FUNDAMENTOS DE PROGRAMACIÓN

Curso 2019-2020

Práctica 4

Juan Sebastian Flor Usma

Ingeniería Informática

Departamento de
Ingeniería de Software y
Sistemas Informáticos

Enunciado

"Realizar un programa en C+/- para gestionar las rutas mensuales de una compañía de autobuses. Las distintas rutas se realizarán entre una lista de pueblos que, para simplificar el programa, será un tipo enumerado con los nombres de cada pueblo ordenados alfabéticamente. Por ejemplo:

```
typedef enum Pueblos { Buitrago, Cabrera, Lozoya, Lozoyuela,
Pedraza, Rascafria, Riaza, Robregordo, Sepulveda, Torrelaguna,
Venturada }
```

El tipo enumerado deberá tener al menos 10 pueblos y sus nombres deberán ser elegidos por cada alumno para realizar su práctica."

Los pueblos de mi elección son los siguientes:

```
typedef enum Pueblos {
                Alpedrete, Escorial, Galapagar, Hoyo, Majadahonda,
                Moralzarzal, Navacerrada, Rozas, Torrelodones, Villalba
          }
```

"Cada ruta tendrá un pueblo origen y un máximo de 5 paradas incluyendo el pueblo destino final. La información de cada ruta será el pueblo origen y entre 1 y 5 tramos, según el número de paradas intermedias.

La información de cada tramo será: el pueblo de parada, la duración del tramo y el precio del tramo. Los tramos intermedios serán consecutivos y tendrán como origen la anterior parada o el origen de la ruta. El pueblo del último tramo será el destino final de la ruta.

Se podrán gestionar un máximo de 10 rutas (identificadas como rutaA, rutaB,…, rutaJ). Cada ruta sólo se podrá programar una vez en el mismo día."

Para conseguir esto definimos un *struct TipoRuta*:

```
typedef struct TipoRuta{
  Pueblos origenRuta;
  Pueblos destinoRuta;
  TipoTramos tramos;
  TipoFechas fechas;
  int numeroTramos;
  int numeroFechas;
};
```

a
l tendrá los siguientes atributos: **origenRuta** (del tipo enum Pueblos), **destinoRuta** (del tipo enum Pueblos), **tramos** (del tipo vector TipoTramos), **fechas** (del tipo vector TipoFechas), **numeroTramos** (tipo int) y **numeroFechas** (tipo int).

OrigenRuta y **destinoRuta** solo podrán tener valores del tipo enum Pueblos.
TipoTramos es un vector de **5 TipoTramo**, de los cuales cada **TipoTramo** tendrá un **origen** (Pueblos), **destino** (Pueblos) y una **duración** y **precio** (ambos int) tal que:

```
const int MAX_Tramos = 5;

typedef struct TipoTramo{
        Pueblos origen;
        Pueblos destino;
        int duracion;
        int precio;
};

typedef TipoTramo TipoTramos[MAX_Tramos];
```

TipoFechas es un vector de **10 TipoFecha**, de los cuales cada **TipoFecha** tendrá **día**, **mes**, **año** y **horaSalidaOrigen** (Como no estaba especificado la cantidad de fechas a programar he elegido 10):

```
const int MAX_Fechas = 10;

typedef struct TipoFecha{
        int dia;
        int mes;
        int anno;
        int horaSalidaOrigen;
};
```

o
Tramos y **numeroFechas** nos serán útiles para llevar la cuenta de cuantos tramos o cuantas fechas tiene programadas cada ruta

"Las operaciones del programa serán las siguientes:

- ✓ *Editar ruta*
- ✓ *Programar calendario de ruta*
- ✓ *Listar datos de ruta y calendario*
- ✓ *Calendario mensual de trayecto*
- ✓ *Información de viaje*

La práctica consiste en realizar el TAD "GestionRutasMes", el programa principal y las correspondientes funciones, procedimientos y TADs que el alumno crea conveniente. En la realización de esta práctica se debe reutilizar en la medida de lo posible el código fuente ya realizado para la tercera práctica, que se redefinirá como un TAD "CalendarioMes"."

