Universidad Nacional De Loja

“Facultad la energía, las industrias y los recursos naturales no renovables”

Carrera de Computación

EVALUACIÓN:Segunda Unidad

INTEGRANTES:

- Anthony Yaguana
- Narcisa Pinzón
- Santiago Villamagua
- Camila Chimbo
- Darwin Correa

FECHA: 17/06/2024

DOCENTE: Ing. Edison Coronel

ASIGNATURA: Teoría de la programación

CICLO
Abril-Agosto
2024

Definición del problema:
Debido a que en el presente periodo académico se nos ha asignado crear una batería que se cargue con energía solar hemos decidido utilizar un panel fotovoltaico, pero, al momento de querer implementarlo nos damos cuenta de que el sol varía su ubicación constantemente y esto se debe al lugar en que lo coloquemos o debido a que el sol se mueve conforme transcurre el día , entonces este panel solar constantemente debe ajustar su posición automáticamente para así optimizar la captación de luz solar a lo largo del transcurso del día, de esta manera conseguiremos que nuestro cargador solar maximice su eficiencia.

Objetivo del algoritmo: Crear un algoritmo que nos permita calcular la altitud y orientación solar en la que se debe colocar el panel solar utilizando los siguientes datos: altitud y latitud de nuestra posición actual, la fecha y hora actual del sistema.

Análisis de variables implementadas en pseudocódigo
Para el presente algoritmo se han identificado varias variables que se utilizaran en el transcurso en el que se desarrolla el algoritmo.

Variables principales: Son variables necesarias para dar inicio al algoritmo.
-Lati_tud y Longi_tud: Las cuales son fáciles de obtener, simplemente hay que dirigirse a Google Maps ya sea desde una computadora o un celular y pinchar en la ubicación que se haya encontrado para que proporcione los datos correspondientes.
-Fecha y Hora actual: Estos datos que cambian constantemente se pueden calcular de una manera automática, solo se debe leer la fecha y hora del sistema para usarlas de forma conveniente.
Dentro de la fecha actual se tienen en cuenta variables como: dia, mes y anio.
Y en la hora actual sólo se utilizaron dos variables para guardar la hora actual y los minutos actuales de estas variables: hora y minutos.

Variables requeridas a lo largo del algoritmo: Estas variables se utilizaron para guardar los datos obtenidos de los diferentes cálculos realizados en el algoritmo.
-N: Guarda el número de días que han transcurrido desde que inició el año hasta el día actual.
-Declinacion: Guarda el valor calculado con la ecuación de la declinación solar por lo tanto es la declinación solar.
-EoT: Aquí se almacena el valor del tiempo solar verdadero calculado con la ecuación del tiempo
-TSV: Guarda el valor del tiempo solar verdadero calculado con su fórmula respectiva.
-hora_solar: Guarda solo la hora del TSV.
-minutos_solar: Aquí se almacena los minutos respectivos del TSV.
-H: Guarda el valor del ángulo horario calculado también con una fórmula.
-AS: Esta variable guarda el valor final de la altura del sol en radianes.
-AS_Grados: Es la variable que almacena la altura final del sol en grados.
-Azimut: Aquí se almacena la orientación solar en radianes.
-Azimut_Grados: Almacena el valor de la orientación solar en grados.

DIAGRAMA DE FLUJO GENERAL DEL ALGORITMO:

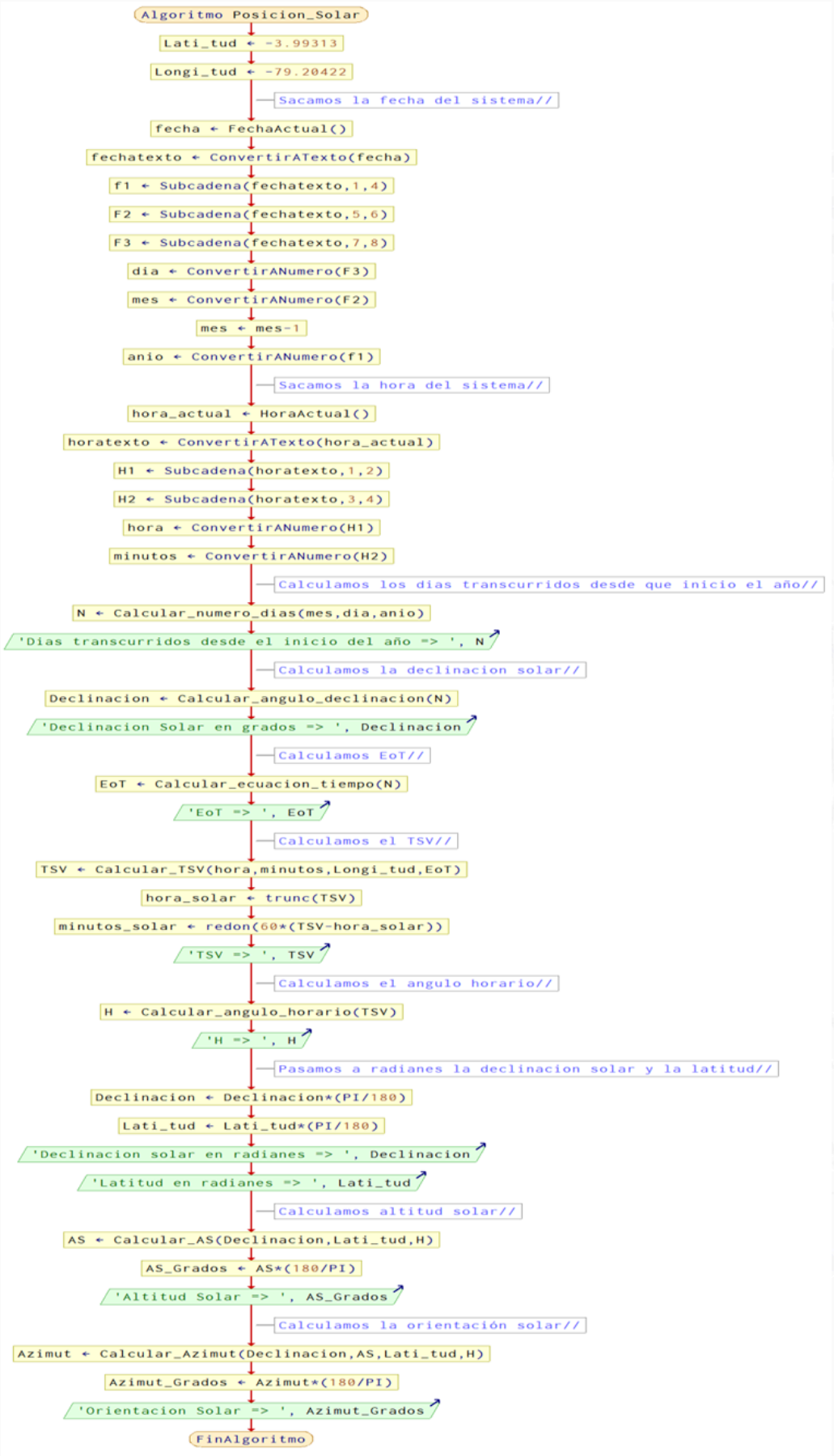


DIAGRAMA DE FLUJO DE LA FUNCIÓN PARA CALCULAR LOS DÍAS TRANSCURRIDOS DESDE EL INICIO DEL AÑO HASTA EL DIA ACTUAL:

