Middle East Technical University | Department of Computer Engineering

CENG 140

C Programming

Spring 2015-2016

Take Home Exam 2

1 Regulations

- **Due date:** 8 June 2016, Wednesday, 23:55
  Not subject to postpone, please do not ask for.
- **Submission:** Electronically. You will be submitting your program source code written in a file which you will name as **the2.c** through the COW. Resubmission is allowed (till the last moment of the due date), the last will replace the previous.
- **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.
- **Team:** There is **no** teaming up. The take home exam has to be done/turned in individually.
- **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.
- **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications.

2 Introduction

This assignment aims to get you familiar with dynamic memory management, structures and linked list operations in C.

A country named Cidoll plans to pursue a space mission to understand whether there is life on the Mars or not, by sending a robot named "ROBdirik", in addition to the "Curiosity" of NASA. However, they do not have any experience about such missions since it is the first time. Fortunately, the research team comes up with a remote programmable task wheel idea. Simply, the tasks the robot will do are stored in a task wheel (initially empty); and the robot will get commands from the Earth Station to add/delete tasks to wheel. Occasionally, the robot will get a run command from the Earth, and in this case it will run the current slice of the wheel, that is, it will do whatever ordered in the task description (something in outside world that we really don't care in this assignment) and turn the wheel (to possibly point to a new slice) as specified in the slice.

Keywords: *Circular Doubly Linked List, Dynamic Memory Allocation*

3 Specifications

- The task wheel is a list of tasks in a circular manner which comes with a *beginning slice* information. It holds the *current slice* and *current direction* information.
- The task wheel comprised of one or more slices. Each slice is a representation of a task. Moreover, they include additional information required by the task wheel system. The information kept in a slice is as follows:
  - the description of the task
  - the direction: 'R' (i.e. right) or 'L'(i.e. left) that points the next slice to be set as the current slice when ROBdirik runs the current slice
  - the number of slices that will be passed by turning the wheel when the slice runs
- There is an exceptional task named **"DRILL"** which does not specify a direction and number of slice. Thus, when a DRILL task is stored in a slice, the direction and the number of slices are set as 'N' (i.e. None) and -1 respectively. Since the drill task doesn't include the turning operation on the wheel, we actually don't care the turning direction and the number of slices.
- At the beginning, the wheel is empty and the direction is **right**. When the first slice is added, it becomes the beginning and the current slice. It is guaranteed that the beginning slice will not be deleted.
- There are five commands, namely *"ADD", "DELETE", "RUN", "TEST", "CONTROL"*:
  - ADD: Adds a new slice in the current direction after the current slice. If the current direction is right, then it adds to the right of the current slice. If the current direction is left, then it adds to the left of the current slice. After addition, the wheel turns such that the current slice is the new one.
  - DELETE: Deletes the current slice and sets the current slice to the previous slice according to the current direction. If the direction is right, then it is set to the slice on the left and otherwise slice on the right.
  - RUN: When the command is RUN, the robot executes the task that is stored in the current slice. For instance, if the task is "Mv10 R 2", the robot will make a move of 10 steps in the physical world, and then turns the task wheel such that the current slice becomes the second slice on the right of the current slice. This is the opposite when the direction is left. RUN is the only command that can change the current direction.

    There is an exceptional task named DRILL (for which we assume that the task of ROBdirik in real world is drilling a hole on the ground to seek for water), and in this case it does not turn the wheel after it runs the task. Thus, the current slice remains the same.
  - TEST: Runs the slices one by one starting from the beginning. When ROBdirik runs the first slice, the wheel stops at another slice. Then, ROBdirik runs the current slice and continues running slices until the slice limit is exceeded or a "DRILL" task is executed. Slice limit restricts the number of slices that ROBdirik runs. It is **not** about the number of slices passed while running the slices.
  - CONTROL: ROBdirik does not run any slices, but controlls the structure of the wheel by turning it when this command is sent. It first turns the wheel such that starting from the beginning, all the slices on the right are passed to finish a full turn and then all the slices on the left are passed to finish a full turn on the opposite direction. The first turn ends at the slice on the left of the beginning slice and the second one finishes at the beginning one.

