Integrantes del proyecto:

- Rafael Rodríguez Calvente
- Alejandro Arias Casquero
- Sergio Cabeza de Vaca Montero

1. DESCRIPCIÓN FUNCIONAL

ESIZON es un simulador simplificado de una plataforma de comercio electrónico online desde la que cualquier cliente registrado podrá adquirir multitud de productos de todo tipo, los cuales estarán clasificados en diferentes categorías. Esta plataforma llevará el control de los pedidos de los clientes. El sistema aceptará códigos promocionales y cheques de descuento que podrán ser aplicados por el cliente en el momento de la realización del pedido. El cliente podrá decidir si el pedido realizado será entregado en su propio domicilio o en un punto de recogida ESILocker al que podrá acudir con un código de entrega para recogerlo él mismo. ESIZON será proveedor de los productos que vende aunque dispondrá también de un conjunto de productos suministrados y gestionados por proveedores externos los cuales se harán cargo de controlar sus propios productos y el stock de los mismos.

El programa dispondrá de cuatro perfiles de usuarios:
Un perfil de cliente: tendrá la posibilidad de acceder a los datos de su perfil, realizar búsquedas de productos, pedidos, devoluciones, añadir dinero a la cartera, etc.
Un perfil de administrador: realizará las tareas de configuración de la plataforma, como tratamiento de clientes, productos propios, pedidos de clientes, devoluciones, stock, ESILockers, proveedores, etc.
Un perfil de proveedor: realizará las tareas de configuración de los productos de su propiedad como la gestión de sus productos, stock, devoluciones, etc. También podrá acceder a los datos de su perfil.
Un perfil de transportista: se encargará de realizar las tareas de actualización de su perfil y repartos de pedidos que le han sido asignados.

Todos los datos de ESIZON estarán almacenados en ficheros para poder conservar toda la información y volverla a utilizar en posteriores ejecuciones del programa. Al iniciar ESIZON, dicha información se volcará desde los ficheros a estructuras de datos en memoria y al terminar la ejecución del programa la información se actualizará en los ficheros. De esta forma todo el funcionamiento de la plataforma tendrá lugar en memoria principal.
Estos ficheros que almacenarán la información serán ficheros de texto con el separador / entre los distintos campos y tendrán los siguientes formatos:

Fichero Clientes.txt, almacenará la información de los clientes del sistema con los siguientes campos.

o Identificador del Cliente
o Nombre del cliente
o Apellidos del cliente
o Dirección
o Localidad
o Provincia
o Email (será usado como nombre de usuario para el acceso a la plataforma)
o Contraseña
o Cartera (almacenará el dinero del que dispone el cliente para poder adquirir los productos)

Formato:

000001/Juan/Pérez/Plaza Juan de Austria 1 11100/Puerto Real/Cádiz/juanperez@gmail.com/psw1234/1000
000002/Pedro/López/Calle Plocia 2 11100/Cádiz/Cádiz/pedrolopez@gmail.com/psw4321/500

Fichero AdminProv.txt, almacenará la información de los administradores y proveedores de productos de ESIZON.

o Identificador de la empresa administradora o proveedora

o Nombre de la empresa (ESIZON si es administrador)

o Email (será usado como nombre de usuario para el acceso a la plataforma)

o Contraseña

o Tipo: admin (administrador)/prov (proveedor)

Formato:

0001/ESIZON/Antonio.martin@esizon.com/antmar01/admin

0002/DISTGEN/donato.lima@distgen.com/donlim04/prov

Fichero productos.txt, almacenará la información de todos los productos que se pueden adquirir desde la plataforma.

o Identificador del producto

o Descripción del producto

o Id de la categoría a la que pertenece

o Id del gestor del producto. Deberá coincidir con el identificador de la empresa administradora o proveedora del producto.

o Stock del producto.

o Compromiso de Entrega. Indicará el no de días máximo que transcurrirá desde la fecha de realización del pedido hasta la entrega al cliente.

o Importe del producto (en euros)

Formato:

000001/televisor 32" 3D/001/0001/50/5/340

000002/Auriculares inalámbricos/002/0001/125/1/45

Fichero Categorías.txt, almacenará la información de todas las categorías de productos que pueden adquirirse en la plataforma.

o Identificador de la categoría

o Descripción de la categoría

Formato:

001/tv video y home cinema

002/audio y HIFI

Fichero Descuentos.txt, almacenará la información de los distintos códigos promocionales de descuentos y cheques regalo a emplear en la compra de productos.

o Identificador del Código promocional o cheque regalo

o Descripción

o Tipo: codpro (código promocional)/cheqreg (cheque regalo)

o Estado: activo/no activo

o Importe del descuento (en euros)

o Aplicabilidad: todos (si es aplicable a todos los productos)/esizon (si sólo es aplicable a los gestionados por ESIZON)

Formato:

black001/Black Friday/codpro/activo/10/esizon

che001/cheque regalo nivel 1/cheqreg/activo/5/todos

Fichero DescuentosClientes.txt, almacenará la información de los distintos códigos promocionales de descuentos y cheques regalo de cada cliente registrado en la plataforma.

o Identificador del cliente

o Identificador del código promocional o cheque regalo

o Fecha de asignación (día mes y año sin formato alguno)

o Fecha de caducidad (día mes y año sin formato alguno)

o Estado:1-Aplicado/0-No aplicado

Formato:

000001/black001/25112019/27112019/0

000002/che001/01122019/01012020/0

Fichero Lockers.txt, almacenará la información de los distintos respositorios utilizados por la plataforma para las entregas de los pedidos.

o Identificador del Locker

o Localidad

o Provincia

o Ubicación

o No de compartimentos total

o No de compartimentos ocupados actualmente

Formato:

Lock001/Puerto Real/Cádiz/local 12/15/0

Fichero CompartimentosLockers.txt, almacenará la información de los distintos productos depositados en cada compartimento y el código locker asociado al producto.

o Identificador del Locker

o No de compartimento

o Código locker

o Estado: ocupado/vacío

o Fecha ocupación (día mes y año sin formato alguno)

o Fecha caducidad (día mes y año sin formato alguno)

Formato:

Lock001/01/134697/ocupado/08012020/13012020

Fichero Pedidos.txt, almacenará la información de los distintos pedidos realizados por los clientes registrados en la plataforma.

o Identificador del pedido

o Fecha del pedido

o Identificador del cliente que realiza el pedido

o Lugar de entrega: domicilio/locker

o Identificador de Locker

o Código de cheque regalo

o Código promocional

Formato:

00000001/02012020/000001/casa//che001/black001

00000002/03012020/000002/locker/Lock001//

Fichero productosPedidos.txt, almacenará la información de todos los productos incluidos en los diferentes pedidos realizados.

o Identificador del pedido

o Identificador del producto

o No de unidades

o Fecha prevista de entrega (día mes y año sin formato alguno)

o Importe del producto (importante para que quede constancia del importe al que compra un cliente un producto si la empresa modifica posteriormente su importe)

o Estado del pedido: enPreparación/enviado/enReparto/enLocker/entregado/devuelto (transportista)

o Identificador del transportista

o Código Locker

o Fecha de entrega/devolución por el transportista

Formato:

00000001/000001/1/07012020/340/En preparación////

00000001/000002/1/03012020/45/ Entregado /00001//

00000002/000001/2/08012020/45/ EnLocker/00002/134697/08012020

Fichero Transportistas.txt, almacenará la información de los repartidores de pedidos.

o Identificador del transportista

o Email (se usa como nombre de usuario para el acceso)

o Contraseña

o Nombre de empresa

o Ciudad de reparto

Formato:

00001/rafael@gutitrans.com/rf001/GutiTrans/Cádiz

00002/pepe@transpo.com/pp001/TransPo/Cádiz

Fichero Devoluciones.txt, almacenará la información de las devoluciones de los distintos productos incluidos en los pedidos entregados.

o Identificador del pedido

o Identificador del producto

o Fecha de devolución

o Motivo

o Estado: pendiente (de aceptación)/aceptado/denegado/enviado (por cliente)/recibido

o Fecha aceptación (día mes y año sin formato alguno)

o Fecha caducidad (día mes y año sin formato alguno)

Formato:

00000002/000001/25012020/desperfecto/pendiente//

Al arrancar el programa aparecerá un mensaje solicitando la entrada a la plataforma mediante usuario y contraseña. En el caso de que el usuario no exista, se dará la posibilidad de dar de alta a dicho usuario con perfil cliente. Existirá un usuario super administrador, por defecto, con permiso para crear al resto de usuarios administradores, clientes, proveedores y transportistas. Dependiendo del perfil de usuario que acceda a la plataforma, el menú principal del sistema será diferente.