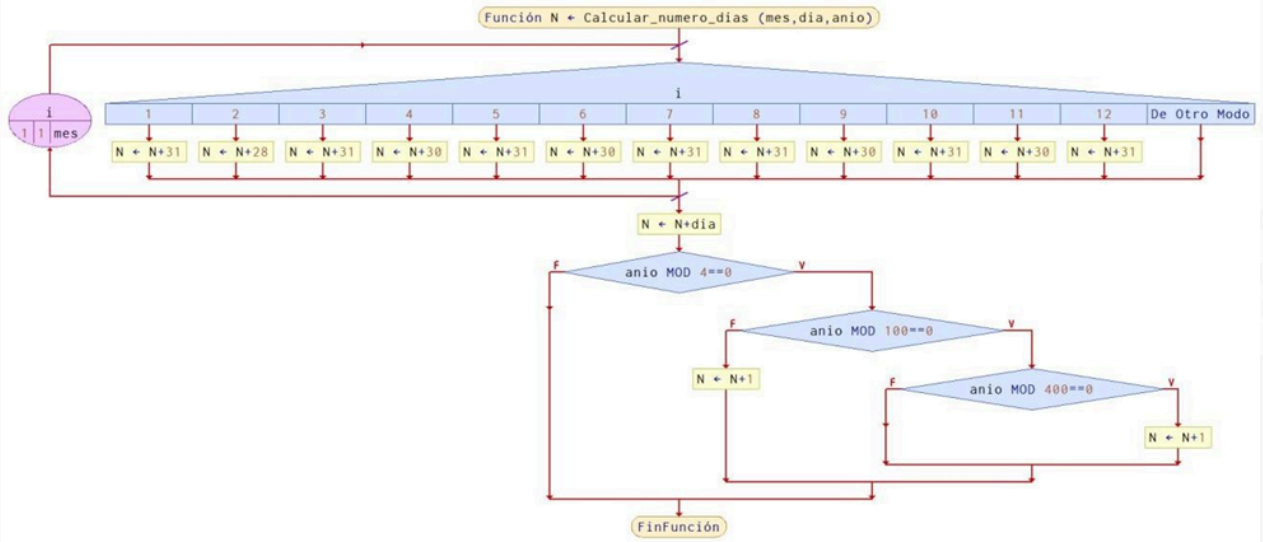


DIAGRAMA DE FLUJO DE LA FUNCIÓN PARA CALCULAR EL ÁNGULO DE DECLINACIÓN:

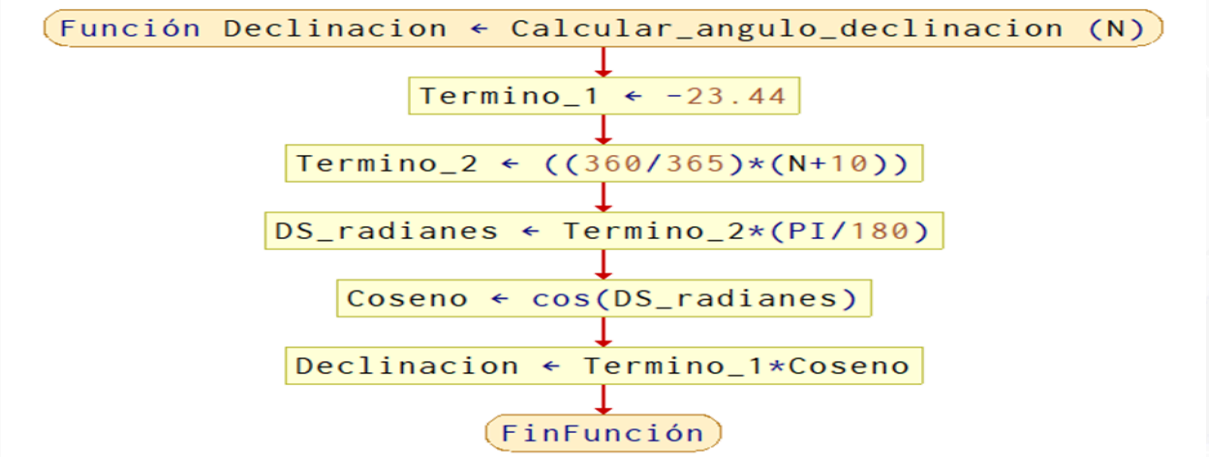


DIAGRAMA DE FLUJO DE LA FUNCIÓN DE LA ECUACIÓN DEL TIEMPO:

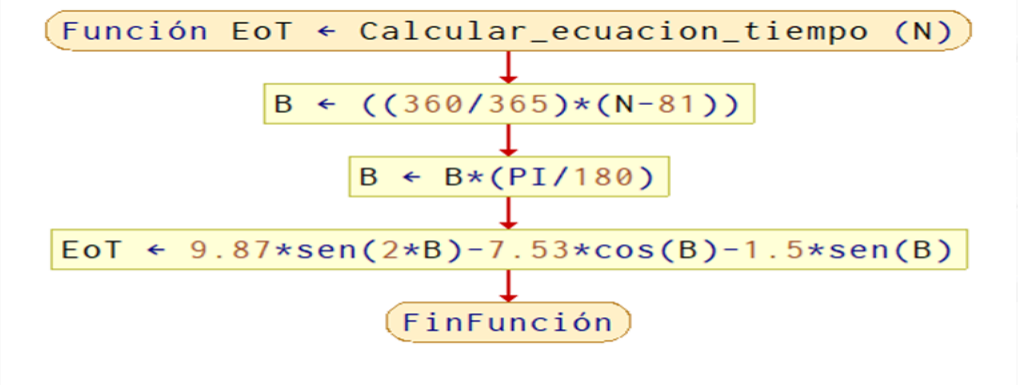


DIAGRAMA DE FLUJO DE LA FUNCIÓN PARA CALCULAR EL TSV:

