Fundamentos de Programación
PR2 - 20232

Fecha límite de entrega: **08/05/2024 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 2: Seleccionar de entre las dos naves asignadas en un puesto avanzado del Imperio, aquella que pueda realizar una misión de exploración determinada por la distancia al objetivo. La nave seleccionada deberá tener la capacidad de transportar un equipo de reconocimiento compuesto por un número de efectivos.

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. **No es necesario realizar ninguna entrega relacionada con este apartado**.

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

type
    tShip = record
        name: string;              {Ship name}
        shipType: tShipType;       {Ship type}
        isInterplanetary: boolean; {Interplanetary capabilities}
        autonomy: real;            {Autonomy, max range}
        maxSpeed: real;            {Max speed}
        troopCapacity: integer;    {Capacity of troops it can be transport}
    end record
end type

{Exercise 2.1}
{...}

{Exercise 2.2}
{...}

{Exercise 2.3}
{...}

{Exercise 2.4}
{...}

algorithm UOCGalacticEmpire
    {Variable definitions}
    var
        ship1, ship2: tShip;
        isValidShip1, isValidShip2: boolean;
        distanceTarget: real;
        teamSize: integer;
    end var

    {Exercise 2.5}
    {Data input Ship1}
    writeString("ENTER DATA FOR SHIP 1");
    {...}

    writeString("DISTANCE TO TARGET [KM]?");
    distanceTarget := readReal();
    {...}