- **The task wheel must be kept dynamically as a Circular Doubly Linked List by using "struct" construct of C. The solutions depending on any other static constructs (arrays, etc.) will get NO credits!**
- **For the task desciption strings, again, the use of static arrays is strictly forbidden. Thus, for the strings you should also use dynamic arrays of characters.**

4 Input

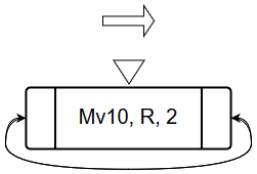
- You are expected to write a program that reads from standard input. You may use redirection for simplicity when testing your code.
- An example input is given below. It can be downloaded as input.txt from COW and tested using redirection.
- Each line represents a command received by ROBdirik and ends with the new line character (i.e. '\n').
- Following items explains the structure of the input lines:
  - Between each part of a command there is exactly one white space.
  - For the ADD command, we have "ADD" followed by an integer indicating the number of characters in the task name, the task name as a string of any length, then the turning direction as a single character ('R' or 'L') and the number of slices.
  - For the ADD command, when the task name is "DRILL", the turning direction and the number of slices are omitted.
  - For the DELETE command, we have "DELETE".
  - For the RUN command, we have "RUN".
  - For the TEST command, we have "TEST" followed by the slice limit.
  - For the CONTROL command, we have "CONTROL".
- All inputs will start with an ADD command to construct a one slice wheel and end with a TEST command.
- An input has exactly one TEST command which defines the end of the input.
- There can be a case in which RUN command is sent to ROBdirik and the current slice has "DRILL" as the task. Then the remaining commands are ignored and the input ends.
- There will be no erroneous input.

- An example input:

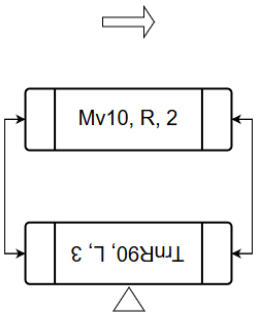
```
ADD 4 Mv10 R 2
RUN
ADD 6 TrnR90 L 3
RUN
ADD 3 Sit L 5
ADD 5 Mv100 L 10
RUN
RUN
ADD 7 TrnL100 R 3
ADD 4 Rise R 1
ADD 6 Stop10 R 7
RUN
DELETE
ADD 5 DRILL
CONTROL
TEST 2
```

- The following figures represents the structure of the wheel created by the input above. The triangle shows the current slice and the arrow shows the current direction.

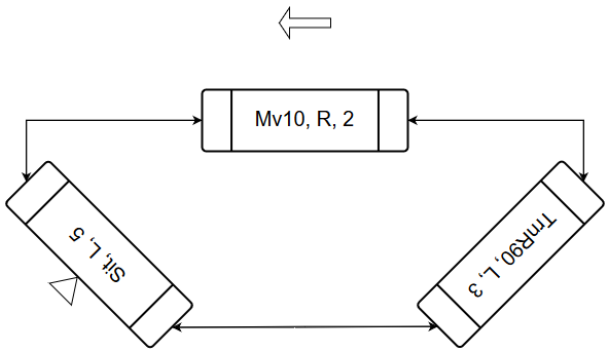
  - The wheel after the first ADD command is runned.



