

Robot Tank Simulation

We've all seen various remote-controlled toys, from miniature racecars to artificial pets. For this project you will implement a simulated robotic tank. The tank will respond to simple commands, mediated by a manager object. The syntax and semantics of the commands are described in a later section of this specification.

Naturally your solution should function correctly, according to this specification. In addition, you should design your solution so that the components are robust and reusable in future projects. This is especially true for the tank.

The requirements of this project are not terribly complex. In other words, it is not especially difficult to produce an implementation that will produce correct results. Because of that, we will place great emphasis on the quality of your design when your project is evaluated.

On startup, the program will read a script file and carry out the commands it contains, writing any output to a log file. The names of the script file and the log file will be specified as command-line arguments to the program.

Program Invocation:

Your program **must** take the names of the input and output files from the command line — failure to do this will irritate the person for whom you will demo your project. The program will be invoked as:

```
Tank <script file name> <log file name>
```

If the specified input file does not exist, the program should print an appropriate error message and either exit or prompt the user for a correction.

Script File Description:

Lines beginning with a semicolon (` ; `) character are comments; your program will ignore comments. An arbitrary number of comment lines may occur in the input file.

Each non-comment line of the command file will specify one of the commands described below. Each line consists of a sequence of “tokens” which will be separated by single tab characters. **Bold** text indicates command keywords that will be used verbatim. A newline character will immediately follow the final “token” on each line. The first command will always be:

```
create <location> [ east | north | west | south ]
```

This causes the creation of a new tank, at the specified location facing in the specified direction. New tanks always start with 10 units of ammunition and an odometer reading of zero centimeters.

There will only be one `create` command in a script file.

```
move <distance>
```

This causes the tank to move forward (in the direction it is currently pointing) for the specified distance. The distance will be specified in centimeters, and will be nonnegative. Moving will cause the tank's odometer to be updated.

```
turn [ left | right ] [ 0 | 90 | 180 | 270 ]
```

This causes the tank to rotate the specified angle in the specified direction. The angle is specified in degrees. Turning does not change the location of the tank, nor its odometer reading.

fire

This causes the tank to fire once.

status

This causes your program to write the current status of the tank to the log file. This will include the current location, direction, odometer reading, and units of ammunition left, each appropriately labeled.

exit

This causes your program to terminate. The script file is guaranteed to contain an `exit` command.

Legend: in the commands above:

`<location>` a tab-separated pair of integers, representing a point in the xy-plane

Each command must be echoed to the log file, along with an informative message indicating the results when the command was processed. The output from each command should be delimited in some manner, similar to the parsing homework assignment.

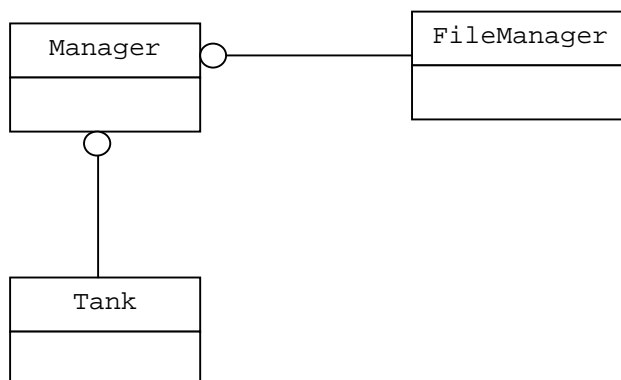
You may assume that the input file will conform to the given syntax, so syntactic error checking is not required. If an error occurs during the parsing of the command file, there's an error in your code. However, your program should still attempt to recover, by "flushing" the current command and proceeding to the next input line. Here is a sample input file:

```
; Sample input script for RoboTank.
;
create      5      9      east
move 100
status
turn left  180
status
fire
move 100
turn right 90
fire
turn right 90
fire
turn right 90
fire
status
move 50
status
turn left  180
move 30
status
fire
fire
fire
fire
turn right 90
fire
fire
fire
move 20
status
exit
```

There will typically be more comment lines than are shown here. Check the website for additional samples.

Design:

Since this is the first project, and much remains to be said about design, we are providing a partial high-level design in the project specification. You are expected to conform to this design. We identify three major classes: Tank, Manager and FileManager. These classes should be related as shown in the following class diagram:



This indicates that there will be a Manager object, which will use a FileManager object and a Tank object. The term "object" is used somewhat loosely here. You must implement the Manager and the Tank as classes. For this project, it is acceptable that the FileManager be purely procedural rather than an instance of a C++ class. There will, of course be additional, purely procedural code, e.g., `main()`.

The determination of attributes and responsibilities for each class is left to you. That does not mean that all decisions are equally good, nor that all decisions will receive full credit. Your goals should be to place responsibilities where they most logically belong, and to provide for maximum cohesion and reusability. We may choose to address the specific implications of these goals for this project in class.

Some explicit requirements are that only the FileManager object should access the script file, and that the FileManager should not be involved in interpreting commands. That is, the FileManager "knows" that it should ignore comment lines, and perhaps that each command line consists of a tab-delimited sequence of tokens, but the FileManager should not "know" what strings are valid commands for this project, much less how to interpret them. Similarly, the Tank object should not involve itself in interpreting input lines.

