The 2025 Internship Software Challenge

This document is designed to be a test of your imagination. As such, you are free to solve the challenge in any way you see fit and using the programming language with which you are most comfortable.

The input data for the various stages of the challenge has been included in the package, and the output schema is loosely defined for each stage.

The data and the formulas for this challenge are loosely based on reality, but we will be working with a massively simplified model. There should be no need to dive into advanced mathematical concepts. Most, if not all, of the stages can be solved with simple equations.

There are seven stages to this challenge and you may solve how many of them you see fit.

Stage One – Perfect spheres in a vacuum

The input file for this part is called "**Planetary_Data.txt**". It contains a list of the currently known solar system planets (including Pluto). Each planet has a known diameter and a known mass (the mass is relative to Earth's mass). The values are, of course, approximations.

From now on, we'll assume that planets are perfect spheres and that there is no interaction between neighbouring planets' gravity wells. We'll also accept that a planet's gravity well (the „volume" of space where the planet's mass has any effect on its surroundings) ends very closely to the planet's radius. So, once a rocket reaches a planet's escape velocity, it's effectively free of that planet's „gravity". Yes, this challenge is about rockets.

You'll need the formula for calculating planetary escape velocity:

$$v = \sqrt{\frac{2GM}{r}}$$

The variables in the formula are the following:

- $G = 6.67 \times 10^{-11}\ m^3 kg^{-1} s^{-2}$ is Newton's gravitational constant;
- r is the planet's radius;
- M is the planet's mass.

All calculations should be done in SI units, so a conversion of the given planetary radii from kilometers to meters is required.

The first stage's requirements are simple: write software that reads the given file and parses it to display the planetary escape velocities for all of the given planets.

For example, plugging the Earth's data into the formula would look like this:

$$v = \sqrt{\frac{2 * (6.67 * 10^{-11}\, m^3 kg^{-1} s^{-2}) * (6 * 10^{24}\, kg)}{\frac{12800}{2} * 10^3\, m}}$$

Computing this yields an escape velocity of **11183 m/s**, which we can approximate to about **11.2 km/s**.

So, in other words, for our rocket to be able to leave Earth's influence and head off into space, it would have to be travelling at more than 11 kilometers per second.

Your application should compute this escape velocity for all of the planets in the input file. A simple console application is sufficient. You could also write the velocities to an output file, whatever suits you. The end goal is this: the user must tell, at a glance, what the escape velocity is for any of the planets.

Some remarks:

- The code that we write is intended for users, not just for developers. If you come up with an application, it should be as user-friendly as possible. The simple app that you write for this first stage should be able to accept a file path pointing to where the input data is.
- An extension of the previous bullet-point is that the application should also be hardened against unusable inputs. This means that poorly formed file paths or non-standard input data should not crash your application. Instead, the user should be notified about the problem.
- These remarks apply to all of the steps in this challenge.

Stage Two – Up, up and away!

Using the same input file as before ("**Planetary_Data.txt**") and a new file ("**Rocket_Data.txt**"), your application should compute how long it would take a hypothetical rocket to reach escape velocity for each of the planets. We'll ignore the rocket's mass, its friction and we'll assume that it has infinite fuel (but that the fuel has no mass). Its engines are also quite good and are capable of unlimited burn times. Furthermore, we'll consider a planet's gravitational influence to be linear, i.e. no matter how high above the planet's surface our rocket gets, it will still feel the same pull towards the planet's center of mass. This is true until the rocket reaches escape velocity, at which point, the planet effectively vanishes from behind the spaceship.

For our current rocket, each engine is capable of boosting the rocket faster by 10 meters per second each second. If we were to burn a single engine for ten seconds, we would be going 100 meters per second at the end of those ten seconds. Our rocket has four of those engines.

Computing how long it would take our rocket to reach Earth's escape velocity would look like this:

$$t = \frac{11183\, ms^{-1}}{4 * 10\, ms^{-2}}$$

This yields a time of about 279 seconds, or a bit more than four and a half minutes. This is considerably faster than mankind's previous attempts, but we'll also ignore the G forces inflicted on the crew during such a fast flight.

