*PRÁCTICA 11*

# *// EJ 1*

**program {**

**unaQueEmpieza := [Norte]**

**unaRutaCualquiera := [Norte, Oeste, Oeste, Este]**

**unaRutaDerecha := [Sur, Sur, Sur]**

**unaRutaQueGiraADerecha := [Este, Oeste, Norte]**

**unaRutaQueGiraAIzquierda := [Oeste, Este, Sur]**

**unaRutaQueGiraEnCurva := [Norte, Sur, Este]**

**return (**

**haySiguienteTramoEn\_(unaRutaCualquiera)**

**)**

**}**

**function haySiguienteTramoEn\_(unaRuta) {**

**/\***

**Prop.: Indica si unaRuta tiene al menos un tramo completo**

**Obs. de repres.:**

**tomamos como un tramo completo,**

**la presencia de dos elementos en**

**la lista de direcciones.**

**Prec.: Ninguna**

**Param.: unaRuta. Lista de direcciones**

**La lista con las direcciones**

**hacia donde se sigue desde un punto**

**hacia la siguiente posicion.**

**Tipo: Booleano**

**\*/**

**return (**

**not esVacía(**

**resto(unaRuta)**

**)**

**)**

**}**

**function sigueRectaEn\_(unaRuta){**

**/\***

**Prop.: Indica si el siguiente tramo de "unaRuta" sigue en linea recta**

**Prec.: Debe haber un siguiente tramo en "unaRuta"**

**Param.: unaRuta. Lista de direcciones**

**La lista con las direcciones**

**hacia donde se sigue desde un punto**

**hacia la siguiente posicion.**

**\*/**

**return(**

**primero(unaRuta) ==**

**primero(resto(unaRuta)**

**)**

**)**

**}**

**function sigueCurvaADerechaEn\_(unaRuta){**

**/\***

**Prop.:**

**Prec.:**

**Param.: unaRuta. Lista de direcciones**

**La lista con las direcciones**

**hacia donde se sigue desde un punto**

**hacia la siguiente posicion.**

**\*/**

**return (**

**siguiente(primero(unaRuta)) == primero(resto(unaRuta))**

**)**

**}**

**function sigueCurvaAIzquierdaEn\_(unaRuta){**

**/\***

**Prop.:**

**Prec.:**

**Param.: unaRuta. Lista de direcciones**

**La lista con las direcciones**

**hacia donde se sigue desde un punto**

**hacia la siguiente posicion.**

**\*/**

**return (**

**siguiente(primero(unaRuta)) == primero(resto(unaRuta))**

**)**

**}**

**function sigueUnaCurvaEn\_(unaRuta){**

**/\***

**Prop.:**

**Prec.:**

**Param.: unaRuta. Lista de direcciones**

**La lista con las direcciones**

**hacia donde se sigue desde un punto**

**hacia la siguiente posicion.**

**\*/**

**return (**

**siguiente(primero(unaRuta)) == primero(resto(unaRuta))**

**)**

**}**

# *// EJ 2*

**type Palo is variant {**

**/\***

**Prop.**

**Modelar un palo de carta.**

**\*/**

**case Basto {}**

**case Copa {}**

**case Espada {}**

**case Oro {}**

**}**

**type Carta is record {**

**/\***

**Prop.**

**Modelar una carta de la baraja española.**

**Inv. Rep.**

**El campo numero tiene que estar comprendido entre 1 y 12 inclusive.**

**\*/**

**field número // Numero**

**field palo // Palo**

**}**

**function anchoDeEspadas() {**

**/\***

**Prop. Describe el 1 de espadas.**

**PRec. Ninguna**

**Tipo: Carta.**

**el CONSTRUCTOR, no es funcion constructora.**

**\*/**

**return (**

**Carta (**

**numero <- 1,**

**palo <- Espada**

**)**

**)**

**}**

**function anchoDeBasto() {**

**/\***

**Prop. Describe el 1 de Basto.**

**PRec. Ninguna**

**Tipo: Carta.**

**el CONSTRUCTOR, no es funcion constructora.**

**\*/**

**return (**

**Carta (**

**numero <- 1,**

**palo <- Basto**

**)**

**)**

**}**

**function sieteDeOro() {**

**/\***

**Prop. DESCRIBE EL 7 de oro.**

**PRec. Ninguna**

**Tipo: Carta.**

**el CONSTRUCTOR, no es funcion constructora.**

**\*/**

**return (**

**Carta (**

**numero <- 7,**

**palo <- Oro**

**)**

**)**

**}**

**function nueveDeBasto() {**

**/\***

**Prop. Describe el 9 de basto.**

**PRec. Ninguna**

**Tipo: Carta.**

**el CONSTRUCTOR, no es funcion constructora.**

**\*/**

**return (**

**Carta (**

**numero <- 9,**

**palo <- Basto**

**)**

**)**

**}**

**function laCarta\_De\_(numeroDeCarta, paloDeCarta) {**

**/\***

**Prop. Describe la carta \*numeroDeCarta\* de \*paloDeCarta\*.