Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

IE-0117 Programación Bajo Plataformas Abiertas

Integrante:

Jean Antonio Zúñiga Mendoza

Profesor:

Juan Carlos Coto

Sede:

Rodrigo Facio

26/06/2023

ÍNDICE

Introducción:

En este documento se especificará el proceso de desarrollo de un programa en lenguaje C, que calcula la órbita de cuerpos celestes en un plano XY desde un archivo.

Para este proceso se utilizaron ciertas herramientas propias de lenguaje C en el compilador llamado replit. Tales como el manejo de listas, memoria, manejo de archivos de texto; tanto como lectura como escritura, funciones para cálculos matemáticos, manejo de datos como char o float y Makefile.

El propósito de este es la lectura del archivo de texto que contiene dos secciones delimitadas por una línea, como entrada, donde contiene la información para los cálculos adecuados. Y como salida sea un archivo con las coordenadas de la ubicación del cuerpo celeste después de finalizado la simulación.

Trasfondo:

Las leyes de Kepler o leyes del movimiento planetario son leyes científicas que describen el movimiento de los planetas alrededor del Sol. Reciben el nombre de su creador, el astrónomo alemán Johannes Kepler (1571-1630).

El aporte fundamental de las leyes de Kepler fue dar a conocer que las órbitas de los planetas son elípticas y no circulares como se creía antiguamente.

Aunque la teoría heliocéntrica sustituyó a la teoría geocéntrica, ambas compartían una creencia común: que las órbitas de los planetas eran circulares. Gracias al hallazgo de Kepler, la teoría heliocéntrica pudo perfeccionarse. Para efectos de este programa, se considera las orbitas como si se movieran en un plano heliocéntrico.

Determinar la forma de la elipse:

La primera ley de Kepler establece que los planetas describen órbitas elípticas con el Sol en uno de los focos de la elipse. Esto significa que la distancia entre un planeta y el Sol varía a lo largo de su órbita, y el punto más cercano se conoce como perihelio, mientras que el punto más alejado se denomina afelio. Para calcular la forma de la órbita, utilizamos las siguientes operaciones matemáticas:

- Determinar la excentricidad de la elipse utilizando la distancia del perihelio y el afelio.

- La excentricidad se calcula como la relación entre la distancia entre los focos de la elipse y la longitud del eje mayor de la elipse.

La excentricidad describe el grado de achatamiento de una curva cerrada. Cuando la excentricidad es igual a 0, la curva forma un círculo perfecto. En cambio, cuando la excentricidad es superior a 0, se achatan los lados de la curva formando una elipse. Si tenemos las componentes del perihelio y el afelio podemos calcular su magnitud con el Teorema de Pitágoras. Con estas podemos formular la ecuación de la excentricidad:

$$e = \frac{afelio - perihelio}{afelio + perihelio}$$

Medida de la distancia promedio entre un planeta y su estrella:

En la tercera ley de Kepler, Kepler estableció que el cuadrado del período de revolución de un planeta es proporcional al cubo de su distancia media al Sol. Esta ley matemática permite establecer una relación entre el período orbital y el semieje mayor de una órbita. En otras palabras, cuanto mayor sea el semieje mayor de una órbita, más largo será el período orbital del planeta.

La importancia del semieje mayor radica en que nos proporciona una medida de la distancia promedio entre un planeta y el Sol. Esto tiene implicaciones significativas para comprender la dinámica orbital y predecir el movimiento de los planetas en el sistema solar.

$$a = \frac{afelio + perihelio}{2}$$

Newton Y Kepler:

La anomalía verdadera es un concepto importante en las leyes de Kepler y se utiliza para describir la posición de un objeto en órbita en relación con su órbita elíptica alrededor de un cuerpo central. Está relacionada con la tercera ley de Kepler, que establece una relación entre el período orbital y la distancia media al cuerpo central.

Las leyes de Kepler establecen que, para una órbita elíptica, la velocidad del objeto en órbita no es constante, sino que varía a medida que se mueve a lo largo de su trayectoria. Estas leyes también proporcionan relaciones entre las anomalías y otros parámetros orbitales, como el período orbital, la excentricidad y el semieje mayor de la órbita.

$$v = \frac{2PI}{T} * t$$

Donde:

v es la anomalía verdadera.

t es el tiempo.

T es el periodo orbital.