You'll also need to compute how far the rocket will travel before reaching escape velocity. The formula for distance travelled with acceleration is as follows:

$$d = v * t + \frac{a * t^2}{2}$$

The variables are:

- **d** is the distance that we're looking for;
- **v** is the starting velocity for our calculation (when launching a rocket, $\mathbf{v = 0}$);
- **t** is the time (an output from our previous calculation);
- **a** is the acceleration.

So, for Earth, our rocket will have travelled:

$$d = 0 * 279\ s + \frac{40\ ms^{-2} *\ 279s^2}{2}$$

or **1556820 meters**, or about 1560 kilometers. We'll assume that the rocket went straight up, so that would put it

$$d = \frac{12800 * 10^3\ m}{2} + 1560 * 10^3\ m$$

or **7956820 meters** away from the Earth's center. That's about 7960 kilometers away from the Earth's center. Keep in mind that planetary positions are given by their centers, but rockets tend to crash into a planet's surface.

Knowing this, **the goal of the second stage is for your program or application to return the time it would take a rocket to reach each of the given planets' escape velocities**. The output should also **include the distance our rocket will have travelled from the planet's surface** before reaching the escape velocity.

Like before, the only constraint on how you feed this output back to the user is user-friendliness.

Stage Three – The fun begins

For this stage, let's consider a simplified solar system, as described in "**Solar_System_Data.txt**".

The goal is to compute travel parameters for simplified, straight-line rocket journeys between any two planets.

First off, our assumptions and constants:

- The Sun has no mass and no gravitational effect on our rocket. It also has no radius, it's merely an anchor point for the planets' orbits;
- Whenever computing travel distances and times for a pair of planets, all the other planets also have no masses and radii. The solar system is then reduced to a pair of planets orbiting a mysterious center;
- Straight-line travel between planets is possible and the planets are positioned in such a way that the distance between then is minimized. For example, if the user were to choose a journey from Earth to Mars, we'll assume that the Sun, Earth and Mars are all aligned, so that Mars and the Earth are as close as they'll ever be;
- The rocket has enough fuel to complete a one-way trip, but the fuel is not infinite anymore. In order to save fuel, the rocket's maximum velocity will be limited to the highest escape velocity of the two planets;
- **1 AU** means one astronomical unit and is equal to **149597870.7 kilometers**;

- Rockets stop by doing what they did to get going, but in reverse. When our rocket is nearing its destination planet, it will maneuver so that its engines are pointing forward relative to its motion and burn those engines until it decelerates to zero velocity at the planet's surface.

An example calculation for this stage, travelling from Mars to Earth, is described below.

We need the distance between the two planets. This, when they're closest, would be **0.52 AU**, or **77790892.76 kilometers**. This is the distance between the two planets' centers.

Mars' escape velocity is about 5 kilometers per second, the Earth's escape velocity is about 11.2 kilometers per second, so our cruising velocity will be **11.2 kilometers per second**.

Knowing the distance between the two planets and the cruising velocity, we can now compute how long the entire journey will take.

We know it takes about 279 seconds for our rocket to get up to speed and that it will travel 1560 kilometers before it reaches that velocity. That means that it will be 1560 kilometers above Mars when it starts its cruise. It also means that it must start its deceleration burn 1560 kilometers above Earth, so the total cruising distance will be

$$d = 77790892.76\ km - 1560\ km - 1560\ km - 6400\ km - 2900\ km$$

or **77778482.76 kilometers**. Knowing our cruising velocity, this distance will be covered in about **6944507 seconds**, or about 80 days. So the total travel time would be an 80 day cruise plus the two burn times.

