Fundamentos de Programación

PAC7 - 20182

Fecha límite de entrega: 04/15/2019

Estudiante

Apellidos: Giménez Gorrís

Nombre: Álvaro

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 **15 de abril 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 y el apartado b del ejercicio 2
- El workspace de Codeliteque contenga **el proyecto y todas las carpetas creadas** al ejercicio 2a

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

Enunciado

En esta PEC continuaremos trabajando la modularidad dividiendo el código en acciones y funciones que nos permitan poderlas reutilizar. 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 UOCAirways, la compañía nos ha pedido nuestra colaboración para seguir trabajando en el siguiente algoritmo, en lenguaje algorítmico, a medio diseñar.

```
type
   tUtility = {COMMERCIAL, PRIVATE, GOVERNMENTAL, MILITAR, EXPERIMENTAL, OTHERS}
tPlane = record
          id: integer;
           model: string;
           year: integer;
           utility: tUtility;
           percentOccupationFirstclass: real;
           percentOccupationBusiness: real;
           percentOccupationTurist: real;
          isActive: boolean;
     end record
end type


algorithm UOCAirways

const
    NUM_CLASSES: integer = 3;
end const

{... algorithm to complete ...}

end algorithm
```

Ejercicio 1: Diseño en lenguaje algorítmico (40%)

Nota: La estructura de datos *tPlane* ha cambiado ligeramente respecto la PEC anterior. Podemos asumir, en este ejercicio de lenguaje algorítmico, que las acciones *planeRead*, *planeWrite* y *planeCopy* de la PEC anterior ya han sido adaptadas para funcionar con la nueva estructura del tipo *tPlane* y que las podemos utilizar sin volver a diseñar.

Apartado a: [50%]Diseña la función *isFirstPlane* que tiene dos parámetros *plane1, plane2* de tipo *tPlane* y devuelve un booleano. La función devuelve true si el primer avión tiene prioridad para aterrizar y *false* en caso contrario. El criterio para

decidir la prioridad para aterrizar se establece siguiendo el siguiente orden según los valores de:

1. Ocupación total (en %) del avión, entendida como media de la ocupación de las tres clases (*percentOccupationFirstclass*, *percentOccupationBusiness*, *percentOccupationTurist*).
2. Ocupación (en%) de la primera clase (*percentOccupationFirstclass*).
3. Ocupación (en%) de la clase business (*percentOccupationBussines*).
4. Ocupación (en%) de la clase turista (*percentOccupationTurist*).

El funcionamiento es el siguiente: en primer lugar se compara la ocupación total (en %) de *plane1* y *plane2.* Si la ocupación total de *plane1* es superior a la de *plane2, se* devuelve *true*. Si la ocupación total de *plane1* es inferior a la de *plane2, se* devuelve *false*. En caso de empate, se pasa a comparar la ocupación (en %) de la primera clase *(percentOccupationFirstclass),* y así sucesivamente. En caso de que todos los campos comparados de los dos aviones sean iguales, la función devolverá *true* (es decir, dará prioridad al primer avión).

```
function isFirstPlane (plane1 : tPlane, plane2 : tPlane): boolean

var

        totalAverage1 : real;
        totalAverage2 : real;
        priority : boolean;

end  var

totalAverage1 := (plane1.percentOccupationFirstclass +
plane1.percentOccupationbussines + plane1.percentOccupationTurist) div 3.0;

totalAverage2 := (plane2.percentOccupationFirstclass +
plane2.percentOccupationbussines + plane2.percentOccupationTurist) div 3.0;

if (totalAverage1 > totalAverage2) then

        priority := true;

else

        if (totalAverage1 < totalAverage2) then

                priority := false;

        else

                if (plane1.percentOccupationFirst > plane2.percentOccupationFirst) then
```

```
                priority := true;

            else
                if (plane1.percentOccupationFirst < plane2.percentOccupationFirst)
                then

                    priority := false;

                else
                    if (plane1.percentOccupationBussines >
                    plane2.percentOccupationBussines) then

                        priority := true;

                    else
                        if (plane1.percentOccupationBussines <
                        plane2.percentOccupationBussines) then

                            priority := false;

                        else
                            if (plane1.percentOccupationTurist >
                            plane2.percentOccupationTurist) then

                                priority := true;

                            else
                                if (plane1.percentOccupationTurist <
                                plane2.percentOccupationTurist) then

                                    priority := false;

                                else

                                    priority := true;

                                end if
                            end if
                        end if
                    end if
                end if
            end if
        end if
end if

return (priority);

end function
```