**

**Prec. 12 >= \*numeroDeCarta\* > 0**

**Param. \*numeroDeCarta\*. Numero. Numero de la carta.**

**\*paloDeCarta\*. Palo. Palo de la carta.**

**Tipo: Carta.**

**\*/**

**return(**

**Carta(**

**numero <- numeroDeCarta,**

**palo <- paloDeCarta**

**)**

**)**

**}**

**program {**

**mano1 := [anchoDeEspadas()]**

**mano2 := [anchoDeBasto()]**

**mano3 := [sieteDeOro()]**

**cambioDeMano := [nueveDeBasto()]**

**return (terceraCartaDeLaMano\_(mano3))**

**}**

**function primerCartaDeLaMano\_(mano1) {**

**/\***

**Proposito: Describe la primera carta de la mano \*mano1\*.**

**Precondiciones: ninguna.**

**Parametros: \*mano1\*.**

**Tipo. Carta.**

**\*/**

**return (anchoDeEspadas())**

**}**

**function segundaCartaDeLaMano\_(mano2) {**

**/\***

**Proposito: Describe la segunda carta de la mano \*mano2\*.**

**Precondiciones: ninguna.**

**Parametros: \*mano2\*.**

**Tipo. Carta.**

**\*/**

**return (anchoDeBasto())**

**}**

**function terceraCartaDeLaMano\_(mano3) {**

**/\***

**Proposito: Describe la segunda carta de la mano \*mano2\*.**

**Precondiciones: ninguna.**

**Parametros: \*mano2\*.**

**Tipo. Carta.**

**\*/**

**return (sieteDeOro())**

**}**

**function laMano\_LuegoDeRobarUnaCartaDe\_(mano3, cambioDeMano) {**

**/\***

**Proposito:**

**Precondiciones:**

**Parametros:**

**Tipo.**

**\*/**

**return ()**

**}**

# *// EJ 3*

**function listaDeCantidadesDeRojasDeLaFila(dirección) {**

**/\***

**Proposito: Describe una lista con las cantidades de bolitas rojas**

**que hay en la fila, donde esta ubicado el cabezal, de Oeste a Este.**

**Precondiciones: El cabezal esta sobre la fila en cuestion.**

**Parametros: \*dirección\*: Dirección - La dirección a mover**

**Tipo. Lista.**

**Obs. Recorrido sobre una fila.**

**\*/**

**IrAlBorde(Opuesto(dirección))**

**listaDeRojas := [nroBolitas(Rojo)] //con corchetes se transforma en lista.**

**while (puedeMover(dirección)) {**

**Mover(dirección)**

**listaDeRojas := listaDeRojas ++ [nroBolitas(Rojo)]**

**}**

**return (listaDeRojas)**

**}**

# *// EJ 4*

**function coloresEnCeldaActual(){**

**/\***

**Prop.: Describe una lista de colores estan en la celda actual**

**Prec.: Debe haber una lista de colores para describir**

**Tipo: Color**

**\*/**

**listaColores := []**

**color := minColor()**

**listaColores := listaColores ++ colorSiHay\_(color)**

**while(color /= maxColor()) {**

**color := siguiente(color)**

**listaColores := listaColores ++ colorSiHay\_(color)**

**}**

**return(listaColores)**

**}**

**function colorSiHay\_(color){**

**/\***

**Prop.: Describe si hay un color en la celda actual**

**Prec.: Ninguna**

**Tipo: Color**

**\*/**

**return (**

**choose**

**[color] when (hayBolitas(color))**

**[] otherwise**

**)**

**}**

**program{**

**return(coloresEnCeldaActual())**

**}**

# *// EJ 5*

**function listaDeColoresEnCeldasDelTablero(){**

**/\***

**Prop.: Describe una lista de colores del Tablero**

**Prec.: Debe haber solo una bolita de color en cada celda del tablero**

**Tipo: Color**

**\*/**

**listaResultante := []**

**IrAlInicioDelRecorridoAl\_Y\_(Norte, Este)**

**while (haySiguienteCeldaEnUnRecorridoAl\_Y\_(Sur, Oeste)){**

**listaResultante := listaResultante ++ [colorDeLaCelda()]**

**IrASiguienteCeldaEnUnRecorridoAl\_Y\_(Sur, Oeste)**

**}**

**return(listaResultante)**

**}**

**program{**

**return(listaDeColoresEnCeldasDelTablero())**

**}**

# *// EJ 6*

**function aparicionesDeColor\_(colorBuscado){**

**/\***

**Prop.: describe una lista de números que indican para cada una de las celdas del tablero la cantidad de bolitas del color buscado en esa celda del tablero recorridas al \*dirPrincipal\* y \*dirSecundaria\***

