Program 1: Slalom text game...

Due date: Friday, November 13, 11:59pm.

1 Purpose

To write a program requiring use of loops, conditionals and arrays.

Programming environment. This is a solo programming project. You are responsible for all the work related to the program development, testing and submission.

Collaboration. **Any** collaboration between peers, as well as any collaboration with outside sources is **strictly prohibited**. If you have any questions, concerning the assignment, please consult the instructor.

2 Program Description

You will implement an old text-based game called slalom. In the game, the user controls an alpine skier whose goal is to get through a slalom course as fast as possible. There is a number of gates to pass through. Each gate comes with a restriction of velocity: if a skier approaches the gate at a higher velocity, (s)he misses the gate and ends the slalom run in infamy. Upon passing each gate, the user receives information on his/her current velocity and the velocity restriction on the next gate. The user then can control the skier's acceleration rate to ensure that the skier passes the next gate successfully. The program keeps track of the time it took the skier to complete the course.

Program Requirements.

SR0. Physics. Recall from high school that motion equations involve five parameters:

Parameter name	Notation
Starting velocity	u
Ending velocity	v
Acceleration	a
Displacement/distance travelled	d
Time in travel	t

Equations of motion, connect these five quantities in the following manner:

$$d = ut + \frac{at^2}{2}$$
$$v = u + at$$

Given any three quantities, we can compute the remaining two. The base equations above compute d and v from u, t and a. Inverting the first equation, we can get the following :

$$t = \frac{-u + \sqrt{u^2 + 2ad}}{a} \text{ [1]}$$

$$a = \frac{2(d - ut)}{t^2}$$

Invertng the second equation we can get:

$$a = \frac{v - u}{t}$$
$$t = \frac{v - u}{a}$$

If $a = 0$, equations of motion simplify to

$$v = u$$
$$d = vt$$

SR1. The program will start by prompting the user to choose one of three actions: play the game, see the instructions or quit. If the user chooses to play the game, the slalom game proper starts. If the user chooses to see the instructions, the instructions are displayed. Finally, if the user chooses to quit the program, the program terminates.

After instructions have been displayed to the user or after a slalom game has been completed, the same menu of three choices (play, see instructions, quit) shall be offered again.

SR2. Slalom game rules. The text-based slalom game works as follows. The user controls an alpine skier who needs to ski through a slalom course. The slalom course **consists of ten (10) gates** Each pair of gates is **spaced 50 meters apart**. Gate 10 is the finish line of the slalom course.

The starting point is 50 meters distant from Gate 1. The skier starts with velocity 0.

[1] We get this by solving the equation $\frac{at^2}{2} + ut - d = 0$ for t and using the positive root, as elapsed time is non-negative.

Each gate has the **maximum velocity restriction**. If the skier arrives to the gate with higher velocity, the skier crashes and fails the slalom course. The velocity restrictions at the gates are specified in the table below:

Gate	Max Velocity
Gate 1:	10 m/s
Gate 2:	18 m/s
Gate 3:	25 m/s
Gate 4:	19 m/s
Gate 5:	24 m/s
Gate 6:	17 m/s
Gate 7:	20 m/s
Gate 8:	25 m/s
Gate 9:	27 m/s
Gate 10:	22 m/s

The user will control the skier at the start of the game and upon successful passage of a gate the user. The control is done by specifying the acceleration of the skier on the next stage of the course (each stage is the part of the course between two consecutive gates). The acceleration can be modified using one of the following seven commands:

Action	**Command**	New Acceleration
Fast acceleration	7	$4m/s^2$
Moderate acceleration	6	$2m/s^2$
Slow acceleration	5	$1m/s^2$
Keep current velocity	4	$0m/s^2$
Slow deceleration	3	$-1m/s^2$
Moderate deceleration	2	$-2m/s^2$
Fast deceleration	1	$-4m/s^2$

To enter the command, the user uses enters one of the seven command codes: "1" — "7" and presses the `<Enter>` key.