Apartado b: [20%]Diseña la acción *planePriorized* que devuelve en el parámetro de salida *planePriority* de tipo *tPlane* el avión que tiene prioridad para aterrizar. La acción tiene, además, dos parámetros de entrada *plane1 y plane2* de tipo *tPlane.* La acción copia todos los campos de *plane1* o *plane2* a *planePriority.* El funcionamiento es el siguiente: copiará todos los campos de *plane1* a *planePriority* en caso de que sea *plane1* el que tenga prioridad. En caso contrario, la acción copiará todos los campos de *plane2* a *planePriority.*

action planePriorized (**in** plane1 : tPlane, **in** plane2 : tPlane, **out** planePriority : tPlane)

if (isFirstPlane(plane1, plane2) = true) **then**

    planePriority := plane1;

else

    planePriority := plane2;

end if

writeString("Data of the plane with priority to land is: ");

end action

Apartado c: [20%]Completa el algoritmo que tenemos a medio diseñar para que lea del canal estándar de entrada la información de dos aviones y la guarde en las variables *plane1* y *plane2.* A continuación, el algoritmo debe mostrar por el canal estándar de salida los campos del avión que tiene prioridad para aterrizar, usando las acciones y funciones que hemos diseñado previamente.

Nota: Debemos asumir que los aviones son comerciales y están en activo. Por lo tanto, no es necesario que el algoritmo lo compruebe y funcionará independientemente del valor de los campos *utility* y *isActive*.

A continuación encontrarás ejemplos de la salida del algoritmo para dos casos diferentes de entrada.

Ejemplo 1:

...
plane1. percentOccupationFirstclass: = 70.5;
plane1. percentOccupationBusiness: = 79.5;
plane1. percentOccupationTurist: = 93.0;
...
plane2. percentOccupationFirstclass: = 77.5;

plane2. percentOccupationBusiness: = 75.5;
plane2. percentOccupationTurist: = 87.0;
...

El resultado que se mostraría por el canal estándar de salida en este caso sería *plane1* ya que *plane1* tiene una ocupación total del 81% y *plane2* tiene una ocupación total del 80%.

Ejemplo 2:
...
plane1. percentOccupationFirstclass: = 70.5;
plane1. percentOccupationBusiness: = 79.5;
plane1. percentOccupationTurist: = 90.0;
...
plane2. percentOccupationFirstclass: = 83.5;
plane2. percentOccupationBusiness: = 69.5;
plane2. percentOccupationTurist: = 87.0;
...

En este caso, la ocupación total de los dos aviones es del 80%. No obstante se mostraría por el canal estándar de salida *plane2* ya que tiene prioridad ante *plane1* debido al campo *percentOccupationFirstclass*.

algorithm UOCplanes

var

```
    plane1 : tPlane;
    plane2 : tPlane;
    planePriority : tPlane;
```

end var

```
planeRead(plane1);
planeRead(plane2);

isFirstPlane(plane1, plane2);

planePriority(plane1, plane2, planePriority);

planeWrite(planePriority);
```

end algorithm

Apartado d: [10%] En el apartado *c* hemos asumido que los aviones son comerciales y están en activo. Desde el punto de vista de la modularización, explica que habría que hacer al algoritmo anterior para que comprobara que el valor del campo

utility fuera COMMERCIAL y el valor del campo *isActive* fuera *true*. No hay que hacer el diseño, sólo hay que explicarlo.

Habría que hacer una función con un parámetro de entrada plane de tipo tPlane y devolviendo un valor booleano. La función contaría con una estructura de control alternativa que comprobaría si el avión está activo y es comercial a la vez y devolvería un valor booleano según si es verdadero o falso para así, poder comprobar el resultado antes de ejecutar la función y la acción anterior.

Ejercicio 2: Programación en C [60%]

Apartado a: [70%] En este ejercicio hay que **codificar en lenguaje C** del ejercicio 1 (apartados a, b y c) y, además, estructurar el código en carpetas. Concretamente hay que hacer lo siguiente:

1. Crea un nuevo proyecto en Codelite llamado *Planes.* Crea una carpeta *include* y una carpeta *src* en este proyecto siguiendo las explicaciones que pueden encontrar en la unidad de la XWiki *Modularidad en Codelite* de la asignatura.
2. Dentro de la carpeta *include,* crea un nuevo archivo llamado **plane.h** que contenga la declaración del tipo estructurado *tPlane,* la del tipo enumerado *tUtility* y la del tipo *boolean*, así como las constantes necesarias.
3. Copia las cabeceras de todas las acciones/funciones que tengas que utilizar ***(planeRead, planeWrite, planePriorized, isFisrtPlane***, etc.)en el archivo **plane.h**.
4. Dentro de la carpeta *src,* crea un nuevo archivo **plane.c.** Pon el código de las acciones/funciones que tengas que utilizar(***PlaneRead***, ***planeWrite, planePriorized, isFisrtPlane***, etc.)