**Prec.:**

**Param.:**

**\*/**

**listaNumeros := []**

**IrAlInicioDelRecorridoAl\_Y\_(Sur, Oeste)**

**while (haySiguienteCeldaEnUnRecorridoAl\_Y\_(Norte, Este)){**

**listaNumeros := listaNumeros ++ [numeroBolitasCeldaActual\_(numero, colorBuscado)]**

**IrASiguienteCeldaEnUnRecorridoAl\_Y\_(Norte, Este)**

**}**

**return(listaNumeros)**

**}**

**function numeroBolitasCeldaActual\_(numero, colorBuscado) {**

**/\***

**Prop.: Describe la cantidad de bolitas de un color dado en la celda actual**

**Prec.: Ninguna**

**Tipo: Número**

**\*/**

**return(**

**choose**

**[número] when (hayBolitas(colorBuscado))**

**[0] otherwise**

**)**

**}**

**program {**

**return(aparicionesDeColor\_(Rojo))**

**}**

# *// EJ 7*

**function direccionesAlBorde() {**

**/\***

**Prop.: Describe una lista de direcciones en las que el cabezal no se puede mover**

**Prec.: Ninguna**

**Tipo: Dirección**

**\*/**

**listaDirecciones := []**

**return(**

**choose**

**[Este, Sur] when (not puedeMover(Oeste) && not puedeMover(Norte))**

**[Este, Norte] when (not puedeMover(Oeste) && not puedeMover(Sur))**

**[Norte, Oeste] when (not puedeMover(Sur) && not puedeMover(Este))**

**[Norte, Sur] otherwise**

**)**

**}**

**program {**

**return(direccionesAlBorde())**

**}**

# *// EJ 8*

# *// EJ 9*

# 

# *// EJ 12*

# *// EJ 13*

**function reversoDe\_(unaLista){**

**/\***

**Propósito: Describe el reverso de la lista dada, es decir, la lista dada vuelta.**

**Parámetros:**

**\* unaLista: Lista de elementos - La lista a describir el reverso.**

**Precondición: Ninguna.**

**Tipo: Lista de elementos.**

**\*/**

**listaResto := unaLista**

**listaReversa := [primero(listaResto)]**

**listaResto := resto(listaResto)**

**while(not esVacía(listaResto)){**

**listaReversa := [primero(listaResto)] ++ listaReversa**

**listaResto := resto(listaResto)**

**}**

**return(listaReversa)**

**}**

# *// EJ 14*

## *BIBLIOTECA*

### A)

**program {**

**unaLista := [1, 2, 4]**

**return(**

**longitudDe\_(unaLista)**

**)**

**}**

**function longitudDe\_(unaLista){**

**/\***

**Propósito: Describe la cantidad de elementos de la lista \*\*unaLista\*\*.**

**Parámetros:**

**\* unaLista: Lista de elementos - La lista a saber su cantidad de elementos.**

**Precondición: Ninguna.**

**Tipo: Número.**

**\*/**

**restoLista := unaLista**

**cantidad := 0**

**while(not esVacía(restoLista)){**

**cantidad := cantidad + 1**

**restoLista := resto(restoLista)**

**}**

**return(cantidad)**

**}**

### B)

**function sumatoriaDe\_(unaListaDeNúmeros){**

**/\***

**Propósito: Describe la suma de todos los elementos de la lista \*\*unaListaDeNúmeros\*\*.