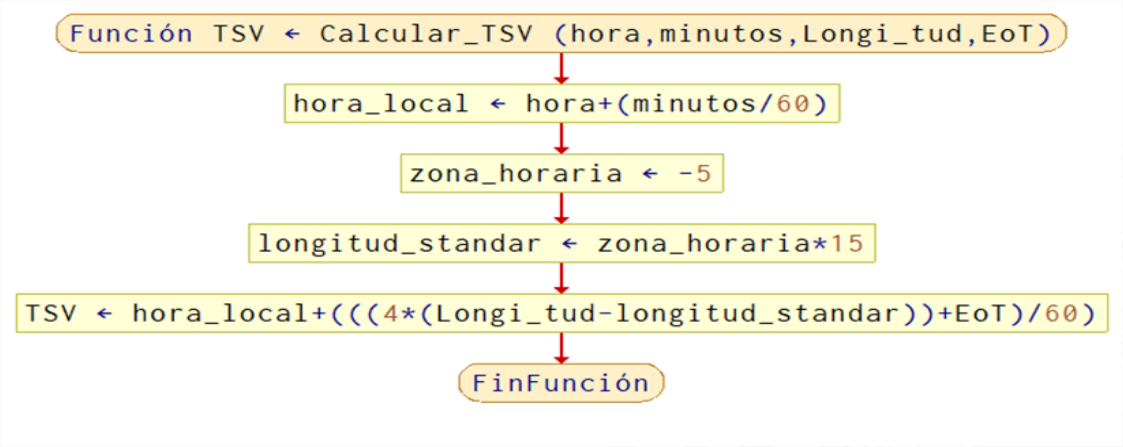


DIAGRAMA DE FLUJO DE LA FUNCIÓN PARA CALCULAR EL ÁNGULO HORARIO:

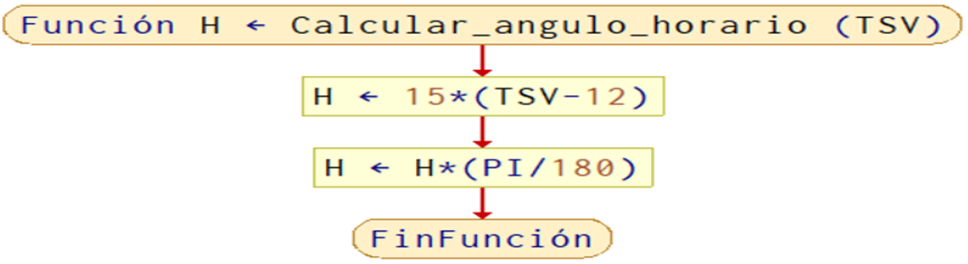


DIAGRAMA DE FLUJO DE LA FUNCIÓN PARA CALCULAR LA ALTITUD SOLAR:

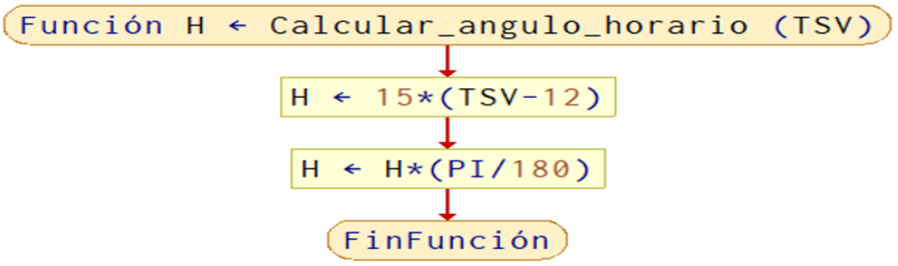
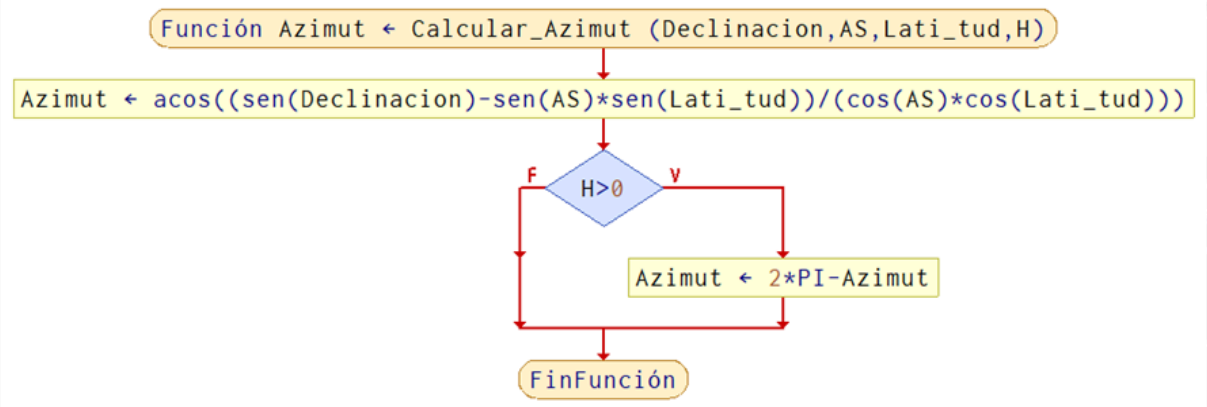


DIAGRAMA DE FLUJO DE LA FUNCIÓN PARA CALCULAR LA ORIENTACIÓN SOLAR:



Explicación del pseudocódigo del algoritmo: Este es el funcionamiento general del algoritmo para calcular la altitud solar y orientación solar en la que debe colocarse el panel fotovoltaico.
Primero se obtienen los datos necesarios para hacer los respectivos cálculos y así obtener los ángulos necesarios, primero se tienen en cuenta los datos de longitud y altitud, después la fecha actual y hora actual del sistema, separándolas en sus respectivas variables.
Con estos datos iniciales ya obtenidos se procede a hacer los cálculos con apoyó de las funciones antes ya definidas que se explicarán a continuación del documento.
Cabe recalcar que la forma en que se obtiene cada dato es secuencial, es decir, calculamos un dato necesario para después usarlo en otra ecuación y así obtener los datos necesarios.
Entonces los pasos del algoritmo son:

1.Definir la longitud y latitud del lugar en que se encuentra.

2.Recuperación de la fecha y hora actual del sistema.

3.Calcular el número de días que han transcurrido (N) desde que inició el año hasta el día actual usando las variables de mes, dia y anio.

4.Con los días totales calculados se procede a hacer el respectivo cálculo de la declinación solar (Declinación).

5.Hacer uso también de los días totales transcurridos (N) para proceder a resolver la ecuación del tiempo (EoT).

6.Calcular el tiempo solar verdadero (TSV), utilizando la hora actual (hora), los minutos actuales (minutos), la longitud de nuestra ubicación (Longi_tud) y el resultado de la ecuación del tiempo (EoT).

7.Hacer el cálculo de la hora solar (hora_solar) y los minutos solares (minutos_solar), para la hora solar utilizando la parte entera del tiempo solar verdadero (TSV) y para los minutos solares.

8.Hacer el cálculo del ángulo horario (H), con ayuda del resultado del tiempo solar verdadero (TSV).

9.Acontecer al cálculo de la declinación solar (Declinación) y la latitud (Lati_tud) a radianes usando la fórmula general de conversión a radianes que sería: variable * (PI/180).

10.Con todos los datos obtenidos anteriormente se procede a la realización de cada uno de los dos datos finales de nuestro algoritmo, la altitud solar (AS), que se denota como los grados de inclinación que debe tener nuestro panel solar obtenido con la declinación solar (Declinación), la latitud (Latitud) y el ángulo horario (H). Cabe recalcar que todos los datos usados para calcular la altitud solar deben estar en radianes.

