Fundamentos de Programación
PR1 - 20232

Fecha límite de entrega: **17/04/2024 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 PR1: A partir de una flota de naves introducidas por el usuario, determinar si es posible que realice una misión de transporte de suministros desde el planeta Tatooine hasta su luna Ghomrassen. Será necesario tener en cuenta la capacidad de carga de las naves de transporte disponibles, además de llevar una debida escolta de protección. Las naves de la flota seleccionadas para ir a la misión, independientemente de su tipo, deberán tener las prestaciones suficientes para superar la fuerza gravitatoria de Tatooine, y alcanzar el espacio exterior.

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

Revisad el algoritmo desarrollado parcialmente que se expone a continuación. 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. La cadena **{...}** os indica

que ahí irá el código necesario para completar el ejercicio. **No es necesario realizar ninguna entrega relacionada con este apartado**.

```
const
    MIN_SHIPS: integer = 3;               {Min. number of ships}
    MAX_SHIPS: integer = 10;              {Max. number of ships}
    TATOOINE_GRAVITY: real = 9.8;         {Gravity of the Tatooine planet, in m/s²}
    ENGINE_THRUST_FORCE: real = 15000.0;  {Engine thrust force, in Newtons}
end const

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

    algorithm UOCGalacticEmpire
    {Variable definitions}
    var
        shipTypes: vector[MAX_SHIPS] of tShipType;          {Ships type}
        shipIsInterplanetary: vector[MAX_SHIPS] of boolean; {Interplanetary capabilities}
        shipEngines: vector[MAX_SHIPS] of integer;          {Ships engines}
        shipWeights: vector[MAX_SHIPS] of real;             {Empty ships weight (KG)}
        shipPayloads: vector[MAX_SHIPS] of real;            {Ships payload (KG)}

        i: integer;
        numShips: integer;           {Number of ships}
        loadWeight: real;            {Load weight to transported to Ghomrassen}
        isMissionPossible: boolean;  {Notify if the mission is possible}
        availableLoadCapacity: real; {Available load capacity}
        numFighters: integer;        {Number of available fighter ships}
        {...}
    end var

    {Exercise 2.1}
    {Data input}
    writeString("INPUT DATA");
    writeString("NUMBER OF SHIPS(3-10)?");
    {...}

    {Data validation}
    {...}
    {Exercise 2.2}
    {Data input}
    for i := 1 to numShips do
        writeString("SHIP #");
        writeInteger(i);
        {...}
    end for

    {Read load weight to transport}
    writeString("LOAD WEIGHT [KG] (A REAL)?");
    {...}

    {Exercise 2.3}
    {Data processing}
    numFighters := 0;
    availableLoadCapacity := 0.0;

    {...}

    {Searching for valid Ships}
    {...}

    {Data processing and Data Outputs}
    {Exercise 2.4}
    {Expression to check if it is possible to complete the mission}
    isMissionPossible := {...}

    writeString("RESULTS");
    writeString("THE MISSION IS POSSIBLE (0-FALSE, 1-TRUE):");
    {...}
```

```
end algorithm
```

2. Diseño algorítmico

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

2.1. Lectura de datos. El algoritmo ha de leer por el canal estándar de entrada, la siguiente información necesaria para la aplicación:

- El número de naves que se darán de alta.

Para la lectura y validación de esta información, se aplicará la siguiente regla:

- El número de naves debe **estar comprendido entre 3 y 10.** En caso contrario, debe mostrarse el siguiente mensaje de error y volver a pedir el dato.

  INVALID DATA, TRY AGAIN!

- El proceso se **repetirá indefinidamente** hasta que el usuario introduzca un valor válido.

Este apartado se encuentra desarrollado parcialmente en el algoritmo del enunciado.

2.2. Lectura de datos. El algoritmo debe leer por el canal estándar de entrada, los datos **de cada una de las naves a registrar**: el tipo de nave, la capacidad interplanetaria, el número de motores, el peso y la capacidad de carga. Cada uno de estos datos se almacenará en el **vector** correspondiente:

Variable	Descripción	Tipo / validación
shipTypes	Conjunto de datos que guarda el tipo de la nave.	Vector de valores de tipo tShipType.
shipIsInterplanetary	Conjunto de datos que indica si la nave tiene capacidades interplanetarias.	Vector de valores de tipo booleano.
shipEngines	Conjunto de datos que guarda el número de motores que dispone cada nave.	Vector de valores de tipo entero.
shipWeights	Conjunto de datos que guarda el peso neto de cada nave, en KG.	Vector de valores de tipo real.

shipPayloads	Conjunto de datos que guarda el peso de la carga útil de cada nave, en KG.	Vector de valores de tipo real.

En este punto de desarrollo del algoritmo, el número de naves a introducir es un dato conocido y válido, que ha sido tratado en el apartado anterior.