The output for this stage **must include**:

- How long it will take the rocket to reach its cruising velocity;
- How far away from the starting planet's surface it will be when it reaches that velocity;
- How long it will cruise at the nominal velocity;
- How far away from the destination planet's surface it has to be when it starts its deceleration burn. Remember, planets' positions are given by their centers;
- How long it will take the rocket to decelerate to zero when it reaches the destination planet's surface.
- The total travel time in seconds. To spare the user's eyesight, this travel time should also be displayed in days, hours, minutes and seconds. For the outer planets, travel times in seconds will be hard to comprehend.

For this stage, **the user should be able to input the starting planet and the destination planet. Your application should then display or output the six relevant pieces of information**, as described above.

Stage Four – The planets are not stationary

The "**Solar_System_Data.txt**" file also contains information about the planets' orbital periods. For each planet, its period is the time (given in days) it needs to complete one full revolution around the Sun.

We'll assume that these orbits are perfectly circular and that they are in the same plane. We'll also assume that all the planets are orbiting in a clockwise manner.

The goal for this stage is to simulate the motion of these planets around the Sun. Like in the previous stage, the Sun is merely an anchor point for the orbits.

At the start of the simulation, t_0, all of the planets are aligned and this is considered to be the "zero degrees of rotation" position. As the days pass, each of the planets rotates around the sun with an angular velocity given by its orbital period.

Your application **should accept a random time input from the user, given in days, and return or display the angular positions for each of the planets on that chosen day**. For example, if the user inputs a value of 200 days, one of the output angular positions, the Earth's, should be 197.26 degrees. When the user inputs a value of 365, the Earth's position is reset to zero degrees, but the other planets will no longer be aligned with it.

Stage Five – Rocket science is not easy

We now have means of computing simplified journeys between planets and a working solar system simulator.

We'll consider a starting time of $t_0 + 100$ years since the beginning of our solar system simulation. That means that one hundred years have passed since the planets were all perfectly aligned and they're now arranged in various positions around the Sun. We'll be searching for optimal transfer windows, or periods when journeys between planets are as easy and as quick as possible.

For this stage, the user will again choose two planets, a starting planet and a destination planet. Your application must then return the same information that it did for stage three and the start of the first optimal transfer window.

We consider a transfer window to be optimal if these two conditions are met:

- The start and destination planets are as close together as possible. This does not mean that we can wait forever for the planets to come back into perfect alignment.
- Since we're using straight-line journeys between two planets, we must take care to avoid transfer windows when our flight would crash us into any planets along the way. If Earth and Saturn were aligned and we'd like to travel from one to the other, we'd better make sure that we don't go through Mars or Jupiter.

Constraints and assumptions:

- We know the diameters of all the planets in the system; therefore, we know if a plotted straight-line journey will crash our rocket into a planet along the way.
- We'll accept that **the solar system freezes once a journey is underway**. When we have our optimal transfer window and start our first burn, the planets will not move until we reach our destination.
- **The maximum wait time is 10 years**. If a suitable transfer window cannot be found for a given pair of planets within that timeframe, we must let the user know and either skip plotting the journey or plot a sub-optimal one.
- Most of the assumptions that were put in place for stage three also apply here, except **that the solar system will no longer be reduced to the start-destination planet pair and, of course, the planets are no longer in the optimal journey positions**. Planets which are not part of the start-destination pair will not exert any gravitational influence on our rocket.
- The output for this stage will **include all the information** that would have been displayed or saved **for stage three (journey parameters), for stage four(positions for all the planets) and the start of the optimal transfer window, given as 100 years plus how many years or days must pass before the transfer window comes around.**

Stage Six – Rocket science is really hard

The goals, constraints and assumptions for this stage are almost identical to those in stage five, except for one: **the solar system will no longer freeze once a rocket journey is underway**. This means that, once the user has chosen the starting planet and the destination planet, the optimal transfer window must be computed knowing that **planets may travel into the rocket's path while the rocket is on its way to its destination**.

Stage Seven – A user interface

The goal is to implement a graphical user interface that displays your application's simulation output. The design is up to you. Good luck!