Program 2: Moon Landing Game

Due date: Monday, November 3, 11:59pm.

Purpose

To write a program requiring use of control structures (loops and conditional statements), arithmetic, relational and boolean expressions.

Assignment

This is an **individual** assignment. All non-collaboration, non-plagiarism policies of the course are in full effect and will be enforced.

Project Description

In this project you will implement a version of one of the oldest known computer games: Moon Landing.

The project will consist of two parts: **General** and **Challenge**. Everyone is required to implement the **General** part of the project. Implementation of the **Challenge** part should be done **only** after the **Regular** part has been successfully implemented.

Your project will be graded based on **Regular** part only.

For the **Regular** part of the project, the premise of the game goes as follows. You start the game being 500 feet away from the surface. Your initial velocity is 50 feet/sec towards the Moon surface. You have 120 units of fuel which you can burn freely in any quantities at any time. Once you run out of fuel your ship enters a free-fall state.

Your objective in the game is to land the spaceship on the moon successfully by controlling the fuel injection on a second-by-second basis. If the ship's velocity at the moment it lands is less than or equal to 10 feet/sec, the landing is successful and your score is computed. Otherwise, your ship crashes and the game is lost.

Regular Moon Landing Game

You are to write a C program implementing the Moon Landing game based on the following specifications.

Initial status. Each game starts in the following state.

- Initial distance to the moon: **500 feet**
- Initial amount of fuel on board: **120 units**
- Initial velocity of the ship: **50 feet/sec**
- Time intervals between fue injections: **1 second**
- Free fall acceleration : 5 feet/sec^2

Physics and Mathematics

In the formulas below: f - fuel burned, v - new velocity, v_0 - previous velocity, a - acceleration, d - distance travelled,

t - time.

$$a = 5 - f$$

$$v = v_0 + at$$

$$d = v_0 t + \frac{at^2}{2}$$

Since we will be computing new velocities and distances covered each second, the last two formulas become simpler:

$$v = v_0 + a$$

$$d = vt + \frac{a}{2}$$

These formulas will allow you to compute velocity and distance travelled during the 1-second intervals.

You will also be required to compute the impact velocity of the free-fall. For this you will have to solve the quadratic equation:

$$\frac{at^2}{2} + v_0 t - d = 0$$

with $a = 5$ (free-fall acceleration) and v_0 and d known. You solve the equation for t. You take the larger of the two solutions of the quadratic equation.

For an equation

$$ax^2 + bx + c = 0$$

the solutions are found as

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Program specifications

Meta-comments. You are writing a **text-based** game — one of the first computer games developed. The "user interface" of the game is a decisive throwback to the days of yore, when all interaction happened on a 25×80 text screen. As, such, think of the text-based menu that you have to implement for this program as an exercise in a programming style, which, while not absolutely necessary anymore, may come handy for some of the programs you write in the future.

Non-Functional Requirements.

NM-1. **Program Style**. Program style will be strictly enforced. Make certain your submission satisfies the programming style of the course.

NM-2. **Output compliance.** For the regular moon landing game, the output of the program **shall exactly co-incide** with the output of instructor's binary, except where noted below:

- The fourth line of the program output shall contain your name. Instructor's binary prints `"Alex Dekhtyar"`.

Functional Requirements. Your program shall perform the following actions:

FM-1. Upon its start, the program shall print information about its purpose (moon landing game) and author:

```
********* Moon Landing Program v. 1.0 ******************


Written by:   Alexander Dekhtyar
```

FM-2. After that the main loop of the program starts. At the beginning of each loop, the program shall display a *text menu* with three choices and prompt the user to make a selection. The menu choices are `''1.  Play Game''`, `''2.  Instructions''`, "3. Quit":

```
Please select an action:

1. Play Game
2. Instructions
3. Quit

Your choice :>>
```

(Note, there is an empty line between the `"Written by:"` line of the output and the `"Please select an action:"` line. Note also that `:>>` is the prompt used in the program.)

FM-3. Your program shall accept user input, indicating the desired action. Acceptable user inputs are integers 1,2,3. If *any other integer number* is enered, the program shall simply output the main menu and the prompt again:

```
********* Moon Landing Program v. 1.0 *****************


Written by:   Alexander Dekhtyar

Please select an action:

1. Play Game
2. Instructions
3. Quit

Your choice :>>8
Please select an action:

1. Play Game
2. Instructions
3. Quit

Your choice :>>
```

Your program is not responsible for handling any other type of input (e.g., if a character or a string has been entered instead of an integer).

