Fundamentos de Programación

PEC8 - 20181

Fecha límite de entrega: 26/11/2018

Estudiante

Apellidos: de Lima B. Campos

Nombre: Lidiane A.

Objetivos

- Comprender el tipo de datos tabla y saber definirlo correctamente.
- Implementar recorridos y búsquedas dentro de las tablas.
- Profundizar en la modularización del código utilizando acciones y funciones
- Profundizar en el uso de parámetros de entrada, parámetros de salida, y parámetros de entrada / salida.

Formato y fecha de entrega

Es necesario entregar la PEC antes del día **26 de noviembre de 2018 a las 23:59**.

Para la entrega se deberá adjuntar un archivo en formato ZIP, que contenga:

- Este documento con la respuesta del ejercicio 1, y capturas de pantalla de las pruebas del ejercicio 2.
- **El workspace de Codelite** que contenga el proyecto y todas las carpetas creadas en el ejercicio 2

La entrega debe realizarse en el apartado de entregas de EC del aula de teoría.

.

Enunciado

Siguiendo con la ayuda que proporcionamos al UOCRailways, la compañía nos ha pedido nuestra colaboración para crear un programa que les ayude a gestionar las bases de datos de sus trenes. En los siguientes ejercicios, trabajaremos con el tipo de datos tabla conjuntamente con la entrada y salida interactiva para gestionar los datos de los trenes.

Nota: Puede darse el caso que los datos que se piden de los trenes varíen ligeramente respecto las PEC anteriores.

Disponemos de los siguientes tipos de datos y constantes ya definidos en lenguaje algorítmico:

Lenguaje algorítmico

```
const
        MAX_TRAINS: integer = 100;
        NO_TRAIN: integer = 9999;
end const
type
                tPropulsion = {ELECTRICAL, CARBON, SOLAR, GASOLINE, DIESEL,
MAGNETIC}
                tTrain = record
                id: integer;
                brand: string;
                model: string;
                year: integer;
                propulsionType: tPropulsion;
                wagonCapacity: real;
                nWagons: integer;
                end record
end type
```

Ejercicio 1: Diseño en lenguaje algorítmico (50%)

a) **Tabla de trenes** (5%). Defina el tipo de datos *tTrainsTable* (en forma de tabla), para guardar los datos de varios trenes. Los datos de cada tren, se guardan dentro de los campos del tipo estructurado *tTrain.* El número máximo de trenes está delimitado por la constante *MAX_TRAINS.*

b) **Inicialización de la tabla de trenes** (5%). Defina la acción *trainsTable_init,* que recibe como parámetro de entrada/salida una tabla de tipo *tTrainsTable,* e inicialice el número de elementos a cero.

c) **Búsqueda en la tabla de trenes** (20%).Defina la función *trainsTable_find,* que recibe como parámetros de entrada:
   - una tabla de tipo *tTrainsTable*
   - un entero con el identificador de un tren

   Si el identificador de tren indicado en el segundo parámetro existe dentro de la tabla de trenes, la función devuelve su posición dentro de la tabla. En caso contrario, devuelve el código de error 9999.

d) **Filtro de trenes por propulsión** (20%). Defina la acción *trainsTable_filterByPropulsion,* que recibe como parámetros:
   - una tabla de trenes tipo *tTrainsTable* (entrada)
   - un tipo de propulsión del tipo *tPropulsion* (entrada)
   - una tabla de trenes tipo *tTrainsTable* donde añadir los trenes que cumplen el filtro (salida)

   Los trenes de la tabla que tengan el tipo de propulsión indicado en el segundo parámetro se tienen que añadir a la tabla de salida. Con los que tengan un tipo de propulsión diferente no hay que hacer nada.

   Nota: Recuerde que hay que inicializar la tabla de salida usando la acción *trainsTable_init* del apartado b.

   Acciones y funciones que podemos utilizar sin implementar

```
{ Acción que copia el contenido del tren trainOrig a trainDest }
action train_cpy(in trainOrig:tTrain, out trainDest:tTrain)
```

Ejercicio 2: Programación en C (50%)

a) Codificación en C (40%)

En este ejercicio hay que codificar en C el ejercicio 1 y además estructurar el código en carpetas y ficheros, de forma similar a la PEC7. Concretamente:

1. Descomprimir el archivo *TrainsTable_20181.zip* que incluye los archivos de proyecto. El proyecto está estructurado en carpetas: carpeta *include* donde se encuentra el fichero ***train.h***, y carpeta *src* donde están los ficheros ***train.c*** y ***main.c.***
2. Dentro del archivo **train.h,** defina el tipo de datos *tTrainsTable* (ejercicio1.a).
3. En el archivo **train.h,** defina las cabeceras de las acciones/funciones *trainsTable_init, trainsTable_find, trainsTable_filterByPropulsion* del (ejercicio 1.b, ejercicio 1.c y ejercicio 1.d)
4. Dentro del archivo **train.c,** implemente la acción *trainsTable_init* (ejercicio 1.b)