Menú para un usuario administrador, el menú principal contendrá:

1. Perfil
2. Clientes
3. Proveedores
4. productos
5. Categorías
6. Pedidos
7. Transportistas
8. Descuentos
9. Devoluciones

10. Salir del sistema

1. Perfil

Esta opción permitirá a un usuario administrador consultar los datos de su perfil y modificarlos.

2. Clientes

Mediante esta opción el administrador podrá acceder a la información de los clientes dados de alta en la plataforma. Mediante el menú correspondiente podrá realizar altas, bajas, búsquedas, listados y modificaciones de clientes.

3. Proveedores

Mediante esta opción el administrador podrá acceder a la información de los proveedores, de productos externos, dados de alta en la plataforma. Mediante el menú correspondiente podrá realizar altas, bajas, búsquedas, listados y modificaciones de proveedores.

4. productos

Mediante esta opción el administrador podrá acceder a la información de los productos dados de alta en la plataforma. Mediante el menú correspondiente podrá realizar altas, bajas, búsquedas, listados y modificaciones de productos.

5. Categorías

Mediante esta opción el administrador podrá acceder a la información de las categorías dadas de alta en la plataforma. Mediante el menú correspondiente podrá realizar altas, bajas, búsquedas, listados y modificaciones de categorías. Además podrá generar listados de productos por categoría.

6. Pedidos

Mediante esta opción el administrador podrá acceder a la información de los pedidos dados de alta en la plataforma. Mediante el menú correspondiente podrá realizar altas, bajas, búsquedas, listados y modificaciones de pedidos. En los listados habrá que contemplar la posibilidad de listarlos según su estado. Esto es necesario para poder localizar rápidamente los pedidos cuya fecha de entrega sea próxima y que, por tanto, deben ser procesados con mayor prioridad. Otras opciones permitidas será la asignación de transportistas a los productos pedidos en función de la dirección del cliente y ciudad de reparto. Así como llevar a cabo la asignación de lockers a los pedidos en base a la localidad de entrega y ubicación

del locker.

7. Transportistas

Mediante esta opción el administrador podrá acceder a la información de todos los transportistas dados de alta en la plataforma. Mediante el menú correspondiente podrá realizar altas, bajas, búsquedas, listados y modificaciones de transportistas.

8. Descuentos

Mediante esta opción el administrador podrá acceder a la información de todos los códigos promocionales y/o cheques regalo dados de alta en la plataforma. Mediante el menú correspondiente podrá realizar altas, bajas, búsquedas, listados y modificaciones de estos descuentos. Además también podrá generar los listados

de clientes que tienen asignado algún cheque regalo/código promocional, así como los listados de clientes que han hecho uso de algún cheque regalo/código promocional. Por supuesto podrá asignar a un cliente determinado un cheque regalo. Y en el caso de la creación de un nuevo código promocional, éste deberá ser asignado a todos los clientes.

9. Devoluciones

Mediante esta opción el administrador podrá acceder a la información de todas las devoluciones de productos. Mediante el menú correspondiente podrá realizar altas de devoluciones (aunque lo habitual es que sean generadas por los clientes se permite esta opción por si el cliente tuviera algún problema y no pudiera realizarla), bajas, búsquedas, listados y modificaciones de devoluciones.

Entre las modificaciones de los pedidos el administrador tendrá la posibilidad de consultar todos las devoluciones, solicitadas por un cliente, que se encuentran pendientes de aceptación, así como modificar el estado de éstas para aceptarlas o no. Si la devolución es aceptada se debe actualizar la fecha de aceptación y la fecha de caducidad de la devolución. También se permitirá modificar el estado a recibido en el momento que reciba el producto. Hay que tener en cuenta que en

este último caso se debe actualizar el stock del producto recibido.

10. Salir del sistema.

Permitirá al administrador salir del sistema, mostrando a continuación el menú de acceso.

Menú para un usuario proveedor, el menú principal contendrá:

1. Perfil
2. productos
3. Pedidos
4. Salir del sistema

1. Perfil

Esta opción permitirá a un usuario proveedor consultar los datos de su perfil y modificarlos.

2. productos

El usuario proveedor podrá realizar las mismas acciones que el administrador pero sólo sobre sus propios productos. Es decir, podrá acceder a la información de sus productos que estén dados de alta en la plataforma. Sobre ellos podrá realizar altas, bajas, búsquedas, listados y modificaciones de productos.

3. Pedidos

Un usuario proveedor podrá acceder únicamente a la información de los pedidos de productos que él mismo suministra. Lógicamente podrá gestionar el estado de dichos pedidos de productos, asignar transportistas, lockers, etc.

4. Salir del sistema

Permitirá al proveedor salir del sistema, mostrando a continuación el menú de acceso.

Menú para un usuario cliente, el menú principal contendrá:

1. Perfil
2. productos
3. Descuentos
4. Pedidos
5. Devoluciones
6. Salir del sistema

1. Perfil

Esta opción permitirá a un usuario cliente consultar los datos de su perfil y modificarlos. Hay que tener en cuenta que el cliente podrá actualizar su cartera que es la que utilizará para las compras en la plataforma.

2. productos

La plataforma permitirá al cliente realizar consultas de productos tanto por categoría como por nombre.

3. Descuentos

Mediante esta opción el cliente podrá consultar todos los códigos promocionales y cheques regalo que tiene asignados.

4. Pedidos

Permitirá al cliente realizar pedidos controlando los códigos promocionales y/o cheques regalo que pueda utilizar en función de si están activos o no y de la aplicabilidad que tengan sobre el producto. También se le permitirá al cliente consultar el estado de cada uno de los productos de su pedido.

Otra opción para el cliente es la recogida de un pedido que haya sido depositado en un ESILocker antes de la fecha de caducidad, para ello se le debe mostrar la opción solicitándole el código locker asociado. Una vez introducido, el sistema mostrará un mensaje en pantalla indicando el no de compartimento que se ha abierto. El estado de los productos recogidos, en consecuencia, se debe actualizar.

Hay que tener en cuenta que en el mismo momento que se haga un pedido de un producto, se deben de dar de baja las unidades pedidas del stock. Además, en base al número de días máximo de entrega del producto pedido y de la fecha del pedido se debe actualizar la fecha prevista de entrega.

5. Devoluciones

Permitirá al cliente llevar a cabo devoluciones de productos y realizar un seguimiento de dichas devoluciones. En el momento en el que se realiza la devolución de un producto, éste pasa a estado pendiente de aceptación. El cliente podrá consultar en todo momento las devoluciones pendientes de aceptación para, una vez aceptadas, poder cambiar el estado del producto a enviado para indicar que ya está de camino. Recalcar que las devoluciones de productos las gestiona ESIZON independientemente de quien las suministre.

6. Salir del sistema

Permitirá al proveedor salir del sistema, mostrando a continuación el menú de acceso.

Menú para un usuario transportista, el menú principal contendrá:

1. Perfil
2. Repartos
3. Retornos

1. Perfil

Esta opción permitirá a un usuario transportista consultar los datos de su perfil y modificarlos.