Según los requerimientos del enunciado mi TAD **GestionRutaMes** es el siguiente:

```
const int MAX_Rutas = 10;

typedef TipoRuta TipoRutas[MAX_Rutas];

typedef struct GestionRutasMes{
    TipoRutas rutas;
    int numeroRutas;

    void init();
    void EditarRuta();
    void ProgramarCalendarioRuta();
    void ListarDiasRutaCalendario();
    void CalendarioMensualTrayecto();
    void InformacionViaje();

    private:
    void ImprimirPueblo(Pueblos pueblo);
    int IndiceRuta(char ruta);
};
```

Que tendrá los atributos **rutas** (del TipoRutas, que es un vector de 10 TipoRuta), **numeroRutas** (int contador del numero de rutas con itinerario) y los métodos **init** (inicializara el programa), **EditarRuta**, **ProgramarCalendarioRuta**, **ListarDiasRutaCalendario**, **CalendarioMensualTrayecto**, **InformacionViaje** y los métodos auxiliares *ImprimirPueblo* (switch que recibe un argumentos del tipo Pueblos y lo imprime por consola) e *IndiceRuta* (switch que recibe un argumento del tipo char que deduce que indice del array de rutas le corresponde)

"El programa principal deberá presentar las siguientes opciones:"

Gestión de Rutas

Editar ruta	**(Pulsar E)**
Programar calendario de ruta	**(Pulsar P)**
Listar datos de ruta y calendario	**(Pulsar L)**
Calendario mensual de trayecto	**(Pulsar C)**
Información de viaje	**(Pulsar I)**
Salir	**(Pulsar S)**

Teclear una opción válida (E\|P\|L\|C\|I\|S)?

Para el menú de opciones me he apoyado en el método init() de mi TAD GestionRutasMes, declaro una variable tecla de tipo char, inicializo numeroRutas en 0 y con un bucle do while podre manejar las opciones del usuario, mediante un switch puedo ejecutar la opcion correspondiente a la tecla pulsada.

"1.- La opción "Editar ruta", tendrá el siguiente formato:"

Editar ruta entre los pueblos:

1 : Buitrago	**2 : Cabrera**	**3 : Lozoya**
4 : Lozoyuela	**5 : Pedraza**	**6 : Rascafria**
7 : Riaza	**8 : Robregordo**	**9: Sepulveda**
10 : Torrelaguna	**11: Venturada**	

Ruta a editar (entre A y J)? F
Pueblo origen? 10
Número de tramos? 4 €

Pueblo Parada 1? 11
Duración tramo 1 en minutos? 15
Precio tramo 1? 5 €

Pueblo parada 2? 4
Duración tramo 2 en minutos? 20
Precio tramo 2? 7 €

Pueblo Parada 3? 1
Duración tramo 3 en minutos? 10
Precio tramo 3? 4 €

Pueblo parada 4? 7
Duración tramo 4 en minutos? 35
Precio tramo 4? 12 €

En EditarRuta para conseguir una experiencia de usuario similar a la de la imagen aportada en el enunciado nos apoyamos en bucles do while con printf y scanf dentro. Cuando el usuario introduzca la ruta (char) llamamos al método IndiceRuta(char ruta) que nos devolverá el índice (int) de la ruta seleccionada respecto al array de rutas y lo guardaremos en la variable posicionRuta. Gracias a esta variable cuando el usuario introduzca pueblo de origen lo podemos asignar de esta manera ***rutas[posicionRuta].origenRuta = pueblo.*** De la misma manera asignamos el numero de tramos y el origen, destino, duración y precio de cada tramo de la ruta elegida. Para finalizar incrementaremos en 1 el valor de nuestra variable numeroRutas e imprimiremos por pantalla lo que hemos programado

"2.- La opción "Programar calendario de ruta" tendrá el siguiente formato:"

Programar calendario de ruta:

Ruta a programar (entre A y J)? F
Día? 9
Mes? 5
Año? 2020
Hora salida de origen? 11
Continuar programando (S/N)? S
Día? 13
Mes? 5
Año? 2020
Hora salida de origen? 16
Continuar programando (S/N)? N

Esta opción la realizaremos el mismo procedimiento para sacar la posicionRuta que nos servirá para programar fechas de la ruta seleccionada. Con otro bucle do while permitiremos al usuario seguir programando (en caso de que asi lo desee), dentro del cual tendremos un try catch (recordamos que cada ruta tenemos un contador de fechas ***numeroFechas*** que incrementaremos cada vez que añadamos una fecha), el cual si de la ruta elegida tiene su atributo numeroFechas > 0 comprobara que las nuevas fechas elegidas no esten repetidas, en caso de estar repetidas lanzara una excepción no dejando guardar las fechas repetidas (ya que cada ruta solo se podrá programar una vez en el mismo dia).

