

In this project, I will be creating an executable that will be able to compute and calculate the exact amount of a given planet's surface gravity using its known mass and radius. The regular edition of this project will be created using C/C++ in the Visual Studio 2022 IDE owned by Microsoft Corporation.

This project also has a ready-to-use browser-supported edition written in JavaScript. The entire script is an object-oriented class that can be initiated and implemented easily even by beginner programmers and coders. Every single code in this project is/will be written by me, the author and the only developer of the project.

Document Contents

1. The primary details about the project
2. The backend details: programming languages, custom editions, and the rest
 - 2.1. *Why use C/C++ instead of easier-to-implement programming languages?*
 - 2.2. *The browser-supported edition: why we chose JavaScript over TypeScript?*
 - 2.3. *Which IDE (Integrated Development Environment) have we preferred?*
3. The development: programming, debugging, and the browser implementation
 - 3.1. *The Regular Edition*
 - 3.2. *The Browser Edition*
4. The outroduction: the contributors of this project, and a few more details
 - 4.1. *Download links, source code, and other project files*
 - 4.2. *More information about the developer of this project*

1. The primary details about the project

This project is basically a simple computer-based tool that can be used to calculate a given planet's surface gravity using its known mass and radius values.

It simply converts a couple of mathematical equations into executable software code which is almost how this entire thing works. The process is pretty much the same on the regular edition and the web-based edition. The only major difference is the programming language.

This project is in fact a school project assigned to me by my local physics teacher, the only catch is the fact that he didn't really ask for something a little crazy like this.

Please see the next page.

2. The backend details: programming languages, custom editions, and the rest

2.1. Why use C/C++ instead of easier-to-implement programming languages?

The reason I decided to go with C/C++ on the regular version is simple. Our scripts are based on the C++ programming language which is one of the fastest and the most powerful programming languages ever created so far. I also believe that C++ is the language that will take us all the way up to Mars. Note this down for the next decade.

We could have used Python, but we didn't. Because there's a huge difference between C++ and Python. For projects like this one, C++ is a much better choice.

The major advantage of C++ is performance. It performs efficiently and the speed is faster when compared to Python. And, it's suitable for almost every platform including embedded systems and even modern spaceship systems whereas Python can be used only on certain platforms that support high-level languages. I'm not saying that modern spaceships don't support Python though. That'd be funny.

Python is a great programming language. But, for this project, C++ is just better.

It's likely that I will keep on developing this project. For now, it only has one function and C++ is not really necessary for a super simple computer program like this. But, as I will keep on developing this project, C++ actually was a great choice.

2.2. The browser-supported edition: why we chose JavaScript over TypeScript?

When ran, TypeScript compiles into JavaScript, which is why there's no need to rule out TypeScript only because it's not supported on some browsers or something. Because TypeScript is converted into JavaScript when ran, which is why it's basically supported by every browser, even the oldest versions of Internet Explorer.

The one and only reason we chose JavaScript over TypeScript is that JavaScript is just much easier to implement, develop, update, and run on any browser. So, it also provides newbie-friendly code for rookie developers and beginners.

2.3. Which IDE (Integrated Development Environment) have we preferred?

I've developed the regular version in Visual Studio 2022. The web-based edition was created on Visual Studio Code. Both IDEs are owned by Microsoft Corporation.

In my opinion, those two IDEs are one of the best. The very first IDE that I have ever used in my entire life was Visual Studio 2010 Express. And, ever since then, I'm more of a Visual Studio guy, no matter what. I also like JetBrains' IDEs though.

3. The development: programming, debugging, and the browser implementation

3.1. The Regular Edition

I've just created a new project on Visual Studio. Here are some of its settings.