2. Repartos

Esta opción permitirá al transportista consultar la lista de productos que tiene asignados para su entrega así como la fecha prevista para la misma, lo que le permite realizar su ruta de reparto. Cuando el transportista realice la entrega del

producto automáticamente debe ser modificado el estado del mismo.
Si la entrega se realiza en un locker el transportista tendrá que asignar un código locker a los productos, asociando de esta manera un compartimento de dicho locker con los productos. Automáticamente se debe actualizar el número de compartimentos ocupados en los CompatimentosLockers.

3. Retornos
El sistema facilitará al transportista la tarea de retornar a origen todos los productos que no hayan sido recogidos de los lockers en el plazo determinado, permitiéndole consultar todos los lockers por localidad y mostrando sus pedidos.
En el momento de la recogida de los productos para su retorno el sistema debe actualizar automáticamente el número de compartimentos ocupados y eliminar el código locker asociado al producto. Así como el estado y el stock de los productos y la cartera del cliente, para que quede reflejada la operación.

2. DESCOMPOSICIÓN DEL PROBLEMA Y MÓDULOS

El programa propuesto trata sobre una simplificación de la conocida plataforma de comercio online Amazon, nos hemos enfocado en la parte de creación de usuarios, pedidos, descuentos, logística, etc...

Para realizar el programa propuesto, lo hemos descompuesto en varios subproblemas con sus respectivos módulos los cuales ayudan a realizar las tareas requeridas. Los subproblemas son los que se muestran en la siguiente tabla:

Problemas	Módulos
1. Manejo de datos y usuarios 2. Manejo de productos y sus categorías 3. Pedidos de los clientes 4. Devoluciones de los pedidos 5. Descuentos los productos 6. Organización y estructuración logística de la plataforma	1. Datos Usuarios 2. Productos/Categorías 3. Pedidos 4. Devoluciones 5. Descuentos 6. Logística

- **Datos Usuarios:** Aquí tenemos la base de todo el programa, en “Datos Usuarios” se encuentra toda la organización de los tipos de usuarios. Desde este módulo podremos realizar registros, búsquedas, listados y modificaciones de todos los tipos de usuarios existentes en ESIZON (clientes, administradores, transportistas y proveedores).

- **Productos/Categorías:** En este módulo se relacionan dos cosas muy importantes de la plataforma de comercio, como son productos y sus categorías, ambas cosas son necesarias para que el cliente pueda realizar la búsqueda de su producto satisfactoriamente. Aquí se desempeñan todas las operaciones como registrar, modificar,

buscar, listar y eliminar productos y categorías de productos. Se corresponde con (“Productos_Categorias.h” y “Productos_Categorias.c”).

- **Pedidos:** En este módulo se realiza todo tipo de gestiones como dar de alta, baja y modificaciones de pedidos. Además de por supuesto poder listarlos y hacer búsquedas concretas de alguno en específico. Se corresponde con (“Pedidos.h” y “Pedidos.h”).

- **Devoluciones:** Gracias a este módulo los administradores tendrán la posibilidad de modificar el estado de las devoluciones, acceder a ellas para,darlas de baja, etc… Por otro lado los clientes podrán comprobar el estado de las devoluciones solicitadas así como comprobar si han sido aceptadas o no entre otras funciones. Se corresponde con (“Devoluciones.h” y “Devoluciones.c”).

- **Descuentos:** Este módulo nos proporciona la posibilidad de dar de alta, baja, modificaciones tanto de los descuentos existentes tanto como los descuentos ya asignados a algún cliente en particular, además de por supuesto poder listar los descuentos activos/no activos.... Se corresponde con (“Descuentos.h” y “Descuentos.c”).

- **Logística:** En este módulo se encuentra todo lo correspondiente a transportistas, lockers y sus compartimentos. Gracias a este módulo podremos dar de alta, baja de los lockers existentes, así como comprobar los lockers disponibles para que los usuarios puedan elegir en cuál de ellos desea recoger su pedido. ¡Todo siempre pensado para mejorar la experiencia de los clientes en sus compras! Se corresponde con (“Logistica.h” y “Logistica.c”).

3. DOCUMENTACIÓN POR MÓDULO

3.1 Datos/Usuarios.

Sección de importación.

Incluye el fichero de cabecera “DatosUsuarios.h”, el cual corresponde al módulo “Datos/Usuarios”. También incluye las librerias ”math.h” y “string.h”.

Sección de exportación.

Exporta los tipos registros ”cliente”, “adminprov”, “usuario” y “transportista” . También exporta las variables enteras globales “ncliente”, “nadminprov” y “ntransport” encargadas de cuantificar el número de clientes, administradores y transportistas respectivamente.

Funciones y procedimientos.

- **void cargar_clienes(cliente **c, usuario **usu):** Carga los datos del fichero “Clientes.txt” en la estructura “*clientes*”. Recibe la estructura “*cliente*” y “usuario”.
- **void guardar_cliente(cliente **c):** Guarda los datos de la estructura “*cliente*” en el fichero “Clientes.txt”. Recibe la estructura “*cliente*”.
- **void cargar_adminprov(adminprov **ap):** Carga los datos del fichero “AdminProv.txt” en la estructura “*adminprov*”. Recibe la estructura “*adminprov*”.

- **void guardar_adminprov(adminprov **ap):** Guarda los datos de la estructura "*adminprov*" en el fichero "AdminProv.txt". Recibe la estructura "*adminprov*".
- **void cargar_transportista(transport **tr):** Carga los datos del fichero "Transportista.txt" en la estructura "*transport*". Recibe la estructura "*transport*".
- **void guardar_transportista(transport **tr):** Guarda los datos de la estructura "*transport*" en el fichero "Transportista.txt". Recibe la estructura "*transport*".
- **void iniciar_sesion(cliente **c, adminmprov **ap, transport **tr, usuario **usu):** Recibe todas las estructuras de tipos usuarios. rellena la estructura usuario y da paso a las demás funciones
- **void acceder_cuenta(cliente **c, adminprov **ap, transport **tr, usuario **usu):** Recibe todas las estructuras de tipo usuario. Comprueba que el perfil existe
- **tipo_usuario(usuario **usu):** Recibe la estructura usuario. Rellena el campo tipo de la estructura usuario.
- **void buscar_cliente(char* mail, cliente **c, usuario **usu):** Recibe como parámetros las estructuras clientes y usuario además del email. Solo pasa por la función si existe el cliente indicado.
- **void buscar_admin(char* mail, adminprov **ap, usuario **usu):** Recibe las estructuras adminprov y usuario además del email. Solo pasa por la función si existe el adminprov indicado.
- **void buscar_transport(char* mail, adminprov **ap, usuario **usu):** Recibe las estructuras adminprov y usuario además de la cadena email. Solo pasa por la función si existe el transport indicado.
- **void modificar_clientes(cliente **c, usuario **usu):** Recibe las estructuras cliente y usuario. Da la posibilidad de modificar cualquier campo de clientes.

- **void modificar_administradores(adminprov **ap, usuario **usu):** Recibe las estructuras adminprov y usuario. Da la posibilidad de modificar cualquier campo de administradores.
- **void modificar_proveedores(adminprov **ap, usuario **usu):** Recibe las estructuras adminprov y usuario. Da la posibilidad de modificar cualquier campo de proveedores.
- **void modificar_transportistas(transport **tr, usuario **usu) :** Recibe las estructuras transport y usuario. Da la posibilidad de modificar cualquier campo de transportistas.
- **void comprobar_pass (char* pass):** Recibe el parámetro pass correspondiente con la contraseña. Da la posibilidad de reintentar o cerrar el programa.
- **void listado_clientes(cliente **c):** Recibe la estructura cliente. Lista los clientes.
- **void listado_admin(adminprov **ap):** Recibe la estructura adminprov. Lista los administradores.
- **void listado_prov(adminprov **ap):** Recibe la estructura adminprov. Lista los proveedores.
- **void listado_transport(transport **tr):** Recibe la estructura transport. Lista los transportistas.
- **void registrar_cliente(cliente **c, usuario **usu):** Recibe las estructuras cliente y usu. Crea una nueva estructura clientes.
- **void eliminar_cliente(cliente **c, usuario **usu):** Recibe las estructuras cliente y usuario. Elimina una estructura cliente
- **void registrar_admin(adminprov **ap, usuario **usu):** Recibe las estructuras adminprov y usuario. Crea una nueva estructura adminprov con tipo admin.
- **void eliminar_admin(adminprov **ap, usuario **usu):** Recibe las estructuras adminprov y usuario. Elimina una estructura adminprov con tipo admin.