**

**Parámetros:**

**\* unaListaDeNúmeros: Lista de números - La lista de números a hacerle la sumatoria.**

**Precondición: Ninguna.**

**Tipo: Número.**

**\*/**

**sumatoria := 0**

**restoLista := unaListaDeNúmeros**

**while(not esVacía(restoLista)){**

**sumatoria := sumatoria + primero(restoLista)**

**restoLista := resto(restoLista)**

**}**

**return(sumatoria)**

**}**

### C)

**function productoriaDe\_(unaListaDeNúmeros){**

**/\***

**Propósito: Describe el producto de todos los elementos de la lista \*\*unaListaDeNúmeros\*\*.**

**Parámetros:**

**\* unaListaDeNúmeros: Lista de números - La lista de números a hacerle la productoria.**

**Precondición: Ninguna.**

**Tipo: Número.**

**\*/**

**productoria := 1**

**restoLista := unaListaDeNúmeros**

**while(not esVacía(restoLista)){**

**productoria := productoria \* primero(restoLista)**

**restoLista := resto(restoLista)**

**}**

**return(productoria)**

**}**

# *// EJ 15*

**function direccionesOpuestasDe\_(unaListaDeDirecciones){**

**/\***

**Propósito: Describe una lista de direcciones donde cada elemento es el opuesto al de la posición original.**

**Parámetros:**

**\* unaListaDeDirecciones: Lista de direcciones - La lista de direcciones a hacerle lista de opuestos.**

**Precondición: Ninguna.**

**Tipo: Lista de direcciones.**

**\*/**

**listaDireccionesOpuestas := []**

**restoLista := unaListaDeDirecciones**

**while(not esVacía(restoLista)){**

**listaDireccionesOpuestas := listaDireccionesOpuestas ++ [opuesto(primero(restoLista))]**

**restoLista := resto(restoLista)**

**}**

**return(listaDireccionesOpuestas)**

**}**

**function siguientesDe\_(unaListaDeColores){**

**/\***

**Propósito: Describe una lista de colores donde cada elemento es el siguiente del original de la lista \*\*unaListaDeColores\*\*.**

**Parámetros:**

**\* unaListaDeColores: Lista de colores - La lista de colores a hacerle una lista de los siguientes.**

**Precondición: Ninguna.**

**Tipo: Lista de colores.**

**\*/**

**listaColoresSiguientes := []**

**restoLista := unaListaDeColores**

**while(not esVacía(restoLista)){**

**listaColoresSiguientes := listaColoresSiguientes ++ [siguiente(primero(restoLista))]**

**restoLista := resto(restoLista)**

**}**

**return(listaColoresSiguientes)**

**}**

**function elementosDe\_multiplicadosPor\_(unaListaDeNúmeros, númeroMultiplicador){**

**/\***

**Propósito: Describe una lista de números donde cada número de la lista \*\*unaListaDeNúmeros\*\* fue multiplicado por \*\*númeroMultiplicador\*\*.**