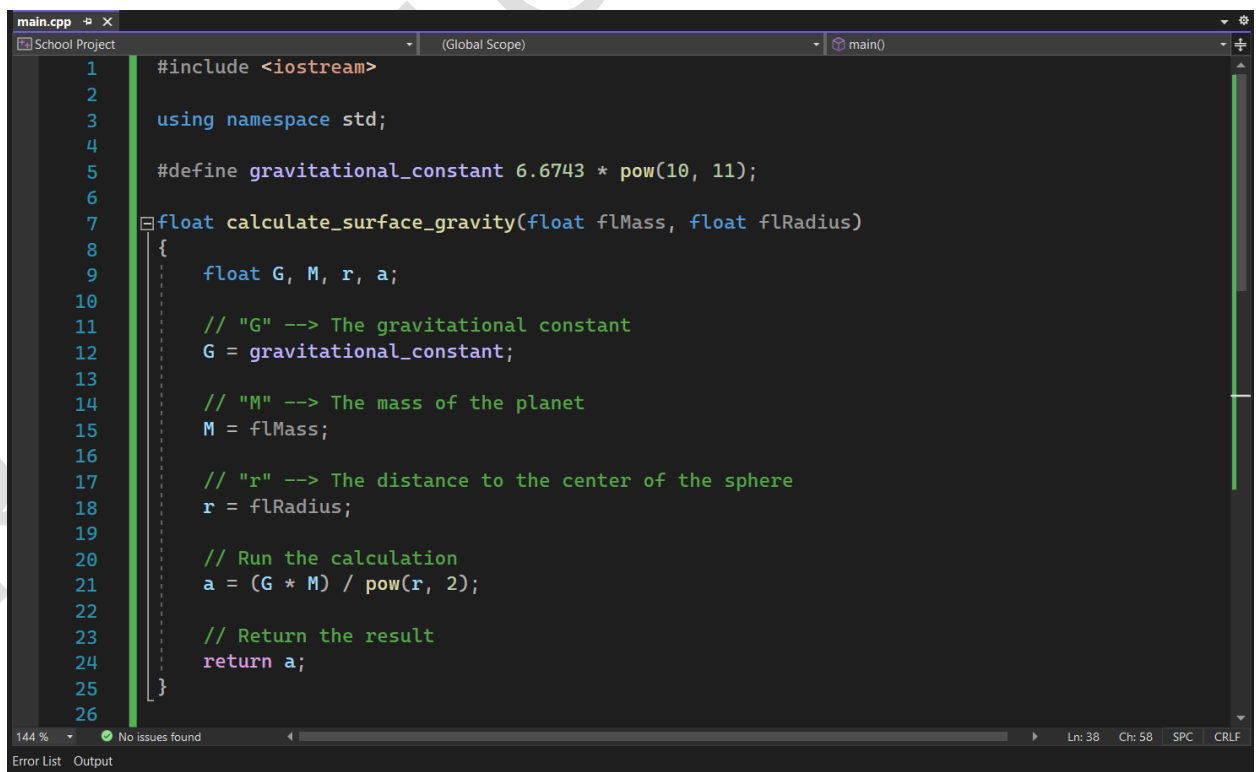
- C++ Language Standard: ISO C++20 Standard (/std:c++20)
- C Language Standard: ISO C17 (2018) Standard (/std:c17)
- Platform Toolset: Visual Studio 2022 (v143)
- Windows SDK Version: 10.0.19041.0

The very first thing that we need to do is create a function that will do the gravity calculations, after that, we should be able to skip to the part where we handle the rest – such as passing the incoming data from the client to that one function.

The following equation is the scientific formula that we will be using. Learn why we use this formula at <https://bit.ly/3Nihd4k> (Princeton University Physics Department)

$$\text{gravitational field strength} = \frac{(\text{gravitational constant})(\text{mass of the planet})}{(\text{distance from the center of the planet})^2}$$

