Universitat Oberta de Catalunya uoc.edu

Fundamentos de Programación
PEC5 - 20181

Fecha tope de entrega: 05/11/2018

Estudiante
Apellidos: de Lima Barbeiro Campos
Nombre: Lidiane Aparecida

Objetivos

- Saber definir correctamente los tipos de datos estructurados.
- Aprender a utilizar la entrada y salida para modificar campos concretos de estructuras de datos sencillos.
- Saber utilizar las funciones de conversión de tipo cuando haga falta.

Formato y fecha de entrega

El plazo límite para la entrega de la PEC es el día **5 de noviembre de 2018 a las 23:59**.

Para la entrega tenéis que entregár un fichero en formato ZIP, que contenga:

- Este documento con la respuesta del ejercicio 1 y capturas de pantalla de las pruebas del ejercicio 2.
- Un proyecto Codelite que contenga los ficheros .c pedidos al ejercicio 2

Hay que hacer la entrega en el apartado de entregas de AC del aula de teoría.

Enunciado

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

Nota: veréis que los datos que se piden de los trenes pueden variar ligeramente respecto PEC anteriores, según el objetivo del ejercicio.

Ejercicio 1: Tipos estructurados de datos [50%]

a) Definición de tipos estructurados de datos [20%]

Definir en lenguaje algorítmico el tipo de datos estructurado *tTrain,* que representa la información de un tren. Los datos que se quieren guardar son las siguientes:

- *id*. Identificador del tren. Se define como un entero.
- *model*. Identificador del modelo del tren. Se define como una cadena de caracteres, con una longitud máxima de 20 caracteres.
- *locLength*. Longitud de la locomotora, se define como un número real.
- *wagonLength*. Longitud de un vagón, se define como un número real.
- *nWagons*. Número de vagones del tren, se define como un entero.
- *spaceBetweenWagons*. Espacio entre los vagones, se define como un número real.
- *propulsionType*.Tipo de propulsión del tren, que puede ser: ELECTRICAL, CARBON, SOLAR, GASOLINE, DIESEL, MAGNETIC. Hay que definir previamente un tipo enumerado que contenga los valores posibles.

b) Entrada y salida interactiva de tipos estructurados de datos [30%]

Implementar en lenguaje algorítmico un algoritmo que permita dar de alta de manera interactiva dos nuevos trenes, y guardar cada uno de los trenes en una variable diferente. NO hay que utilizar vectores. El algoritmo va pidiendo al usuario que introduzca el valor de cada campo del tren mediante la entrada y salida estándar (teclado/pantalla), y lo guarda al campo correspondiente de la estructura de datos definida al ejercicio anterior. Previamente a la lectura de cada uno de los campos, la función muestra un mensaje informativo de ayuda que indica al usuario los valores posibles, la longitud máxima del campo, u otras informaciones de interés.

Para leer los tipos de datos enumerados, en lenguaje algorítmico se dispone de la función *readPropulsionType().*

Una vez leídos todos los datos de cada tren, el algoritmo tiene que mostrar por pantalla:

- La suma de las longitudes totales de los dos trenes, para saber si podrían estacionarse en una vía determinada de la estación. No hay que dejar ningún espacio entre los dos trenes.
- El tren que tiene la locomotora de longitud más grande (identificador y modelo), que tendría prioridad de estacionamiento en caso de necesidad. En caso de que las locomotoras tengan la misma longitud, se mostrará por pantalla el primer tren entrado por el canal estándar.

Ejercicio 2: [50%]

a) Codificación en C (80%)

Implementar en lenguaje C, el algoritmo del ejercicio anterior.
Recordad que en lenguaje C, hay que definir previamente la longitud máxima de las cadenas de caracteres, típicamente mediante una constante.

b) Pruebas (20%)

Como las anteriores PEC se pide que mostréis el funcionamiento del algoritmo **haciendo dos capturas** de pantalla donde se vea la salida de vuestro programa de los valores de trenes que se han introducido en diferentes estructuras, y el resto de datos que se pide. Las capturas de pantallas las tenéis que adjuntar en el documento donde hay la respuesta del ejercicio 1.

Criterios de corrección:

- Que se siga la notación algorítmica utilizada en la asignatura. Ver el documento *Nomenclator* en la xWiki de contenido.
- 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 el programa se adecúe a las indicaciones dadas.
- Que el programa compile y funcione de acuerdo con lo que se pide.
- Que se declaren los tipos adecuados según el tipo de datos que representa.

Respuestas

```
Algorithm record

type
tPropulsion ={ ELECTRICAL, CARBON, SOLAR, GASOLINE, DIESEL,
MAGNETIC };
end type

type
tTrain = record
    idTrain:entero;
    model: vector[20] of char;
    locLength: real;
    wagonLength: real;
    nWagons: entero;
    spaceBetweenWagons: real;
    propulsionType: tPropulsion;
end type

var
train1: tTrain;
train2:  tTrain;
totalLength1: real;
totalLength2: real;
end var

    for i:=1 to 2 do

        writeString("Enter the train id: ");
        writeString("Enter the train model: ");
        writeString("Enter the locomotive's length ");
        writeString("Enter the wagon's Length: ");
        writeString("Enter the number of Wagons: ");
        writeString("Enter the space between the vagons: ");
        writeString("Enter the propulsion type ("1-ELECTRICAL, 2-CARBON, 3-
        SOLAR, 4-GASOLINE, 5-DIESEL, 6-MAGNETIC ) ");

    if i=1  then
        train1.idTrain:=readInteger();
        train1.locLength:= readReal();


                        train1.model[j]:=readChar();


        train1.wagonLength:= readReal();
        train1.nWagons:= readInteger();
        train1.spaceBetweenVagons:= readReal();
        train1.propulsionType:= readPropulsionType();
    else
        train2.idTrain:=readInteger();
        train2.locLength:= readReal();
```

```
                    train2.model[]:=readChar();


        train2.wagonLength:= readReal();
        train2.nWagons:= readInteger();
        train2.spaceBetweenVagons:= readReal();
        train2.propulsionType:= readPropulsionType();
end if
end for


totalLength1:=(train1.locLength +  train1.wagonLength +  train1.nWagons +
                ( train1.spaceBetweenVagons* train1.nWagons);

totalLength2:=(train2.locLength +  train2.wagonLength +  train2.nWagons +
                ( train2.spaceBetweenVagons* train2.nWagons));


if (train1.locLength >=  train2.locLength) then
    writeInteger(train1.dTrain);

    writeChar(train1.model);

else
        writeInteger(train2.dTrain);

        writeChar(train1.model);


end if



end algorithm
```