Universitat Oberta de Catalunya uoc.edu

Fundamentos de Programación

PEC7 - 20191

Fecha límite de entrega: 18/11/2019

Estudiante

Apellidos: DELGADO FLORES

Nombre: PABLO JOSÉ

Objetivos

- Saber modularizar el código utilizando acciones y funciones.
- Comprender la diferencia entre acción y función.
- Entender que es un parámetro actual y distinguirlo de un parámetro formal.
- Estructurar proyectos en Codelite.

Formato y fecha de entrega

La PEC se debe entregar antes del día **18 de noviembre de 2019 a las 23:59**.

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

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

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

Enunciado

En esta PEC continuaremos trabajando la modularidad, creando acciones con parámetros de salida y dividiendo el código en acciones y funciones que nos permita reutilizarlo. En este sentido, el ejercicio de codificación no se limita a traducir a C el ejercicio de lenguaje algorítmico. Se pide, además, estructurar el proyecto de Codelite en carpetas y ficheros.

Siguiendo con la ayuda que proporcionamos a UOCBookings, la compañía nos ha pedido nuestra colaboración para seguir trabajando en el siguiente algoritmo, en lenguaje algorítmico, a medio diseñar.

```
type
   tTypeHotel = {BUDGET, INN, RESORT, CONDO, LUXURY, COUNTRY}
   tHotel = record
         id: integer;
         brand: string;
         name: string;
         type: tTypeHotel;
         city: string;
         category: integer;
         priceDouble: real;
         distanceFromCityCenter: real;
         hasPool: boolean;
         hasGym: boolean;
         closeToSubway: boolean;
         percentOccupation: real;
    end record
end type

algorithm UOCBookings

{... algorithm to complete …}

end algorithm
```

Ejercicio 1: Diseño en lenguaje algorítmico (40%)

Nota: En lenguaje algorítmico podéis utilizar las acciones *hotelRead* y *hotelWrite* sin necesidad de diseñarlas. Recordad que sus cabeceras son:

```
action hotelRead(out hotel: tHotel);
action hotelWrite(in hotel: tHotel);
```

Apartado a: [25%] Diseñar la acción *hotelCopy* que copie los campos de una variable de tipo *tHotel* en otra del mismo tipo. La acción debe tener un parámetro de entrada, *src* de tipo *tHotel* (con los datos del hotel de origen) y un parámetro de salida, *dst*, también de tipo *tHotel* (que contiene los datos del hotel donde van a parar los datos que se copian).

Apartado b: [25%] Diseñar la función *hotelComputePoints*.

Esta función debe recibir un parámetro de tipo *tHotel*, *hotel,* y dos parámetros de tipo real, *price* y *distance* que, como en la función *hotelAcceptable* de la PEC6, representan un precio y una distancia que nos parecen razonables, y debe devolver un entero que se calcula a partir de la suma de los siguientes conceptos:

- 5 puntos si tiene piscina o gimnasio
- 5 puntos si tiene metro cerca.
- El resultado de la operación: 100 * (*price* - precio habitación doble) / *price*
- El resultado de la operación: 100 * (*distance* - distancia del hotel del centro) / *distance*

Mostrar también el resultado por el canal estándar de salida: nombre del hotel y el resultado de la puntuación.

Apartado c: [25%] Diseñar la acción *hotelCmpPoints* con tres parámetros de tipo *tHotel*, dos de entrada y uno de salida. Tenéis que utilizar la función *hotelComputePoints* del apartado anterior. La acción debe:

- Leer del canal estándar de entrada el precio y la distancia que nos parecen razonables.
- Calcular el número de puntos de los dos hoteles correspondientes a los parámetros de entrada y retornar en el de salida una copia del hotel que tiene el número superior de puntos. En caso de empate devuelve el primer hotel.

Apartado d: [20%] Completar el algoritmo que tenemos a medio diseñar para que:

- Lea del canal estándar de entrada la información de dos hoteles y la guarde en dos variables *hotel1* y *hotel2* de tipo *tHotel*. Es obligatorio utilizar la acción *hotelRead*.
- Muestre por el canal estándar de salida los campos del hotel (usando la acción *hotelWrite*) que tiene más puntos, haciendo uso de las acciones y funciones que se han diseñado previamente.