**Parámetros:**

**\*unaListaDeNúmeros: Lista de números - La lista de números a multiplicar.**

**\*númeroMultiplicador: Número - El número para multiplicar los números de la lista.**

**Precondición: Ninguna.**

**Tipo: Lista de números.**

**\*/**

**listaNúmerosMultiplicados := []**

**restoLista := unaListaDeNúmeros**

**while(not esVacía(restoLista)){**

**listaNúmerosMultiplicados := listaNúmerosMultiplicados ++ [((primero(restoLista))\*númeroMultiplicador)]**

**restoLista := resto(restoLista)**

**}**

**return(listaNúmerosMultiplicados)**

**}**

# *// EJ 16*

**function númerosParesDe\_(unaListaDeNúmeros){**

**/\***

**Propósito: Describe una lista de números de los números pares que aparezcan en la lista \*\*unaListaDeNúmeros\*\*.**

**Parámetros:**

**\* unaListaDeNúmeros: Lista de números - La lista de números a saber los pares.**

**Precondición: Ninguna.**

**Tipo: Lista de números.**

**\*/**

**listaNúmerosPares := []**

**restoLista := unaListaDeNúmeros**

**while(not esVacía(restoLista)){**

**listaNúmerosPares := listaNúmerosPares ++ listaSiEsPar\_(primero(restoLista))**

**restoLista := resto(restoLista)**

**}**

**return(listaNúmerosPares)**

**}**

**function listaSiEsPar\_(número){**

**/\***

**Propósito: Describe una lista con el elemento \*\*número\*\* si es par, si no lo es describe una lista vacía.**

**Parámetros:**

**\* número: Número - El número a saber si es par.**

**Precondición: Ninguna.**

**Tipo: Lista de números.**

**\*/**

**return(**

**choose**

**[número] when ((número mod 2) == 0)**

**[] otherwise**

**)**

**}**

**function laLista\_SinElElemento\_(unaLista, elementoAQuitar){**

**/\***

**Propósito: Describe la lista que resulta de quitar todas las apariciones del elemento \*\*elementoAQuitar\*\* de la lista \*\*unaLista\*\*.**

**Parámetros:**

**\* unaLista : Lista de elementos - La lista de elementos.**

**\* elementoAQuitar: Elemento - El elemento a quitar de la lista.**

**Precondición: Ninguna.**

**Tipo: Lista de elementos.**

**\*/**

**listaDeElementosFinal := []**

**restoLista := unaLista**

**while(not esVacía(restoLista)){**

**listaDeElementosFinal := listaDeElementosFinal ++ listaSiElElemento\_NoEs\_(primero(restoLista), elementoAQuitar)**

**restoLista := resto(restoLista)**

**}**

**return(listaDeElementosFinal)**

**}**

**function listaSiElElemento\_NoEs\_(elementoDeLista, elementoDeComparación){**

**/\***

**Propósito: Describe una lista con el elemento \*\*elementoDeLista\*\* si este no es igual a \*\*elementoDeComparación\*\*, si es igual describe una lista vacía.**

**Parámetros:**

**\* elementoDeLista : Elemento - El elemento de la lista.**

**\* elementoDeComparación: Elemento - El elemento a comparar.**

**Precondición: Ninguna.**

**Tipo: Lista de elementos.**

**\*/**

**return(**

**choose**

**[elementoDeLista] when (elementoDeLista /= elementoDeComparación)**

**[] otherwise**

**)**

**}**

**function losMayoresA\_De\_(umbral, unaLista){**

**/\***

**Propósito: Describe una lista con los elementos de la lista \*\*unaLista\*\* que sean mayores a \*\*umbral\*\*.**

**Parámetros:**

**\* umbral : Elemento - El elemento de umbral.**

**\* unaLista: Lista de elementos - La lista de elementos.**

**Precondición: Ninguna.**

**Tipo: Lista de elementos.**

**\*/**

**listaElementosFinal := []**

**restoLista := unaLista**

**while(not esVacía(restoLista)){**

**listaElementosFinal := listaElementosFinal ++ listaSi\_EsMayorA\_(primero(restoLista), umbral)**

**restoLista := resto(restoLista)**

**}**

**return(listaElementosFinal)**

**}**

**function listaSi\_EsMayorA\_(elemento, umbral){**

**/\***

**Propósito: Describe una lista con el elemento \*\*elemento\*\* si este es mayor a \*\*umbral\*\*, si no lo es describe una lista vacía.**