5. Dentro del archivo **train.c,** implemente la función *trainsTable_find* (ejercicio 1.c)
6. Dentro del archivo **train.c,** implemente la acción *trainsTable_filterByPropulsion* (ejercicio 1.d)
7. Dentro el fichero **main.c,** implemente el siguiente código:
    a. Declarar una tabla de trenes de tipo *tTrainsTable*.
    b. Inicializar la tabla de trenes a cero.
    c. Solicitar al usuario que introduzca un número de trenes a registrar, y actualizar con este valor el número real de trenes guardados en la tabla.
    d. Leer los datos de todos los trenes a registrar y guardarlos en la tabla de tipos *tTrainsTable.*
    e. Solicitar al usuario un identificador de tren.
    f. Si dentro de la tabla existe algún tren con este identificador, escribir por pantalla un mensaje indicando la posición en la que se encuentra el tren con el identificador indicado [si hubiera más de un tren con el mismo identificador, hay que mostrar la posición del primer tren encontrado].
    g. Si el tren no existe, mostrar por pantalla un mensaje de error.
    h. Declarar una nueva tabla de trenes *(filterResult*) de tipo *tTrainsTable.*
    i. Solicitar al usuario un tipo de propulsión.
    j. Filtrar la tabla original por este tipo de propulsión (usando la acción *trainsTable_filterByPropulsion*) y guardar los resultados en la nueva tabla *filterResult*.
    k. Si la tabla *filterResult* no tiene ningún tren mostrar un mensaje por pantalla indicando que no hay ningún tren con el tipo de propulsión pedido.
    l. Si la tabla *filterResult* sí que tiene trenes, escribir el número de trenes y a continuación los datos de cada uno de los trenes por pantalla.
8. Comprobar el funcionamiento del programa, probando con diferentes datos, diferentes identificadores y con diferentes tipos de propulsión.

b) Pruebas (10%)

Como en las anteriores PEC se pide mostrar el funcionamiento del algoritmo **haciendo dos capturas** de pantalla donde se vea el funcionamiento de vuestro programa. Las capturas de pantalla las tenéis que adjuntar en el documento donde están las respuestas del ejercicio 1.

Criterios de corrección:

En el ejercicio 1:

- Que se siga la notación algorítmica utilizada en la asignatura. Véase documento *Nomenclator* en la xWiki de contenido.
- Que se sigan las instrucciones dadas y el algoritmo responda al problema planteado.
- Que se diseñen y se llamen correctamente las acciones y funciones pedidas.

En el ejercicio 2:

- Que el programa se adecue a las indicaciones dadas.
- Que el programa compile y funcione de acuerdo con lo que se pide.
- Que se respeten los criterios de estilo de programación C. Ver la *Guía de estilo de programación en C* que tenéis en la Wiki de contenido.
- Que se implemente correctamente la modularización del proyecto, dividiendo el código en carpetas y poniendo lo que corresponda en cada carpeta.

Solución

Lenguaje algorítmico

```
const
        MAX_TRAINS: integer = 100;
        NO_TRAIN: integer = 9999;
end const
type
                tPropulsion = {ELECTRICAL, CARBON, SOLAR, GASOLINE, DIESEL,
MAGNETIC}
                tTrain = record
                id: integer;
                brand: string;
                model: string;
                year: integer;
                propulsionType: tPropulsion;
                wagonCapacity: real;
                nWagons: integer;
                end record

        tTrainsTable=record
                  trains:vector[MAX_TRAINS] of tTrain;
                  numTrains: integer;
         end record
end type

action trainsTable_init(inout trainTable: tTrainsTable)
trainTable.numTrains:=0;
endaction
```

```
function trainsTable_find(trainsTable: tTrainsTable, train: tTrain):  integer
 var
   posicion: integer;
   y: integer;
 end var
   posicion: =  NO_TRAIN;
    y: = 1;

    while (i <=trainsTable.numTrains and posicion = NO_TRAIN) do
          if (trainsTable.trains[i].id = train.id) then
          posicion:= y;
          end if
     y:= i + 1;
   end while
return posicion;
end function



action trainsTable_filterByPropulsion(in trainsTable: tTrainsTable, in train: tTrain, out
newtrainsTable: tTrainsTable)

 var
   posicion: integer;
  end var
   posicion: =  NO_TRAIN;
 trainsTable_init(newtrainsTable);

    while (i <=trainsTable.numTrains and posicion <> NO_TRAIN) do
          if (trainsTable.trains[i].propulsionType = train.propulsionType) then
              train_cpy(trainsTable.trains[i], newtrainsTable.trains[i]);
           end if
     i:= i + 1;
   end while
return
end function
```