El período orbital es el tiempo que tarda un objeto en completar una vuelta alrededor de otro cuerpo central en su órbita. El cálculo del período orbital depende de la masa del cuerpo central y la distancia media entre el objeto en órbita y el cuerpo central.

De acuerdo con la tercera ley de Kepler, el período orbital (T) está relacionado con el semieje mayor de la órbita (a) mediante la siguiente fórmula:

$$2 * pi \sqrt{\frac{a^3}{G * M}} = T$$

Donde:

T es el período orbital.

a es el semieje mayor de la órbita.

G es la constante de gravitación universal.

M es la masa del cuerpo central alrededor del cual el objeto está orbitando.

En este caso, el período orbital se calcula utilizando la tercera ley de Kepler, que relaciona el período orbital con el semieje mayor de la órbita, la constante de gravitación universal y la masa del cuerpo central. Esta fórmula proporciona una estimación del tiempo que tarda un objeto en completar una vuelta en su órbita alrededor de otro cuerpo en el espacio.

Angulo verdadero:

El ángulo verdadero es un parámetro utilizado para describir la posición de un objeto en una órbita elíptica en un momento determinado. Se refiere al ángulo medido entre el punto de perihelio (o perigeo en el caso de órbitas alrededor de la Tierra) y la posición actual del objeto en su órbita, con respecto al centro del cuerpo central.

El ángulo verdadero es un parámetro importante para describir la posición de un objeto en su órbita elíptica. Juega un papel crucial en el cálculo de la posición y velocidad del objeto en coordenadas polares. Además, el ángulo verdadero se utiliza en diversas aplicaciones, como la determinación de la visibilidad de un objeto en el cielo, la planificación de misiones espaciales y la predicción de eventos astronómicos.

La ecuación para estos efectos es:

$$i = atan\frac{y}{x}$$

Donde:

i es el ángulo verdadero.

y es la coordenada “y” en el plano al inicio de la simulación.

x es la coordenada “x” en el plano al inicio de la simulación.

Sin embargo, al estar relacionado con la anomalía verdadera, se puede expresar de distintas formas. En este caso se usó esta expresión por tener los datos necesarios para esta.

Coordenadas XY:

Primero, la distancia desde el centro del Sol al punto orbital de un planeta se puede calcular utilizando la fórmula de la distancia en una órbita elíptica, que depende del semieje mayor y la excentricidad de la órbita, así como de la anomalía verdadera en un momento determinado. Se puede calcular utilizando la siguiente ecuación:

$$r = \frac{a(1 - e^2)}{1 + e * cos\theta}$$

Donde:

r es la distancia desde el centro del Sol al punto orbital del planeta.

a es el semieje mayor de la órbita.

e es la excentricidad de la órbita.

θ es la anomalía verdadera en ese momento.

Es importante destacar que esta fórmula asume una órbita elíptica alrededor del Sol. Si el objeto sigue una órbita circular o una órbita alrededor de otro cuerpo celeste, se requerirían fórmulas diferentes.

Segundo, las coordenadas en el plano orbital (x, y) utilizando las coordenadas polares. Los cuerpos celestes al suponer que orbitan en un plano, las ecuaciones se simplifican al punto de:

$$x = r * cos(v)$$

$$y = r * sin(v) * cos(i)$$

Donde:

r es la distancia desde el centro del Sol al punto orbital del planeta.

i es la inclinación.

v es la anomalía verdadera.

Estas fórmulas son aplicables para convertir coordenadas polares a cartesianas en el plano orbital y se utilizan comúnmente en la representación y cálculo de las posiciones de objetos en órbitas elípticas, circulares u otras trayectorias curvas.

Diseño general:

Funciones.c:

Se decidió que el nombre del archivo donde se declararían todas las funciones necesarias para calcular las órbitas se llamase Funciones.c

```c
#include "Funciones.h"
#include <math.h>
#include <stdio.h>

#define PI 3.1416
#define M 1.989*pow(10,30)
#define G 6.67*pow(10,-11)

float calcularHipotenusa1(float cateto1, float cateto2) {
    float hipotenusa;

    // Aplicar el teorema de Pitágoras para calcular las magnitudes del perihelio y afelio. Y otros...
    hipotenusa = sqrt(pow(cateto1, 2) + pow(cateto2, 2));

    return hipotenusa;
}
```

En esta parte se definieron algunas constantes necesarias para próximos cálculos. El comando "pow" en C se utiliza para calcular la potencia de un número. Su aplicación se basa en “pow(base, exponente)”. Esto resulta muy útil para transformar las constantes, como la de gravitación universal, en unidades del Sistema Internacional de Unidades (SI) y, también se usó en la función “calcularHipotenusa” que esta sirve para calcular las magnitudes del perihelio y el afelio. También, en vez de usar “pow” se pudo multiplicar dos veces las entradas.