**Parámetros:**

**\* elemento : Elemento - El elemento a comparar.**

**\* umbral : Elemento - El umbral.**

**Precondición: Ninguna.**

**Tipo: Lista de elementos.**

**\*/**

**return(**

**choose**

**[elemento] when (elemento > umbral)**

**[] otherwise**

**)**

**}**

# *// EJ 17*

**function contiene\_A\_(unaLista, unElemento){**

**/\***

**Propósito: Indica si la lista \*\*unaLista\*\* contiene al elemento \*\*unElemento\*\*.**

**Parámetros:**

**\* unaLista: Lista de elementos - La lista a saber si está el elemento.**

**\* unElemento: Elemento - El elemento a saber si está en la lista.**

**Precondición: Ninguna.**

**Tipo: Booleano.**

**\*/**

**restoLista := unaLista**

**while(not esVacía(restoLista) && primero(restoLista) /= unElemento){**

**restoLista := resto(restoLista)**

**}**

**return(not esVacía(restoLista) && primero(restoLista) == unElemento)**

**}**

**function algunoMayorQué\_En\_(unElemento, unaLista){**

**/\***

**Propósito: Indica si la lista \*\*unaLista\*\* contiene algún elemento que sea mayor a \*\*unElemento\*\*.**

**Parámetros:**

**\* unElemento: Elemento - El elemento a saber si hay uno mayor.**

**\* unaLista : Lista de elementos - La lista de elementos.**

**Precondición: Ninguna.**

**Tipo: Booleano.**

**\*/**

**return(not esVacía(losMayoresA\_De\_(unElemento, unaLista)))**

**}**

**function hayAlgunoDe\_Entre\_Y\_(unaListaDeNúmeros, nroDesde, nroHasta){**

**/\***

**Propósito: Indica si la lista \*\*unaListaDeNúmeros\*\* contiene algún elemento que sea mayor a \*\*nroDesde\*\* y menor que \*\*nroHasta\*\*.**

**Parámetros:**

**\* unaListaDeNúmeros: Lista de números - La lista de elementos a analizar.**

**\* nroDesde : Número - El número desde.**

**\* nroHasta : Número - El número hasta.**

**Precondición: Ninguna.**

**Tipo: Booleano.**

**\*/**

**restoLista := unaListaDeNúmeros**

**while(not esVacía(restoLista) && ((primero(restoLista) < nroDesde) || (primero(restoLista) > nroHasta))){**

**restoLista := resto(restoLista)**

**}**

**return(not esVacía(restoLista) && (primero(restoLista) > nroDesde) && (primero(restoLista) < nroHasta))**

**}**

**function hayAlgúnElementoImparDe\_(unaListaDeNúmeros){**

**/\***

**Propósito: Indica si la lista \*\*unaListaDeNúmeros\*\* contiene algún elemento que sea impar.**

**Parámetros:**

**\* unaListaDeNúmeros: Lista de números - La lista a ver si hay algún impar.**

**Precondición: Ninguna.**

**Tipo: Booleano.**

**\*/**

**restoLista := unaListaDeNúmeros**

**while(not esVacía(restoLista) && ((primero(restoLista) mod 2) == 0)){**

**restoLista := resto(restoLista)**

**}**

**return(not esVacía(restoLista) && ((primero(restoLista) mod 2) /= 0))**

**}**

# *// EJ 18*

**function sinDuplicados\_(unaLista){**

**/\***

**Propósito: Describe una lista de elementos que contenga los elementos de la lista \*\*unaLista\*\* sin repetir.**

**Parámetros:**

**\* unaLista: Lista de elementos - La lista a sacarle los duplicados que haya.**

**Precondición: Ninguna.**

**Tipo: Lista de elementos.**

**\*/**

**listaSinDuplicados := []**

**restoLista := unaLista**

**while(not esVacía(restoLista)){**

**listaSinDuplicados := listaSinDuplicados ++ listaDeElemento\_SiNoEstáEnLista\_(primero(restoLista), listaSinDuplicados)**

**restoLista := resto(restoLista)**

**}**

**return(listaSinDuplicados)**

**}**

**function listaDeElemento\_SiNoEstáEnLista\_(unElemento, unaLista){**

**/\***

**Propósito: Describe una lista de elementos con el elemento \*\*unElemento\*\* si no está en \*\*unaLista\*\*, si ya está en la lista describe una lista vacía.**