    {Exercise 2.6}
    {Data processing and Data Output}

    writeString("RESULT");
    writeString("SELECTED SHIP:");
    {...}

    isValidShip1 := isValidShip(ship1, distanceTarget, teamSize);
    {...}

    if isValidShip1 and isValidShip2 then
        {...}
    end if
end algorithm
```

Definición de tipos de datos.

- El tipo de dato estructurado *tShip* representa a una nave perteneciente a la flota del Imperio Galáctico. Los campos son los siguientes:

Campo	Descripción	Tipo / validación
name	Nombre de la nave, sin espacios en blanco	Valor de tipo cadena de caracteres.
shipType	Tipo de nave	Valor de tipo tShipType.
isInterplanetary	Capacidad interplanetaria de la nave.	Valor de tipo booleano.
autonomy	Autonomía operativa de la nave.	Valor de tipo real
maxSpeed	Máxima velocidad.	Valor de tipo real.
troopCapacity	Capacidad de tropas transportadas.	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.

2. Diseño algorítmico

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

2.1. Desarrollo de funciones/acciones. Desarrollar la acción *readShip* que reciba como parámetro una variable de tipo *tShip* sin inicializar, y devuelva la misma variable con los campos informados con los datos leídos por el canal estándar de entrada.

En este caso se presupondrá que el usuario respetará el tipo de datos, rango de valores esperado o conjunto de valores posibles, y, por lo tanto, **no será necesario realizar ninguna comprobación al respecto**.

A continuación se da la descripción del parámetro.

Parámetro	Tipo	Clase	Descripción
ship	*tShip*	Salida	Salida - los campos del parámetro *ship* han de ser inicializados con los datos

	leídos por el canal estándar de entrada.

El parámetro *ship* representa los datos básicos de una nave definida mediante el tipo estructurado *tShip*. Por ello, para leer los datos de la nave, hay que leer todos los campos que forman parte de la tupla.

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

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

2.2. Desarrollo de funciones/acciones. Desarrollar la acción *writeShip*. Esta recibe como parámetro de entrada una variable de tipo *tShip* ya inicializada y muestra por el canal estándar de salida la siguiente información:

- Nombre de la nave.
- Tipo de nave.
- Si dispone de capacidades interplanetarias.
- Autonomía (Km).
- Máxima velocidad (Km/h).
- Capacidad de tropas transportadas.

A continuación se da la descripción del parámetro.

Parámetro	Tipo	Clase	Descripción
ship	*tShip*	Entrada	Parámetro que representa una nave previamente inicializada a mostrar por el canal de salida estándar.

El parámetro *ship* representa los datos de una nave mediante el tipo estructurado *tShip*. Por ello, para mostrar los datos de la nave, hay que mostrar todos los campos que forman parte de la variable.

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

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

2.3. Desarrollo de funciones/acciones. Desarrollar la función *isValidShip* que recibe como parámetro una nave, una distancia hacia un objetivo, y un número de tropas necesarias para realizar una misión. Retornará verdadero si una nave es válida para una misión de exploración según los siguientes criterios:

- Las naves válidas deberán ser de tipo EXPLORER o TRANSPORT.
- Debe tener suficiente autonomía para alcanzar la distancia al objetivo y volver.
- Debe tener la suficiente capacidad de transporte de tropas para el equipo de reconocimiento empleado en la misión.

En caso de no cumplirse alguno de los anteriores criterios, retorna falso.

A continuación se da la descripción de los parámetros y el valor retornado.

Parámetro	Tipo	Clase	Descripción
ship	*tShip*	Entrada	Nave a comprobar su validez.
distance	*real*	Entrada	Distancia hasta el objetivo.
troops	*entero*	Entrada	Número de tropas del equipo de reconocimiento empleado en la misión.
	booleano	Valor de retorno de la función	True si la nave es válida, False en caso contrario.

2.4. Desarrollo de funciones/acciones. Desarrollar la función *bestShip* que recibe como parámetro dos naves (*ship1* y *ship2*) y las compara según los siguientes criterios y en el orden indicado:

- Una nave de tipo EXPLORER es mejor.
- En caso de empate del punto anterior, la que tiene más autonomía es mejor.
- En caso de empate del punto anterior, la que tiene más velocidad es mejor.
- En caso de empate del punto anterior, la que es interplanetaria es mejor.

En caso de haber coincidencia en el primer criterio (ambas son de tipo EXPLORER), se pasará al siguiente criterio y así sucesivamente.

A continuación se da la descripción de los parámetros y el valor retornado.

Parámetro	Tipo	Clase	Descripción
ship1	tShip	Entrada	Primera nave a comparar.
ship2	tShip	Entrada	Segunda nave a comparar.
	entero	Valor de retorno de la función	1 si ship1 es mejor que ship2. 0 si ship1 es igual que ship2. -1 si ship2 es mejor que ship1.

2.5. Lectura de datos. Leer, por el canal estándar de entrada, los datos de dos naves, la distancia hasta el objetivo a explorar, y el número de tropas de reconocimiento a transportar.

2.6. Procesamiento y salida de datos. Mostrar la nave más adecuada para realizar la misión de exploración. Si ambas naves son válidas, seleccionar la mejor de ellas, y si ambas son iguales, la primera introducida será la seleccionada.

Mostrar por el canal de salida los datos de la nave seleccionada. En caso de no ser válida ninguna de las dos naves, mostrar el siguiente mensaje:

SELECTED SHIP:
NOT AVAILABLE

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

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.

Solución (algoritmo completo)

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

type
    tShip = record
        name: string;              {Ship name}
        shipType: tShipType;       {Ship type}
        isInterplanetary: boolean; {Interplanetary capabilities}
        autonomy: real;            {Autonomy, max range}
        maxSpeed: real;            {Max speed}
        troopCapacity: integer;    {Capacity of troops it can be transport}
    end record
end type

{Exercise 2.1}
action readShip(out ship: tShip)
    {Input data}
    writeString("NAME (A STRING)?");
    ship.name := readString();
    writeString("SHIP TYPE (TRANSPORT=1, FIGHTER=2, MEDICAL=3, EXPLORER=4)?");
    ship.shipType := readShipType();
    writeString("IS INTERPLANETARY (0-FALSE, 1-TRUE)?");
    ship.isInterplanetary := readBoolean()
    writeString("AUTONOMY [KM] (A REAL)?");
    ship.autonomy := readReal();
    writeString("MAX SPEED [KM/H](A REAL)?");
    ship.maxSpeed := readReal();
    writeString("TROOP CAPACITY (AN INTEGER)?");
    ship.troopCapacity := readInteger();
end action

{Exercise 2.2}
action writeShip(in ship: tShip)
    {Output data}
    writeString("NAME:");
    writeString(ship.name);
    writeString("SHIP TYPE (TRANSPORT=1, FIGHTER=2, MEDICAL=3, EXPLORER=4:");
    writeShipType(ship.shipType);
    writeString("IS INTERPLANETARY (0-FALSE, 1-TRUE):");
    writeBoolean(ship.isInterplanetary);
    writeString("AUTONOMY [KM]:");
    writeReal(ship.autonomy);
    writeString("MAX SPEED [KM/H]:");
    writeReal(ship.maxSpeed);
    writeString("TROOP CAPACITY:");
    writeInteger(ship.troopCapacity);
end action

{Exercise 2.3}
function isValidShip(ship: tShip, distance: real, troops: integer): boolean
   return (ship.shipType = EXPLORER or ship.shipType = TRANSPORT) and
       ship.autonomy ≥ distance * 2.0 and  ship.troopCapacity ≥ troops;
end function

{Exercise 2.4}
function bestShip(ship1: tShip, ship2: tShip): integer
    var
        result: integer;
    end var
    result := 0
    if ship1.shipType = EXPLORER and ship2.shipType ≠ EXPLORER then
        result := 1
    else
        if ship1.shipType ≠ EXPLORER and ship2.shipType = EXPLORER then
            result := -1;
```

```
        else
            if ship1.autonomy > ship2.autonomy then
                result := 1;
            else
                if ship1.autonomy < ship2.autonomy then
                    result := -1;
                else
                    if ship1.maxSpeed > ship2.maxSpeed then
                        result := 1;
                    else
                        if ship1.maxSpeed < ship2.maxSpeed then
                            result := -1;
                        else
                            if ship1.isInterplanetary and not ship2.isInterplanetary then
                                result := 1;
                            else
                                if not ship1.isInterplanetary and ship2.isInterplanetary then
                                    result := -1;
                                end if
                            end if
                        end if
                    end if
                end if
            end if
        end if
    end if
    return result;
end function

algorithm UOCGalacticEmpire
    {Variable definitions}
    var
        ship1, ship2: tShip;
        isValidShip1, isValidShip2: boolean;
        distanceTarget: real;
        teamSize: integer;
    end var

    {Exercise 2.5}
    {Data input Ship1}
    writeString("ENTER DATA FOR SHIP 1");
    readShip(ship1);
    {Data input Ship2}
    writeString("ENTER DATA FOR SHIP 2");
    readShip(ship2);

    writeString("DISTANCE TO TARGET [KM]?");
    distanceTarget := readReal();
    writeString("RECON TEAM SIZE?");
    teamSize := readInteger();

    {Exercise 2.6}
    {Data processing and Data Output}

    writeString("RESULT");
    writeString("SELECTED SHIP:");

    isValidShip1 := isValidShip(ship1, distanceTarget, teamSize);
    isValidShip2 := isValidShip(ship2, distanceTarget, teamSize);

    if isValidShip1 and isValidShip2 then
        if bestShip(ship1, ship2) ≥ 0
            writeShip(ship1);
        else
            writeShip(ship2);
        end if
    else
        if isValidShip1 then
            writeShip(ship1);
        else
            if isValidShip2 then
```

```
                writeShip(ship2);
            else
                writeString("NOT AVAILABLE");
            end if
        end if
    end if
end algorithm
```

