

Universidad de Costa Rica

I Semestre 2023

**Proyecto de Programación Bajo Plataformas Abiertas –
IE0117**

Fecha Entrega: 25/6/23

Profesor: Juan Carlos Coto Ulate

Estudiante: Sebastián Bonilla Vega

C01263

Índice

Introducción	2
Trasfondo	3
Diseño General	4
Retos	11
Conclusiones	11
Bibliografía	11

Introducción

El siguiente documento fué elaborado con el objetivo de que este mismo sirva tanto como un manual de cómo utilizar el programa, así como explicar el funcionamiento del código, el análisis y razonamiento detrás de él y el propósito del programa en general. Se ve necesario verlo como un manual ya que fué hecho bajo la interpretación de las indicaciones dadas, lo cual provee cierta libertad para elaborar el código a criterio propio y con libertad de darle la forma que sea deseada. Principalmente, el programa fue hecho para modelar órbitas elípticas, utilizando parámetros dados. Entre estos parámetros, están los puntos iniciales del afelio y del perihelio y la masa del objeto, y en base a ellos se debería poder obtener la órbita, al menos a nivel lógico, ya que no hay parte gráfica en el proyecto.

Trasfondo

El modelo principal utilizado para modelar las órbitas en el proyecto fue el modelo planetario de Kepler. Dando una explicación breve sobre el modelo de Kepler, se basa en 3 leyes que fueron establecidas, propuestas por Johannes Kepler en el siglo XVII. La Primera Ley de Kepler, conocida como la Ley de las Órbitas, establece que todos los planetas en el sistema solar se mueven de manera elíptica alrededor del Sol, con dos puntos focales, donde el Sol está en uno de los focos. La Segunda Ley de Kepler, conocida como la Ley de Áreas, la que establece que un planeta cubre áreas iguales en tiempos iguales. Esto significa que, cuando un planeta se acerca al sol, este se mueve más rápido y por ende cubre más área en el mismo tiempo, y que cuando está más lejos del sol, cubre menos área. Finalmente, la Tercera Ley de Kepler, o la Ley de Periodos, establece que el tiempo que le toma a un planeta completar una órbita alrededor del Sol está directamente relacionada a su distancia promedio del Sol. Dicha relación, escrita de manera detallada, es que el periodo al Cuadrado de un planeta es proporcional a la distancia promedio del Sol al cubo.

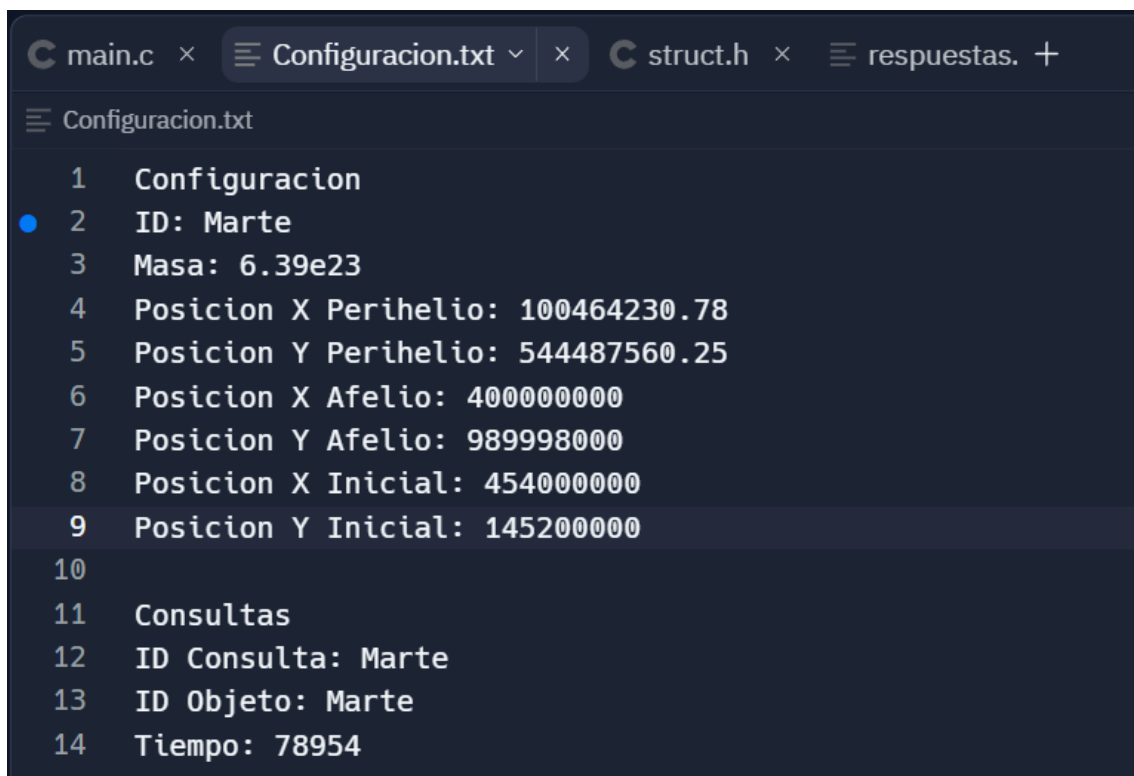
$$T^2 = \frac{4\pi^2 * a^3}{k^2(M + m)}$$

