Fundamentos de Programación
PR3 - 20232

Fecha límite de entrega: **29/05/2024 a las 23:59**

Enunciado

El Imperio Galáctico nos ha pedido ayuda para la organización de su flota de naves. Para esto desarrollaremos una aplicación para la gestión de sus naves, así como de las bases estelares repartidas a lo largo de la galaxia.

Para dar respuesta a esta petición, a lo largo de las distintas PR iremos desarrollando una pequeña parte de la aplicación final que se encargará de la gestión de la flota del Imperio mediante la definición del modelo de datos y la implementación de distintas funcionalidades y algoritmos.

Aplicación a desarrollar en esta PR: Leer y mostrar el listado de bases imperiales ubicadas en Tatooine, posteriormente seleccionar una de estas bases para mostrar su flota de naves ordenadas según la capacidad de transporte de tropas.

El desarrollo de la aplicación en esta PR tiene **cuatro partes**:

1. Interpretación de un algoritmo
2. Diseño algorítmico
3. Codificación en C
4. Prueba del programa en C

---

1. Interpretación de un algoritmo

Leer el algoritmo desarrollado parcialmente que se expone a continuación; **no es necesario realizar ninguna entrega relacionada con este apartado**. Dicho algoritmo os servirá de base para generar vuestra solución.

```
const
    MAX_SHIPS: integer = 10          {Max. ships}
    MAX_IMPERIAL_BASES: integer = 5 {Max. Imperial bases}
end const

type
    {User defined types}
    {Error type}
    tError = {OK, ERR_CANNOT_READ, ERR_MEMORY, ERR_NOT_FOUND}

    {Ship type}
    tShipType = {TRANSPORT, FIGHTER, MEDICAL, EXPLORER}

    tShip = record
        name: string;            {Ship name}
        shipType: tShipType;     {Ship type}
        troopCapacity: integer; {Capacity of troops it can be transport}
    end record

    tShipTable = record
        ships: vector[MAX_SHIPS] of tShip;
        numShips: integer;
    end record

    tImperialBase = record
        name: string;               {Imperial base name}
        shipTable: tShipTable;      {Ships deployed}
    end record

    tImperialBaseTable = record
        imperialBases: vector[MAX_IMPERIAL_BASES] of tImperialBase;
        numImperialBases: integer;
    end record
end type


{Exercise 2.1}
action selectImperialBase(in imperialBaseTable: tImperialBaseTable,
    in baseName: string, out imperialBase: tImperialBase, out retVal: tError)
    {...}
end action

{Exercise 2.2}
action sortShipTableByCapacity(inout shipTable: tShipTable)
    var
        i: integer;       {Index}
        j: integer;       {Index}
        posMin: integer; {Minimum position}
        shipAux: tShip;  {Auxiliary ship for swapping}
    end var

    i := 1; {Initialize the iteration index}
    while i ≤ shipTable.numShips do {Iterate over all ships in the ship table}
        posMin := i; {Assume the current ship has the minimum troop capacity}
        j := i + 1;  {Start comparing with the following ships onwards}
        {...}
        {Swap the ships: place the ship with the minimum troop capacity at the current}
        {position (i), and move the current ship (i) to the position where the ship
with
        {minimum capacity was (posMin)}
        shipAux := shipTable.ships[posMin];
        {...}

        {Move to the next ship to continue the sorting process}
        i := i + 1;
    end while
end action

algorithm UOCGalacticEmpire
    var
```

```
        imperialBaseTable: tImperialBaseTable;
        selectedImperialBase: tImperialBase;
        basename: string
        filename: string;
        retVal: tError;
    end var

    {Exercise 2.3}
    {Imperial Bases table initialization}
    {...}

    {Load data from file}
    writeString("LOAD DATA FROM FILE. ENTER FILE NAME:");
    {...}

    {Data processing}
    if retVal = OK then
        {Write Imperial Bases list loaded}
        {...}
        {Exercise 2.4}
        {Select a Imperial Base by name}
        {Data input}
        writeString("BASE NAME (A STRING)?");
        {...}
    else
        {No data recovered}
        writeString("NO IMPERIAL BASES RECOVERED");
    end if
end algorithm
```

Definición de tipos de datos.