11.Finalmente se realiza la obtención de la orientación solar (Azimut) que son los grados en que debe estar dirigido el panel fotovoltaico con respecto al norte, para calcular la orientación solar (Azimut) usamos los datos respectivos de la declinación solar (Declinación), la altitud solar (AS), la latitud con respecto a la ubicación actual (Latitud) y el ángulo horario (H). En este caso se utilizó todos los ángulos en radianes.

Pseudocódigo principal:

```
Algoritmo Posicion_Solar
    Lati_tud ← -3.99313
    Longi_tud ← -79.20422
    // Sacamos la fecha del sistema//
    fecha ← FechaActual()
    fechatexto ← ConvertirATexto(fecha)
    f1 ← Subcadena(fechatexto,1,4)
    F2 ← Subcadena(fechatexto,5,6)
    F3 ← Subcadena(fechatexto,7,8)
    dia ← ConvertirANumero(F3)
    mes ← ConvertirANumero(F2)
    mes ← mes-1
    anio ← ConvertirANumero(f1)
    // Sacamos la hora del sistema//
    hora_actual ← HoraActual()
    horatexto ← ConvertirATexto(hora_actual)
    H1 ← Subcadena(horatexto,1,2)
    H2 ← Subcadena(horatexto,3,4)
    hora ← ConvertirANumero(H1)
    minutos ← ConvertirANumero(H2)
    // Calculamos los dias transcurridos desde que inicio el año//
    N ← Calcular_numero_dias(mes,dia,anio)
    Escribir 'Dias transcurridos desde el inicio del año => ', N
    // Calculamos la declinacion solar//
    Declinacion ← Calcular_angulo_declinacion(N)
    Escribir 'Declinacion Solar en grados => ', Declinacion
    // Calculamos EoT//
    EoT ← Calcular_ecuacion_tiempo(N)
    Escribir 'EoT => ', EoT
    // Calculamos el TSV//
    TSV ← Calcular_TSV(hora,minutos,Longi_tud,EoT)
    hora_solar ← trunc(TSV)
    minutos_solar ← redon(60*(TSV-hora_solar))
    Escribir 'TSV => ', TSV
    // Calculamos el angulo horario//
    H ← Calcular_angulo_horario(TSV)
    Escribir 'H => ', H
    // Pasamos a radianes la declinacion solar y la latitud//
    Declinacion ← Declinacion*(PI/180)
    Lati_tud ← Lati_tud*(PI/180)
    Escribir 'Declinacion solar en radianes => ', Declinacion
    Escribir 'Latitud en radianes => ', Lati_tud
    // Calculamos altitud solar//
    AS ← Calcular_AS(Declinacion,Lati_tud,H)
    AS_Grados ← AS*(180/PI)
    Escribir 'Altitud Solar => ', AS_Grados
    // Calculamos la orientación solar//
    Azimut ← Calcular_Azimut(Declinacion,AS,Lati_tud,H)
    Azimut_Grados ← Azimut*(180/PI)
    Escribir 'Orientacion Solar => ', Azimut_Grados
FinAlgoritmo
```

Ahora se procede a detallar las respectivas funciones que se definen en el algoritmo
Función Calcular_numero_dias (N): En esta función se pasan como parámetros el mes actual (mes), el día actual(dia) y el año actual (anio), tomando un bucle “Para” y un “Según” para hacer una iteración desde el mes 1 que sería enero hasta un mes anterior al actual esto debido a que el mes actual no está totalmente cursado para determinar el número de mes y por cada número de mes se suma la cantidad de días que tiene cada mes almacenando esta cantidad de días en la variable (N).
Al finalizar se realiza un análisis para verificar si el año actual es un año bisiesto, de ser así se suma un 1 dia a la cantidad de días calculado anteriormente ya que los años bisiestos cuentan con un dia mas que los años normales.

```
Funcion N ← Calcular_numero_dias (mes, dia, anio)
    Para i←1 Hasta mes Con Paso 1 Hacer
        Segun i Hacer
            1:
                N = N + 31
            2:
                N = N + 28
            3:
                N = N + 31
            4:
                N = N + 30
            5:
                N = N + 31
            6:
                N = N + 30
            7:
                N = N + 31
            8:
                N = N + 31
            9:
                N = N + 30
            10:
                N = N + 31
            11:
                N = N + 30
            12:
                N = N + 31
        Fin Segun
    Fin Para
    N = N + dia
    si anio % 4 == 0 Entonces
        si anio % 100 == 0 Entonces
            si anio % 400 == 0 Entonces
                N = N + 1
            Fin si
        SiNo
            N = N + 1
        FinSi

    FinSi
Fin Funcion
```

Función Calcular_angulo_declinación (Declinacion): Esta función recibe como parámetro el número de días (N) que han transcurrido desde que inició el año en el mes de enero hasta el día actual mediante l formula:

$$\delta = -23.45^{\circ} \times \cos\left(\frac{360}{365} \times (d + 10)\right)$$

[imagen 1]

A esta fórmula se le complementan ciertos parámetros que se manifiestan de la siguiente manera:

Término _1: Este valor se debe a que es el ángulo de inclinación máximo de la tierra respecto al plano de su órbita alrededor del sol.

Término_2: Se establece una división de 360° entre el número de días del año multiplicado al número de días que han pasado más 10, para compensar cierto tiempo que se pudo haber perdido

Finalmente, al realizar estos cálculos se obtiene el ángulo de la declinación solar para el día N.

```
Funcion Declinacion ← Calcular_angulo_declinacion  ( N )
    Termino_1 = -23.44
    Termino_2 = ((360/365) * (N + 10))
    DS_radianes = Termino_2 * (PI/180)
    Coseno = cos(DS_radianes)
    Declinacion = Termino_1 * Coseno
Fin Funcion
```

Función Calcular_ecuacion_tiempo (EoT): Para llevar a cabo este cálculo se tiene como referencia la Ecuación del Tiempo (EoT) en minutos denotada como una ecuación que corrige la excentricidad de la órbita de la tierra y la inclinación del eje de la tierra donde:

$$EoT = 9.87 \cdot \sin(2B) - 7.53 \cdot \cos(B) - 1.5 \cdot \sin(B)$$

[imagen 2]

Esta fórmula se utiliza para calcular el ángulo del horario del sol restando al día 81 para ajustar el día del año para que se centre alrededor del solsticio de invierno.

$$B = \frac{360}{365}(d - 81)$$

Luego se convierte el ángulo del horario del sol de grados a radianes multiplicando (B * PI/180) y finalmente se utiliza el ángulo calculado para determinar la Ecuación del tiempo

```
Funcion EoT ← Calcular_ecuacion_tiempo ( N )
    B = ((360/365) * (N - 81))
    B = B * (PI/180)
    EoT = 9.87 * sen(2 * B) -7.53 * cos(B) -1.5 * sen(B)
Fin Funcion
```