- **void registrar_prov(adminprov **ap, usuario **usu):** Recibe las estructuras adminprov y usuario. Crea una estructura adminprov con tipo prov.
- **void eliminar_prov(adminprov **ap, usuario **usu):** Recibe las estructuras adminprov y usuario. Elimina una estructura adminprov con tipo prov.
- **void registrar_transport(transport **tr, usuario **usu):** Recibe las estructuras adminprov y usuario. Crea una estructura transportista.
- **void eliminar_transport(transport **tr, usuario **usu):** Recibe las estructuras adminprov y usuario. Elimina una estructura transportista.

3.2 Productos/Categorías.

Sección de importación.

Incluye el fichero de cabecera “Productos_Categorias.h”, el cual corresponde al módulo “Productos/Categorías”. También incluye las librerias ”math.h” y “string.h”.

Sección de exportación.

Exporta los tipos registros ”catg” y “prod”. También exporta las variables enteras globales “ncatg” y “nprod” encargadas de cuantificar el número de categorías y productos respectivamente.

Funciones y procedimientos.

- **void cargar_categorias(catg **cat):** Carga los datos del fichero “Categorias.txt” en la estructura “*catg*”. Recibe la estructura catg.
- **void guardar_categorias(catg **cat):** Guarda los datos de la estructura “*catg*” en el fichero “Categorias.txt”. Recibe la estructura “*catg*”.

- **void cargar_productos(prod **pr):** Carga los datos del fichero “Productos.txt” en la estructura “*prod*”. Recibe la estructura prod.
- **void guardar_productos(prod **pr):** Guarda los datos de la estructura “*prod*” en el fichero “Categorias.txt”. Recibe la estructura “*prod*”.
- **int busqueda_producto(char* nombre, prod **pr):** Recibe la estructura prod y la cadena de caracteres nombre. devuelve la dirección de memoria de un vector de enteros donde la posición 0 es la cantidad de coincidencias encontradas y el resto del vector son las posiciones en el vector productos de las coincidencias, si no encuentra la primera posición 0 del vector es -1.
- **int buscar_producto(char* nombre, prod **pr):** Recibe la estructura prod y la cadena de caracteres nombre. devuelve la posición en el vector de estructuras producto de la coincidencia completa, si no encuentra devuelve -1.
- **void registrar_producto(prod **pr, catg **ca, adminprov **ap, usuario **usu):** Recibe las estructuras prod, catg, adminprov y usuario. Da de alta un nuevo producto.
- **void eliminar_producto(prod **pr):** Recibe la estructura prod. Elimina un producto existente.
- **int busqueda_categoria(char* nombre, catg **ca):** Recibe la cadena nombre y la estructura catg. devuelve la dirección de memoria de un vector de enteros donde la posición 0 es la cantidad de coincidencias encontradas y el resto del vector son las posiciones en el vector categorías de las coincidencias, si no encuentra la primera posición 0 del vector es -1.
- **int buscar_categoria(char* nombre, catg **ca):** Recibe la cadena nombre y la estructura catg. devuelve la posición en el vector de estructuras categoría de la coincidencia completa, si no encuentra devuelve -1.
- **void listado_categorias(catg **ca):** Recibe la estructura catg. Lista todas las categorías.

- **void listado_productos(prod **pr, int *vec):** Recibe la estructura prod y el parámetro vec. crea un listado de todos los productos indicados.
- **void categoria_producto(prod **pr, int cat, int op):** Recibe la estructura prod y los enteros cat y op.asigna la id correcta a los productos despues de eliminar una categoría, significando 0 que no tiene categoria asignada.
- **void registrar_categoria(catg **ca, adminprov **ap, usuario **usu):** Recibe las estructuras catg y usuario. Da de alta una nueva categoría.
- **void eliminar_categorias(catg **ca, prod **pr):** Recibe las estructuras catg y prod. Elimina una categoría existente.
- **void modificar_categoria(catg **ca, prod **pr):** Recibe las estructuras catg y prod. Modifica el nombre de un categoría.

3.3 Descuentos.

Sección de importación.

Incluye el fichero de cabecera “Descuentos.h”, el cual corresponde al módulo “Descuentos”. También incluye la librería “string.h”.

Sección de exportación.

Exporta los tipos registros ”descuentos” y “descliente”. También exporta las variables enteras globales “ndescuentos” y “ndescliente” encargadas de cuantificar el número de descuentos, y descuentos de cada cliente respectivamente.

Funciones y procedimientos.

- **void cargar_descuentos(descuentos **des):** Carga los datos del fichero "Descuentos.txt" en la estructura "*descuentos*". Recibe la estructura descuentos.
- **void guardar_descuentos(descuentos **des):** Guarda los datos de la estructura "*descuentos*" en el fichero "Descuentos.txt". Recibe la estructura "*descuentos*".
- **void cargar_descliente(descliente **desc):** Carga los datos del fichero "DescuentosClientes.txt" en la estructura "*descliente*". Recibe la estructura descliente.
- **void guardar_descliente(descliente **desc):** Guarda los datos de la estructura "*descliente*" en el fichero "DescuentosClientes.txt". Recibe la estructura "*descliente*".
- **void listar_descuentos(descuentos **des):** Recibe la estructura descuentos y los lista.
- **int busqueda_des(descuentos **des):** Recibe la estructura descuentos y devuelve -1 si no encuentra el descuento o un entero mayor que -1 con la posición dentro de la estructura descuentos del que se busca**.**
- **int val_codprom(descuentos **des):** Recibe la estructura descuentos y devuelve un entero, 1 si el descuento está activo y 0 si no lo está.
- **void alta_descuento(descuentos **des):** Recibe la estructura descuentos y realiza el alta de un nuevo descuento en el sistema.
- **void baja_des(descuentos **des):** Recibe la estructura descuentos y realiza la baja de un descuento del sistema. Se comprueba previamente que dicho descuento existe.
- **void modificar_descuento(descuentos **des):** Recibe la estructura descuentos y realiza la modificación de un descuento del sistema. Se comprueba previamente que dicho descuento existe.

- **int busqueda_cliente(descliente **desc):** Recibe la estructura descliente y devuelve -1 si no encuentra el descuento del cliente o un entero mayor que -1 con la posición dentro de la estructura descliente del que se busca**.**
- **void list_todos_desc_cada_cliente(descliente **desc):** Recibe la estructura descliente y lista todos los descuentos del cliente que se elija. Se comprueba que dicho cliente existe previamente.
- **void listar_desc(descliente **desc):** Recibe la estructura descliente y lista todos los descuentos de los clientes almacenados en los registros/estructuras.
- **void listar_desc_usados(descliente **desc):** Recibe la estructura descliente y lista todos los descuentos los cuales hayan sido ya usados.

3.4 Logística.

Sección de importación.

Incluye el fichero de cabecera “Logistica.h”, el cual corresponde al módulo “Logistica”. También incluye la librería “string.h”.

Sección de exportación.

Exporta los tipos registros ”comp_lockers”, “lockers” y “transportistas”. También exporta las variables enteras globales “ncomp_lockers”, “nlockers” y “ntransportistas” encargadas de cuantificar el número de compartimentos lockers, lockers y transportistas respectivamente.