- El tipo de datos estructurado *tShip* representa una nave desplegada en una base Imperial. Los campos son los siguientes:

Campo	Descripción	Tipo / validación
name	Nombre de la nave	Valor tipo *cadena de caracteres*
shipType	Tipo de nave.	Valor de tipo *tShipType*: TRANSPORT, FIGHTER, MEDICAL, EXPLORER.
troopCapacity	Capacidad de tropas que puede transportar	Valor de tipo *integer*.

- El tipo de datos estructurado *tShipTable* representa una tabla de elementos de tipo *tShip*. Los campos son los siguientes:

Campo	Descripción	Tipo / validación
ships	Naves	Valor de tipo *vector de tShips.*
numShips	Número de naves de la tabla. Indica la cantidad de elementos que posee el vector *ships*.	Valor de tipo *integer.*

- El tipo de datos estructurado *tImperialBase* representa una base del Imperio Galáctico. Los campos son los siguientes:

Campo	Descripción	Tipo / validación
name	Nombre de la base, sin espacios en blanco.	Valor de tipo *cadena de caracteres.*
shipTable	Flota de naves desplegadas en la base.	Valor de tipo *tShipTable.*

- El tipo de datos estructurado *tImperialBaseTable representa* una tabla de Bases Imperiales. Los campos son los siguientes:

Campo	Descripción	Tipo / validación
imperialBases	Bases imperiales.	Valor de tipo *vector de tImperialBase.*
numImperialBases	Número de bases imperiales. Indica la cantidad de elementos que posee el vector imperialBases.	Valor de tipo *entero.*

La estructura del algoritmo, así como los tipos de datos, constantes y variables que ya están declarados, **servirán de base para el diseño algorítmico posterior**.

Acciones auxiliares.

Se dispone de las siguientes acciones auxiliares ya predefinidas, por tanto, **no es necesario implementarlas.**

- ***action*** *initImperialBaseTable(**out** imperialBaseTable: tImperialBaseTable);*

Esta acción inicializa una tabla de tipo *imperialBaseTable.* Una tabla de bases imperiales inicializada es aquella que se encuentra vacía.

- ***action*** *loadImperialBaseTable(**in** filename: **string**, **out** imperialBaseTable: tImperialBaseTable, **out** retVal: tError);*

  Esta acción lee datos de bases imperiales desde el fichero *filename* y los carga en la tabla *imperialBaseTable*. El resultado de la operación de lectura se guarda dentro del parámetro de salida *retVal*, si la lectura del fichero es correcta, el valor de *retVal* será *OK*, en caso de error será *ERR_CANNOT_READ*.

- ***action*** *writeImperialBase(**in** imperialBase: tImperialBase);*

  Esta acción muestra por la salida estándar los datos de una base imperial, que son su nombre y el listado de las naves desplegadas en ella. Para cada nave muestra su nombre, el tipo, y su capacidad de transporte de tropas.

```
BASE NAME
-----------------------------
BESTINE

SHIP NAME   TYPE     CAPACITY
-----------------------------
BE_T1          2          500
BE_T2          2          750
BE_F1          3            0
```

- ***action*** *writeImperialBaseTable(**in** imperialBaseTable**:** tImperialBaseTable);*

  Esta acción muestra por la salida estándar las bases imperiales contenidas en tabla *tImperialBaseTable*.

```
BASE NAME
------------------------------
BESTINE

SHIP NAME   TYPE     CAPACITY
------------------------------
BE_T1          2          500
BE_T2          2          750
BE_F1          3            0


BASE NAME
------------------------------
FORT_TUSKEN

SHIP NAME   TYPE     CAPACITY
------------------------------
FT_T1          2          500
FT_T2          2          750
FT_F1          3            0
FT_E2          2           25


BASE NAME
------------------------------
MOS_EISLEY

SHIP NAME   TYPE     CAPACITY
------------------------------
ME_T1          2          350
ME_T2          2          300
ME_T3          2          150
ME_M1          4          500
ME_E1          4           15
```

2. Diseño algorítmico

Diseñar un algoritmo que incluya las funcionalidades que se detallan a continuación.