Log File Description:

Since this assignment will be graded by TAs, rather than the Curator, the format of the output is left up to you. Of course, your output should be clear, concise, well labeled, and correct. The first few lines should contain your name, course (CS 2704), and project title.

The remainder of the log file output should come directly from your processing of the script file. You are required to echo each command that you process to the log file so that it's easy to determine which command each section of your output corresponds to. You are also required to delimit the output from each command. Here is a sample log file that corresponds to the input given earlier:

Bill McQuain
CS 2704 Project 1: RoboTank

Command: create 5 9 east
Tank created.

Command: move 100
Tank moved 100 cm.

Command: status
Location: 105 9
Direction: east
Odometer: 100
Ammo: 10

Command: turn left 180
Tank turned 180 degrees to the left

Command: status
Location: 105 9
Direction: west
Odometer: 100
Ammo: 10

Command: fire
Tank fired.

Command: move 100
Tank moved 100 cm.

Command: turn right 90
Tank turned 90 degrees to the right

Command: fire
Tank fired.

Command: turn right 90
Tank turned 90 degrees to the right

Command: fire
Tank fired.

Command: turn right 90
Tank turned 90 degrees to the right

Command: fire
Tank fired.

Command: status
Location: 5 9
Direction: south
Odometer: 200
Ammo: 6

Command: move 50
Tank moved 50 cm.

Command: status
Location: 5 -41
Direction: south
Odometer: 250
Ammo: 6

```
Command: turn    left 180
         Tank turned 180 degrees to the left
-----
```

```
Command: move    30
         Tank moved 30 cm.
-----
```

```
Command: status
Location:      5  -11
Direction: north
Odometer:     280
Ammo:         6
-----
```

```
Command: fire
         Tank fired.
-----
```

```
Command: fire
         Tank fired.
-----
```

```
Command: fire
         Tank fired.
-----
```

```
Command: fire
         Tank fired.
-----
```

```
Command: turn    right 90
         Tank turned 90 degrees to the right
-----
```

```
Command: fire
         Tank fired.
-----
```

```
Command: fire
         Tank fired.
-----
```

```
Command: fire
         Tank failed to fire.
-----
```

```
Command: move    20
         Tank moved 20 cm.
-----
```

```
Command: status
Location:     25  -11
Direction: east
Odometer:    300
Ammo:        0
-----
```

```
Command: exit
         Exiting...
-----
```

Submitting Your Program:

You will submit a zipped file containing your project to the Curator System (read the *Student Guide*), and it will be archived until you demo it for one of the GTAs. Instructions for submitting are contained in the *Student Guide*. You will find a list of the required contents for the zipped file on the course website. Follow the instructions there carefully; it is very common for students to suffer a loss of points (often major) because they failed to include the specified items.

Be very careful to include all the necessary source code files. It is amazingly common for students to omit required header or cpp files. In such a case, it is obviously impossible to perform a test of the submitted program unless the student is allowed to supply the missing files. When that happens, to be fair to other students, we must assess the late penalty that would apply at the time of the demo.

You will be allowed up to five submissions for this assignment, in case you need to correct mistakes. Test your program thoroughly before submitting it. If you discover an error you may fix it and make another submission. Your last submission will be graded, so fixing an error after the due date will result in a late penalty.

The submission client can be found at: <http://spasm.cs.vt.edu:8080/curator/>

Programming Standards:

The GTAs will be carefully evaluating your source code on this assignment for programming style, so you should observe good practice. See the Programming Standards page on the course website for specific requirements that should be observed in this course.

Evaluation:

You will schedule a demo with your assigned GTA. The procedure for scheduling your demo will be announced later. At the demo, you will download your submitted project, perform a build, and run your program on the supplied test data. The GTA will evaluate the correctness of your results. In addition, the GTA will evaluate your project for good internal documentation and software engineering practice.

Here is a **rough estimate** of how much weight will be given to each of the factors that the GTA will consider:

Factor	Est, Weight
create command	10%
move command	10%
turn command	10%
fire command	5%
Tank class design <ul style="list-style-type: none"> ▪ good use of public/private access ▪ good identification of attributes ▪ good identification of responsibilities 	20%
Manager class design <ul style="list-style-type: none"> ▪ good use of public/private access ▪ good identification of attributes ▪ good identification of responsibilities 	20%
Conformance to specified design	10%
Quality of internal documentation	15%

This is only a preliminary estimate. It is subject to change. Note that we must use the `status` command to verify most of your results. Therefore, we have not assigned any points to the correct implementation of that command. If it does not work, you will lose many points for correctness of operation.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your main source code file.

```
// On my honor:  
//  
// - I have not discussed the C++ language code in my program with  
// anyone other than my instructor or the teaching assistants  
// assigned to this course.  
//  
// - I have not used C++ language code obtained from another student,  
// or any other unauthorized source, either modified or unmodified.  
//  
// - If any C++ language code or documentation used in my program  
// was obtained from another source, such as a text book or course  
// notes, that has been clearly noted with a proper citation in  
// the comments of my program.  
//  
// - I have not designed this program in such a way as to defeat or  
// interfere with the normal operation of the Curator System.  
//  
// <Student Name>
```

Failure to include this pledge in a submission is a violation of the Honor Code.