Función Calcular_TSV (TSV): Esta función se encarga de calcular la hora solar verdadera (TSV) en función de la hora local, longitud, zona horaria y ecuación del tiempo (EoT).

$$TSV = \text{Hora local} + \frac{4 \cdot (\text{Longitud local} - \text{Longitud estándar}) + EoT}{60}$$

[imagen 3]

```
Funcion TSV ← Calcular_TSV (hora, minutos, Longi_tud, EoT)
    hora_local = hora + (minutos/60)
    zona_horaria = -5
    longitud_standar = zona_horaria * 15
    TSV = hora_local + (((4*(Longi_tud - longitud_standar)) + EoT)/60)
Fin Funcion
```

Función Calcular_angulo_horario (H): Pasamos como parámetros el tiempo solar verdadero (TSV) calculado en la anterior función, para calcular este ángulo horario nos ayudamos de la fórmula:

$$H = 15^\circ \times (TSV - 12)$$

[imagen 4]

Luego convertimos en radianes con la fórmula general de conversión a radianes para así poder trabajar este dato adecuadamente.

```
Funcion H ← Calcular_angulo_horario (TSV)
    H = 15 * (TSV - 12)
    H = H * (PI/180)
Fin Funcion
```

Función Calcular_AS: En esta función se usan como parámetros los valores anteriormente calculados, para hacer uso de la declinación solar (Declinacion), la latitud de nuestra ubicación actual (Lati_tud) y el ángulo horario (H).

Con los parámetros mencionados se procede a utilizar la siguiente fórmula para poder calcular la altura solar (AS):

De esta manera obtenemos la inclinación correspondiente del panel solar (AS).

$$\alpha = \sin^{-1}(\sin(\delta) \cdot \sin(\phi) + \cos(\delta) \cdot \cos(\phi) \cdot \cos(H))$$

Función Calcular_Azimut: Con esta función se procede a calcular la orientación (Azimut) que debe tener nuestro panel solar con respecto al norte, pasando los parámetros correspondientes que serían: la declinación solar (Declinacion), la altitud solar (AS), la latitud de nuestra ubicación actual (Lati_tud) y el ángulo horario (H). De la misma manera que en las anteriores funciones, tomando la fórmula correspondiente para calcular el azimut:

$$\text{Azimut} = \cos^{-1}\left(\frac{\sin(\delta) - \sin(\alpha) \cdot \sin(\phi)}{\cos(\alpha) \cdot \cos(\phi)}\right)$$

[imagen 5]

```
Funcion Azimut ← Calcular_Azimut (Declinacion, AS, Lati_tud, H)
    Azimut = acos((sen(Declinacion) - sen(AS) * sen(Lati_tud)) / (cos(AS) * cos(Lati_tud)))
    si H > 0 Entonces
        Azimut =2 * PI - Azimut
    FinSi
Fin Funcion
```

Implementación del algoritmo empleado en el lenguaje de programación C.

El programa realizado a continuación tiene como objetivo optimizar la orientación de un panel solar para maximizar la captación de energía solar. La captación eficiente de energía solar depende de la orientación precisa del panel respecto al sol en cualquier momento del día y del año.

Inicio del Programa en C.

Definición de bibliotecas y constantes utilizadas en el código:

```c
#include <stdio.h>
#include <math.h>
#include <time.h>
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif
```

<stdio.h> .- Librería empleada para la entrada y salida de datos.
<math.h>.- Librería utilizada para resolver funciones matemáticas.
<time.h>.- Librería para obtener la fecha y hora en en tiempo real.
#define .- Se implementa para tomar una variable como constante en el programa.

Inicialización de la función encargada de calcular la fecha actual usando tres variables como enteros(int meses, int dias, int anios).

```
int Calcular_Numero_Dias(int meses, int dias, int anios)
{
    int N_Dias = 0;
    for (int i = 1; i <= meses; i++)
    {
        switch (i)
        {
        case 1:
            N_Dias = N_Dias + 31;
            break;
        case 2:
            N_Dias = N_Dias + 28;
            break;
        case 3:
            N_Dias = N_Dias + 31;
            break;
        case 4:
            N_Dias = N_Dias + 30;
            break;
        case 5:
            N_Dias = N_Dias + 31;
            break;
        case 6:
            N_Dias = N_Dias + 30;
            break;
        case 7:
            N_Dias = N_Dias + 31;
            break;
        case 8:
            N_Dias = N_Dias + 31;
            break;
        case 9:
            N_Dias = N_Dias + 30;
            break;
        case 10:
            N_Dias = N_Dias + 31;
            break;
        case 11:
            N_Dias = N_Dias + 30;
            break;
        case 12:
            N_Dias = N_Dias + 31;
            break;
        }
    }
    N_Dias = N_Dias + dias;
    if (anios % 4 == 0)
    {
        if (anios % 100 == 0)
        {
            if (anios % 400 == 0)
            {
                N_Dias = N_Dias + 1;
            }
        }
        else
        {
            N_Dias = N_Dias + 1;
        }
    }
    return N_Dias;
```

Inicialización del contador de días a 0 en la variable de (N_Dias).
Se inicializa el **Bucle For** el cual se encarga de:

- Recorrer los meses desde el 1 hasta el valor de `meses`.
- En cada iteración del bucle, se utiliza una estructura `switch` para sumar los días correspondientes al mes actual (`i`).
- Dependiendo del valor de `i`, se añaden los días correspondientes al mes en `N_Dias`.

Suma de días del mes actual:

- `N_Dias = N_Dias + dias;` suma los días del mes actual a `N_Dias`.

Luego dentro del mismo Bucle se incorpora un ajuste para saber si un año es o no bisiesto:

- El primer `if` verifica si el año (`anios`) es divisible por 4.
- El segundo `if` verifica si el año es divisible por 100.
- El tercer `if` verifica si el año es divisible por 400.
- Si el año es divisible por 4 pero no por 100, o es divisible por 400, se considera bisiesto y se suma un día adicional (29 de febrero).

Retorno del total de días:

- `return N_Dias;` devuelve el total de días calculados.

Función con una variable (double) esta variable maneja decimales esta función nos ayudará para calcular el ángulo de Declinación incluyendo la variable (int N_Dias).

Esta función calcula la declinación solar para un día específico del año, representado por (**`N_Dias`**), considerando el efecto de la inclinación del eje terrestre y la variación anual del ángulo solar.

```
double Calcular_Angulo_Declinacion_Solar(int N_Dias)
{
    double Termino1, Termino2, DS_Radianes, Coseno, Declinacion_Solar;
    Termino1 = -23.44;
    Termino2 = (360.0 / 365.0) * (N_Dias + 10);
    DS_Radianes = Termino2 * (M_PI / 180.0);
    Coseno = cos(DS_Radianes);
    Declinacion_Solar = Termino1 * Coseno;
    return Declinacion_Solar;
}
```

Inicialización de las variables a usar como double en este caso: **(double Termino1, Termino2, DS_Radianes, Coseno, Declinacion_Solar).**

- `Termino1` se establece en -23.44. Este valor representa la máxima declinación solar en grados.
- `Termino2` se calcula como la fracción del año transcurrida más 10 días, convertida a grados. La fórmula `(360.0 / 365.0) * (N_Dias + 10)` convierte el número de días en una fracción del año a grados y ajusta el cálculo sumando 10 días para compensar el desfase entre el inicio del año y el punto en que la declinación solar es cero.