Deberán declararse las variables que se consideren necesarias, escogiendo el tipo de datos más apropiado, pudiéndose combinar con las que ya están declaradas en el algoritmo parcial del enunciado.

2.1. Desarrollo de funciones/acciones. Desarrollar la acción *selectImperialBase* para seleccionar una base imperial por nombre. Esta acción recibirá una tabla de bases imperiales y el nombre de base a seleccionar. Si la base existe, se retornará en un parámetro de salida de tipo *tImperialBase*. Además, dispondrá de un parámetro de salida de tipo *tError* para informar si la selección ha sido efectiva o no. Disponéis de la cabecera de esta acción en el apartado 1 del enunciado.

2.2. Desarrollo de funciones/acciones. Desarrollar la acción *sortShipTableByCapacity* que ordenará un grupo de naves en función de la capacidad de cada nave. La ordenación será en orden ascendente. Esta acción recibirá un parámetro de entrada/salida de tipo *tShipTable*. El tipo de ordenación aplicable se contempla en los recursos de aprendizaje de esta PR3, lo podéis encontrar en el apartado ***6. Ordenación de Tablas*** del tema ***17. Tipos de datos estructurados***. Tenéis esta acción parcialmente desarrollada en el apartado 1 del enunciado.

2.3. Carga y posterior visualización de datos. Leer del canal estándar de entrada el nombre del fichero que contiene los datos de las bases imperiales necesarios para asignar valores a la tabla *imperialBaseTable,* definida en el algoritmo del enunciado y **previamente inicializada**. Para la carga de datos hará uso de la acción predefinida *loadImperialBaseTable*. Seguidamente, deberá mostrarse el listado de las bases imperiales contenidas en la tabla, para esto se hará uso de la acción auxiliar predefinida *writeImperialBaseTable*. En caso de error en la carga de datos, el programa finalizará y se mostrará el siguiente mensaje informativo:

NO IMPERIAL BASES RECOVERED

2.4. Procesamiento y salida de datos. Mostrar por el canal estándar de salida la información de una base imperial previamente seleccionada por su campo nombre. Su listado de naves deberá ir ordenado de forma ascendente según el campo de capacidad.

Leer desde el canal estándar de entrada de datos el nombre de la base imperial a seleccionar. Posteriormente se recuperará la base imperial desde la tabla de bases imperiales según el nombre introducido.

A continuación, se ordenarán las naves desplegadas en la base seleccionada según la capacidad de estas y en orden ascendente. Finalmente se mostrará la base seleccionada usando la acción auxiliar predefinida *writeImperialBase*, **finalizando así la ejecución del algoritmo**.

En caso de no recuperarse la base, **el algoritmo finaliza** mostrando el siguiente mensaje informativo:

