const { Queue } = require("../DS");

// ⚠️ NO MODIFICAR NADA POR ENCIMA DE ESTA LÍNEA ⚠️

// 🛒 HENRY MARKET 🛒

// En el día de inauguración del nuevo supermercado Henry Market, estamos encargados

// del correcto funcionamiento de los programas de cada una de las computadoras que

// realizan las distintas tareas en el negocio.

//

// Cada uno de los ejercicios nos pedirá implementar una función en la que utilizaremos

// lo aprendido en este módulo.

// Leer atentamente cada uno de los enunciados y guiarse por los ejemplos!

//

//

//

//

// 1️⃣ \*\*\*\*\* EJERCICIO 1 \*\*\*\*\* - henryParking() 1️⃣

// Implementar la función henryParking, que nos permitirá organizar los movimientos de

// los vehículos del estacionamiento de Henry Market.

// La función recibirá por parámetro un array que representa en orden las entradas

// y salidas de vehículos que se producen en el estacionamiento, y deberá crear

// una QUEUE, donde iremos registrando los ingresos y egresos.

//

// El array de movimientos tendrá la siguiente forma:

//

// [23, 43, "OUT", 65, "OUT", 32, 55, "OUT"] , donde cada número representa el ingreso

// de un vehículo al que se le asigna dicho valor; y cada "OUT" representa la salida

// del vehículo que lleva más tiempo dentro del estacionamiento.

// Finalmente, la función debe retornar la QUEUE que representa el estado del estacionamiento

// al momento de finalizar los eventos del array.

//

// EJEMPLOS:

//

//  - henryParking([23, 43, "OUT", 65]) => Queue [43, 65]

//      - Ingresó vehículo 23. [23] 🔺

//      - Ingresó vehículo 43. [23, 43] 🔺

//      - Egresó vehículo 23.  [43] 🔻

//      - Ingresó vehículo 65. [43, 65] 🔺

//

// REQUISITOS:

//  🟢 La función debe retornar la QUEUE resultante de procesar los movimientos.

//  🟢 Si la función intenta retirar un vehículo cuando la Queue se encuentra vacía,

//    debe retornar false

//  🟢 ATENCIÓN! La QUEUE que retorna la función debe ser una instancia de la clase QUEUE.

function henryParking(arr) {

   let newQueue = new Queue ();

   for (var i= 0; i <arr.length; i++) {

    if(arr[i] !== "OUT") {

      newQueue.enqueue(arr[i]);

    }

     if(newQueue.size()=== 0 ){

      return false;

     }

    if (arr[i] === "OUT") {

      newQueue.dequeue();

    }

   }

    return newQueue;

     }

// ⚠️ NO MODIFICAR NADA POR DEBAJO DE ESTA LÍNEA ⚠️

module.exports = {

  henryParking

};

// 2️⃣ \*\*\*\*\* EJERCICIO 2 \*\*\*\*\* - calcularIndice() 2️⃣

// En este ejercicio, debemos implementar la función calcularIndice, la debe retornar

// un valor numérico que es parte de una secuencia de números que representa los índices

// de precios que se aplicarán en Henry Market en cada año de actividad.

// La función recibirá por parámetro el año de actividad del supermercado (valor númérico 0 o mayor),

// y calculará el índice en base a la siguiente secuencia:

//

// - Para el año 0 (n=0) de actividad, el índice es de 15

// - Para el año 1 (n=1) de actividad, el índice es 25

// - A partir del año 2, el índice se calcula con la fórmula f(n-1)+f(n-2)

//

// EJEMPLOS:

// - calcularIndice(0) => 15

// - caluclarIndice(1) => 25

// - calcularIndice(2) => calcularIndice(2-1) + calcularIndice(2-2) => 40

// - calcularIndice(3) => calcularIndice(3-1) + calcularIndice(3-2) => 65

//

// REQUISITOS:

//  🟢 La función debe obtener el índice hallando de forma recursiva el valor correspondiente a n dentro de la secuencia.

//  🟢 La función debe retornar el valor numérico correspondiente a n dentro de la secuencia.