La función “calcularHipotenusa” culmina con el comando “sqrt” que sirve para calcular la raíz cuadrada. Esta sección también se pudo usar “pow” pero se evito para evitar confusiones.

```
//Calcular el semieje mayor (a)
float semieje_mayor(float perihelio, float afelio){
  float a;
  a = perihelio + afelio / 2;
  return a;

}

//Calcular la excentricidad (e)
float excentricidad (float perifelio, float afelio){
  float e;
  e = (afelio - perifelio)/(afelio + perifelio);
  return e;
}

```

En esta sección las funciones “semieje_mayor” y “excentricidad” reciben dos parámetros que son las magnitudes del perihelio y afelio, y siguen las operaciones matemáticas declaras en la sección anterior.

```
// periodo orbital: masa es la masa del objeto y a es el semieje mayor
float periodo_orbital (float masa){
  float a = semieje_mayor(4, 2);
  float T = 2*PI/sqrt(pow(a,3)/M*G);
  return T;
}
//inclinacion  (ω):
float inclinacion (float y_i, float x_i){
  float i = atan(y_i/x_i);
  return i;
}
// velocidad del cuerpo:
float velocidad (float t, float T){
  float v = (2 * (PI) / T)*t;
  return v;
}
```

La función “periodo_orbital” recibe como parámetro a masa como flotante, pero este no es usado en la función en absoluto. La razón se debe a que se confundió la constante M con la masa.

La función “inclinacion” y la función “velocidad” sufrieron algo parecido y es que se le etiquetaron con nombres que no representan sus cálculos.

En “inclinacion” lo que se calcula es el ángulo verdadero y en “velocidad” se calcula la anomalía verdadera.

Este error se debe a la denotación de estas igualdades; “i” para el ángulo verdadero que puede ser confundida con la inclinación, ya que sería lo más intuitivo y “v” para la anomalía verdadera que también puede ser confundida con “velocidad”. Estas confusiones se tomaron en cuenta para el calculo de las componentes XY finales.

```
49
50  // calcular la distancia desde el centro del sol
51  float distancia (float i, float v, float e, float a) {
52    float r, x, y;
53    r = a * (1 - pow (e, 2)) / (1 + e * cos(v));
54    x = r * cos(v);
55    y = r * sin(v) * cos(i);
56    printf("La posición en x es: %.2f\n", x);
57    printf("La posición de y es: %.2f\n", y);
58    return 0;
59    // a es semieje mayor. I es inclinacion. v es velocidad. e es exsentricidad
60  }
```

En la función “distancia” se calculan las componentes XY finales. Esta recibe cuatro parámetros que son extraídos desde el archivo main.c e imprime las componentes.

main.c:

En main.c se usó una herramienta generadora de código que, a pesar de que se tuvo que modificar para obtener lo esperado, no cambio mucho en su topología.

Su edición se basó en el nombre de las variables y tipo de datos como double a float.

La entrada fue:

> “Un programa en C que reciba un archivo con una sección llamada [descripcion] delimitada por una línea. Cada línea esta de sección contiene los siguientes datos, separados por comas: id, masa, x_p, y_p, x_a, y_a, x_i,y_i”

```
1  #include "Funciones.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  float semieje, incli, velo, excen, periodo;
7
8  #define MAX_LINE_LENGTH 100
9
10 int main() {
11     FILE *archivo;
12     char linea[MAX_LINE_LENGTH];
13
14     archivo = fopen("datos.txt", "r"); // Cambia "datos.txt" por el nombre de tu archivo
15
16     if (archivo == NULL) {
17         printf("No se pudo abrir el archivo.\n");
18         return 1;
19     }
```

```
int dentro_de_seccion = 0;
while (fgets(linea, MAX_LINE_LENGTH, archivo) != NULL) {
    // Verificar si estamos dentro de la sección "[descripcion]"
    if (strcmp(linea, "[descripcion]\n") == 0) {
        dentro_de_seccion = 1;
        continue;
    } else if (strcmp(linea, "\n") == 0) {
        dentro_de_seccion = 0;
    }
```

En primer lugar, se inicializa la variable dentro_de_seccion con el valor 0, lo que indica que no estamos dentro de la sección "[descripcion]".