**Parámetros:**

**\* unElemento: Elemento - El elemento a ver si está en la lista.**

**\* unaLista : Lista de elementos - La lista a ver si está el elemento.**

**Precondición: Ninguna.**

**Tipo: Lista de elementos.**

**\*/**

**return(**

**choose**

**[unElemento] when (not contiene\_A\_(unaLista, unElemento))**

**[] otherwise**

**)**

**}**

# *// EJ 19*

**function posiciónDe\_enLaQueAparece\_(unaLista, unElemento){**

**/\***

**Propósito: Describe la posición en \*\*unaLista\*\* donde esté el elemento \*\*unElemento\*\*.**

**Parámetros:**

**\* unaLista : Lista de elementos - La lista de elementos a saber la posición del elemento.**

**\* unElemento: Elemento - El elemento a saber su posición.**

**Precondición: Debe estar \*\*unElemento\*\* en la lista \*\*unaLista\*\*.**

**Tipo: Número.**

**\*/**

**posición := 0**

**restoLista := unaLista**

**while(primero(restoLista) /= unElemento){**

**posición := posición + 1**

**restoLista := resto(restoLista)**

**}**

**return(posición)**

**}**

# *// EJ 20*

**function lista\_estáIncluidaEn\_(primerLista, segundaLista){**

**/\***

**Propósito: Indica si la lista \*\*primerLista\*\* se encuentra contenida en la lista \*\*segundaLista\*\*.**

**Parámetros:**

**\* primerLista : Lista de elementos - La lista a saber si está contenida en la otra.**

**\* segundaLista: Lista de elementos - La lista a saber si contiene a la otra.**

**Precondiciones:**

**\* Las dos listas no contienen elementos repetidos.**

**\* La lista \*\*primerLista\*\*\* no puede estar vacía.**

**Tipo: Booleano.**

**\*/**

**restoLista := primerLista**

**while(not esVacía(restoLista) && contiene\_A\_(segundaLista, primero(restoLista))){**

**restoLista := resto(restoLista)**

**}**

**return(esVacía(restoLista) || contiene\_A\_(segundaLista, primero(restoLista)))**

**}**

# *// EJ 21*

## A)

**function intersecciónDe\_Con\_(primerLista, segundaLista){**

**/\***

**Propósito: Describe una lista de todos los elementos que se encuentren a la vez en \*\*primerLista\*\* y en \*\*segundaLista\*\*.**

**Parámetros:**

**\* primerLista : Lista de elementos - La primer lista.**

**\* segundaLista: Lista de elementos - La segunda lista.**

**Precondiciones:**

**\* Las dos listas no contienen elementos repetidos.**

**Tipo: Listas de elementos.**

**\*/**

**listaIntersección := []**

**restoLista := primerLista**

**while(not esVacía(restoLista)){**

**listaIntersección := listaIntersección ++ listaSiContiene\_A\_(segundaLista, primero(restoLista))**

**restoLista := resto(restoLista)**

**}**

**return(listaIntersección)**

**}**

**function listaSiContiene\_A\_(unaLista, unElemento){**

**/\***

**Propósito: Describe una lista con el elemento \*\*unElemento\*\* si este está en la lista \*\*unaLista\*\*.**

**Parámetros:**

**\* unaLista : Lista de elementos - La lista de elementos.**

**\* unElemento: Elemento - El elemento a saber si está en la lista.**

**Precondiciones: Ninguna.**

**Tipo: Listas de elementos.**

**\*/**

**return(**

**choose**

**[unElemento] when (contiene\_A\_(unaLista, unElemento))**

**[] otherwise**

**)**

**}**

## B)

**function uniónDe\_Con\_(primerLista, segundaLista){**

**/\***

**Propósito: Describe una lista sin repetidos que contenga todos los elementos que aparezcan en las listas \*\*primerLista\*\* y en \*\*segundaLista\*\*.**