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:

```
ENTER DATA FOR SHIP 1
NAME (15 CHAR MAX, NO SPACES)?
SHIP1
SHIP TYPE (TRANSPORT=1, FIGHTER=2, MEDICAL=3, EXPLORER=4)?
4
IS INTERPLANETARY (0-FALSE, 1-TRUE)?
1
AUTONOMY [KM] (A REAL)?
2000
MAX SPEED [KM/H] (A REAL)?
1500
TROOP CAPACITY (AN INTEGER)?
20
ENTER DATA FOR SHIP 2
NAME (15 CHAR MAX, NO SPACES)?
SHIP2
SHIP TYPE (TRANSPORT=1, FIGHTER=2, MEDICAL=3, EXPLORER=4)?
4
IS INTERPLANETARY (0-FALSE, 1-TRUE)?
1
AUTONOMY [KM] (A REAL)?
2000
MAX SPEED [KM/H] (A REAL)?
1700
TROOP CAPACITY (AN INTEGER)?
20
DISTANCE TO TARGET [KM]?
500
RECON TEAM SIZE?
15
RESULT
SELECTED SHIP:
NAME: SHIP2
SHIP TYPE (TRANSPORT=1, FIGHTER=2, MEDICAL=3, EXPLORER=4): 4
IS INTERPLANETARY (0-FALSE, 1-TRUE): 1
AUTONOMY [KM]: 2000.00
MAX SPEED [KM/H]: 1700.00
TROOP CAPACITY: 20
```