Luego, el bucle while se ejecuta mientras se pueda leer una línea del archivo con la función fgets(). Cada línea se almacena en la variable linea (que se ha definido previamente).

Dentro del bucle, se realizan las siguientes comprobaciones:

if (strcmp(linea, "[descripcion]\n") == 0): Esta condición verifica si la línea leída coincide exactamente con el texto "[descripcion]" seguido de un salto de línea. Si la condición se cumple, se establece dentro_de_seccion en 1, lo que indica que estamos dentro de la sección "[descripcion]". La instrucción continue salta a la siguiente iteración del bucle, sin ejecutar el resto del código en el bloque del bucle.

else if (strcmp(linea, "\n") == 0): Esta condición verifica si la línea leída es un salto de línea vacío. Si se cumple, se establece dentro_de_seccion en 0, lo que indica que hemos salido de la sección "[descripcion]".

```
21  char dentro_de_seccion_descripcion = 0;
22  char dentro_de_seccion_consultas = 0;
23  while (fgets(linea, MAX_LINE_LENGTH, archivo) != NULL) {
24      // Verificar si estamos dentro de la sección "[descripcion]"
25      if (strcmp(linea, "[descripcion]\n") == 0) {
26          dentro_de_seccion_descripcion = 1;
27          dentro_de_seccion_consultas = 0;
28          continue;
29      }
30      // Verificar si estamos dentro de la sección "[consultas]"
31      else if (strcmp(linea, "[consultas]\n") == 0) {
32          dentro_de_seccion_descripcion = 0;
33          dentro_de_seccion_consultas = 1;
34          continue;
35      } else if (strcmp(linea, "\n") == 0) {
36          dentro_de_seccion_descripcion = 0;
37          dentro_de_seccion_consultas = 0;
38      }
```

En este caso, se agregó una nueva condición para verificar si estamos dentro de la sección "[consultas]". Si la línea coincide con "[consultas]\n", se establece

dentro_de_seccion_descripcion en 0 y dentro_de_seccion_consultas en 1. Esto significa que estamos dentro de la sección "[consultas]" y fuera de la sección "[descripcion]".

Además, se mantiene la condición para verificar si encontramos una línea vacía, lo que indica que estamos fuera de ambas secciones.

```
// Procesar los datos si estamos dentro de la sección
if (dentro_de_seccion) {
    int id;
    float masa, x_p, y_p, x_a, y_a, x_i, y_i;

    // Utilizar sscanf para separar los datos por comas
    sscanf(linea, "%d,%f,%f,%f,%f,%f,%f,%f", &id, &masa, &x_p, &y_p, &x
```

En esta sección condicional, se declaran varias variables: ID de tipo char array de tamaño 20, y las variables masa, x_p, y_p, x_a, y_a, x_i y y_i de tipo float.

Luego, se utiliza la función sscanf para analizar (parsear) el contenido de la variable linea según el formato especificado. El formato %[^,],%f,%f,%f,%f,%f,%f,%f indica que se espera leer una cadena de caracteres hasta encontrar una coma, seguida de 7 valores de tipo float separados por comas.

La función sscanf asigna los valores analizados a las variables correspondientes. En este caso, se asignan los valores a ID, masa, x_p, y_p, x_a, y_a, x_i y y_i en ese orden.

```
41      if (dentro_de_seccion_descripcion) {
42          char ID[20];
43          float masa, x_p, y_p, x_a, y_a, x_i, y_i;
44
45          sscanf(linea, "%[^,],%f,%f,%f,%f,%f,%f,%f", ID, &masa, &x_p, &y_p, &x_a, &y_a, &x_i,&y_i);
46
47          float afelio = calcularHipotenusa1(x_a, x_a);
48          float perihelio = calcularHipotenusa1(x_p, y_p);
49          float periodo = periodo_orbital(masa);
50          float semieje = semieje_mayor(perihelio, afelio);
51          float excen = excentricidad(perihelio, afelio);
52          float incli = inclinacion(y_i, x_i);
53
54
55      }
56
```

```
        if (dentro_de_seccion_consultas) {
            char ID[20], id2[20];
            float tiempo;

            sscanf(linea, "%[^,],%[^,],%f", ID, id2, &tiempo);
            float velo = velocidad(tiempo, periodo);
          // a es semieje mayor. I es inclinacion. v es velocidad. e es ecsentricidad
            printf("ID: %s\n", ID);
            float dis = distancia (semieje, incli, velo, excen);
            return dis;
        }


    }

    fclose(archivo);
    return 0;
}
```

También podemos usar la misma lógica para la sección [consultas].