//  🟢 Si el valor de n recibido por parámetro es menor a 0, debe retornar false.

const calcularIndice = (n) => {

  // Tu código aquí:

  if (n < 0) return false;

  if(n === 0) return 15;

    else if ( n=== 1) return 25;

        else ( n >= 2)

          return calcularIndice(n-1) + calcularIndice(n-2);

}

const { Stack } = require("../DS");

// ⚠️ NO MODIFICAR NADA POR ENCIMA DE ESTA LÍNEA ⚠️

// 3️⃣ \*\*\*\*\* EJERCICIO 3 \*\*\*\*\* - apilarCajas() 3️⃣

// Se debe implementar la función apilarCajas, la cual ayudará a los encargados de depósito de

// Henry Market a armar pilas de cajas de productos de forma tal que no supere el peso máximo permitido

// y evitar así que se dañen los productos.

// La función recibirá un array que representará las cajas que se desean apilar, con el nombre del producto

// y el peso total de cada caja, y deberá crear y retornar un STACK con LOS NOMBRES de productos que vaya apilando,

// sin superar el peso máximo permitido.

// En caso de que los productos superen dicho peso máximo, deberá retornar el string "No se puede crear la pila"

// Si el array recibido no contiene cajas, debe retornar el string "Error".

// El array recibido tendrá, por ejemplo, la siguiente forma:

//

// [

//  {nombre: arroz, peso: 10},

//  {nombre: fideos, peso: 15},

//  {nombre: cafe, peso: 5},

//  {nombre: arroz, peso: 10},

//  {nombre: azucar, peso: 20}

// ]

//

// TIPs:

//  - El array recibido es una ARRAY DE OBJETOS

//  - El array puede contener productos repetidos

//

//                     \*\*\*\*\*\*\*\*\* El peso máximo permitido para cada STACK es de 50 kg! \*\*\*\*\*\*\*\*\*\*\*

//

// EJEMPLOS:

//  - apilarCajas( [{nombre: arroz, peso: 10},

//                  {nombre: fideos, peso: 15},

//                  {nombre: cafe, peso: 5}] ) => Stack [arroz, fideos, cafe]

//                  \* El peso total (30), no supera el máximo (50)

//

//  - apilarCajas( [{nombre: leche, peso: 30},

//                  {nombre: fideos, peso: 15},

//                  {nombre: cafe, peso: 5}] ) => Stack [leche, fideos, cafe]

//                  \* El peso total (50), no supera el máximo (50)

//

//  - apilarCajas( [{nombre: leche, peso: 30},

//                  {nombre: fideos, peso: 15},

//                  {nombre: fideos, peso: 15},

//                  {nombre: cafe, peso: 5}] ) => "No se puede crear la pila"

//                  \* El peso total (65), excede el peso máximo permitido (50)

//

//  - apilarCajas( [] ) => "Error"

//

// REQUISITOS:

//  🟢 La función debe crear y retornar el Stack creado con las cajas de productos recibidas en el array

//  🟢 El Stack que retorna debe ser una instancia de la clase Stack

//  🟢 En caso de exceder el peso máximo permitido, debe retornar en string "No se puede crear la pila"

//  🟢 En caso de recibir un array vacío, debe retornar el string "Error"

//

// ATENCIÓN! Las respuestas en strings son case sensitive!

//

function apilarCajas(arr) {

    if (apilarCajas===[]) {

        return "Error";

    }

    let newStack = new Stack();

    let sumaPeso = 0;

    for (var i= 0; i < arr.length; i++){

        sumaPeso+= arr[i];

       if (sumaPeso < 50 || sumaPeso === 50) {

            newStack.push([i]);

        if (sumaPeso > 50) {

            newStack.pop([i])

            return "No se puede crear la pila"

        }

        }

       return newStack;

    }

}

// ⚠️ NO MODIFICAR NADA POR DEBAJO DE ESTA LÍNEA ⚠️

module.exports = {

    apilarCajas

};

// 4️⃣ \*\*\*\*\* EJERCICIO 4 \*\*\*\*\* - devuelveMayores() 4️⃣

// Implementar la función devuelveMayores, que recibe por parámetro una LinkedList con precios

// de productos de una determinada góndola de Henry Market, y un precio máximo a evaluar.