Donde T es el periodo, a es la distancia promedio, K es la constante de gravitación universal, M es la masa del Sol, y m la del cuerpo a analizar.

Diseño General:

El programa, en general, consiste en leer un archivo de configuración .txt, que a su vez está dividido en una parte de Descripción y en otra parte de consultas. Entonces, el programa va a leer el archivo de descripción, y de ahí va a leer y extraer la información necesaria para así poder modelar las órbitas. Posteriormente, iba a imprimir sus resultados en otro archivo, llamado "respuestas.txt". Las respuestas van a incluir la ID del objeto, así como las posiciones 'x' y 'y' en un tiempo dado.

Entonces, este programa permite mucha versatilidad, ya que la parte de configuración del archivo se puede editar libremente, lo que en teoría permite modelar cualquier objeto en el sistema solar, sin verse limitado solamente a planetas. Queda a libertad del usuario editar el archivo de configuración.



```
main.c x Configuracion.txt x struct.h x respuestas. +
Configuracion.txt
1 Configuracion
2 ID: Marte
3 Masa: 6.39e23
4 Posicion X Perihelio: 100464230.78
5 Posicion Y Perihelio: 544487560.25
6 Posicion X Afelio: 400000000
7 Posicion Y Afelio: 989998000
8 Posicion X Inicial: 454000000
9 Posicion Y Inicial: 145200000
10
11 Consultas
12 ID Consulta: Marte
13 ID Objeto: Marte
14 Tiempo: 78954
```

Figura 1. Configuración .txt con datos prueba.

Como se puede en la figura 1, el archivo de configuración está lleno con datos prueba. La parte de la masa fue directamente sacada de la página de NSSDCA de la NASA, y en base a la información en esa página también se escogieron valores de Perihelio, Afelio e Iniciales estratégicamente. A la hora de editar la configuración, las posiciones deben estar dadas en kilómetros.

Ahora, analizando las respuestas escritas en el archivo 'respuestas.txt', se puede observar que:

```
respuestas.txt
1
2  ID de la Consulta: Marte
3  Coordenada X en el tiempo 78954.00: 453921046.000000
4  Coordenada Y en el tiempo 78954.00: 145200018.000000
5  ID de la Consulta: Marte
6  Coordenada X en el tiempo 78954455040.00:
   -2147483648.000000
7  Coordenada Y en el tiempo 78954455040.00:
   -8543644672.000000
8  ID de la Consulta: Marte
9  Coordenada X en el tiempo 78954456.00: 377142878.000000
10 Coordenada Y en el tiempo 78954456.00: 163277307.000000
11
```

Figura 2. Respuestas.txt con datos respuesta.