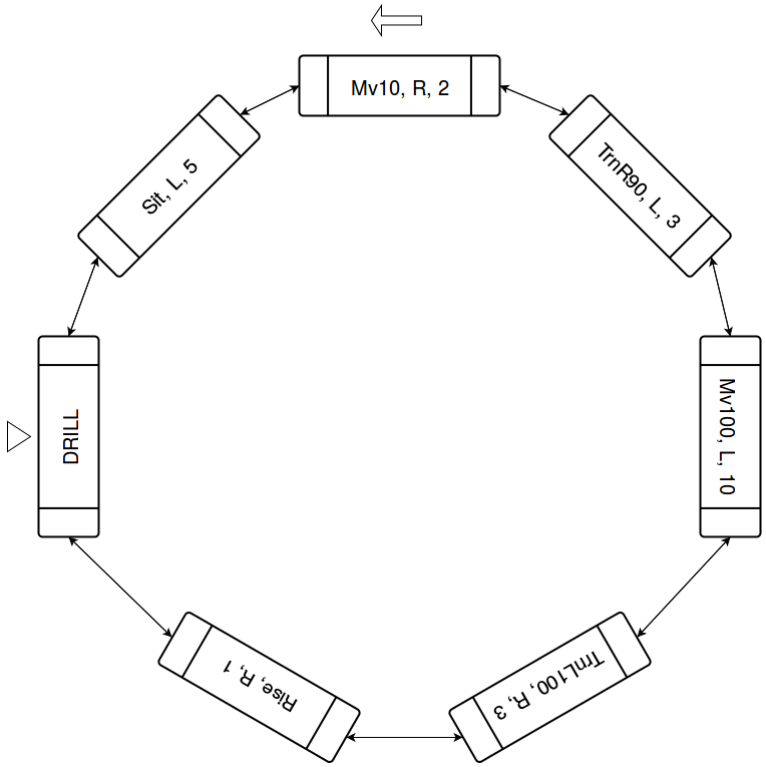
  - The wheel after all the input lines until the second ADD command (including the second ADD command) are runned.



– The wheel after all the input lines until the third ADD command (including the third ADD command) are runned.



– The wheel after the TEST command (i.e. all of the input lines) is runned.



5 Output

- Your program will write to standard output.
- Each line of the output should represent the execution of last command and end with the new line character (i.e. '\n').
- The following items explain the structure of the output lines:
  - **Between each part of an output line, there should be exactly one whitespace.**
  - For the ADD command, the output line should contain a letter 'A', the task names of the slice on the left of the new one, the added slice and the slice on the right of the added one.
  - For the DELETE command, the output line should contain a letter 'D', the task names of the slice on the left of the slice that will be deleted, the one that is to be deleted and the slice on the right of the slice to be deleted.

  - For the RUN command, the output line should contain a letter 'R', task names of the slices in the order while the wheel is turning, omitting the slice where the run operation starts. For example, assume that the slice that includes "Mv10" task with 'R' as direction and 8 as the number of slices is the current slice and a RUN command is received, the output should contain the letter R followed by 8 strings and one whitespace per pair in between. The strings should start from the task name of the slice on the right of the slice including "Mv10" task where the RUN operation starts.
  - For the TEST command, the output line should contain a letter 'T', the task name of the beginning slice and the task names of the slices in the order when the run operations are done one by one until the test stops. If the test has stopped due to the slice limit, "`DRILL_NOT_FOUND`" should be printed after the task names. If a "DRILL" task is found in the slice limit since ROBdirik will run it and stop the execution "`DRILL`" will be printed after the task names.
  - For the CONTROL command, the output line should contain a letter 'C', the task names of the passed slices while controlling the structure of the wheel. Since the first turn starts from the beginning slice and ends at the slice on the left of it (i.e. the last slice), the passed slides are the ones in between. Likewise, the second turn starts from the last slice and ends at the beginning one; thus, the passed slices are the ones in between.

- Sample output for the input above:

```
A Mv10 Mv10 Mv10
R Mv10 Mv10
A Mv10 TrnR90 Mv10
R Mv10 TrnR90 Mv10
A TrnR90 Sit Mv10
A TrnR90 Mv100 Sit
R TrnR90 Mv10 Sit Mv100 TrnR90 Mv10 Sit Mv100 TrnR90 Mv10
R TrnR90 Mv100
A Mv100 TrnL100 Sit
A TrnL100 Rise Sit
A Rise Stop10 Sit
R Sit Mv10 TrnR90 Mv100 TrnL100 Rise Stop10
D Rise Stop10 Sit
A Rise DRILL Sit
C Mv10 TrnR90 Mv100 TrnL100 Rise DRILL Sit DRILL Rise TrnL100 Mv100 TrnR90 Mv10
T Mv10 TrnR90 Mv100 TrnR90 Mv10 Sit DRILL
```

6 Compile, Run and Test

- Make sure that you can compile and run your code on Ineks with:

```
gcc the2.c -Wall -pedantic-errors -ansi -o the2
./the2 < input.txt
```