// La función deberá recorrer la LinkedList recibida y retornar la CANTIDAD DE PRECIOS QUE SUPEREN

// el valor de precio máximo que se desea evaluar.

// En caso de que ningún precio supere dicho valor, debe retornar en string "Sin precios"

//  EJEMPLOS:

//

//  listaPrecios1 = 5 => 10 => 8 => 7 => 14 => 20

//  devuelveMayores(listaPrecios1, 10) => 2

//  (2 precios (14 y 20) superan el valor a evaluar (10))

//

//  listaPrecios2 = 99 => 100 => 81 => 74 => 14 => 22

//  devuelveMayores(listaPrecios2, 50) => 4

//  (4 precios (99, 100, 81 y 74) superan el valor a evaluar (50))

//

//  listaPrecios3 = 99 => 80 => 81 => 75 => 15 => 21

//  devuelveMayores(listaPrecios3, 100) => "Sin precios"

//  (Ningún precio de la lista supera el valor a evaluar (100))

//

//  REQUISITOS:

//    🟢 La función debe retornar un valor numérico, que representa la CANTIDAD DE PRECIOS que superen

//      el valor a evaluar.

//    🟢 En caso de que ningún precio supere dicho valor, debe retornar el string "Sin precios"

//

//  ATENCIÓN! Las respuestas en strings son case sensitive!

function devuelveMayores(lista, valor) {

  // Tu código aquí:

}

// ⚠️ NO MODIFICAR NADA POR DEBAJO DE ESTA LÍNEA ⚠️

module.exports = {

  devuelveMayores

};

// 5️⃣ \*\*\*\*\* EJERCICIO 5 \*\*\*\*\* - cobrarClientes 5️⃣

// Implementar la función cobrarClientes, que recibirá una

// Queue (fila) de clientes que esperan para realizar el pago

// de sus compras del supermercado.

//

// Cada cliente dentro de la fila estará como objeto, y tendrá

// la siguiente estructura:

//

// {

//   nombre: "Jorge",

//   dinero: 1500,

//   precioProductos: 1200

// }

//

// - nombre: Nombre del cliente

// - dinero: Dinero disponible que tiene el cliente

// - precioProductos: Suma de los productos que desea comprar

//

// La función debe evaluar si el primer cliente de la fila poseee dinero suficiente

// para abonar los productos que desea comprar. En caso de ser así, debe remover

// al cliente de la fila.

// En caso de presentarse un cliente que no posee dinero suficiente, debe dejar de

// evaluar los clientes de la fila, y retornar un array con los NOMBRES de los clientes

// que fueron correctamente retirados de la fila.

//

// EJEMPLO:

//

// clientes=[

//  {

//   nombre: "Jorge",

//   dinero: 1500,

//   precioProductos: 1200

//  },

//  {

//   nombre: "Mateo",

//   dinero: 2000,

//   precioProductos: 1900

//  },

//  {

//   nombre: "Mora",

//   dinero: 5000,

//   precioProductos: 5500

//  },

//]

//

// cobrarClientes(clientes) => ["Jorge","Mateo"]

//

// ⚠️ ATENCIÓN ⚠️

// Recuerden que los "clientes" recibidos por parámetro se encuentran

// en una Queue! Por lo cual tendremos que utilizar los métodos que

// tenemos implementados para esta clase.

//

// CONSIGNAS:

//  🟢 Cuando termine de evaluar clientes, la función debe retornar un array con los nombres de los clientes que

//    han abonado correctamente.

//  🟢 En caso de quedarse sin clientes en la fila, la función debe retornar false.

function cobrarClientes(clientes) {

  // Tu código aquí:

   }

// ⚠️ NO MODIFICAR NADA POR DEBAJO DE ESTA LÍNEA ⚠️

module.exports = {

  cobrarClientes

};

const { BinarySearchTree } = require("../DS");

// ⚠️ NO MODIFICAR NADA POR ENCIMA DE ESTA LÍNEA ⚠️

// 6️⃣ \*\*\*\*\* EJERCICIO 6 \*\*\*\*\* - agregarProductos 6️⃣

// Implementar la función agregarProductos(producto) al prototipo de BinarySearchTree, que nos servirá para

// agregar LOS NOMBRES de los productos en una góndola de supermercado en base a las comparacionesde sus precios,