   Nota: En el ejercicio 1 hemos supuesto que las acciones *planeRead*, *planeWrite* y *planeCopy* (de la PEC anterior) ya habían sido adaptadas en lenguaje *algorítmico.* Ahora, en este ejercicio de codificación en C, es necesario que las modifiques para que funcionen correctamente con la nueva estructura del tipo *tPlane.*

5. Codifica el algoritmo del ejercicio 1, apartado c, dentro de la función principal **main.c**
6. Comprueba que compila y funciona correctamente.

Apartado b: [30%] Como en las anteriores PEC se pide que muestres el funcionamiento del algoritmo **haciendo juegos de prueba.** Es decir que completes las siguientes tablas indicando, para unos valores de aviones que se han introducido en las diferentes estructuras, qué salida se espera en la ejecución del programa.

b1) La ocupación total (en%) de un avión (entendida como media de la ocupación de las tres clases) es superior al otro (como en el ejemplo 1):

Datos de entrada	
Nombre de la variable	**Valor de entrada**
Id	1111
Model	First
Year	1988
Utility	0
percentOccupationFirstclass	70.5
percentOccupationBussines	79.5
PercentOccupationTurist	93.0
isActive	1
Id	2222
Model	Second
Year	2000
Utility	0
percentOccupationFirstclass	77.5
percentOccupationBussines	75.5
PercentOccupationTurist	87.0
isActive	1

Salida
Plane1

```
Identifier for the plane (integer):
1111
Model for the plane (string):
first
Year for the plane(integer):
1988
Utility for the plane (enter a number being 0=COMMERCIAL, 1=PRIVATE, 2=GOVERNA
MENTAL, 3=MILITAR, 4=EXPERIMENTAL, 5=OTHERS): .
0
Percent occupation of the first class:
70.5
Percent occupation of the bussines class:
79.5
Percent occupation of the turist class:
93.0
Is the plane active? (true/false):
1
Identifier for the plane (integer):
2222
Model for the plane (string):
second
Year for the plane(integer):
2000
Utility for the plane (enter a number being 0=COMMERCIAL, 1=PRIVATE, 2=GOVERNA
MENTAL, 3=MILITAR, 4=EXPERIMENTAL, 5=OTHERS): .
0
Percent occupation of the first class:
77.5
Percent occupation of the bussines class:
75.5
Percent occupation of the turist class:
87.0
Is the plane active? (true/false):
1


Data of the plane with priority to land:
Plane id:
1111
Plane model:
first
Plane fabrication year:
1988
Plane utility:
0
```

b2) La ocupación total (en%) de los dos aviones (entendida como media de la ocupación de las tres clases) esigual y tienen diferente ocupación en alguna de las otras clases (como en el caso del ejemplo 2):

Datos de entrada	
Nombre de la variable	**Valor de entrada**
Id	1111
Model	First
Year	1988
Utility	0
percentOccupationFirstclass	70.5
percentOccupationBussines	79.5
PercentOccupationTurist	90.0
isActive	1
Id	2222
Model	Second
Year	2000
Utility	0
percentOccupationFirstclass	93.5
percentOccupationBussines	69.5
PercentOccupationTurist	87.0
isActive	1