"3.- La opción de "Listar datos de ruta y calendario" tendrá el siguiente formato:

Listar datos de ruta y calendario:

Ruta a listar (entre A y J)? F
Mes? 5
Año? 2020

Listando a continuación el itinerario de la ruta, las fechas y horas programadas en el mes y año elegidos con el siguiente formato:"

La ruta F realiza el itinerario:

Torrelaguna
Venturada
Lozoyuela
Buitrago
Riaza

En las siguientes fechas y horas de mayo - 2020:

02 mayo. Salida de origen: 11 horas
06 mayo. Salida de origen: 16 horas
09 mayo. Salida de origen: 11 horas
13 mayo. Salida de origen: 16 horas
16 mayo. Salida de origen: 11 horas
20 mayo. Salida de origen: 16 horas
23 mayo. Salida de origen: 11 horas
27 mayo. Salida de origen: 16 horas
30 mayo. Salida de origen: 11 horas

En este método sigo el mismo enfoque anterior de obtención de datos por parte del usuario siempre haciendo hincapié en saber el índice de la ruta elegida. A la hora de imprimir el itinerario me ayudo del ***numeroTramos*** de la ruta para saber si hay itinerarios programados o no, en caso de no haberlos enseñaría el correspondiente mensaje por consola. En cuanto a las fechas me ayudo del ***numeroFechas*** de la ruta, si no es > 0 imprimo el correspondiente error por consola (no hay fechas programadas para la ruta), si lo es uso un try catch, dentro del cual una variable ***auxImpresion*** (int contador inicializada a 0) y un bucle for que recorrerá todas las fechas de la ruta que tendrá dentro un if que evaluara si de las fechas elegidas coincide alguna fecha programada, en caso afirmativo la imprimirá e incrementara el contador auxiliar. Después del bucle si el auxiliar de impresión sigue a 0 lanzara una excepción para informar al usuario por consola que no coinciden las fechas, así pues tenemos controlados los posibles casos de error .

“4.- La opción de “Calendario mensual de trayecto” tendrá el siguiente formato:

Calendario mensual de trayecto entre los pueblos:

1 : Buitrago	***2 : Cabrera***	***3 : Lozoya***
4 : Lozoyuela	***5 : Pedraza***	***6 : Rascafria***
7 : Riaza	***8 : Robregordo***	***9: Sepulveda***
10 : Torrelaguna	***11: Venturada***	

Origen del trayecto? 11
Destino del trayecto? 7
Mes del trayecto? 5
Año del trayecto? 2020

que mostrará el calendario programado para el trayecto solicitado con el siguiente formato:

Trayecto entre Venturada y Riaza

Mayo **2020**

L	***M***	***M***	***J***	***V***	***S***	***D***
				--	***02***	***--***
--	***--***	***06***	***--***	***--***	***09***	***--***
--	***--***	***13***	***--***	***--***	***16***	***--***
--	***--***	***20***	***--***	***--***	***23***	***--***
--	***--***	***27***	***--***	***--***	***30***	***--***

Indicando mediante “--“ los días que no hay disponible el trayecto solicitado”

Aqui nos apoyamos en el calendario creado en la practica anterior como dice el enunciado, el cual será el TAD CalendarioMes que tendrá un atributo ***diasRuta*** del tipo ***DiasRuta*** y dos métodos ***ImprimirMes*** (recibe como argumento un mes int,que usaremos de apoyo en varias partes de la aplicación cuando necesitemos imprimir un mes por pantalla) e ***ImprimirCalendario*** (recibe mes y anno del tipo int y diasRuta del tipo DiasRuta e imprime solo los días que hayan en el array diasRuta en el calendario)

```
                    Const int MAX_DiasMes = 31;

                    typedef int DiasRuta[MAX_DiasMes];

                    typedef struct CalendarioMes{
                       DiasRuta diasRuta;
                       void ImprimirCalendario(int Mes, int Anno,
    DiasRuta diasRuta);
                       void ImprimirMes(int mes);
                    };
```