// siguiendo la siguiente tabla:

//

//    ┌─────────┬────────┐

//    │Productos│ Precios│

//    ├─────────┼────────┤

//    │  Leche  │  100   │

//    │  Queso  │  150   │

//    │   Pan   │   50   │

//    │ Frutas  │  110   │

//    │  Arroz  │   40   │

//    │  Jugo   │   80   │

//    └─────────┴────────┘

//

//  EJEMPLO:

//

//  Si partimos del árbol inicial llamado "arbolDeProductos" con un solo nodo:

//

//                                              Leche

//                                              /    \

//

//    arbolDeProductos.agregarProductos(Queso)

//

//                                              Leche

//                                              /    \

//                                                   Queso

//

// El producto "Queso" se insertó a la derecha, ya que su precio (150), es mayor que el precio Leche (100).

//

//    arbolDeProductos.agregarProductos(Pan)

//

//                                             Leche

//                                             /    \

//                                          Pan      Queso

//

// El producto "Pan" se insertó a la izquierda, ya que su precio (50), es menor que el precio Leche (100).

//

//    arbolDeProductos.agregarProductos(Frutas)

//

//                                             Leche

//                                            /     \

//                                         Pan       Queso

//                                                  /

//                                               Frutas

//

// El producto "Frutas" en la posición correspondiente, ya que su precio (110), supera al de Leche (100) y

// es menor que precio de Queso (150).

//

// CONSIGNAS:

//  🟢 En caso de querer insertar un producto que ya se encuentra en el árbol, debe retornar

//    el string "Ya existe el producto".

//  🟢 En caso de querer insertar un producto que no se encuentra en la tabla de precios, debe retornar

//    el string "Producto inexistente".

//  🟢 El método debe insertar el producto en la posición correspondiente, basando su ubicación en el

//    precio del producto. (Ver ejemplos)

//  🟢 En caso de insertar el producto correctamente, debe retornar el nuevo nodo que ha sido insertado.

//

//

// IMPORTANTE! La lista de productos se encuentra en el objeto productos a continuación:

//⚠️ NO MODIFICAR NADA POR DEBAJO DE ESTA LÍNEA ⚠️

//⚠️ NO MODIFICAR NADA POR ENCIMA DE ESTA LÍNEA ⚠️

BinarySearchTree.prototype.agregarProductos = function (nombreProducto, productos) {

  // Tu código aquí:

};

// ⚠️ NO MODIFICAR NADA POR DEBAJO DE ESTA LÍNEA ⚠️

module.exports = {

    BinarySearchTree

};

// 7️⃣ \*\*\*\*\* EJERCICIO 7 \*\*\*\*\* - ordenarPrecios() 7️⃣

// Implementar la función ordenarPrecios, la cual recibirá un array que representa una lista desordenada

// de precios de Henry Market y, a través de alguno de los métodos de ordenamiento vistos en el módulo (a elección),

// deberá retornar un array con los precios ordenados de menor a mayor.

// Asímismo, en caso de que la función se encuentre con algún precio 0 (cero) dentro del array, debe desestimar

// el ordenamiento, y retornar false.

//

// EJEMPLOS:

//  - ordenarPrecios([20,15,45,10,5]) => [5,10,15,20,45]

//  - ordenarPrecios([20,15,0,10,5]) => false

// CONSIGNAS:

//  🟢 En caso de recibir un 0 (cero) dentro del array, la función debe retornar false

//  🟢 Caso contrario, debe retornar un array con los precios ordenados, utilizando alguno de los método de ordenamiento

//    estudiados. NO SE PUEDE USAR EL MÉTODO SORT() DE ARRAY.

function ordenarPrecios(arr) {

  if(arr.includes(0)) return false;

    for (let i = 1; i < arr.length; i++) {

      if (arr[i] === 0 ) return false;

      let current = arr[i];

      let j;

      for (j = i - 1; j >= 0 && arr[j] > current; j--) {

        var aux = arr [j + 1];

        arr[j + 1] = arr[j];

        arr[j] = aux;

    }

    }

    return arr;

}

// ⚠️ NO MODIFICAR NADA POR DEBAJO DE ESTA LÍNEA ⚠️

module.exports = {

  ordenarPrecios

};

const { BinarySearchTree } = require("../DS");

// ⚠️ NO MODIFICAR NADA POR ENCIMA DE ESTA LÍNEA ⚠️

// 8️⃣ \*\*\*\*\* EJERCICIO 8 \*\*\*\*\* - searchMax() 8️⃣

// Implementar la función searchMax dentro del prototipo de BynarySearchTree, que nos servirá para buscar el precio

// máximo que se encuentra dentro de un BinarySearchTree que contendrá números que representan a los precios de una

// de las góndolas de Henry Market.