Se puede ver como los valores de 'x' y 'y' dependen del tiempo anotado en la parte 'tiempo' del archivo de configuración. Un tiempo muy pequeño causa una diferencia muy pequeña entre los puntos finales e iniciales, este es el del primer caso, donde el tiempo es de 78954 segundos. En una perspectiva astronómica, este tiempo es sumamente pequeño, ya que se trata de aproximadamente 22 horas, casi un día. Sin embargo, al aumentar el valor del tiempo a uno casi que aleatorio y considerablemente más grande, de 78954455040 segundos, nos da ese valor de 'x' y 'y' negativo, lo cual tiene sentido, ya que las coordenadas están simplificadas a una elipse cartesiana, entonces cabe dentro de la posibilidad que se esté en la parte negativa en 'x' y 'y' de dicha elipse. Para el tercer caso, se toma un tiempo relativamente más sensato, el cual aún valida que la posición cambia con respecto al tiempo, el cual fue uno de los más grandes retos, pero se adentrará a más profundidad más adelante. Ya habiendo terminado de describir los archivos de .txt, el archivo principal se divide en un header llamado 'struct.h' y el 'main.c'. La idea del archivo .h era definir solo structs ahí, pero se terminaron definiendo funciones adicionales dentro de ese archivo.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <math.h>
5  #define MAX_LINE_LENGTH 1024
6  #define G 6.67430e-11
7  #define M 1.989e30
8
9
10 typedef struct {
11     char idObjeto[MAX_LINE_LENGTH];
12     float masa;
13     float X_P;
14     float Y_P;
15     float X_A;
16     float Y_A;
17     float X_I;
18     float Y_I;
19     char C_cons[MAX_LINE_LENGTH];
20     char C_obj[MAX_LINE_LENGTH];
21     float tiempo;
22 } Configuracion;
23
24 typedef struct {
25     float x;
26     float y;
27 } Point;

```

Figura 3. Estructuras Definidas en “struct.h”.

De la figura anterior, se pueden ver las dos estructuras principales utilizadas en todo el programa. Estas son “Configuracion” y “Point”. El struct “Configuracion”, en particular, se encarga de almacenar los valores leídos y extraídos del archivo Configuracion.txt, para que puedan ser utilizados en otras partes del programa libremente. El struct “Point” fue definida con el proyecto más avanzado, ya que era necesario la definición de éste struct para así poder ‘guardar’ los datos en forma de puntos cartesianos, y así poder utilizar estos puntos cartesianos en otras funciones que se verán más adelante.