Conversión de `Termino2` de grados a radianes:

- `Termino2` se convierte de grados a radianes multiplicándose por `(M_PI / 180.0)`, ya que las funciones trigonométricas en C utilizan radianes.

Cálculo del coseno del ángulo en radianes (`DS_Radianes`):

- El coseno del ángulo `DS_Radianes` se calcula usando la función `cos()`. Esto proporciona el coseno del ángulo correspondiente al día del año en radianes.

Cálculo de la declinación solar:

- La declinación solar se calcula multiplicando `Termino1` (-23.44) por el coseno del ángulo (`Coseno`). Este valor representa la declinación solar en grados para el día especificado.

Retorno del valor calculado:

- La función devuelve el valor de la declinación solar, que es el ángulo de declinación solar en grados para el número de días transcurridos en el año (`N_Dias`)

Función con una variable (double) esta variable maneja decimales, esta función nos ayudará para calcular la Ecuación del tiempo incluyendo la variable (int N_Dias).

Esta función calcula la ecuación del tiempo para un día específico del año, teniendo en cuenta las variaciones debidas a la excentricidad de la órbita de la Tierra y la inclinación de su eje. .

```
double Ecuacion_Del_Tiempo(int N_Dias)
{
    double B, EoT;

    B = ((360.0 / 365.0) * (N_Dias - 81));
    B = B * (M_PI / 180.0);
    EoT = 9.87 * sin(2 * B) - 7.53 * cos(B) - 1.5 * sin(B);
    return EoT;
}
```

Definición de la función y declaración de variables:

- La función `Ecuacion_Del_Tiempo` toma un entero `N_Dias` que representa el día del año.
- Se declaran dos variables de tipo `double`: `B` y `EoT`.

Cálculo del valor de B en grados:

- B se calcula usando la fórmula `((360.0 / 365.0) * (N_Dias - 81))`.
- Esta fórmula convierte el número de días transcurridos en una fracción del año en grados, restando 81 para ajustar el cálculo al solsticio de primavera, cuando la ecuación del tiempo es aproximadamente cero.

Conversión de B de grados a radianes:

- B se convierte de grados a radianes multiplicándose por `(M_PI / 180.0)`.
- Esto se debe a que las funciones trigonométricas en C utilizan radianes en lugar de grados.

Cálculo de la ecuación del tiempo (EoT):

- EoT se calcula usando la fórmula `9.87 * sin(2 * B) - 7.53 * cos(B) - 1.5 * sin(B)`.
- Esta fórmula combina varias funciones trigonométricas (seno y coseno) para modelar la variación de la ecuación del tiempo a lo largo del año. La ecuación del tiempo mide la diferencia entre el tiempo solar aparente y el tiempo solar medio, que varía debido a la excentricidad de la órbita terrestre y la inclinación del eje terrestre.

Retorno del valor calculado:

- La función devuelve el valor de `EoT`, que es la ecuación del tiempo en minutos para el día del año especificado por `N_Dias`.

Función con una variable (double) esta variable maneja decimales esta función nos ayudará para calcular el Tiempo Solar. Esta función permitirá ajustar la orientación de los paneles solares de manera más precisa y eficiente, siguiendo el movimiento del sol en el cielo a lo largo del día.

```
double Calcular_Tiempo_Solar_Verdadero(int horas, int minutos, double Longitud, double EoT)
{
    int Zona_Horaria;
    double Hora_Local, Longitud_Estandar, TSV;

    Hora_Local = horas + (minutos / 60.0);
    Zona_Horaria = -5;
    Longitud_Estandar = Zona_Horaria * 15;
    TSV = Hora_Local + (((4 * (Longitud - Longitud_Estandar)) + EoT) / 60.0);
    return TSV;
}
```

Conversión de la Hora Local a Formato Decimal:

- La hora local, compuesta por horas y minutos, se convierte en un valor decimal. Por ejemplo, 10 horas y 30 minutos se convierten en 10.5 horas.

Definición de la Zona Horaria:

- Se define la zona horaria de la ubicación en relación al tiempo universal coordinado (UTC). Por ejemplo, si la zona horaria es UTC-5, el valor será -5.

Cálculo de la Longitud Estándar de la Zona Horaria:

- La longitud estándar de la zona horaria se calcula multiplicando el valor de la zona horaria por 15 grados. Esto se debe a que cada zona horaria cubre aproximadamente 15 grados de longitud (360 grados divididos por 24 horas).

Cálculo del Tiempo Solar Verdadero (TSV):

- Se calcula la diferencia entre la longitud geográfica de la ubicación específica y la longitud estándar de la zona horaria.
- Esta diferencia en longitud se convierte en minutos, ya que la Tierra gira aproximadamente 1 grado cada 4 minutos.
- La ecuación del tiempo (EoT) se añade a este valor en minutos para ajustar el desfase debido a la órbita elíptica de la Tierra y su inclinación axial.

- El total se convierte de minutos a horas y se añade a la hora local en formato decimal para obtener el tiempo solar verdadero.

Retorno del Tiempo Solar Verdadero:

- El tiempo solar verdadero en horas decimales se obtiene, representando el tiempo que marca un reloj de sol en la ubicación específica y momento dados.

Función con una variable (double) esta variable maneja decimales esta función nos ayudará para calcular el Ángulo .

Esta función calcula el ángulo horario se utiliza para ajustar la inclinación de los paneles de manera que estén orientados hacia el sol para captar la máxima cantidad de radiación solar.

```
double Calcular_Angulo_Horario(double TSV)
{
    double Angulo_Horario;
    Angulo_Horario = 15 * (TSV - 12);
    Angulo_Horario = Angulo_Horario * (M_PI / 180.0);
    return Angulo_Horario;
}
```

Definición de la Función y Declaración de Variables:

- La función `Calcular_Angulo_Horario` toma como entrada TSV, que es el tiempo solar verdadero en horas.

Cálculo del Ángulo Horario:

- `Angulo_Horario = 15 * (TSV - 12);`

  Se calcula el ángulo horario multiplicando 15 grados por la diferencia entre el tiempo solar verdadero (TSV) y 12 horas. Esto ajusta el ángulo para que sea cero al mediodía solar y positivo o negativo dependiendo de si es antes o después del mediodía.

Conversión a Radianes:

- `Angulo_Horario = Angulo_Horario * (M_PI / 180.0);`

  Se convierte el ángulo horario de grados a radianes, ya que muchas funciones trigonométricas en C utilizan radianes como unidad de medida.