Funciones y procedimientos.

- **void cargar_lockers(lockers **lo):** Carga los datos del fichero “Lockers.txt” en la estructura “*lockers*”. Recibe la estructura lockers.
- **void guardar_lockers(lockers **lo):** Guarda los datos de la estructura “*lockers*” en el fichero “Lockers.txt”. Recibe la estructura “*lockers*”.
- **void cargar_compartimentolockers(comp_lockers **cl):** Carga los datos del fichero “CompartimentosLockers.txt” en la estructura “*comp_lockers*”. Recibe la estructura comp_lockers.
- **void guardar_comp_lockers(comp_lockers **cl):** Guarda los datos de la estructura “*comp_lockers*” en el fichero “CompartimentosLockers.txt”. Recibe la estructura “*comp_lockers*”.

- **void cargar_transportista(transportistas **tr):** Carga los datos del fichero “Transportistas.txt” en la estructura “*transportistas*”. Recibe la estructura transportistas.
- **void guardar_descuentos(transportistas **tr):** Guarda los datos de la estructura “*transportistas*” en el fichero “Transportistas.txt”. Recibe la estructura “*transportista*”.
- **int disp_locker(lockers **lo):** Recibe la estructura lockers y devuelve el id del locker que el propio usuario elige.
- **void alta_lockers(lockers **lo):** Recibe la estructura lockers y da de alta un nuevo locker en el sistema dentro de la estructura lockers.
- **void baja_lockers(lockers **lo):** Recibe la estructura lockers y da de baja un locker.. Previamente se comprueba que dicho locker está disponible en el sistema.
- **void busqueda_lockers(lockers **lo):** Recibe la estructura lockers y devuelve un entero, -1 si no encuentra el locker introducido u otro entero mayor a -1 el cual corresponde con la posición de la estructura lockers de donde se encuentra el locker buscado.

- **void modificar_lo(lockers **lo):** Recibe la estructura lockers y modifica algún campo de la estructura locker disponible en el sistema.
- **void listar_lockers(lockers **lo):** Recibe la estructura lockers y lista todos los lockers almacenados en el sistema junto a todas sus características.
- **void modificar_transportistas(transportistas **tr):** Recibe la estructura transportistas y modifica algún campo de la estructura transportistas almacenado en el sistema.
- **void baja_transportista(transportistas **tr):** Recibe la estructura transportistas y da de baja un usuario transportista. Previamente se comprueba su existencia en el sistema.
- **int busqueda_transportistas(transportistas **tr):** Recibe la estructura transportistas y devuelve -1 si no se encuentra el transportista o un entero mayor a -1 correspondiente con la posición dentro del vector de estructuras donde se encuentra.
- **void listar_transportistas(transportistas **tr):** Recibe la estructura transportistas y lista todos los transportistas almacenados en el sistema.
- **void listar_usuariotransportista(transportistas **tr):** Recibe la estructura transportistas y lista las características del usuario transportista elegido. Se comprueba que dicho usuario transportista se encuentra existente en el sistema.

3.5 Devoluciones.

Sección de importación.

Incluye el fichero de cabecera “Logistica.h”, el cual corresponde al módulo “Devoluciones.h”. También incluye la librería “string.h”, “math.h” y “time.h”.

Sección de exportación.

Exporta los tipos registros ”devoluciones”, “pedidos” y “prod_pedidos” y “cliente”. También exporta las variables enteras globales nt_pedidos, nt_prodpedidos, ncliente, nadminprov y ntransport correspondientes al número de pedidos, productos pedidos, clientes administradores proveedores y transportistas. Por otro lado también contamos con las variables globales puerta_listar y puerta_alta (para saber si se puede o no lista y/o dar de alta) encargadas de cuantificar el número de compartimentos lockers, lockers y transportistas respectivamente.

Funciones y procedimientos.

- **void cargar_clientes(cliente **c, usuario **usu):** carga los datos del fichero correspondiente.
- **void listar_pedidos_cliente(pedidos **pe,prod_pedidos **ppe,int cliente):** le pasamos un cliente y se visualizará una lista de sus pedidos realizados en la plataforma.
- **void dev_alta_cliente(devoluciones **de,pedidos **pe,prod_pedidos **ppe,usuario **usu,cliente **c):** dar de alta una devolución de un cliente
- **void dev_alta_admin(devoluciones **de,pedidos **pe,prod_pedidos **ppe,usuario **usu,cliente **c):** dar de alta una devolución de un cliente.
- **dev_baja_admin(devoluciones **dev,pedidos **pe,prod_pedidos **ppe,usuario **usu,cliente **c):** se le da de baja a una devolución.
- **void dev_baja_admin(devoluciones **dev,pedidos **pe,prod_pedidos **ppe,usuario **usu,cliente **c):** se da de baja una devolución.
- **void dev_seguimiento(devoluciones **dev,pedidos **pe,usuario **usu,cliente **c):** para que el cliente pueda ver el estado de sus devoluciones.
- **void dev_enviar(devoluciones **dev,pedidos **pe,usuario **usu,cliente **c):** con esta función el cliente podrá ver las devoluciones que han sido aceptadas.

- **int dev_existe_devolucion(devoluciones **dev,int devol):** se le pasa el id de un pedido y se devuelve -1 si el pedido no se ha devuelto y !=-1 si existe en devoluciones.
- **int dev_existe_devolucion_seguimiento(devoluciones **dev,int devol):** se le pasa el id de un pedido y se devuelve -1 si el pedido no se ha devuelto y !=-1 si existe en devoluciones.
- **int dev_pendiente(devoluciones **dev, pedidos **pe,int id_aux_pedido):** se le pasa el id de un pedido y devuelve 1 si el estado del pedido es “pendiente”.
- **int dev_enviado(devoluciones **dev, pedidos **pe,int id_aux_pedido):** se le pasa el id pedido y devuelve 1 si su estado es “recibido”.
- **int dev_aceptado(devoluciones **dev, pedidos **pe,int id_aux_pedido):** se le pasa el id del pedido y devuelve 1 si su estado es “aceptado”.
- **int dev_pendiente_recibido_posicion(devoluciones **dev, pedidos **pe,int id_aux_pedido):** se le pasa el id del pedido y devuelve la posición dentro de la estructura devoluciones.
- **void dev_listar(devoluciones **dev,pedidos **pe):** gracias a esta función el admin visualizará todas las devoluciones del sistema.
- **void dev_baja(devoluciones **de):** se le da de baja a una devolución
- **void dev_modificar(devoluciones **de,pedidos **pe):** esta función se encarga de modificar el estado de la devolución. En el caso de ser aceptada la misma fecha de aceptación y fecha de caducidad, entonces se actualiza.
- **int dev_cliente_pedidos(int pedido,int cliente,pedidos **pe):** se le pasa un cliente y un pedido, y devuelve 1 si el pedido y el cliente coinciden, 0 si no.

3.6 Pedidos.

Sección de importación.

Incluye el fichero de cabecera “Pedidos.h”, el cual corresponde al módulo “Pedidos”. También incluye la librería “string.h”, “math.h” y “time.h”.

Sección de exportación.

Exporta los tipos registros ”pedidos”, “prod_pedidos”, “cliente”, “prod”, “usuario”, “comp_lockers”, “lockers”, “transportistas”, “descuentos” y “descliente”. También exporta las variables enteras globales “nt_pedidos”, “nprod_pedidos”, “cesta”, “ncesta”, “ncliente”, “nprod”, “ncomp_lockers” “nlockers”, “ntransportistas” , “ndescuentos” y “ndescliente” encargadas de cuantificar el número de pedidos, productos pedidos, productos en cesta, número de clientes, productos, compartimentos lockers, lockers y transportistas respectivamente.

Funciones y procedimientos.