SOLUCIÓN apartados a-b-c-d

```
const
      MAX_BRAND: integer = 15;
      MAX_NAME: integer = 15;
      MAX_CITY: integer = 15;
end const

type
    tTypeHotel = {BUDGET, INN, RESORT, CONDO, LUXURY, COUNTRY}
    tHotel = record
        id: integer;
        brand: string;
        name: string;
        type: tTypeHotel;
        city: string;
        category: integer;
        priceDouble: real;
        distanceFromCityCenter: real;
        hasPool: boolean;
        hasGym: boolean;
        closeToSubway: boolean;
        percentOccupation: real;
    end record
end type

action hotelCopy(in src:tHotel,out dst:tHotel)
    dst.id:=src.id;
    dst.brand:=src.brand;
    dst.name:=src.name;
    dst.type:=src.type;
    dst.city:=src.city;
    dst.category:= src.category;
    dst.priceDouble:= src.priceDouble;
    dst.distanceFromCityCenter:= src.distanceFromCityCenter;
    dst.hasPool:= src.hasPool;
    dst.hasGym:= src.hasGym;
    dst.closeToSubway:= src.closeToSubway;
    dst.percentOccupation:= src.percentOccupation;
end action

function hotelComputePoints(in hotel:tHotel,in price:real,in
distance:float):integer
    var
        calc:real;
        points:integer;
    end var
    calc:=0;
    points:=0;
    if ((hotel.hasGym=true) or (hotel.hasPool=true)) then
        calc:=calc+5;
    end if
    if (hotel.closeToSubway=true) then
```

```
        calc:=calc+5;
    end if
    calc:=calc+(100*(price-hotel.priceDouble) div price);
    calc:=calc+(100*(distance-hotel.distanceFromCityCenter) div distance);
    points:=realToInteger(calc);
    writeString("PUNCTUATION FOR HOTEL ");
    writeString(hotel.name);
    writeString(": ");
    writeInteger(points);
    return points;
end function

action hotelCmpPoints(in hotel1:tHotel,in hotel2:tHotel,out h:tHotel)
    var
        acceptablePrice:real;
        acceptableDistance:real;
        pointsOfH1:integer;
        pointsOfH2:integer;
    end var
    writeString("WHAT'S THE OPTIMAL DISTANCE? >> ");
    readReal(acceptableDistance);
    writeString("WHAT'S THE OPTIMAL PRICE? >> ");
    readReal(acceptablePrice);
    pointsOfH1:=hotelComputePoints(hotel1, acceptablePrice,
acceptableDistance);
    pointsOfH2:=hotelComputePoints(hotel2, acceptablePrice,
acceptableDistance);
    if (pointsOfH1≥pointsOfH2) then
        hotelCopy(hotel1,h);
    else
        hotelCopy(hotel2,h);
    end if
end action

algorithm UOCBookings
    var
        hotel1:tHotel;
        hotel2:tHotel;
        bestHotel:tHotel;
    end var
    writeString("INTRODUCE DATA FOR HOTEL 1");
    hotelRead(in hotel1:tHotel);
    writeString("INTRODUCE DATA FOR HOTEL 2");
    hotelRead(in hotel2:tHotel);
    hotelCmpPoints(in hotel1, in hotel2, in out bestHotel);
    writeString("THE BEST CHOICE IS >> ");
    hotelWrite(out bestHotel);
end algorithm
```

Apartado e: [5%] ¿Cómo habría que modificar la estructura de *tHotel* y la función *hotelComputePoints* si quisiéramos guardar en un campo de la tupla *tHotel* el número de puntos?

No hay que hacer ningún diseño, únicamente explicar qué haría falta hacer.

SOLUCIÓN

Habría que declarar en la estructura una nueva variable que recoja el número de puntos, mientras que en la función *hotelComputePoints* habría que declarar un parámetro más de salida, que sería el puntero que modificaría el valor de puntuación en el struct.

Ejercicio 2: Programación en C [60%]

En este ejercicio hay que **codificar en lenguaje C** el ejercicio 1 y, además, estructurar el código en carpetas. Concretamente hay que hacer lo siguiente:

1. Crea un nuevo proyecto en Codelite llamado *Hotels.* Crea una carpeta *include* y una carpeta *src* en este proyecto siguiendo las explicaciones que puedes encontrar en la unidad de la XWiki *Modularidad en Codelite* de la asignatura.
2. Dentro de la carpeta *include,* crea un nuevo archivo llamado **hotel.h** que contenga la declaración del tipo estructurado *tHotel,* la del tipo enumerado *tTypeHotel* y la del tipo *boolean*, así como las constantes necesarias.
3. Copia las cabeceras de todas las acciones/funciones que tengas que utilizar (***hotelRead, hotelWrite, hotelComputePoints, hotelCopy, hotelCmpPoints*** etc.) en el archivo **hotel.h**.
4. Dentro de la carpeta *src,* crea un nuevo archivo **hotel.c.** Pon el código de las acciones/funciones que tengas que utilizar (***hotelRead, hotelWrite, hotelComputePoints, hotelCopy***, ***hotelCmpPoints*** etc.)

   Nota: Puedes copiar de la PEC6 el código de las acciones *hotelRead* y *hotelWrite.*
5. Codifica el algoritmo del ejercicio 1, apartado d, dentro de la función principal **main.c** (este archivo tiene que estar dentro de la carpeta *src*).
6. Comprueba que compila y funciona correctamente.

Criterios de corrección:

En el ejercicio 1:

- Que sigue la notación algorítmica utilizada en la asignatura. Véase documento *Nomenclator* la XWiki de contenido.
- Que se adecua a las instrucciones dadas y el algoritmo responda al problema planteado.
- Que se diseña y se llama correctamente las acciones y funciones demandadas.
- Que se utilice correctamente la estructura alternativa y el tipo de datos estructurado.
- Que se razone correctamente la respuesta del apartado e de la primera pregunta.

En el ejercicio 2:

- Que el programa se adecua las indicaciones dadas.
- Que el programa compila y funciona de acuerdo con lo que se pide.
- Que se respetan los criterios de estilo de programación C. Véase la *Guía de estilo de programación en C* que tiene en la Wiki de contenido.
- Que se implementa correctamente la modularización del proyecto, dividiendo el código en carpetas y poniendo lo que corresponde a cada carpeta.