NO IMPERIAL BASE RECOVERED

Solución (algoritmo completo)

```
const
    MAX_SHIPS: integer = 10          {Max. ships}
    MAX_IMPERIAL_BASES: integer = 5 {Max. Imperial bases}
end const

type
    {User defined types}
    {Error type}
    tError = {OK, ERR_CANNOT_READ, ERR_MEMORY, ERR_NOT_FOUND}

    {Ship type}
    tShipType = { TRANSPORT, FIGHTER, MEDICAL, EXPLORER}

    tShip = record
        name: string;            {Ship name}
        shipType: tShipType;     {Ship type}
        troopCapacity: integer; {Capacity of troops it can be transport}
    end record

    tShipTable = record
        ships: vector[MAX_SHIPS] of tShip;
        numShips: integer;
    end record

    tImperialBase = record
        name: string;               {Imperial base name}
        shipTable: tShipTable;      {Ships deployed}
    end record

    tImperialBaseTable = record
        imperialBases: vector[MAX_IMPERIAL_BASES] of tImperialBase;
        numImperialBases: integer;
    end record
end type


{Exercise 2.1}
action selectImperialBase(in imperialBaseTable: tImperialBaseTable,
    in baseName: string, out imperialBase: tImperialBase, out retVal: tError)

    var
        i: integer;
    end var

    i := 1; {Initialize the iteration index}
    retVal := ERR_NOT_FOUND;
    {Iterate over the imperial bases table until the end or until the base is found}
    while i ≤ imperialBaseTable.numImperialBases and retVal = ERR_NOT_FOUND do
        {Check if the name of the current base matches the provided shipName}
        if baseName = imperialBaseTable.imperialBases[i].name then
```

```
            imperialBase := imperialBaseTable.imperialBases[i];
            retVal := OK
        end if
        i := i + 1;
    end while
end action

{Exercise 2.2}
action sortShipTableByCapacity(inout shipTable: tShipTable)
    var
        i: integer;       {Index}
        j: integer;       {Index}
        posMin: integer; {Minimum position}
        shipAux: tShip;  {Auxiliary ship for swapping}
    end var

    i := 1; {Initialize the iteration index}
    while i ≤ shipTable.numShips do {Iterate over all ships in the ship table}
        posMin := i; {Assume the current ship has the minimum troop capacity}
        j := i + 1;  {Start comparing with the following ships onwards}
        while j ≤ shipTable.numShips do

            {If the next ship has less troop capacity, update the minimum position to j}
            if shipTable.ships[j].troopCapacity <
                shipTable.ships[posMin].troopCapacity then
                posMin := j;
            end if
            {Move to the next ship to compare}
            j := j + 1;
        end while
        {Swap the ships: place the ship with the minimum troop capacity at the current}
        {position (i), and move the current ship (i) to the position where the ship
with
        {minimum capacity was (posMin)}
        shipAux := shipTable.ships[posMin];
        shipTable.ships[posMin] := shipTable.ships[i];
        shipTable.ships[i] := shipAux;

        {Move to the next ship to continue the sorting process}
        i = i + 1;
    end while
end action


algorithm UOCGalacticEmpire
    var
        imperialBaseTable: tImperialBaseTable;
        selectedImperialBase: tImperialBase;
        basename: string
        filename: string;
        retValLoad: tError;
        retValSel: tError;
    end var

    {Exercise 2.3}
    {Imperial Bases table initialization}
    initImperialBaseTable(imperialBaseTable);

    {Load data from file}
    writeString("LOAD DATA FROM FILE. ENTER FILE NAME:");
    filename := readString();

    loadImperialBaseTable(filename, imperialBaseTable, retValLoad);

    {Data processing}
    if retValLoad = OK then
        {Write Imperial Bases list loaded}
        writeImperialBaseTable(imperialBaseTable);
        {Exercise 2.4}
```

```
        {Select an Imperial Base by name}
        {Data input}
        writeString("BASE NAME (A STRING)?");
        baseName := readString();
        {Recover imperial base from table}
                selectImperialBase(imperialBaseTable,  baseName,  selectedImperialBase,
retValSel);

        if retValSel = OK then
            {Sort the ships from the selected Imperial Base}
            sortShipTableByCapacity(selectedImperialBase.shipTable);
            {Write selected imperial base}
            writeImperialBase(selectedImperialBase);
        else
            {No data recovered}
            writeString("NO IMPERIAL BASE RECOVERED");
        end if
    else
        {No data recovered}
        writeString("NO IMPERIAL BASES RECOVERED");
    end if
end algorithm
```

3. Codificación en C

En este ejercicio hay que codificar en C el algoritmo anterior. Para esta PR se proporciona el enunciado del programa con una parte del código ya implementado. **Revisad la configuración ya preestablecida del proyecto**: en el directorio de trabajo *bin* se generará el archivo final ejecutable, mientras que los archivos de compilación se crearán en el directorio *Debug*. Concretamente, hay que hacer lo siguiente:

1. Descomprimir el archivo que incluye el proyecto *Codelite*. El proyecto está estructurado en carpetas: en la carpeta *include* dónde está el archivo ***galacticempire.h***, y en la carpeta *src* donde se encuentran los archivos ***galacticempire.c*** y ***main.c***. La codificación debe seguir la estructura de carpetas y archivos del workspace facilitado y que se explica a continuación. También se pueden encontrar los ficheros *bases1.txt*, *bases2.txt* y *bases3.txt,* que se proporcionan para poder cargar datos en la tabla de bases imperiales. En el siguiente ejemplo podéis observar cómo se le indica el nombre y la localización del fichero a la hora de ejecutar la aplicación. En este caso, dicho fichero se encuentra en el directorio padre de donde se ejecuta, así es como lo tenéis disponible en el proyecto enunciado:

