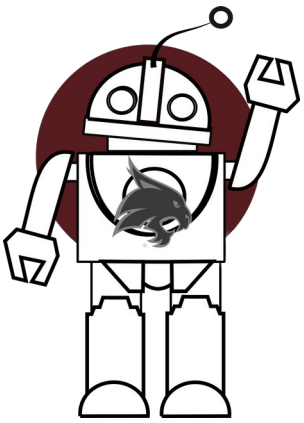
Bobcat Robotics

Software Requirements Document

Software Name: Multi-Robo Guider (MRG)
Author: Gentry Atkinson
Version: 1.3.0



1. Introduction

Product Purpose: this product is being developed as a generalized robotic guidance platform. This product will take guidance input from the console and translate those values into positional data for a robotic platform.

2. Standards

Hardware: this product should be capable of running on a Linux platform with the gcc compiler and a means of text I/O.

Schedule: completion of this product is expected no later than 30 June.

Language: this project will be implemented using the C++ programming language.

3. System Description

System Context: this will be a self-contained piece of software that will not rely on external platforms for its basic operation.

User Characteristics: users will be trained robot operators who are proficient with computer operation. This project will not be responsible for user training.

4. Functional Requirements

4.1 Startup (same as Version 1.1): when the software starts, it should ask the user for a number of robots to begin with. It should also ask the user for a unique identifier for each robot. You can assume that the user is entering unique names for each robot, you do not need to check.

4.2 Menu (updated for Version 1.3): the user should be shown the menu of possible commands and should be able to select one command from the menu. Your program should print an error message if the user enters a letter that is not in the menu. The following commands are available in **MRG**:

- **M**- move one robot
- **D**- print the distance each robot has moved
- **U**- Update robot list
- **R-** rename one robot
- **Q**- quit the program

4.3 Move (same as Version 1.2): The user should be prompted to enter the unique identifier of one robot and a direction. The program should print an error if the user enters an invalid direction or if they enter an invalid robot identifier. The following directions are allowed:

- **U**- up or positive y
- **D**- down or negative y
- **R**- right or positive x
- **L**- left or negative x
- **S-** stop or start moving.

Your program should update the position of the correct robot by an amount equal to the robots speed (defined in **Section 5.1**) and then print the new position. Every robot starts at position 0,0 and can move infinitely in any direction. The position should always be printed as **X,Y**.

4.4 Distance (same as Version 1.1): This menu option should print the unique identifier of each robot and the distance that it has moved in neatly formatted columns, sorted with the robot that has moved the farthest at the top.

4.5 Update (new for Version 1.3): This menu option should allow the user to add or delete robots from the list. The options for updating the list are:

- **A**- add a robot
- **D**- delete a robot

Either option should be followed by a robot name. “Add” should add a robot with the new name to the list of robots. “Delete” should remove the robot with the matching name from the list, or do nothing if no robot in the list has a name which matches.

4.6 Rename (same as Version 1.2): The user should enter two names following this menu option. The first identifies the robot to rename, the second is its new name. Do nothing if the first name is not in the list.

4.7 Quit (same as Version 1.1): Your program should print a short parting message when the user selects ‘Q’.

4.8 Errors (same as Version 1.1): Your program should print an appropriate error message and return to the top level menu (described in 4.2) as soon as it receives unrecognized input.

5. Non-functional Requirements (updated for Version 1.3)

- The program should be submitted as three .cpp file and two .h files. The names of the files should be: **main.cpp**, **Robot.h**, R**obot.cpp**, **RobotList.h**, and **RobotList.cpp**.
- The first three lines of the program should be the author’s name, the date that the assignment was written, and a list of collaborators.
- The author is expected to follow the Style Guidelines.
- Your program should store all information about Robots in a linked list (described below). The size of this list should be able to grow or shrink at runtime.
- Correct any issues that you had with Version 1.3
- Your program should delete all dynamically allocated memory.
- Your functions should be well commented. The comments for each function should include:
  - Name
  - Parameters
  - Returns
  - Side Effects

5.1 Robot class (same as Version 1.2)
The program should store the robot’s name, speed, and positional data in a class with the following member variables:

- **X**- the current X value of the robot’s position
- **Y**- the current Y value of the robot’s position
- **lastCommand**- the last direction that the robot moved
- **currentSpeed**- the speed that the robot is traveling
- **distance**- the total distance that the robot has traveled
- **name**- the name of the robot

You may add additional class members if you choose too.

The initial values of the X and Y members should be set to 0. This should be done with a **constructor.** You may also set values for the other member variables in the constructor if you choose.