Salida
Plane 2

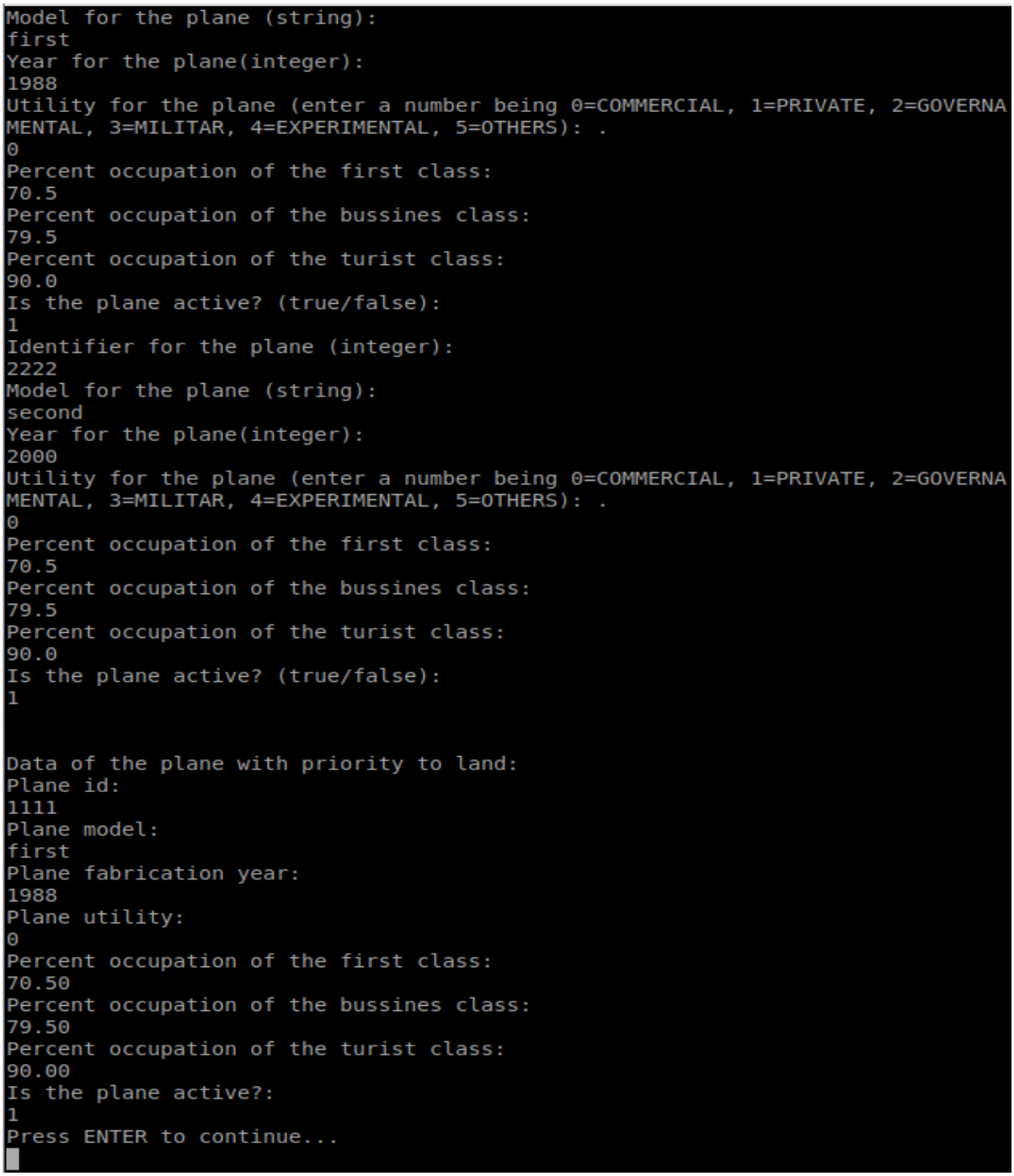
```
Model for the plane (string):
first
Year for the plane(integer):
1988
Utility for the plane (enter a number being 0=COMMERCIAL, 1=PRIVATE, 2=GOVERNA
MENTAL, 3=MILITAR, 4=EXPERIMENTAL, 5=OTHERS): .
0
Percent occupation of the first class:
70.5
Percent occupation of the bussines class:
79.5
Percent occupation of the turist class:
90.0
Is the plane active? (true/false):
1
Identifier for the plane (integer):
2222
Model for the plane (string):
second
Year for the plane(integer):
2000
Utility for the plane (enter a number being 0=COMMERCIAL, 1=PRIVATE, 2=GOVERNA
MENTAL, 3=MILITAR, 4=EXPERIMENTAL, 5=OTHERS): .
0
Percent occupation of the first class:
83.5
Percent occupation of the bussines class:
69.5
Percent occupation of the turist class:
87
Is the plane active? (true/false):
1


Data of the plane with priority to land:
Plane id:
2222
Plane model:
second
Plane fabrication year:
2000
Plane utility:
0
Percent occupation of the first class:
83.50
Percent occupation of the bussines class:
69.50
Percent occupation of the turist class:
87.00
Is the plane active?:
1
Press ENTER to continue...
```

b3) Los dos aviones tienen la ocupación total igual y la ocupación de las tres clases también igual:

Datos de entrada	
Nombre de la variable	Valor de entrada
Id	1111

Model	First
Year	1988
Utility	0
percentOccupationFirstclass	70.5
percentOccupationBussines	79.5
PercentOccupationTurist	90.0
isActive	1
Id	2222
Model	Second
Year	2000
Utility	0
percentOccupationFirstclass	70.5
percentOccupationBussines	79.5
PercentOccupationTurist	90.0
isActive	1

Salida
Plane1



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 d 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.