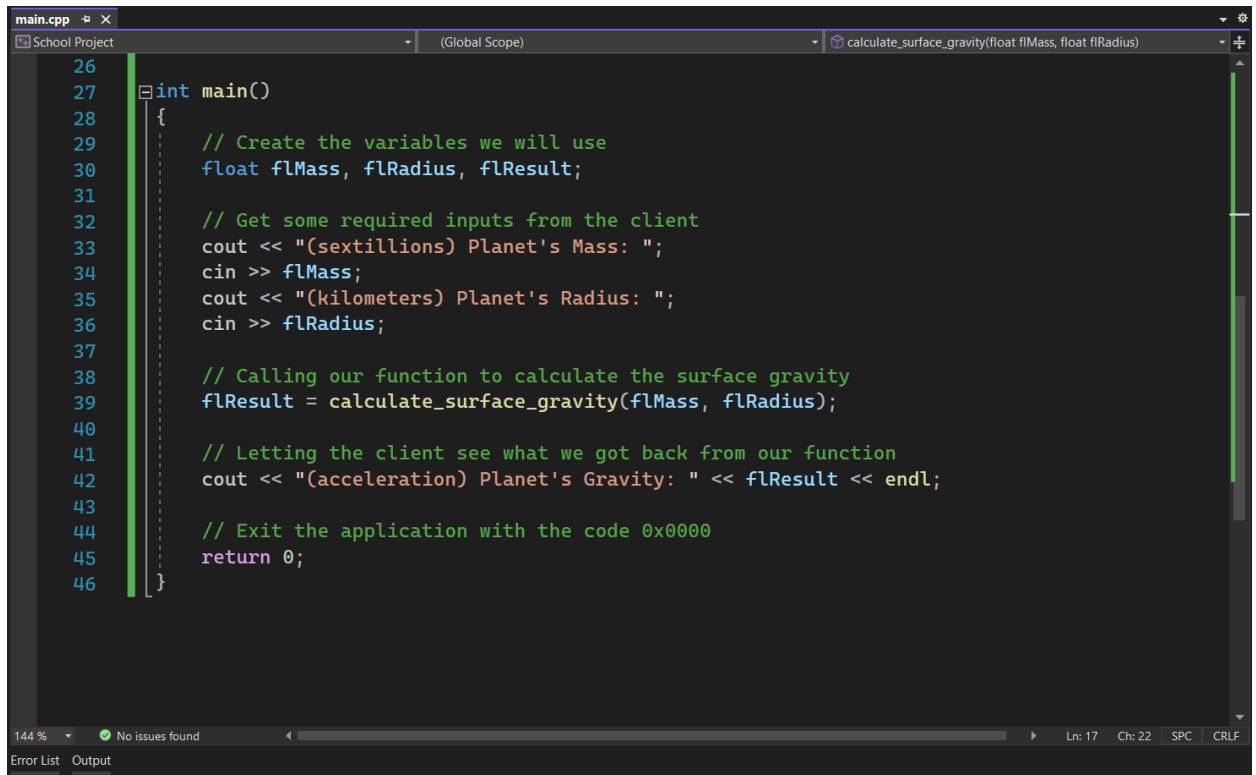
I have converted the equation above into C++ code. See the image below.



```
1  #include <iostream>
2
3  using namespace std;
4
5  #define gravitational_constant 6.6743 * pow(10, 11);
6
7  float calculate_surface_gravity(float fMass, float fRadius)
8  {
9      float G, M, r, a;
10
11      // "G" --> The gravitational constant
12      G = gravitational_constant;
13
14      // "M" --> The mass of the planet
15      M = fMass;
16
17      // "r" --> The distance to the center of the sphere
18      r = fRadius;
19
20      // Run the calculation
21      a = (G * M) / pow(r, 2);
22
23      // Return the result
24      return a;
25  }
26
```

Now it's time for us to create the main entry of the application.

It's going to be a simple function named main which will return an integer.

A screenshot of a C++ IDE window titled 'main.cpp'. The code defines a 'main()' function that initializes three float variables: flMass, flRadius, and flResult. It prompts the user for planet mass and radius, calls a function 'calculate_surface_gravity' with these values, and then prints the resulting gravity value. The code is as follows:

```
26
27 int main()
28 {
29     // Create the variables we will use
30     float flMass, flRadius, flResult;
31
32     // Get some required inputs from the client
33     cout << "(sextillions) Planet's Mass: ";
34     cin >> flMass;
35     cout << "(kilometers) Planet's Radius: ";
36     cin >> flRadius;
37
38     // Calling our function to calculate the surface gravity
39     flResult = calculate_surface_gravity(flMass, flRadius);
40
41     // Letting the client see what we got back from our function
42     cout << "(acceleration) Planet's Gravity: " << flResult << endl;
43
44     // Exit the application with the code 0x0000
45     return 0;
46 }
```

The IDE interface includes a toolbar at the top, a status bar at the bottom showing '144 %' and 'No issues found', and tabs for 'Error List' and 'Output'.

What the method above called `main()` does is that it creates three float variables named Mass, Radius, and Result. The data we will be getting from the client will be saved into two of these variables we have created which are Mass and Radius.

The other variable, Result, will be used to get a response from our function and save its incoming response. And, then, we will print the variable Result to the console.