The goal of the user is to ensure that the skier successfully completes the slalom course and does it in the fastest way possible.

SR3. Main menu. Upon start, your program shall display the following text:

```
-------------------------------

      Slalom Race, Version 1.0

1. Play
2. Instructions
3. Quit

Your choice:
```

In the rest of the document we refer to this text as the *main menu*. The program will then wait for user input.

SR4. If user inputs any number other than `1, 2` or `3`, the program shall repeat the entire greetings/main menu text shown above, and wait again for user input.

SR5. Quit. If the user enters `"3"`, the program shall skip a line and output

```
Welcome to the Slalom Race!

Your task is to complete the slalom race.
The track has 10 gates. Gates are 50 meters apart.
You start at the top with the speed of 0 m/s.
After start and after each gate you can correct your run
by entering an acceleration/deceleration command,
The following commands can be given:

7 : fast acceleration
6 : moderate acceleration
5 : small acceleration
4 : no change in speed
3 : small deceleration
2 : moderate acceleration
1 : fast deceleration


Each gate has maximum velocity on which it can be passed.
If you exceed the max velocity of the gate, you lose control and lose the game.
Max gate velocities:

Gate 1: 10.00 m/s
Gate 2: 18.00 m/s
Gate 3: 25.00 m/s
Gate 4: 19.00 m/s
Gate 5: 24.00 m/s
Gate 6: 17.00 m/s
Gate 7: 20.00 m/s
Gate 8: 25.00 m/s
Gate 9: 27.00 m/s
Gate 10: 22.00 m/s

Enjoy the game!
```

Figure 1: Slalom game: the instructions

```
Good bye!
```

After that, the program shall terminate.

SR6. Instructions. If the user enters "2" the program shall output the game playing instructions. The text of instructions is shown in Figure 1. After the instructions are printed, the program shall output the main menu and wait for user input.

SR7. Game. If the user selects "1", you program shall start the gameplay. The program shall print be message specifying that the skier is at the start, that the current velocity is 0 and no time has elapsed. Current velocity (0.00 m/s), next gate number and the velocity restriction at the next gate shall also be specified. After this text, the program shall prompt the user for action. The message shall look as follows:

```
You are at the start.
```

```
Time elapsed: 0.000000 sec
Current velocity: 0.000000 m/sec
Next Gate: 1.   Velocity limit: 10.00 m/sec
Your action (1 - 7):
```

The program then, shall read user input. If user inputs a number other than 1—7, the program shall repeat the

```
Your action (1 - 7):
```

line and wait for input again.

SR8. One step of the game. Each time the program accepts correct user input command (1 – 7), the following actions shall take place:

1. Based on the command entered the program shall set the acceleration of the current leg (stage) of the course according to the acceleration table from **SR2.**
2. With the established acceleration, the program shall compute the velocity of the skier at the next gate, and the time it takes the skier to get to the gate.
3. The program then shall update the current time the skier spent on the course.
4. The program shall establish whether skier successfully reaches the gate (i.e., whether the skier reaches the gate with non-negative velocity).
5. The program shall establish whether the skeir's velocity at the next gate exceeds the gate's velocity restrictions.

SR9. Successful passage of the gate. If the skier reaches the next gate with **non-negative velocity** which **does not exceed** the gate's velocity restrictions, the skier is considered to **have successfully passed the gate**. In this case, the program shall output the status message similar to the status message it outputs at the start, specifying the current location of the skier on the track, skier's current velocity, next gate number and the velocity restriction at the next gate, and prompt the user for action.