//

// EJEMPLO:

//  - En caso de que nuestro árbol de precios sea el siguiente:

//

//             17

//          /      \

//        7         24

//      /  \       /   \

//     3    15    18    32

//    / \                 \

//       4                45

//

// Deberá retornar 45.

BinarySearchTree.prototype.searchMax = function () {

  // Tu código aquí:

};

// ⚠️ NO MODIFICAR NADA POR DEBAJO DE ESTA LÍNEA ⚠️

module.exports = {

    BinarySearchTree

};

const { BinarySearchTree } = require("../DS");

// ⚠️ NO MODIFICAR NADA POR ENCIMA DE ESTA LÍNEA ⚠️

// 9️⃣ \*\*\*\*\* EJERCICIO 9 \*\*\*\*\* - searchPrice() 9️⃣

// Implementar la función searchPrice dentro del prototipo de BynarySearchTree, que nos servirá para buscar un precio

// recibido por parámetro dentro de un BinarySearchTree que contendrá números que representan a los precios de una de

// las góndolas de Henry Market.

// 🟢 En caso de encontrar el precio recibido por parámetro, debe retornar true.

// 🟢 En caso de no encontrar dicho precio, debe retornar false.

// 🟢 En caso de recibir por parámetro un precio igual o menor a 0 (cero), debe retornar el string "Error"

//

// EJEMPLOS:

//  - En caso de que nuestro árbol de precios sea el siguiente:

//

//             17

//          /      \

//        7         24

//      /  \       /   \

//     3    15    18    32

//    / \                 \

//       4                45

//

// arbolDePrecios.searchPrice(24) => true

// arbolDePrecios.searchPrice(4) => true

// arbolDePrecios.searchPrice(50) => false

// arbolDePrecios.searchPrice(1) => false

// arbolDePrecios.searchPrice(0) => "Error"

// arbolDePrecios.searchPrice(-10) => "Error"

BinarySearchTree.prototype.searchPrice = function (precio) {

    // Tu código aquí

};

// ⚠️ NO MODIFICAR NADA POR DEBAJO DE ESTA LÍNEA ⚠️

module.exports = {

 BinarySearchTree

};

// 1️⃣0️⃣  EJERCICIO 10  ingresoEmpleado 1️⃣0️⃣

// Implementar la función ingresoEmpleado que servirá para registrar el ingreso de los empleados del Henry Market,

// apuntando aquellos que hayan llegado a horario.

// IMPORTANTE! Los horarios serán números enteros (9,10,11,12), NO tendrán formato hora (09:00, 10:00, etc)

//

// La función ingresoEmpleado debe retornar la función ingresosHorario, la cual recibirá un arreglo de empleados que ingresan

// al supermercado con la siguiente forma:

// let empleados =  [

//    {nombre:"Jorge", ingresoA: 9}

//    {nombre:"Mora", ingresoA: 8}

//    {nombre:"Mati", ingresoA: 10}

//    {nombre:"Juani", ingresoA: 7}

//  ]

//, y retornará un nuevo arreglo con LOS NOMBRES de aquellos que se hayan presentado antes del horario de ingreso.

//

// Ejemplo

//  const ingresosHorario =  ingresoEmpleado(9)

//  Si ejecuto ingresosHorario(empleados) me daria de resultado ["Jorge"]

function ingresoEmpleado(horario) {

// Tu código aquí:

const ingresosHorario =  ingresoEmpleado(horario);

return

}

// ⚠️ NO MODIFICAR NADA POR DEBAJO DE ESTA LÍNEA ⚠️

module.exports = {

    ingresoEmpleado

};