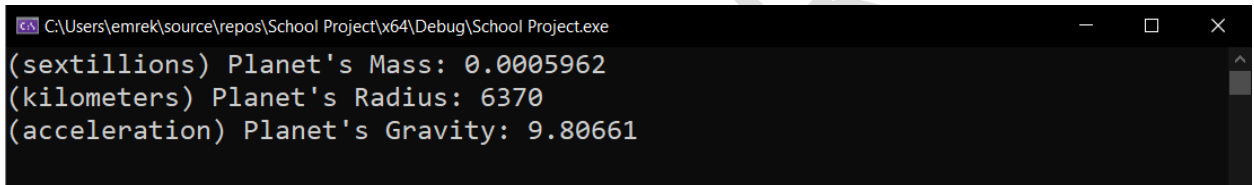
This is basically it.

Now, it's time to see whether or not the application works properly as expected.

Planet Earth's mass is 5.972×10^{24} kilograms. We are going to need to convert that into sextillion tons which is equal to 1×10^{21} kilograms. So, in this case, our planet's known mass, in sextillion tons, would be equal to about 0.0005962.

And, on the other hand, Planet Earth's equatorial radius is around 6370 kilometers.

Let's pass these numbers to the application and see what result it gives us back. By the way, there are consequently slight deviations in the magnitude of gravity across the surface. Gravity on the Earth's surface varies by around 0.7%, from 9.7639 m/s^2 on the Huascarán mountain in Peru to 9.8337 m/s^2 at the surface of the Arctic Ocean. This means that the result should be between 9.75 m/s^2 and 9.85 m/s^2



```
C:\Users\emrek\source\repos\School Project\x64\Debug\School Project.exe
(sextillions) Planet's Mass: 0.0005962
(kilometers) Planet's Radius: 6370
(acceleration) Planet's Gravity: 9.80661
```

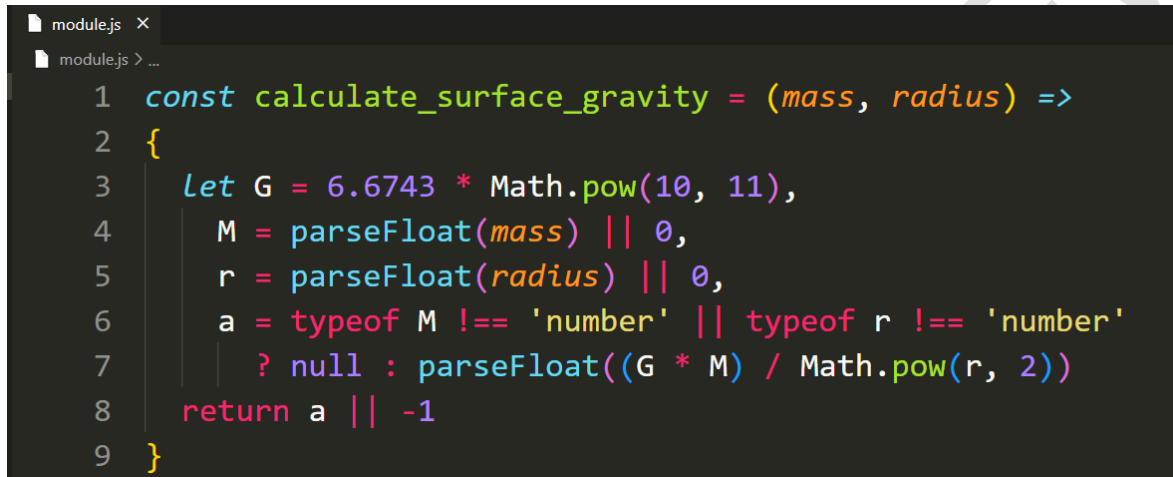
Voilà! Planet Earth's approximate surface gravity is 9.807 m/s^2 and we got back 9.8066 m/s^2 from our function. This proves the program works just as it should.

3.2. The Browser Edition

The browser edition will be much easier to create and implement as it'll be written in JavaScript, which is a language that almost every single browser supports.

Let's get started by creating a JavaScript file on Visual Studio Code.