Primero que todo instanciamos el calendario e inicializamos el vector diasRutas asignando en todas sus posiciones 0.
Si el numero de rutas no es mayor que 0 dara el error por consola, si lo es la ejecución del programa entrara en un try catch dentro del cual habrá un bucle for comprobando en primer lugar si hay alguna ruta con el itinerario que coincida con el origen seleccionado y el destino (en caso negativo tendremos ***auxOrigenDestino*** inicializado a 0, solo se incrementara si hay alguna coincidencia, si después del bucle sigue a 0 se lanzara una excepción para decirle al usario por consola que no hay coincidencias en origen y destino seleccionado) y en segundo lugar se comprueba si la ruta que haya pasado la comprobación de origen destino tiene fechas programadas (***numeroFechas***), si las tiene comprobaremos si alguna de esas fechas programadas coincide en el mes y año seleccionado (en caso afirmativo se imprime el calendario con los días y se incrementa otro auxiliar ***auxMesAnno***, en caso negativo si después de estas comprobaciones este auxiliar sigue a 0 lanzaremos el segundo tipo de excepción que contemplamos en este modulo, la cual informara al usuario de que no hay fechas que coincidan). En caso de impresión del calendario, antes en cada fecha que coincida con los requisitos de mes y año sacaremos su dia y lo asignaremos en diasRuta a la posición dia – 1, asi luego el calendario sabra que dia imprimir según las posiciones de diasRutas que esten rellenas con números que coincidan en el bucle de impresión del calendario, por lo que los que sean = 0 imprimiran --

“ 5.- La opción de “Información de viaje” tendrá el siguiente formato:

Información de viaje entre los pueblos:

1 : Buitrago	***2 : Cabrera***	***3 : Lozoya***
4 : Lozoyuela	***5 : Pedraza***	***6 : Rascafria***
7 : Riaza	***8 : Robregordo***	***9: Sepulveda***
10 : Torrelaguna	***11: Venturada***	

Origen del viaje? 11
Destino del viaje? 1
Día del viaje? 9
Mes del viaje? 5
Año del viaje? 2020

Como resultado, mostrará la siguiente información del viaje solicitado:

Información de viaje:
Fecha: 9 mayo 2020
Origen: Venturada
Salida: 11 horas y 15 minutos
Destino: Buitrago
Llegada: 11 horas y 45 minutos
Precio: 11 euros

Si hay varias rutas en el mismo día entre los pueblos solicitados, se listarán todas las alternativas disponibles con su horario y precio. En el caso de que no sea posible realizar el viaje solicitado, se deberá dar el correspondiente mensaje."

Siguiendo el enfoque de antes pediremos los datos al usuario y comprobaremos si hay rutas con ***numeroRutas,*** en caso negativo mostraremos el error por consola, en caso afirmativo recorreremos las rutas en busca de coincidencias de origen, destino, mes, año y dia. Siendo fiel el enfoque de dar información detallada al usuario de los errores para una mejor experiencia de usuario informaremos cuando no haya coincidencias en origen-destino y en segundo lugar en fechas. De haber coincidencias se imprimirían todas las posibles.

Ayuda corrección

Para ayudar a la correccion del programa en el método init inicializamos las rutas J e I. Las dos tendrán el mismo origen y destino, la J será programada para las fechas 15 y 25 de febrero de 2020 y la ruta I para los días 5 y 15 del mismo mes y año, lo cual pretende facilitar la corrección del modulo de Información de viaje (ya que este requiere que si existe algún otro itinerario con las fechas seleccionadas se imprima tambien) y para el modulo de calendario (ya que al tener el mismo origen y destino y tener fechas para el mismo mes y año se imprimirán conjuntamente en el calendario)