- **void cargar_clientes(cliente **c,usuario **usu):** recibe la estructura cliente y usuario y carga los datos del fichero correspondiente.
- **void guardar_cliente(cliente **c):** recibe la estructura cliente y guarda los datos en el fichero correspondiente.
- **void cargar_productos(prod **pr):** recibe la estructura prod y carga los datos del fichero correspondiente.
- **void guardar_productos(prod **pr):** recibe la estructura prod y guarda los datos en el fichero correspondiente.
- **void cargar_lockers(lockers **lo):** recibe la estructura lockers y carga los datos del fichero correspondiente.

- **void guardar_lockers(lockers **lo):** recibe la estructura lockers y guarda los datos en el fichero correspondiente.
- **void cargar_compartimentolockers(comp_lockers **cl):** recibe la estructura comp_lockers y carga los datos en el fichero correspondiente.
- **void guardar_comp_lockers(comp_lockers **cl):** recibe la estructura comp_lockers y guarda los datos en el fichero correspondiente.
- **void cargar_descuentos(descuentos **des):** recibe la estructura descuentos y carga los datos del fichero correspondiente.
- **void guardar_descuentos(descuentos **des):** recibe la estructura descuentos y guarda los datos en el fichero correspondiente.
- **void cargar_descliente(descliente **desc):** recibe la estructura descliente y carga los datos del fichero correspondiente.
- **void guardar_descliente(descliente **desc):** recibe la estructura descliente y guarda los datos en el fichero correspondiente
- **void cargar_transportista(transportistas **tr):** recibe la estructura transportistas y carga los datos del fichero correspondiente.
- **void guardar_transportistas transportistas **tr):** recibe la estructura transportistas y guarda los datos en el fichero correspondiente.
- **void locker_compartimentos_act(comp_lockers **cl, lockers **lo,int locker):** recibe la estructura comp_lockers, lockers y un entero y actualza el almacenamiento de ese locker.
- **int val_codprom(descuentos **des):** recibe la estructura descuentos y devuelve 1 si el descuento está activo, 0 si no.

- **int busqueda_des(descuentos **des):** recibe la estructura descuentos y devuelve -1 si no encuentra el descuentos o un entero mayor que -1 con la posición dentro de la estructura descuentos correspondiente al descuento que se busca.
- **int busqueda_cliente(descliente **desc,int cliente)**: recibe la estructura descliente y devuelve -1 si no encuentra el descuento del cliente o un entero mayor que -1 con la posición dentro de la estructura descliente del que se busca.
- **void listado_lockers(lockers **lo, char *ciudad):** recibe la estructura lockers y los lista.
- **void prueba_impresion(pedidos **pe,prod_pedidos **ppe):** recibe la estructura pedidos y prod_pedidos y realiza una prueba de impresión.
- **int pe_existe_pedido(pedidos **pe,int pedido):** se le pasa un ID_pedido y si pos!=-1 ese pedido existe, si pos==-1 ese pedidp no existe.
- **int pe_existe_producto(prod_pedidos **ppe,int pedido):** se le pasa un ID_pedido y devuelve un vector con los índices de las estructuras prod_pedidos para saber dónde están los productos de ese pedido.
- **int pe_existe_cliente(pedidos **pe,int aux_cliente):** se le pasa un ID_cliente y

  devuelve un vector con los índices de las estructuras pedidos para saber dónde están los productos de ese cliente.
- v**oid pe_asignar_trans(pedidos **pe, prod_pedidos **ppe, cliente **c, transportistas **tr,int aux_pedido):** se le pasa un id_pedido, la realiza en el sistema cuando se da de alta un pedido y asigna a los transportistas, los pedidos en base a la localidad de entrega y ubicación del mismo.
- **void pe_asignar_locker(pedidos **pe, prod_pedidos **ppe, cliente **c, lockers **lo, comp_lockers **cl, int aux_pedido):** se le pasa un id_pedido, la realiza en el sistema cuando se da de alta un pedido y asigna a los lockers, los pedidos en base a la localidad de entrega y ubicación del locker.

- **void incrementar_fecha_locker(comp_lockers **cl, int incremento, char fechayhora[9],int pos):** cuando la estructura está inicializada ,se le pasa un incremento, fecha de incrementar y la posición del campo, y luego incremente la fecha acorde al incremento para el locker.
- **void dar_baja_pedidos(pedidos **pe, prod_pedidos **ppe, cliente **c):** cuando las estructuras están inicializadas, se elimina un pedido.
- **void dar_alta_pedidos(pedidos **pe,prod_pedidos **ppe, cliente **c,descuentos **des, usuario **usu, prod **pr, descliente **desc, lockers **lo,comp_lockers **cl, transportistas **tr):** cuando las estructuras están inicializadas se crea un pedido, con sus correspondientes productos asignados a el.
- **void pe_modificar(pedidos **pe):** modifica algún campo de un pedido en concreto.
- **void pe_modificar_prod(prod_pedidos **ppe,pedidos **pe,cliente **c):** cuando las estructuras están inicializadas se modifica algún campo de un producto pedido en concreto.
- **void pe_listar_admin(pedidos **pe,cliente **c):** cuando las estructuras están inicializadas se listan todos los pedidos.
- **void pe_listar_estado_admin(pedidos **pe):** cuando las estructuras están inicializadas, se listan todos los pedidos.
- **void pe_asignar_locker_admin(pedidos **pe, prod_pedidos **ppe, cliente **c, lockers **lo, comp_lockers **cl):** cuando las estructuras están inicializadas se asigna los lockers a los pedidos en base a la localidad de entrega y ubicación del locker.
- **void pe_asignar_trans_admin(pedidos **pe, prod_pedidos **ppe, cliente **c, transportistas **tr):** cuando las estructuras están inicializadas se asigna los transportistas a los pedidos en base a la localidad de entrega y ubicación del mismo.

- **void incrementar_fecha(prod_pedidos **ppe, int incremento, char fechayhora[9]):** la estructura debe estar inicializada, luego se le pasa un incremento, fecha a incrementar y la posición del campo, y se incrementa la fecha acorde al incremento para el locker.
- **void listar_pedidos_cliente(pedidos **pe,int cliente):** se le debe pasar la estructura pedidos y el ID cliente, luego el cliente podrá visualizar una lista de sus pedidos realizados en la plataforma.
- **void consultar_estado(pedidos **pe ,prod_pedidos **ppe,int cliente):** recibe las estructuras pedidos, prod_pedidos y el ID cliente, y realiza una consulta del estado de sus pedidos.
- **void recogida_pedido(pedidos **pe,prod_pedidos **ppe,lockers **lo,comp_lockers **cl,int cliente):** recibe las estructuras pedidos, prod_pedidos, lockers, comp_lockers y el ID cliente y realiza la correspondiente recogida y actualización de todos los datos que conlleva.
- **void estado_pedido(pedidos **pe,prod_pedidos **ppe,transportistas **tr,int aux_trans):** se le pasa por parámetros su id_trans. Se cambia el estado de los productos pedidos.
- **int listar_pedidos_proveedor(pedidos **pe, prod_pedidos **ppe, prod **pr, usuario **usu):** las estructuras deben estar inicializadas. Lista todos los pedidos de ese proveedor.
- **void pe_asignar_locker_proveedor(pedidos **pe, prod_pedidos **ppe, cliente **c, lockers **lo, comp_lockers **cl,usuario **usu,prod **pr):** necesita tener las estructuras inicializadas. Asigna los lockers a los pedidos en base a la localidad de entrega y ubicación del locker.

4. DESCRIPCIÓN DEL PROCESO DE INSTALACIÓN Y DE EJECUCIÓN

Instalación.

Para poder instalar el el programa, simplemente debes de tener el ejecutable en una carpeta cualquiera de tu ordenador y en el mismo directorio los ficheros de los datos, si existen. De lo contrario los ficheros serán creados por el propio juego.

Ejecución.