FM-4. If the user chooses ``Quit'', i.e., enters 3 the program shall print a good-bye message and exit.

```
Please select an action:

1. Play Game
2. Instructions
3. Quit

Your choice :>>3
Bye ! Thanks for playing Moon Landing
```

FM-5. If the user chooses ``Instructions'', i.e., enters 2 the instructions on how to play the game shall be printed After that the program shall loop back to the menu. The text of the instructions is as follows:

```
Please select an action:

1. Play Game
2. Instructions
3. Quit

Your choice :>>2
* * * * * *  * * The Game * * * * * * * * * * * * * * * * *
 You are to land a spaceship on the moon.
 You control the speed of your spaceship by telling it
 how much fuel has to be burned every second.
 5 units of fuel cancel the Moon gravity, more - will deccelerate your ship.
 Do not run out of fuel.
 To land successfully your velocity should not exceed 10 feet/sec
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Please select an action:

1. Play Game
2. Instructions
3. Quit

Your choice :>>
```

FM-6. If the user chooses ``Play Game'', i.e., enters 1, the game shall commence. See requirements below for the in-game specifications. After the game is over, the program shall loop back to the menu.

FM-7. **In the game.** The game consists of three parts: initialization, main game loop, and the assessement of the results.

FM-7-1. During the game initialization stage, the program shall set the initial values for the distance to the Moon, amount of fuel left and ship velocity. These parameters were specified above.

FM-7-2. The flow of the game is implemented in the main game loop. In this loop, the following shall be done. See sample output for the exact output format.

FM-7-2-1. At the beginning of each game loop iteration, your program shall display current status of the game. Current status includes

i. time elapsed since the beginning fo the landing (in seconds);
ii. distance left to the Moon (in feet);
iii. amount of fuel left (units);
iv. current velocity (feet/sec).

FM-7-2-2. After the status is displayed, the program shall prompt the user to enter amount of fuel to be burned (injected) over the next second. The program shall read the amount of fuel (an `int` value).

The amount of fuel entered by the user shall be tested: it shall not be negative and it cannot exceed amount of fuel left. The prompt for the amount of fuel to burn shall be repeated and a new value read until a valid value had been entered.

FM-7-2-3. Once a valid amount of fuel is read the program compute the new velocity of the ship and the new distance to the moon. Use the formulas above to compute this.

FM-7-2-4. Once the new distance to the Moon and the new velocity are computed the program shall check whether the **game can be finished at this step**. The game can be finished in one of two cases:

i. **Moon Landing/Crash.** The ship has reached the Moon surface, i.e., the new distance to the Moon is 0 or less. In this case the program shall exit the game loop and proceed with the assessement of results.
ii. **Free Fall.** The ship has ran out of fuel. In this case the user can no longer control the ship. Using the knowledge about the current distance to the Moon and the current velocity of the ship, the program shall compute how much time it will take for the ship to reach the surface in free-fall. Based on this time the program shall compute the impact velocity of the ship (see **Physics and Mathematics** section). After that the program shall exit the loop and proceed with the assessement of the results.

If neither condition holds, the program proceeds with the next round of the loop. (See requirements FM-7-2-1 — FM-7-2-4.)

FM-7-3. **Assessment stage.** During the result assesement part, the program shall consider the impact velocity of the ship to determine the outcome of the game. If the user has won the game, the program also shall compute the number of points awarded to the user.

FM-7-3-1. The program shall output the information about the time it took the ship to land on the Moon and the impact velocity of the ship.

In, addition, *if the game ended because the ship went into a free-fall mode*, the program shall output that information as well. The output formats are as below:

Regular landing.

```
You have reached the Moon after 8 units of time
Your impact velocity was : 1.00
```

Free Fall landing.

```
You ran out of fuel after this step and went into free fall
You have reached the Moon after 32 units of time
Your impact velocity was : 95.00
```

FM-7-3-2. The program shall determine the outcome of the game based on the impact velocity at the moment of the impact.

> *A landing is **successful** if **the impact velocity is less than or equal to 10 feet/sec**. Any velocity over that means that the ship crashed and the game had been lost.*

There are 4 possible landings, described in the following table:

Impact Velocity	Description	Outcome
$[0-5]$ feet/sec	**Wonderful landing**	*Game won*
$(5-10]$ feet/sec	**Badly executed landing**	*Game won*
$(10-20]$ feet/sec	**"almost made it" landing**	*Game lost*
$[20-\ldots]$ feet/sec	**Big Crash landing**	*Game lost*

Your program shall be able to recoginze which of the four landings occured and shall inform the user about this. The appropriate messages for each landing are shown below.