```

29 int calcularElipse(Point p1, Point p2, float masa) {
30     Configuracion config;
31     double cx = (p1.x + p2.x) / 2.0; // Coordenada x del centro de la elipse
32     double cy = (p1.y + p2.y) / 2.0; // Coordenada y del centro de la elipse
33     double a = fabs(cx - p1.x); // Semi-eje mayor
34     double b = fabs(cy - p1.y); // Semi-eje menor
35     //la funcion fabs me da el valor absoluto
36     double denominador = G * (M + masa);
37     double T = 2 * M_PI * sqrt(pow(a*10e4, 3) / denominador);
38     return T;
39 }
40 int calcularVel(Point p3, int T) {
41     Configuracion config;
42     int distance = sqrt(pow(p3.x, 2) + pow(p3.y, 2));
43     int velocity = (2 * M_PI * distance) / T;
44     return velocity;
45 }
46 int calcularCos(double X_I, double distance, double ang){
47     return X_I + distance * cos(ang);
48 }
49 int calcularSen(double Y_I, double distance, double ang){
50     return Y_I + distance * sin(ang);
51 }
52 // las funciones calcularSen, calcularCos, calcularVel y calcularElipse fueron hechas con
    chatgpt

```

Figura 3. Funciones Definidas en “struct.h”.

Las funciones definidas en esta parte calculaban otros valores necesarios para poder determinar los puntos finales en un tiempo dado. La función “calcularElipse” fue nombrada así porque, como su nombre lo dice, calcula una elipse. Hace esto tomando ambos structs, y con ellos calcula primero el centro de dicha elipse, utilizando los valores del afelio y perihelio del archivo de configuración. Posteriormente, calcula el semieje mayor y el menor. Con estos valores luego se procede a calcular el periodo orbital de acuerdo a la definición de la Tercera Ley de Kepler, definimos una variable “denominador” que depende de la constante Gravitatoria Universal, y la Masa del Sol, que ambas ya son definidas al inicio del Header, y también se toma la masa del objeto que se encuentra en el archivo config. Luego, se toma el valor del semieje “a”, y se eleva al cubo. También se le agrega un factor de conversión.

La función “calcularVel” fue nombrada así porque, como su nombre lo dice nuevamente, calcula la velocidad. En esta función, se toma p3, que incluye los puntos ‘x’ y ‘y’ iniciales anotadas en la configuración, que nuevamente, pueden ser alterados libremente, para calcular una distancia. Luego, toma la distancia y la multiplica por 2π y la divide entre el periodo. Se hace esto porque se asemeja a una órbita elíptica. Para los puntos p1, p2 y p3 es que se necesitó definir el struct ‘Point’. Esto devuelve un valor ‘distancia’ determinado con Pitágoras, el cual después es usado para determinar la velocidad con el periodo orbital. Esta es la velocidad con la que se supone que se recorre el arco. Finalmente, se definen las funciones “calcularCos” y “calcularSen”, ya que son las que definen la posición final en el arco, porque es un movimiento que sigue una elipse. Como está anotado en el código, las funciones de la Elipse, Velocidad, Seno y Coseno fueron sugeridas por ChatGPT al consultar sobre una manera de simular un sistema solar, sugeridas una a la vez. Los prompts se incluirán en la parte de bibliografía y referencias.

Hasta acá, se termina la parte de análisis a la parte del header. Posteriormente, se describirá el funcionamiento del código que se encuentra en la parte del main. En resumen, el main se fue hecho más con el propósito de leer y tomar la información deseada del archivo de configuración, que fue otro de los retos más elaborados de este proyecto, el cual se analizará más a fondo en su parte correspondiente.


```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5 #include "struct.h"
6 #define G 6.67430e-11
7 #define M 1.989e30
8 #define MAX_TEXT_LENGTH 100
9
10 int main() {
11     const char* nombre_archivo = "Configuracion.txt";
12     Configuracion config;
13     if (leer_config(nombre_archivo, &config)) {
14         printf("ID Objeto: %s\n", config.idObjeto);
15         printf("Masa: %.2e kg\n", config.masa);
16         printf("Posicion X Perihelio: %.2f m\n", config.X_P);
17         printf("Posicion Y Perihelio: %.2f m\n", config.Y_P);
18         printf("Posicion X Afelio: %.2f m\n", config.X_A);
19         printf("Posicion Y Afelio: %.2f m\n", config.Y_A);
20         printf("Posicion X Inicial: %.2f m\n", config.X_I);
21         printf("Posicion Y Inicial: %.2f m\n", config.Y_I);
22         printf("ID Consulta: %s\n", config.C_cons);
23         printf("ID Objeto: %s\n", config.idObjeto);
24         printf("Tiempo: %.2f\n", config.tiempo);
25     } else {
26         printf("No se encontró la configuración en el archivo.\n");
27     }
28     Point p1 = {config.X_A, config.Y_A};
29     Point p2 = {config.X_P, config.Y_P};
30     Point p3 = {config.X_I, config.Y_I};
31     ;
32
33     //Se toman los datos del struct y por ende, del .txt de configuracion para calcular la elipse
34     int T = calcularElipse(p1, p2, config.masa);
35     //Se calcula una elipse porque las leyes de Kepler establecen que los cuerpos de nuestros sistemas se mueven de manera eliptica
36     float vel = calcularVel(p3, T);
37     double ang = (2 * M_PI * config.tiempo) / T;
38     //Para calcular X e Y, es necesario calcular el angulo al que se encuentra el cuerpo del centro, que estamos asumiendo que es el Sol.
39     double dist = vel * config.tiempo;
40     double X_f = calcularCos(config.X_I, dist, ang);
41     double Y_f = calcularSen(config.Y_I, dist, ang);
42     return 0;
43 }

```

Figura 4. Archivo 'main.c' con sus funciones.

Investigando en páginas como stack overflow y en Youtube, se encontró cómo hacer para leer un archivo y sacar información relevante de él, que fue otro gran reto en el que se ahondará en su sección. En la primera parte, básicamente se le pide al programa que abra Configuracion.txt y se le mandó a imprimir cada valor individualmente. Por ejemplo, se ordenó imprimir "ID Objeto", después de ejecutar la función "leer_config", que se va a describir más adelante, para así asegurar que estaba imprimiendo el valor deseado, que era el que correspondía a la parte ID Objeto en el archivo de configuración. Así se hizo para todos los valores del archivo.