**Parámetros:**

**\* primerLista : Lista de elementos - La primer lista.**

**\* segundaLista: Lista de elementos - La segunda lista.**

**Precondiciones:**

**\* Las dos listas no contienen elementos repetidos.**

**Tipo: Listas de elementos.**

**\*/**

**listaUnión := primerLista**

**restoLista := segundaLista**

**while(not esVacía(restoLista)){**

**listaUnión := listaUnión ++ listaDeElemento\_SiNoEstáEnLista\_(primero(restoLista), listaUnión)**

**restoLista := resto(restoLista)**

**}**

**return(listaUnión)**

**}**

# *// EJ 22*

## A)

**function mínimoElementoDe\_(unaListaDeNúmeros){**

**/\***

**Propósito: Describe el elemento más chico que se encuentra en la lista \*\*unaListaDeNúmeros\*\*.**

**Parámetros:**

**\* unaListaDeNúmeros : Lista de números - La lista de números a buscar el más chico.**

**Precondiciones:**

**\* Debe haber al menos un elemento en la lista.**

**Tipo: Número.**

**\*/**

**mínimoElemento := primero(unaListaDeNúmeros)**

**restoLista := resto(unaListaDeNúmeros)**

**while(not esVacía(restoLista)){**

**mínimoElemento := mínimoEntre\_Y\_(mínimoElemento, primero(restoLista))**

**restoLista := resto(restoLista)**

**}**

**return(mínimoElemento)**

**}**

## B)

**function sinElMínimoElemento\_(unaListaDeNúmeros){**

**/\***

**Propósito: Describe la lista que se obtiene después de eliminar una única vez el elemento más chico de la lista \*\*unaListaDeNúmeros\*\*.**

**Parámetros:**

**\* unaListaDeNúmeros : Lista de números - La lista de números a eliminar una única vez el más chico.**

**Precondiciones:**

**\* Debe haber al menos un elemento en la lista.**

**Tipo: Lista de números.**

**\*/**

**listaSinElMínimo := []**

**restoLista := unaListaDeNúmeros**

**while(primero(restoLista) /= mínimoElementoDe\_(unaListaDeNúmeros)){**

**listaSinElMínimo := listaSinElMínimo ++ [primero(restoLista)]**

**restoLista := resto(restoLista)**

**}**

**listaSinElMínimo := listaSinElMínimo ++ resto(restoLista)**

**return(listaSinElMínimo)**

**}**

## C)

**function lista\_ordenada(unaListaDeNúmeros){**

**/\***

**Propósito: Describe la lista con los mismos elementos de \*\*unaListaDeNúmeros\*\*, pero ordenada de menor a mayor.**

**Parámetros:**

**\* unaListaDeNúmeros : Lista de números - La lista de números a ordenar de menor a mayor.**

**Precondiciones:**

**\* Debe haber al menos un elemento en la lista.**

**Tipo: Lista de números.**

**\*/**

**listaOrdenada := [mínimoElementoDe\_(unaListaDeNúmeros)]**

**restoLista := sinElMínimoElemento\_(unaListaDeNúmeros)**

**while(not esVacía(restoLista)){**

**listaOrdenada := listaOrdenada ++ [mínimoElementoDe\_(restoLista)]**

**restoLista := sinElMínimoElemento\_(restoLista)**

**}**

**return(listaOrdenada)**

**}**

# *// EJ 23*

## A)

**function máximoElementoDe\_(unaListaDeNúmeros){**

**/\***

**Propósito: Describe el elemento más grande que se encuentra en la lista \*\*unaListaDeNúmeros\*\*.**

**Parámetros:**

**\* unaListaDeNúmeros : Lista de números - La lista de números a buscar el más grande.**

**Precondiciones:**

**\* Debe haber al menos un elemento en la lista.**

**Tipo: Número.**

**\*/**

**máximoElemento := primero(unaListaDeNúmeros)**

**restoLista := resto(unaListaDeNúmeros)**

**while(not esVacía(restoLista)){**

**máximoElemento := máximoEntre\_Y\_(máximoElemento, primero(restoLista))**

**restoLista := resto(restoLista)**

**}**

**return(máximoElemento)**

**}**

## B)

**function sinElMáximoElemento\_(unaListaDeNúmeros){**

**/\***

**Propósito: Describe la lista que se obtiene después de eliminar una única vez el elemento más grande de la lista \*\*unaListaDeNúmeros\*\*.**

**Parámetros:**

**\* unaListaDeNúmeros : Lista de números - La lista de números a eliminar una única vez el más grande.**

**Precondiciones:**

**\* Debe haber al menos un elemento en la lista.**

**Tipo: Lista de números.**

**\*/**

**listaSinElMáximo := []**

**restoLista := unaListaDeNúmeros**

**while(primero(restoLista) /= máximoElementoDe\_(unaListaDeNúmeros)){**

**listaSinElMáximo := listaSinElMáximo ++ [primero(restoLista)]**

**restoLista := resto(restoLista)**

**}**

**listaSinElMáximo := listaSinElMáximo ++ resto(restoLista)**

**return(listaSinElMáximo)**

**}**

# *// EJ 24*

# *// EJ 25*

# *// EJ 26*

**—------------------------------------------------------------------------------------------------------------**