Retorno del Valor Calculado:

- La función devuelve `Angulo_Horario`, que es el ángulo horario en radianes para el tiempo solar verdadero dado.

Función con una variable (double) esta variable maneja decimales esta función nos ayudará para calcular el Altitud.

Con esta función podremos obtener la altitud solar que se utiliza para calcular la intensidad de la radiación solar incidente en los paneles y ajustar la inclinación de estos paneles para maximizar la captación de energía solar.

```
double Calcular_Altitud_Solar(double Declinacion_Solar, double Latitud, double Angulo_Horario)
{
    double Altitud_Solar;
    Altitud_Solar = asin(sin(Declinacion_Solar) * sin(Latitud) + cos(Declinacion_Solar) * cos(Latitud) * cos(Angulo_Horario));
    return Altitud_Solar;
}
```

Definición de la Función y Declaración de Variables:

- La función `Calcular_Altitud_Solar` toma tres parámetros como entrada:
- `Declinacion_Solar`: El ángulo de declinación solar en radianes.
- `Latitud`: La latitud geográfica del lugar en radianes.
- `Angulo_Horario`: El ángulo horario del sol en radianes.

Cálculo de la Altitud Solar:

- `Altitud_Solar = asin(sin(Declinacion_Solar) * sin(Latitud) + cos(Declinacion_Solar) * cos(Latitud) * cos(Angulo_Horario));`

Se utiliza la función `asin()` para calcular el arcoseno, que representa el ángulo de elevación del sol sobre el horizonte.

La fórmula dentro de `asin()` calcula la suma de dos componentes:

1. `sin(Declinacion_Solar) * sin(Latitud)`: Este término representa la componente vertical debido a la inclinación del eje de la Tierra (declinación solar) y la latitud del lugar.
2. `cos(Declinacion_Solar) * cos(Latitud) * cos(Angulo_Horario)`: Este término representa la componente horizontal, que depende del ángulo horario del sol.

Retorno del Valor Calculado:

- La función devuelve `Altitud_Solar`, que es el ángulo de elevación del sol sobre el horizonte en radianes.

Función con una variable (double) esta variable maneja decimales esta función nos ayudará para calcular la Azimut.

En sistemas de seguimiento solar, el azimut solar se utiliza para ajustar continuamente la orientación de los paneles solares a lo largo del día y del año.

```
double Calcular_Azimut(double Declinacion_Solar, double Altitud_Solar, double Latitud, double Angulo_Horario)
{
    double Azimut;
    Azimut = acos((sin(Declinacion_Solar) - sin(Altitud_Solar) * sin(Latitud)) / (cos(Altitud_Solar) * cos(Latitud)));
    if (Angulo_Horario > 0)
    {
        Azimut = 2 * M_PI - Azimut;
    }
    return Azimut;
}
```

Definición de la Función y Declaración de Variables:

- La función `Calcular_Azimut` toma cuatro parámetros como entrada:

  `Declinacion_Solar`: El ángulo de declinación solar en radianes.

  `Altitud_Solar`: El ángulo de elevación del sol sobre el horizonte en radianes.

  `Latitud`: La latitud geográfica del lugar en radianes.

  `Angulo_Horario`: El ángulo horario del sol en radianes.

Cálculo del Azimut Solar:

- `Azimut = acos((sin(Declinacion_Solar) - sin(Altitud_Solar) * sin(Latitud)) / (cos(Altitud_Solar) * cos(Latitud)));`

    Se utiliza la función `acos()` para calcular el arcocoseno, que representa el ángulo de azimut solar.

    La fórmula dentro de `acos()` calcula el coseno del ángulo de azimut utilizando los siguientes componentes:

        `sin(Declinacion_Solar)`: Componente vertical debido a la inclinación del eje de la Tierra (declinación solar).

        `sin(Altitud_Solar) * sin(Latitud)`: Componente horizontal que depende de la altitud solar y la latitud del lugar.

        `cos(Altitud_Solar) * cos(Latitud)`: Factor de normalización para convertir las coordenadas a un sistema cartesiano.

Ajuste del Azimut según el Ángulo Horario:

- `if (Angulo_Horario > 0)`

    Si el ángulo horario es positivo (es decir, si es después del mediodía solar), se ajusta el azimut agregando `2 * M_PI` (360 grados en radianes) y restando el valor calculado. Esto se hace para obtener el azimut en el rango correcto de 0 a 2 * M_PI (0 a 360 grados).

Retorno del Valor Calculado:

- La función devuelve `Azimut`, que es el ángulo de azimut solar en radianes.

Presentación y llamado de funciones en el (main)

```c
int main()
{
    // Definimos todas las variables necesarias para nuestro programa //
    int horas, minutos, segundos;
    int anios, meses, dias;
    int N_Dias;
    double Longitud, Latitud;
    double Declinacion_Solar, EoT, TSV, Angulo_Horario, Altitud_Solar, Orientacion_Solar, Azimut, Hora_Solar, Minutos_Solar;
    /* Recuperamos la fecha y hora actual del sistema con ayuda de la libreria "time.h" y metemos cada uno de los datos en su respectiva
    variable*/
    time_t tiempoahora;
    time(&tiempoahora);
    struct tm *mitiempo = localtime(&tiempoahora);
    horas = mitiempo->tm_hour;
    minutos = mitiempo->tm_min;
    segundos = mitiempo->tm_sec;
    anios = mitiempo->tm_year + 1900;
    meses = mitiempo->tm_mon;
    dias = mitiempo->tm_mday;
    printf("=====================================================\n");
    printf("// FECHA Y HORA ACTUAL DEL SISTEMA //\n");
    printf("Horas => %d\n", horas);
    printf("Minutos => %d\n", minutos);
    printf("Segundos => %d\n", segundos);
    printf("Anio => %d\n", anios);
    printf("Mes => %d\n", meses);
    printf("Dia => %d\n", dias);
```

Variables utilizadas:

- int horas, minutos, segundos.
- int anios, meses, dias.
- int N_Dias.
- double Longitud, Latitud.
- double Declinacion_Solar, EoT, TSV, Angulo_Horario, Altitud_Solar, Orientacion_Solar, Azimut, Hora_Solar, Minutos_Solar.

Obtención de Fecha y Hora exactas.

Declaración de variable:

time_t tiempoahora.

Nos ayuda a almacenar la Fecha y Hora como un número entero.

Obtencion de Fecha y Hora actuales:

time(&tiempoahora).

Lama a la función `time()` de la librería estándar `time.h`, que obtiene la cantidad de segundos desde el 1 de enero de 1970 y la guarda en la variable `tiempoahora`.

Conversión a tiempo local.

struct tm *mitiempo = localtime(&tiempoahora);

`localtime(&tiempoahora)`; es una función que toma el valor de `time_t` almacenado en `tiempoahora` y lo convierte en una estructura `tm` que contiene los componentes de fecha y hora (como horas, minutos, segundos, año, mes, día, etc.) en la zona horaria local.