```

> make -s
> ./main
ID Objeto: Marte
Masa: 6.39e+23 kg
Posicion X Perihelio: 100464232.00 m
Posicion Y Perihelio: 544487552.00 m
Posicion X Afelio: 400000000.00 m
Posicion Y Afelio: 989998016.00 m
Posicion X Inicial: 454000000.00 m
Posicion Y Inicial: 145200000.00 m
ID Consulta: Marte
ID Objeto: Marte
Tiempo: 90000.00
> 

```

Figura 5. Valores imprimidos en consola.

Además, después de mandar a imprimir los valores para que fuera más fácil asegurar que todo anduviera bien, se procedió a utilizar el struct "Point". En este se llena con las variables p1, p2 y p3. Se tiene que p1 incluye a los puntos "x" y "y" del afelio, p2 incluye los puntos "x" y "y" del perihelio, se usaron ambos para determinar la elipse y p3 incluye los puntos "x" y "y" iniciales, para así determinar los puntos "x" y "y" finales. Posteriormente, en la función main después se llamó a las funciones del header, para así poder guardar las variables en el archivo principal y ejecutarlas en la función "leer_config", porque de otro modo el código no corría. Se guardaron las variables del Periodo (T), velocidad (vel), Ángulo (ang), Posición X Final (X_F) y Posición Y Final (Y_F). Cabe destacar que la función calcularSen utiliza la distancia que depende del tiempo de config, el ángulo que depende tanto del periodo como el tiempo en config y la posición X_I de config. Paralelamente, la función calcularCos utiliza la misma distancia y ángulo, pero utiliza la posición Y_I de config.