Big Crash.

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*- WOW ! It sure does look that spaceship pilot  -*
*- was not the best occupation for you !         -*
*- Earth has one bad pilot less and Moon has one -*
*- medium-size crater more                       -*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Almost made it.

```
 *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*- ....a few more inches... a few more drops of fuel.... -*
*- ... and you would have made it...                     -*
*- but... your ship is in ruins and your dead body had   -*
*- been found near it by the next, more successful crew  -*
*- to make it here... SORRY. YOU LOSE. But you will be   -*
*- remembered as a national hero.                        -*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Badly executed landing.

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*- HMMMMMM..... Well, well, well...                          -*
*- An old rusty axe would have probably flown better         -*
*- Not a flight to be proud of, eh ?                         -*
*- However.... You got off ALIVE and the ship had been landed-*
*- with only some minor damage to the body. A few broken ribs-*
*- of the pilot will heal soon. So... Congratulations !      -*
*- You are a survivor !                                      -*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Wonderful landing.

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*- CONGRATULATIONS: you have proved yourself to be         -*
*- among the best pilots in the world. Your landing will   -*
*- go into the annals of historyand thousands of people    -*
*- will try to follow your footsteps                       -*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

FM-7-3-3. If the user has won the game, the program shall compute the user's score. The score is computed as follows: $\mathsf{score} = \frac{f*10}{l}$ where f is the amount of fuel left and l is equal to 1 if the landing was "wonderful" and 2 if the landing was "badly executed" (see the table above for landing conditions). The score is then printed out.

Challenge part

The challenge part of the assignment is **EXTRA CREDIT**. You can earn up to 50% of the total score (i.e., up to 2.5% towards the final grade).

For the Challenge part, you shall modify your Moon Landing program to include a richer set of functionality surrounding the game. **Notice,** that you **must** submit the regular Moon Landing game as a **separate program**, if you are submitting the Challenge part.

The following counts towards the completion of the **Challenge** part of the project.

- Modify your main menu to include two more choices another choice: "*Setup*" and "*Reset*". If "Setup" is chosen, ask the user to enter the new initial values for the
  - initial distance to the moon
  - initial amount of fuel
  - initial velocity of the ship (this number can be both positive and negative)
  - free-fall acceleration

  After the setup is performed, return to main menu. When user chooses "Play Game", start playing the game with the new initial values. Keep these values until either a new setup is performed or "Reset" is chosen. "Reset" should reset the initial values to their defaults. "Setup" should be selection number 4 and "Reset" should be selection number 5.

- Keep track of all the games played by the user. Upon exit from the program (after user chose "Quit"), print out the following information:
  - total number of games played
  - games won
  - games lost
  - number of games ended in each of the four results
  - largest number of points gained in a winning game
  - smallest number of points gained in a winning game

If you have implemented the "Setup" and "Reset" selection, implement selection "*Stats*" (selection 6) which will output current statistics for the player without finishing the program.

- To make the game even more interesting distinguish between more than 4 different outcomes. For each outcome you should provide a special comment (preferebly funny), and if you have implemented the statistics, you should keep statistics for each outcome as well.
- After each step, provide a little bit of help to the user by computing the impact velocity, should the ship enter free fall at current moment. Output the time to impact and the impact velocity as a prediction before you query the user for the fuel.

Compiling your programs

In order to compute the solution for a quadratic equation you will need to use a `square root` function `sqrt()`. This function is part of the Standard C Library and it resides in `math.h`. When compiling your program, please make sure, `-lm` option is included (as we generally do).

Submission

What to submit. For regular game submission, you need to submit just the program file. For challenge submission, you need to submit two files: the program file and a README file describing the exact nature of modifications you have implemented.

Name your regular Moon Landing program `moon.c`.

Name your challenge Moon Landing program `landing.c`.

Name the README file for the challenge assignment `README.landing`.

Handin. You will submit to `dekhtyar`. As usual, ssh to vogon. The `handin` commands are:

```
> handin dekhtyar program02-09 <files>
```

and

```
> handin dekhtyar program02-11 <files>
```

for Sections 09 and 11 of the course respectively.