   ```
   LOAD DATA FROM FILE. ENTER FILE NAME:
   ../bases1.txt
   ```

2. Dentro del archivo ***galacticempire.h,*** ya tenéis declaradas las cabeceras de las acciones y funciones a desarrollar: ***selectImperialBase*** *y* ***sortShipTableByCapacity.***
3. Dentro del archivo ***galacticempire.c,*** implementar las acciones y funciones*,* ***selectImperialBase*** *y* ***sortShipTableByCapacity.***
4. Dentro del fichero ***main.c***:
   a. Se debe completar la declaración de variables para codificar ***main.c***.
   b. Se debe completar la codificación del ***main.c*** con los ejercicios 2.4 y 2.5.

El programa en C debe cumplir con las siguientes particularidades:

- En los archivos ***galacticempire.h*** y ***galacticempire.c*** también disponéis de las cabeceras e implementaciones de las acciones ***initImperialBaseTable***, ***loadImperialBaseTable****,* ***writeImperialBase*** y ***writeImperialBaseTable****,* necesarias para mostrar los datos de forma unificada.
- El resto de las acciones implementadas en este archivo son de uso interno para las acciones auxiliares descritas.

4. Prueba del programa en C

Ejecutar y superar los juegos de prueba automáticos disponibles en la herramienta DSLab.

El proceso de validación y corrección de la herramienta DSLab se basa en una comprobación **literal** de la salida obtenida por el programa sometido a prueba, con los resultados esperados de los juegos de prueba introducidos previamente. Por ello, los textos de la interfaz de usuario deben ser exactamente idénticos a los esperados. **Podéis recuperar dichos textos del ejemplo de ejecución que hay al final del enunciado**.

Hay que tener en cuenta que el proceso de copiar y pegar textos entre distintos editores, entornos y herramientas puede generar caracteres ocultos que hagan que la comparación de los textos literales sea incorrecta a pesar de que los textos literales aparentemente sean idénticos.

Ejemplos de ejecución.

Ejemplo 1:

```
LOAD DATA FROM FILE. ENTER FILE NAME:
../bases1.txt

BASE NAME
----------------------------
BESTINE

SHIP NAME    TYPE      CAPACITY
----------------------------
BE_T1           2           500
BE_T2           2           750
BE_F1           3             0


BASE NAME
----------------------------
FORT_TUSKEN

SHIP NAME    TYPE      CAPACITY
----------------------------
FT_T1           2           500
FT_T2           2           750
FT_F1           3             0
FT_E2           2            25


BASE NAME
----------------------------
MOS_EISLEY

SHIP NAME    TYPE      CAPACITY
----------------------------
ME_T1           2           350
ME_T2           2           300
ME_T3           2           150
ME_M1           4           500
ME_E1           4            15

BASE NAME (A STRING)?
BESTINE

BASE NAME
```

```
----------------------------
BESTINE

SHIP NAME    TYPE      CAPACITY
----------------------------
BE_F1           3             0
BE_T1           2           500
BE_T2           2           750
```

Ejemplo 2:

```
LOAD DATA FROM FILE. ENTER FILE NAME:
../bases1.txt

BASE NAME
----------------------------
BESTINE

SHIP NAME    TYPE      CAPACITY
----------------------------
BE_T1           2           500
BE_T2           2           750
BE_F1           3             0


BASE NAME
----------------------------
FORT_TUSKEN

SHIP NAME    TYPE      CAPACITY
----------------------------
FT_T1           2           500
FT_T2           2           750
FT_F1           3             0
FT_E2           2            25


BASE NAME
----------------------------
MOS_EISLEY

SHIP NAME    TYPE      CAPACITY
----------------------------
ME_T1           2           350
ME_T2           2           300
ME_T3           2           150
```

```
ME_M1           4           500
ME_E1           4            15

BASE NAME (A STRING)?
NEW_BASE
NO IMPERIAL BASE RECOVERED
```