```

44 ~ int leer_config(const char* archivo, Configuracion* config) {
45 ~     FILE* archivo_txt = fopen("Configuracion.txt", "r");
46 ~     if (archivo_txt == NULL) {
47 ~         printf("No se pudo abrir el archivo.\n");
48 ~         return 0;
49 ~     }
50 ~     char linea[MAX_LINE_LENGTH];
51 ~     while (fgets(linea, sizeof(linea), archivo_txt) != NULL) {
52 ~         if (strstr(linea, "ID Objeto:") == linea) {
53 ~             sscanf(linea, "ID Objeto: %[^\\n]", config->idObjeto);
54 ~         } else if (strstr(linea, "Masa:") == linea) {
55 ~             sscanf(linea, "Masa: %f", &(config->masa));
56 ~         } else if (strstr(linea, "Posicion X Perihelio:") == linea) {
57 ~             sscanf(linea, "Posicion X Perihelio: %f", &(config->X_P));
58 ~         } else if (strstr(linea, "Posicion Y Perihelio:") == linea) {
59 ~             sscanf(linea, "Posicion Y Perihelio: %f", &(config->Y_P));
60 ~         } else if (strstr(linea, "Posicion X Afelio:") == linea) {
61 ~             sscanf(linea, "Posicion X Afelio: %f", &(config->X_A));
62 ~         } else if (strstr(linea, "Posicion Y Afelio:") == linea) {
63 ~             sscanf(linea, "Posicion Y Afelio: %f", &(config->Y_A));
64 ~         } else if (strstr(linea, "Posicion X Inicial:") == linea) {
65 ~             sscanf(linea, "Posicion X Inicial: %f", &(config->X_I));
66 ~         } else if (strstr(linea, "Posicion Y Inicial:") == linea) {
67 ~             sscanf(linea, "Posicion Y Inicial: %f", &(config->Y_I));
68 ~         } else if (strstr(linea, "ID Consulta:") == linea) {
69 ~             sscanf(linea, "ID Consulta: %[^\\n]", config->C_cons);
70 ~         } else if (strstr(linea, "ID Objeto:") == linea) {
71 ~             sscanf(linea, "ID Objeto: %[^\\n]", config->C_obj);
72 ~         } else if (strstr(linea, "Tiempo:") == linea) {
73 ~             sscanf(linea, "Tiempo: %f", &(config->tiempo));
74 ~         }
75 ~     }
76 ~     fclose(archivo_txt);

```

Figura 6. Función "leer_config", parte que lee.

De la figura 6 se puede observar cómo está estructurada la parte de lectura del archivo config. De nuevo, investigando en Youtube fue sugerida esta manera de leer un el archivo config. Primero se abre el archivo con “fopen” y con “r” se le indica que solo va a leer. Se le asigna una memoria determinada de 100 caracteres por línea, y después lo que hace, mediante un while, es leer línea a línea buscando lo que se le indica. Tomando “ID Objeto” de ejemplo nuevamente, el programa se abre, en modo lectura, busca la primera línea, y si no esta “ID Objeto” va a saltarse esa línea hasta que alcance una línea que tenga “ID Objeto”, y luego va almacenar el string en el struct que ya había sido definido antes mediante la función “sscanf”. Luego, va a moverse a la siguiente condición. Este proceso, va a ser repetido uno a uno hasta que el struct esté lleno. Ya cuando se haga todo el proceso, se cierra el archivo.