Asignación de componentes:

- horas = mitiempo->tm_hour;
- minutos = mitiempo->tm_min;
- segundos = mitiempo->tm_sec;

Estas son las variables de la estructura `tm` que representan la hora, minutos y segundos actuales del sistema, respectivamente.

- anios = mitiempo->tm_year + 1900;

Contiene el número de años transcurridos desde 1900. Al sumarle 1900, obtenemos el año actual.

- meses = mitiempo->tm_mon;

Representa el mes actual, dónde enero es 0, febrero es 1, y así sucesivamente.

- dias = mitiempo->tm_mday;

Es el día del mes actual.

En resumen, este código obtiene la fecha y hora actuales del sistema y almacena cada componente (horas, minutos, segundos, año, mes, día) en variables separadas para su posterior uso en el programa.

Llamado de funciones y presentación en pantalla.

```
    Latitud = -3.99313;
    Longitud = -79.28422;

    // Iniciamos el Programa llamando y asando los valores a cada una de las funciones antes definidas //
    printf("=====================================================\n");
    printf("// VERIFICACION DE DATOS //\n");
    N_Dias = Calcular_Numero_Dias(meses, dias, anios);
    printf("> Dias transcurridos desde que inicio el año => %d\n", N_Dias);

    // Calculamos la Declinacion Solar //
    Declinacion_Solar = Calcular_Angulo_Declinacion_Solar(N_Dias);
    printf("> Declinacion Solar => %f\n", Declinacion_Solar);

    // Calculamos la ecuacion del tiempo (EoT) //
    EoT = Ecuacion_Del_Tiempo(N_Dias);
    printf("> Ecuacion del tiempo => %f\n", EoT);

    // Calculamos el tiempo solar verdadero //
    TSV = Calcular_Tiempo_Solar_Verdadero(horas, minutos, Longitud, EoT);
    Hora_Solar = trunc(TSV);
    Minutos_Solar = round(60 * (TSV - Hora_Solar));
    printf("> Tiempor solar verdadero => %f\n", TSV);

    // Calculamos el angulo horario //
    Angulo_Horario = Calcular_Angulo_Horario(TSV);
    printf("> Angulo Horario => %f\n", Angulo_Horario);

    // Pasamos la Declinacion Solar y la Latitud de grados a radianes //
    Declinacion_Solar = Declinacion_Solar * (M_PI / 180.0);
    Latitud = Latitud * (M_PI / 180.0);
    printf("> Declinacion Solar en radianes => %f\n", Declinacion_Solar);
    printf("> Latitud en radianes => %f\n", Latitud);

    // Calculamos la Altitud Solar en radianes //
    Altitud_Solar = Calcular_Altitud_Solar(Declinacion_Solar, Latitud, Angulo_Horario);
    // Pasamos la Altitud Solar a grados //
    Altitud_Solar = Altitud_Solar * (180.0 / M_PI);

    // Calculamos la orientacion Solar o Azimut en radianes//
    Azimut = Calcular_Azimut(Declinacion_Solar, Altitud_Solar, Latitud, Angulo_Horario);
    // Pasamos la orientacion solar a grados //
    Azimut = Azimut * (180.0 / M_PI);

    printf("=====================================================\n");
    printf("// RESULTADOS FINALES DE ALTITUD DEL SOL Y SU ORIENTACION //\n");
    printf("> La posicion de la altitud del sol es de => %f grados de elevacion desde el piso.\n", Altitud_Solar);
    printf("> La posicion de la orientación del sol es de => %f grados desde el norte.\n", Azimut);
    printf("=====================================================\n");
    return 0;
}
```

Discusión para la implementación del código en un sistema de paneles solares reales.

El software utilizado para realizar el panel solar puede extenderse ampliamente beneficiando a todo aquel que lo necesite y sea un amante de los recursos naturales que ofrece la naturaleza como fuente de energía propiciando al ahorro de la energía eléctrica, además no solo puede un panel solar puede tener múltiples usos, empezando como un cargador solar hasta como un implemento de alumbrado en las viviendas, como aire acondicionado o inclusive como una fuente de energía para duchas de agua caliente, el software puede ser modificado para infinidad de usos y no solo destinarse a uno solo. Pero para lograrlo debemos considerar algunos aspectos como:

- **Interfaz con hardware**: El código proporcionado debe comunicarse eficientemente con los controladores de los paneles solares.
- **Gestión de datos**: Debe manejar los datos de manera eficiente y segura. Esto incluye el almacenamiento local en el sistema de control y la transmisión segura de datos a través de redes.
- **Control y monitoreo en tiempo real**: El sistema debe ser capaz de monitorear constantemente el rendimiento de los paneles, la energía generada, y otros parámetros relevantes.
- **Seguridad y redundancia**: Se deben implementar medidas de seguridad robustas para proteger el sistema contra accesos no autorizados y para garantizar que el sistema pueda manejar situaciones de emergencia de manera adecuada.
- **Optimización de eficiencia**: El código debe ser optimizado para minimizar el consumo de recursos computacionales y energéticos, asegurando que el sistema funcione de manera eficiente tanto en términos de hardware como de energía.

Mejoras futuras:

- **Integración con IoT y sistemas de energía inteligente:** Utilizar IoT para una integración más completa con otros dispositivos del hogar y para permitir la gestión inteligente de la energía generada y consumida.
- **Optimización del sistema de almacenamiento de energía:** Mejorar los algoritmos de gestión de la batería para optimizar la carga y descarga, considerando patrones de consumo y generación.
- **Automatización avanzada:** Implementar lógica de control avanzada para ajustar automáticamente la orientación de los paneles solares y optimizar el seguimiento del sol.
- **Monitoreo remoto y gestión de mantenimiento predictivo:** Utilizar análisis de datos avanzados para detectar anomalías en el rendimiento antes de que causen problemas mayores, permitiendo intervenciones de mantenimiento predictivo.

Referencias:

[1] Tutorialspoint. (s.f.). C Programming - Tutorialspoint. Recuperado de https://www.tutorialspoint.com/cprogramming/index.htm

[2] PV Education. (s.f.). Ángulo de Declinación Solar. Recuperado de https://www.pveducation.org/es/fotovoltaica/2-propiedades-de-la-luz-del-sol/angulo-de-declinación

[3] Algoritmos. (s.f.). Tiempo Solar. Recuperado de https://algoritmos9511.gitlab.io/_downloads/e1ac04d57c11925f0283040c533417bb/tiempo.pdf

[4] Universidad Santo Tomás, "Título del documento," en *Repositorio Institucional de la Universidad Santo Tomás*, año. [En línea]. Disponible en: https://repository.usta.edu.co/handle/11634/16519. [Consultado: 16 junio 2024].

imagen 2: https://susdesign.com/popups/sunangle/eot.php

imagen 3: https://www.e-education.psu.edu/eme810/node/531

imagen 4: https://solarsena.com/solar-hour-angle-calculator-formula/

imagen 5: https://www.certicalia.com/blog/como-calcular-azimut