Al ejecutarlo, el programa funcionará con total normalidad. Podremos disfrutar de una increíble experiencia de usuario de nuestra plataforma online “ESIZON”.

5. CASOS DE PRUEBA

5.1 Descuentos.

5.1.1 Pruebas de caja negra.

Vamos a realizar las pruebas de caja negra con un fragmento de la función “baja_descuentos”:

```
while (aux == 0){

    listar_des(des);
    printf("\nA continuacion daremos de baja un descuento.\n\n");

    while (aux2 == 0){
        if ((i = busqueda_des(des)) >= 0){

            descuentos aux;
            //Intercambiamos valores
            aux = (*des)[ndescuentos-1];
            (*des)[ndescuentos-1] = (*des)[i];
            (*des)[i] = aux;
            //Eliminamos último elemento
            ndescuentos--;
            *des=(descuentos*)realloc((*des),(ndescuentos+1)*sizeof(descuentos));

            aux2 = 1;

            printf("\nDescuento eliminado con exito!!\n\n\n");
        }
        else{
            printf("Id. Codigo Descuento erroneo...\n\n");
        }
    }
    guardar_descuentos(des);

    printf("¿Desea realizar otra baja? s/n: ");
    scanf("%s", cadena);

    if(!strcmp(cadena,"n") || (!strcmp(cadena,"N")))
        aux = 1;
    else
        aux2 = 0;
}
```

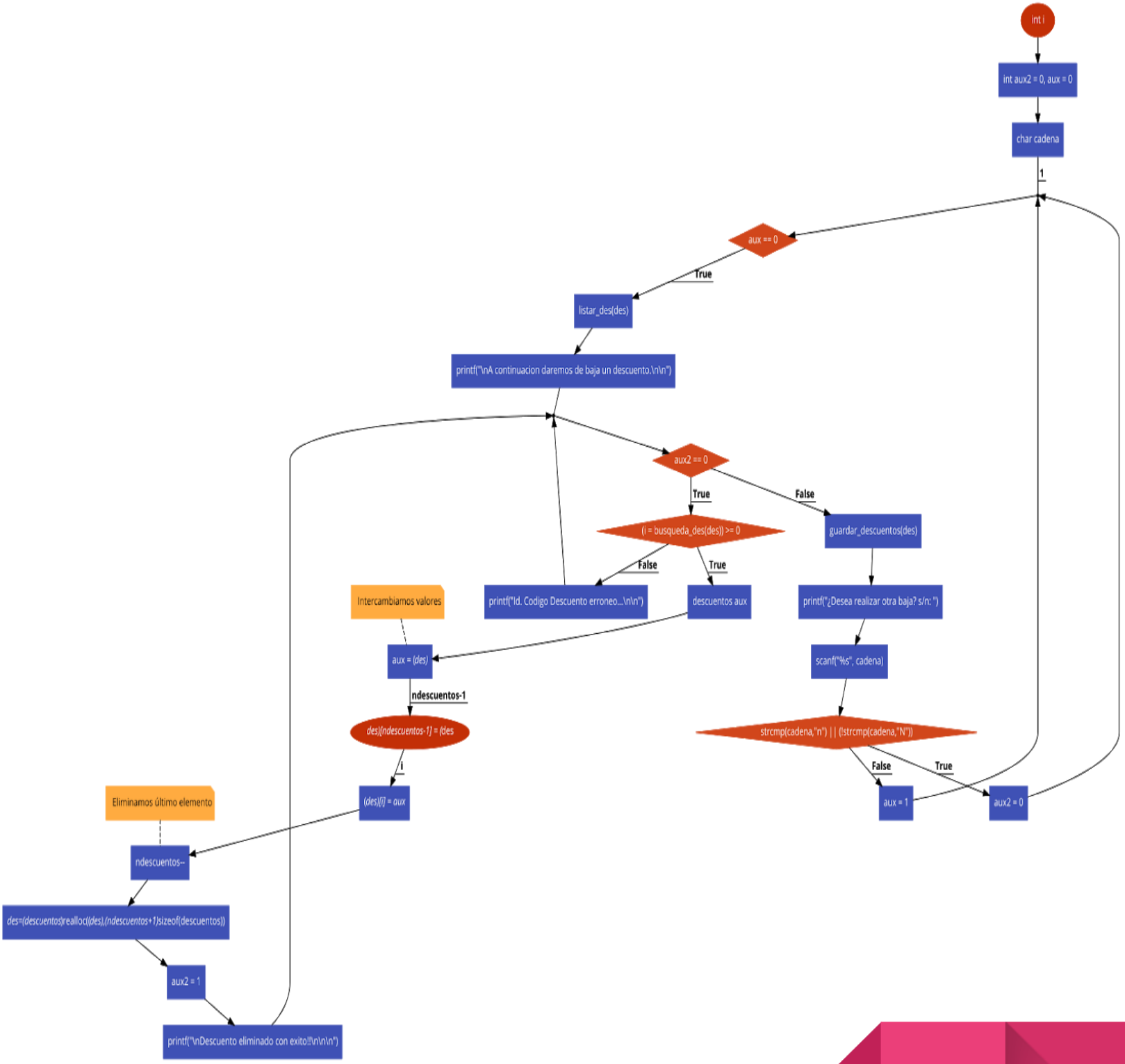
ENTRADA	SALIDA
Introducimos “n/N”	El programa finaliza
Introducimos otro carácter	El programa vuelve a preguntarnos por otro descuento para proceder a darle de baja

Este fragmento cumple con creces las entradas/salidas de acuerdo con las condiciones establecidas.

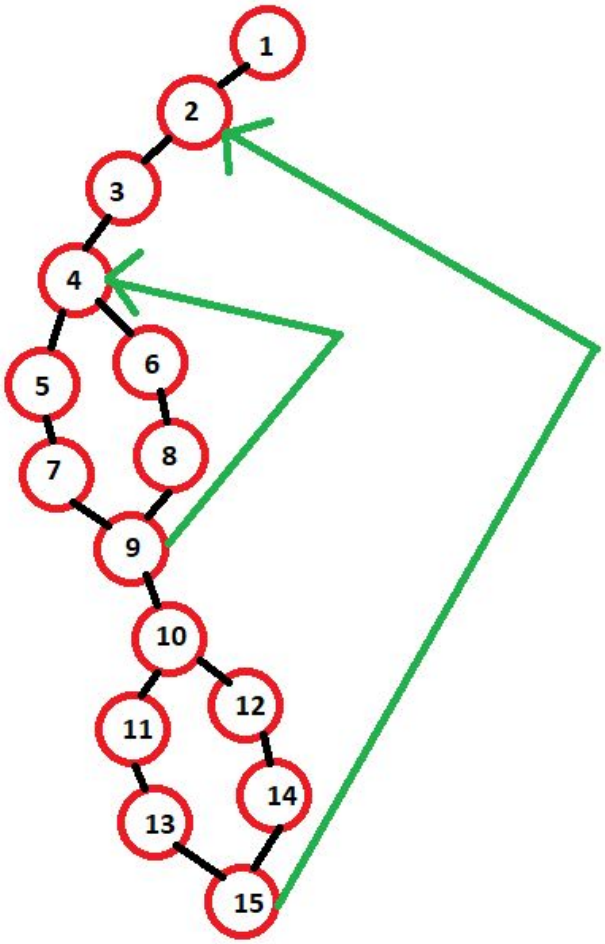
5.1.2 Pruebas de caja blanca.

Prueba de ruta básica.