I've created the file and just written the function. Here's how it looks.



```
1 const calculate_surface_gravity = (mass, radius) =>
2 {
3   let G = 6.6743 * Math.pow(10, 11),
4       M = parseFloat(mass) || 0,
5       r = parseFloat(radius) || 0,
6       a = typeof M !== 'number' || typeof r !== 'number'
7         ? null : parseFloat((G * M) / Math.pow(r, 2))
8   return a || -1
9 }
```

I was going to create an object-oriented thing, however, for now, we don't really need to use object-oriented programming as the script has only one actual function.

You can just run this JavaScript code on Node.js – but I am now going to create an HTML file to implement and use this code on an actual locally-hosted website.

Well then, let's create the HTML and run the code on a website.

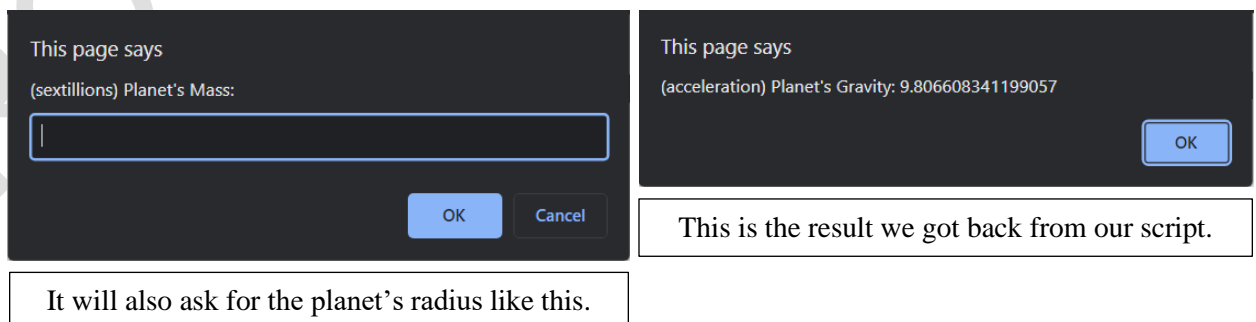
I've created the HTML file. Here's what it looks like.

```
index.html x
index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Hey! This is where the magic happens.</title>
8  </head>
9  <body>
10   <script src="module.js"></script>
11   <script type="text/javascript">
12     let mass = prompt("(sextillions) Planet's Mass:");
13     radius = prompt("(kilometers) Planet's Radius:");
14     result = calculate_surface_gravity(mass, radius)
15     alert(`(acceleration) Planet's Gravity: ${result}`)
16   </script>
17 </body>
18 </html>
```

This HTML has two script objects in it. One is for integrating our script, which you don't really need to import from another file at all, and the other one is to run it.

When you run this code, a box should show up on your screen asking you to type in the planet's mass – and, when this is done, another one should pop up right away, but, this time, it will be asking you to type in the planet's radius, and not the mass.

And, right after you type all these variables down, the calculated result should pop up on your screen just like the input boxes did. Here's what it looks like on a PC. On mobile devices, these alert boxes would probably look like native notifications.



It will also ask for the planet's radius like this.

4. The outroduction: the contributors of this project, and a few more details

4.1. Download links, source code, and other project files

Due to limited time, I couldn't really develop this project much. I was going to add more tools, make the whole thing more advanced, etc.

For now, it only has one function, both available on smart devices and browsers. However, by time, I'll probably develop this project more during the free time which I currently don't have.

Source code and downloads: <https://github.com/MisterXRD/School-Project>

4.2. More information about the developer of this project

Every single code in this project is/will be written by me, the author and the only developer of the project. Just to come clean about my identity, this is KÜTÜKER, E. (10-D/548)

I usually prefer to use and go by my other name NORMAN, J. Charles on the internet though.

One of my most known aliases is Mr. Radix – I chose that nickname as I'm a huge fan, investor, and a trusted community member of Radix DLT, which is basically a DeFi (Decentralized Finance) protocol. Learn more about Radix here: <https://radixdlt.com>

Still not sure who I actually am?

Here are the everything (accounts, websites, and even official/unofficial companies) I currently own and run:

- <https://twitter.com/MisterXRD> (My Twitter Account)
- <https://twitter.com/DinosaursXRD> (My NFT Project's Twitter Account)
- <https://radicaldinosaurs.com> (My NFT Project's Website)
- <https://weradix.com> (My Decentralized Apps Company's Website)
- WeRadix Interactive (My Decentralized Apps Company)
- Synclapse, Inc. (All My Projects' Parent/Holding Company)