The message shall look as follows:

```
----------------------------------
You are at gate <GATE>
Time elapsed: <TIME> sec
Current velocity: <VELOCITY> m/sec
Next Gate: <NEXT_GATE>.   Velocity limit: <VLIMIT> m/sec
Your action (1 - 7):
```

For example, here is how passing through Gate 2 might look:

```
----------------------------------
You are at gate 2
Time elapsed: 15.000000 sec
Current velocity: 10.0000 m/sec
Next Gate: 3.   Velocity limit: 25.00 m/sec
Your action (1 - 7):
```

SR10. Exceeding the velocity limit. If the skier arrives to a gate with the velocity exceeding the gate's velocity limit, the skier is considered to *have failed the race*. The race stops, and your program shall output the following message:

```
Your velocity at gate <GATE> was <VELOCITY> m/sec
You have lost control and went into a nearby snow pile!
You failed the race and went face down into the snow!
Better luck next time!
```

For example, failure at Gate 4 can look as follows:

```
Your velocity at gate 4 was 26.4575 m/sec
You have lost control and went into a nearby snow pile!
You failed the race and went face down into the snow!
Better luck next time!
```

After this message, the program shall return to main menu.

SR11. Failing to reach the gate. Under certain starting conditions and acceleration, the skier will reach velocity $0m/s$, i.e., will stop, before reaching the next gate on the course. If this situation occurs during the race, the skier is considered to *have failed the race*. The race stops and your program shall output the following message:

```
You stopped before reaching the gate.
Your attempt to pick up the speed landed you in a snow pile.
You failed the race due to your overburdening caution.
Better luck next time.
```

After this message, the program return to main menu.

SR12. Completing the race. Gate 10 is the finishing line of the slalom course. Upon *successful passage through Gate 10* (see requirement **SR9.** the race ends. At this point, your program shall compute the total time it took the skier to race through the slalom course, and output the following text:

```
You are at the finishing line!
************* CONGRATULATIONS !! ************

You've done it!
Race time: <TIME> seconds


Can you beat it next time?
```

Here is an output for a run in which the skier accelerated on the first two stages, and kept the velocity afterwards:

```
You are at the finishing line!
************* CONGRATULATIONS !! ************
```

```
You've done it!
Race time: 42.426407 seconds


Can you beat it next time?
```

After this message, the program shall return to main menu.

Design notes.

Legacy. The overall organization of this program is remniscent of the many legacy text-based games, and other text-based programs that operated for years and years (and some – still operate) on text-based terminals. Such programs usually consisted of a main menu that allowed the user to select what (s)he wanted to do with the program by entering a number, and the code that combined interaction with the user with the computations necessary to perform the tasks/play the game.

While most of the programs today use different means of user interaction, and while current graphical user interfaces make the legacy text-based interactive programs obsolete, it is well worth understanding the design and internal organization of such programs.

Overall organization. Essentially, your program should consist of two loops. The outter loop is responsible for the main menu and reactions to the main menu choices. The reaction to the `"Quit"` choice is to leave the loop and terminate the program. The reaction to the `"Instructions"` choice is the print some text and start a new iteration of the outter loop. The inner loop of the program will appear in the code responsible for the game play. Each step of the game play loop covers one stage of the race.

Data. Your program shall use a one-dimensional array to store velocity restrictions at gates. At the beginning of the program, this array shall be initialized with the numbers provided to you in the velocity restrictions table in Requirement **SR2.**

Your program may also choose to use an array to store the accelerations for each user command, but this can be avoided.

3 Submission Instructions

Submission.

Program name. Name your program `slalom.c`.

Files to submit. You shall submit the `slalom.c` file.

No other files can be submitted. In fact, if you submit *other* file, or submit one of the three files about with an *incorrect filename*, you will receive an email informing you about a submission error, and asking you to resubmit.

Submission procedure. You will be using `handin` program to submit your work. The procedure is as follows:

- `ssh` to `vogon` (`vogon.csc.calpoly.edu`). Submit using the following command:

```
> handin dekhtyar program02 slalom.c
```

Late submission. You may submit late for a 24-hour period following the deadline. Late submissions are subject to the standard 10%—30% penalty at the instructor's discretion.