```

77 FILE* fr;
78 Point p1 = {config->X_A, config->Y_A};
79 Point p2 = {config->X_P, config->Y_P};
80 Point p3= {config->X_I, config->Y_I};
81 int T = calcularElipse(p1, p2, config->masa);
82 float vel = calcularVel(p3, T);
83 double dist = vel * config->tiempo;
84 double ang= (2 * M_PI*config->tiempo)/ T;
85 double X_F= calcularCos(config->X_I,dist,ang);
86 double Y_F= calcularSen(config->Y_I,dist,ang);
87 fr = fopen("respuestas.txt", "a");
88 fprintf(fr, "ID de la Consulta: %s\n", config->C_cons);
89 fprintf(fr, "Coordenada X en el tiempo %.2f: %f\n", config->tiempo, X_F);
90 fprintf(fr, "Coordenada Y en el tiempo %.2f: %f\n", config->tiempo, Y_F);
91 fclose(fr);
92 return 1;
93 }

```

Figura 7. Función “leer_config”, parte que imprime en un nuevo .txt.

Finalmente, se llega a la parte de impresión en un nuevo .txt. Abriendo FILE*fr, donde fr seria nuestra file de respuestas, se define todo a utilizar. Se llaman los puntos p1, p2 y p3, el periodo T, la velocidad vel, ángulo ang, distancia dist, X_F y X_Y. Después, se le pide a fr que abra un nuevo archivo, pero en función “a”, que debe agregar al texto, de esta forma quedan guardado absolutamente todas las consultas que se hagan del cuerpo que sea. Para que en las respuestas txt salga la consulta deseada, cada ID debe ser cambiado por el objeto deseado. Si, por ejemplo, quiero analizar a Júpiter, en el config debo cambiar el valor de la masa por el de Júpiter, y todo lo que diga ‘ID’ debe decir Júpiter. Los puntos del afelio y el perihelio pueden ser escogidos en base a la página de NSDCC.

Principales Retos

Por ahora, se ha hecho mención de ciertas partes del código que han presentado más dificultad que otras. En orden, primero se mencionó que lograr que las posiciones dependieran del tiempo fue algo particularmente difícil. En lo personal, lo fue porque no podía lograr referenciar el tiempo de una manera correcta, ya que esperaba el valor específico que decía en la configuración, pero el programa solo tiraba "0" y después de 3 iteraciones donde todo daba 0 era fácil frustrarse.

Para leer el archivo y sacar información relevante de ahí se ocupó una nueva perspectiva, ya que era algo que yo no había hecho antes, al menos no de manera tan directa, se ocupó mucha investigación para poder lograr que saliera. Elaborar todo este proyecto como tal se puede considerar un reto muy grande, ya que como tal C es un lenguaje complicado de aprender e implementar, pero todo reto deja una enseñanza.

Finalmente, la parte de implementar tantas funciones y asegurar que todas funcionen como deben hacerlo para así poder lograr obtener los datos necesarios o esperados de una simulación de tal magnitud. Otro reto es tener que investigar como subir este trabajo a github porque, de nuevo, todo esto es nuevo. Es una programación más avanzada al nivel que ya traía, por lo que tuve que ajustarlo al nivel necesario.

Conclusiones

En general, la experiencia estuvo intensa e interesante, como la mayoría de cursos universitarios. Sin embargo, no fue sin su provecho, ya que puedo agregar que ya sé, o que al menos sé un poco más de lo que sabía antes sobre un lenguaje tan complejo como C. De las lecciones más importantes fue a como trabajar bajo presión y con tiempos limitados, pero de una manera más directa, aprendí a leer y sacar información de archivos, que eso me parece sumamente importante y tiene gran cantidad de aplicaciones laborales. Para la parte de dudas sin responder, no me quedan muchas. Las que tenía se fueron respondiendo una a una conforme iba realizando el proyecto. Otras dudas surgirán con otros proyectos.

Referencias Bibliográficas

Si bien no se referencio nada de manera directa en el reporte, todos estos links fueron utilizados para elaborar el proyecto.

- Williams, D. (2023). Planetary Fact Sheet-Metric. Nasa. <https://nssdc.gsfc.nasa.gov/planetary/factsheet>
- TNB. (2014). Playlist [The New Boston]. Youtube. <https://www.youtube.com/watch?v=2NWeucMKrLI&list=PL6qx4Cwl9DGAkIXv8Yr6nhGJ9Vlcjyymq>
- OpenAI. (2023). *ChatGPT* (May 24 Version) [Como hago una elipse a base de dos puntos en C]. <https://chat.openai.com/share/983d7f87-cd52-4cb2-a0ca-fbcf11b89930>
- OpenAI. (2023). *ChatGPT* (May 24 Version) [Periodo Orbital de Marte]. <https://chat.openai.com/share/9fd8391c-de54-4b69-b06a-f45ee49143ac>
- OpenAI. (2023). *ChatGPT* (May 24 Version) [Puedes determinar la velocidad con el periodo orbital, un punto inicial dado y la masa en C]. <https://chat.openai.com/share/61b144cc-5982-4818-9d1e-3ffe77b7e128>
- OpenAI. (2023). *ChatGPT* (May 24 Version) [Calcula los puntos finales con el periodo orbital, la velocidad y los puntos iniciales]. <https://chat.openai.com/share/7ab1d28c-b743-42f4-99a6-0e571e11b558>