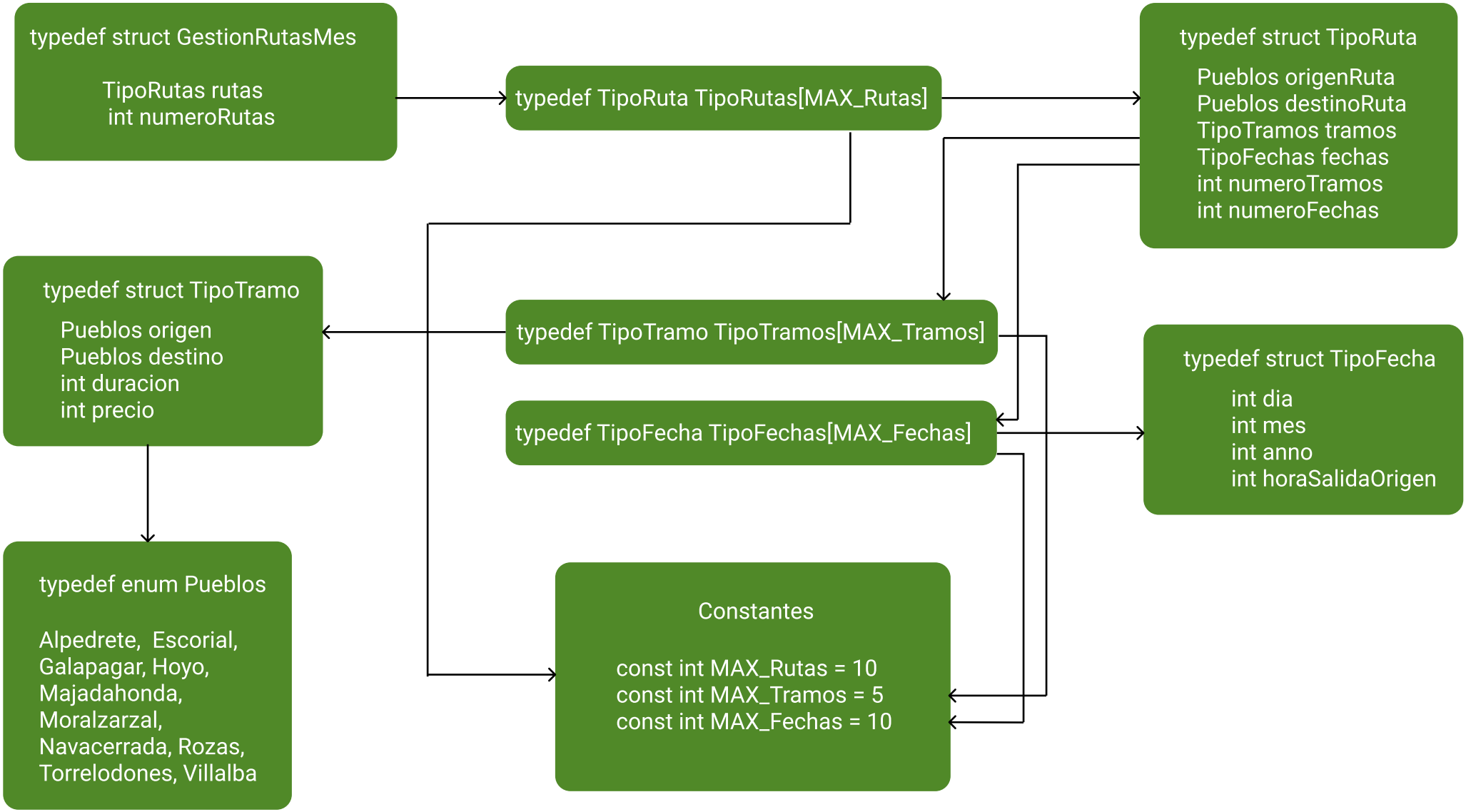
Repositorio

El código de la aplicación también estará disponible online en la siguiente direccion:

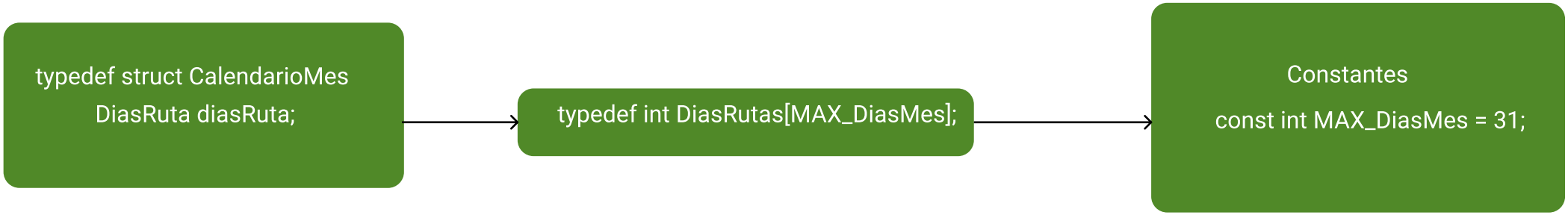
https://github.com/sebastianflor/practica4