En el proceso de lectura de datos de este apartado, se presupondrá que el usuario, al entrar los datos, respetará el tipo de datos, rango de valores esperado o conjunto de valores posibles y, por lo tanto, ***no será necesario realizar ninguna validación al respecto****.*

Este apartado se encuentra desarrollado parcialmente en el algoritmo del enunciado.

2.3. Procesamiento de datos. Completar el algoritmo de acuerdo con el siguiente flujo de ejecución y restricciones:

- Se introducirá por el canal estándar de entrada el **peso total** de todos los suministros que deberán cargar entre todas las naves de transporte de la flota disponible.
- Seguidamente, se deberán contabilizar las naves **con capacidad de realizar la misión**. Una nave podrá realizar la misión si cumple las siguientes condiciones:
    - Tiene capacidad interplanetaria.
    - Puede superar la fuerza gravitacional del planeta Tatooine. Será posible si se cumple que:
      Fuerza empuje motor * **número motores** > **fuerza gravitacional**, siendo la **fuerza gravitacional** = peso total nave * gravedad Tatooine.

- Tened en cuenta la capacidad total de carga disponible entre **todas las naves de transporte seleccionadas** ya que será determinante para saber la carga que puede transportar toda la flota en su conjunto.

En notación algorítmica, los tipos de datos enumerados no tienen una correspondencia numérica, y por este motivo no pueden leerse ni escribirse como si fueran enteros. En su lugar, en esta PR podemos usar la/s siguiente/es función/es, que podemos considerar definida/s "adhoc":

```
{...}
writeString("TYPE (...)?");
shipType := readShipType();
{...}
```

```
writeString("TYPE (...):");
writeShipType(shipType);
{...}
```

Este apartado se encuentra desarrollado parcialmente en el algoritmo del enunciado.

2.4. Procesamiento y salida de datos. La flota disponible para esta misión estará formada únicamente por las naves que tienen **capacidad para realizar la misión**. La flota podrá completar la misión si se cumplen las siguientes condiciones:

- La capacidad de carga de la flota tiene que ser igual o superior al peso total de los suministros a enviar a la luna Ghomrassen. La capacidad de carga de la flota es la suma de las capacidades de carga de **todas sus naves de transporte**.
- Como mínimo el 50% del total de naves que componen la flota tienen que ser cazas, con el objetivo de garantizar la seguridad de la flota.

El uso de una expresión sería la forma más óptima de implementar este procesamiento.

La salida informará si la misión es o no posible de la forma siguiente:

```
RESULTS
THE MISSION IS POSSIBLE (0-FALSE, 1-TRUE): 1
```

Este apartado se encuentra desarrollado parcialmente en el algoritmo del enunciado.

Para guardar la información, podrán declararse las variables, constantes y tipos de datos que se consideren necesarios, pudiéndose combinar con las que ya están declaradas en el algoritmo parcial del enunciado.

Hay que evitar siempre que sea posible el uso de valores numéricos directos en el algoritmo, y utilizar en su lugar constantes previamente definidas.

3. Codificación en C

Codificar en C el algoritmo diseñado anteriormente.

El programa en C debe cumplir con las siguientes particularidades:

- Los números reales deben mostrarse con una precisión de dos decimales.
- Los tipos enumerados en C, tienen una correspondencia numérica. Por este motivo, podrán leerse y mostrarse como si fueran enteros, siguiendo las reglas establecidas y siempre mediante la ayuda de una interfaz de usuario para interpretar los datos a leer/mostrar.

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.

Ejemplo de ejecución:

```
INPUT DATA
NUMBER OF SHIPS (3-10)?
3
SHIP #1
SHIP TYPE (TRANSPORT=1, FIGHTER=2, MEDICAL=3, EXPLORER=4)?
2
IS INTERPLANETARY (0-FALSE, 1-TRUE)?
1
ENGINES (AN INTEGER)?
2
WEIGHT [KG] (A REAL)?
1500.0
PAYLOAD [KG] (A REAL)?
1000.0
SHIP #2
SHIP TYPE (TRANSPORT=1, FIGHTER=2, MEDICAL=3, EXPLORER=4)?
2
IS INTERPLANETARY (0-FALSE, 1-TRUE)?
1
ENGINES (AN INTEGER)?
2
WEIGHT [KG] (A REAL)?
1500.0
PAYLOAD [KG] (A REAL)?
1000.0
SHIP #3
SHIP TYPE (TRANSPORT=1, FIGHTER=2, MEDICAL=3, EXPLORER=4)?
1
IS INTERPLANETARY (0-FALSE, 1-TRUE)?
1
ENGINES (AN INTEGER)?
4
WEIGHT [KG] (A REAL)?
2000.0
PAYLOAD [KG] (A REAL)?
4000.0
LOAD WEIGHT TO TRANSPORT [KG] (A REAL)
3000.0
RESULTS
THE MISSION IS POSSIBLE (0-FALSE, 1-TRUE): 1
```