Las variables semieje, incli, velo, excen se han declarado como variables globales para poder ser usadas en esta sección.

Funciones.h:

Funciones.h > ...

```h
#ifndef FUNCIONES_H
#define FUNCIONES_H

float calcularHipotenusa1(float cateto1, float cateto2);
float calcularHipotenusa2(float cateto1, float cateto2);
float periodo_orbital(float masa);
float semieje_mayor(float perihelio, float afelio);
float excentricidad (float perifelio, float afelio);
float inclinacion (float y_i, float x_i);
float velocidad (float t, float T);
float distancia (float i, float v, float e, float a);
#endif  // FUNCIONES_H

```

En el archivo "Funciones.h", se declaran los prototipos de las funciones.

Principales retos:

- El principal reto fue la investigación de las Leyes de Kepler, ya que ninguna página web cumplía con los requisitos. Y muchas mezclaban Kepler con Newton. En mi caso usé ciertos tópicos de la mecánica newtoniana para poder realizar los cálculos
- El segundo reto fue el manejo de tipo de datos. Llegué a borrar todo el código y volverlo a escribir porque no era capaz de encontrar el problema.
  Buscando códigos note que a la hora de imprimir los resultados cambiaba ciertas letras como: ("La posición en x es: %.2f\n", resultado).
  Para el caso anterior se usa %.2f para flotantes y %s para char. Cada tipo de dato se puede transformar usando este tipo de notación, y es una maravilla.
- El tercer reto fue la lectura de archivos. Anteriormente ya había utilizado chatGPT para que generase un código que lo hiciera. Sin embargo, la primera vez generó un código muy extraño, por no decir incomprensible.
  La segunda vez generó el que se usó, aun así, se tuvo que investigar sobre que decía el programa para modificarlo a nuestras necesidades, pero la investigación fue más sencilla.
- La cuarta podríamos decir que fue la impresión de “respuestas.txt”, pero quedó resuelta por consulta del profesor.
- El uso de “main.c”, “Funciones.h”, “Funciones.c” fue laborioso ya que no sabia que no era necesario incluir “Funciones.c” en “main.c”. Por lo que, al hacer lo antes descrito, las funciones me daban error.

Conclusiones:

La experiencia de desarrollar un código en C fue desafiante y enriquecedora. Investigar el modelo de Kepler por completo y condensarlo para poder ser ingresado en lenguaje C fue frustrante, y a su vez, satisfactoria. Ya que, a decir verdad, utiliza ciertos tópicos de física que pasamos por alto e incentivarnos a repasar estos conceptos le añade cierto interés.

Claro, se tienen que utilizar tópicos de programación que no se quedan atrás cuando se habla de desafíos. El hecho de digitar un trozo de código y este no funcione como lo acordado es desalentador y más cuando se le agrega otra fila que, a priori, encaja con lo razonado y esta devuelve un error. Sin embargo, ese hecho es el hace comprender los tópicos necesarios para la solución de ese error. El error, en lo personal, que más detesté fue el de tipo de datos. Siempre olvidaba cambiarlos o que “char” es diferente de “float” y por una letra o punto, el programa no compilaba.

En resumen, la experiencia de desarrollar un código en C proporcionó una amplia gama de lecciones valiosas, desde el entendimiento del lenguaje y la planificación adecuada, que varió mucho a lo largo del desarrollo, hasta la investigación de un tema en concreto para la resolución de un problema que lo involucre. Estas lecciones no solo son aplicables a futuros proyectos en C, sino que también se pueden transferir a otros lenguajes de programación.

Referencias:

Fernández, J. L. (s. f.). *Leyes de Kepler*. Fisicalab.

https://www.fisicalab.com/apartado/leyes-kepler

Leyes de Kepler. (s. f.). http://www.sc.ehu.es/sbweb/fisica/celeste/kepler/kepler.htm

Significados. (2020). Leyes de Kepler. *Significados*. https://www.significados.com/leyes-de-kepler/

OpenAI. (2023). ChatGPT (Modelo de lenguaje grande) [Software]. Recuperado de https://openai.com/chatgpt