All member variables should be private. Add accessor functions (**getter**) and mutator functions (**setter**) for every member variable that needs to be accessed from outside the class. Only add a getter or a setter if it is necessary.

A robot speeds up the longer it moves in one direction. A robot's first move in a direction will move one unit in that direction, the second move in the same direction increases the robot's speed to two, the third move in the same direction to three, and then to four. Four is the maximum speed for these robots. Changing direction resets the robot's speed to 1. Stopping and the starting again resets the robot's speed to 1.

A robot that has been told to stop should not change it's position until it has started again. A stopped robot should print an error message if it is told to move. A robot that has stopped and then started again should only move one space even if it is moving in the same direction as its last successful move. Every robot should be moving, not stopped, when the program starts.

The distance value should store the total number of spaces that the robot has moved, not its euclidean distance from the starting position. For example, a robot that moves four spaces up and then three spaces right should have a distance of seven even though the straight line distance back to (0,0) is only five.

5.2 RobotList class (new for Version 1.3)

The Robot objects should be stored in a linked list implemented as a class. You can include another **Node** class as a private data type of the RobotList class. The data members of the **Node** class can be public (as they are invisible outside of RobotList).

The public data members of the Node class are at least:

- Robot* val
- Node* next

The private data members of the RobotList class are at least:

- Node* head

You can add more data members to either Node or RobotList if you would like, but the data members of RobotList should all be private.

You should add member functions to this class that let you insert and delete new Robots.

The destructor of this class should de-allocate all dynamically allocated memory that this class is responsible for.

5.3 findRobot function (updated for Version 1.3)

This function be re-implemented as a member function of the RobotList class. Find the Node of one Robot in the linked list of Robots using its unique identifier.

The parameters of this function are:

- **string identifier**: the target unique identifier

Return a pointer to the robot with a matching identifier or NULL if the identifier is not in the list.

5.4 moveRobot member function (same as Version 1.2)
The position of the Robot objects should be updated using a member function of the Robot class called moveRobot.

The parameters of this member function are:

- **char d**: a character value representing one direction

The robot's position should be updated as described **in Section 4.3** (so if d == 'U', then increase r.y by r.currentSpeed, etc.)

This function also needs to update the current speed and the distance traveled for the robot. The robot's speed increases by one unit every time it moves in the same direction up to a maximum speed of four. See **Section 5.1** for more information about speed and distance.

Only this member function should be used to change the robot's position after the initial position has been set.

Make this function a class member of Robot. This is not a stand-alone function.

5.4 makeRoboList function (removed for Version 1.1)
This function is no longer necessary. Use a constructor for you RoboList class instead.

5.5 Interface (same as Version 1.1): the display should be neatly formatted and easily readable by the user. You may choose the exact formatting of the output, but it must be consistent on a terminal with an 80 character width.

Your program should let users enter characters as upper OR lower case.

5.6 File Structure (updated for Version 1.3):
You should build this project as five separate files:

1. Robot.h: the header file for the Robot class
2. Robot.cpp: the implementation file for the Robot class
3. RobotList.h: the header file for the RobotList class
4. RobotList.cpp: the implementation file for RobotList
5. main.cpp: the driver file

6. Guidance

- The robot should speed up every time it moves in the same direction. You need to be able to compare characters regardless of whether they are upper or lower case. The ctype.h library might help.
- The program should keep running until the user selects Q. There should be a loop in your main function that breaks when a user types Q.
- The iomanip library might help you with your formatting, but it is not required. Just make sure everything looks good.
- You will need a search algorithm to find one Robot in the list of Robots. Think about which one is best for this project.
- The **new** keyword is used to dynamically allocate memory. Remember to free that memory using the **delete** keyword.
- You still need to print the distances that the robots have moved in sorted order. You *can* do this by sorting the linked list of of Robots, but sorting a linked list can be difficult. An easier option *might* be to write the distance values into an array, and then sort that array.

7. Sample Test Cases

Input	**Output**
Start program 3 Fatemeh Ariel Carlotta	Welcome to MultiRobo Guider. Please select: M- move D- distance Q- quit
M Fatemeh u	Fatemeh’s position is 0,1
m Carlotta R	Carlotta’s position is 1,0
K	Invalid command
m Carl R	Robot not in list.
M Fatemeh f	Invalid direction.
m Fatemeh U	Fatemeh’s position is 0,3
U a Bob	Bob has been added
U d Carlotta	
m Ariel s	Ariel is now stopped
m Ariel l	Ariel cannot move while stopped
m Carlotta R	Robot not in list.
R Fatemeh Ada	Fatemeh is now called Ada
D	Ada 3 Ariel 0 Bob 0
Q	Goodbye