Diseño TADs y diagrama flujo del programa

TAD GestionRutasMes



TAD CalendarioMes



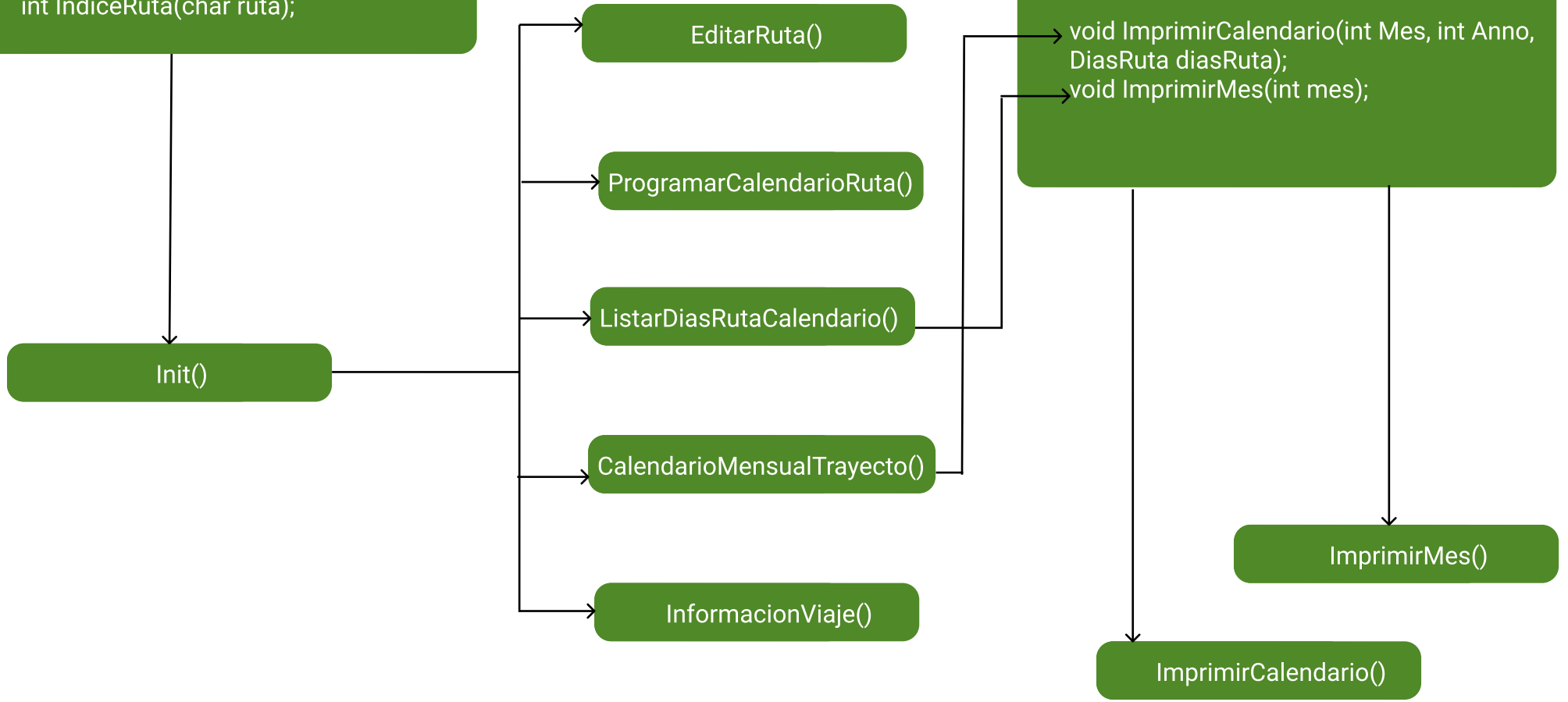
TAD GestionRutasMes / TAD CalendarioMes

typedef struct GestionRutasMes

void init();
void EditarRuta();
void ProgramarCalendarioRuta();
void ListarDiasRutaCalendario();
void CalendarioMensualTrayecto();
void InformacionViaje();
void ImprimirPueblo(Pueblos pueblo);
int IndiceRuta(char ruta);

typedef struct CalendarioMes

void ImprimirCalendario(int Mes, int Anno, DiasRuta diasRuta);
void ImprimirMes(int mes);



Flujo Informacion