El diagrama de flujo de la función bajar_des


int i
int aux2 = 0, aux = 0
char cadena
1
aux == 0
True
listar_des(des)
printf("\nA continuacion daremos de baja un descuento.\n\n")
aux2 == 0
True
False
(i = busqueda_des(des)) >= 0
False
True
guardar_descuentos(des)
Intercambiamos valores
printf("Id. Codigo Descuento erroneo...\n\n")
descuentos aux
printf("¿Desea realizar otra baja? s/n: ")
aux = (des)
scanf("%s", cadena)
ndescuentos-1
strcmp(cadena,"n") || (!strcmp(cadena,"N"))
des[ndescuentos-1] = (des
False
True
i
(des)[i] = aux
aux = 1
aux2 = 0
Eliminamos último elemento
ndescuentos--
des=(descuentos)realloc((des),(ndescuentos+1)sizeof(descuentos))
aux2 = 1
printf("\nDescuento eliminado con exito!!\n\n\n")

El grafo es el siguiente:



Calculemos, ahora pues, la complejidad ciclomática para saber el número de rutas independientes:

- V(G)= NNP+1= 4 + 1 = 5

➢ V(G)= 18-15+2= 5

Veamos ahora las 5 rutas independientes que salen del grafo:

RUTA Nº	
1	1-2-3-4-5-7-9-10-11-13-15
2	1-2-3-4-5-7-9-10-12-14-15
3	1-2-3-4-6-8-9-10-12-14-15
4	1-2-3-4-6-8-9-10-11-13-15
5	1-2-3-4-6-8-9-4-6-8-9-10-12-14-15

Por lo tanto, nuestras rutas independientes coinciden y el flujo del fragmento es correcto.

5.2 Logística.

5.2.1 Pruebas de caja negra.

Vamos a realizar las pruebas de caja negra con un fragmento de la función “modificar_lo”:

```
while (aux == 0){
    printf(" (1) Id. Código Locker\n (2) Localidad\n (3) Provincia\n (4) Ubicacion\n "
    "(5) Nº total compartimentos\n (6) Nº compartimentos ocupados\n");
    printf("Indica el numero correspondiente al campo a modificar: ");
    scanf("%i", &campo);

    if ((campo < 1) || (campo > 6))
        printf("\nIntroduce un numero del 1 al 6\n\n");
    else
        aux = 1;
}
```

ENTRADA	SALIDA
Introducimos un nº menor que 1 o mayor que	Pide de nuevo el valor

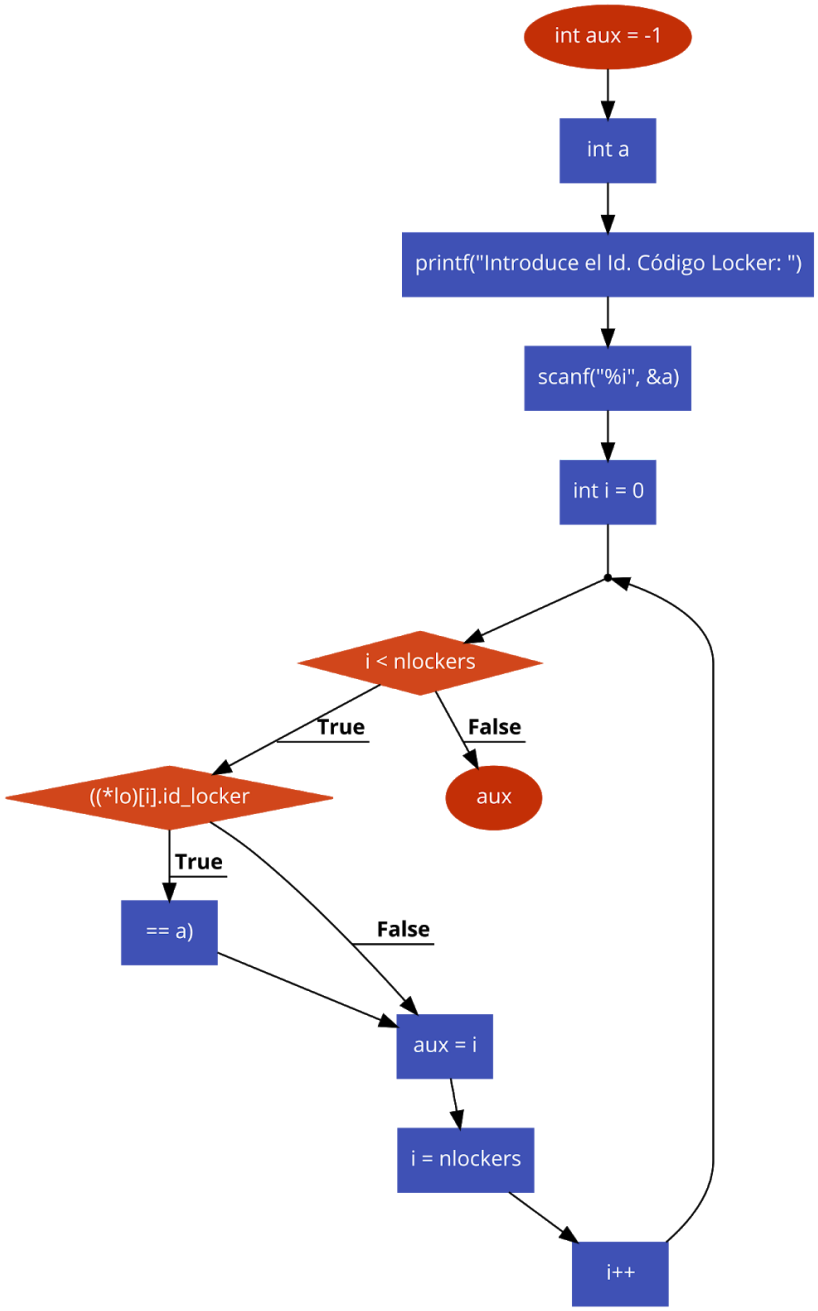
6	
Introducimos un nº entre 1 y 6	Se guarda el valor en la variable campo y sale del bucle

Este fragmento cumple con creces las entradas/salidas de acuerdo con las condiciones establecidas. La variable campo es la encargada de que el flujo del programa funcione correctamente, luego cumple con los objetivos para los que ha sido diseñada.

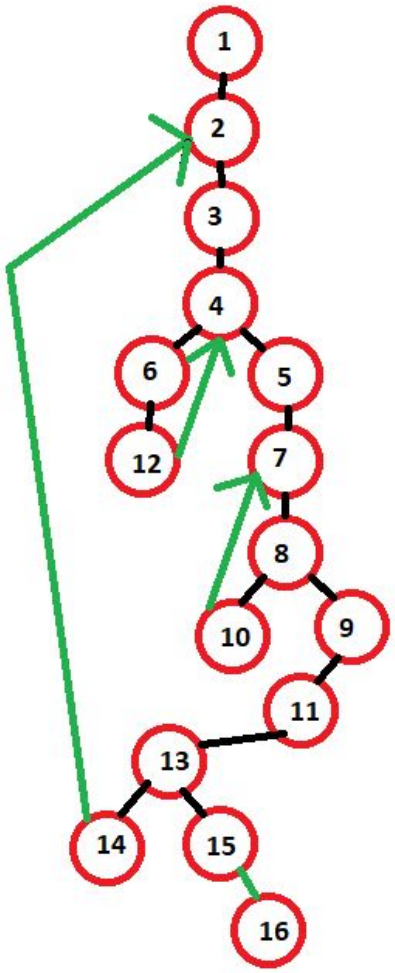
5.1.2 Pruebas de caja blanca.

Prueba de ruta básica.

El diagrama de flujo de la función modificar_lo


int aux = -1
int a
printf("Introduce el Id. Código Locker: ")
scanf("%i", &a)
int i = 0
i < nlockers
True
False
aux
((*lo)[i].id_locker
True
== a)
False
aux = i
i = nlockers
i++

El grafo es el siguiente:



Calculemos, ahora pues, la complejidad ciclomática para saber el número de rutas independientes:

- V(G)= NNP+1= 3 + 1 = 4

➢ V(G)= 18-16+2= 4

Veamos ahora las 4 rutas independientes que salen del grafo:

RUTA Nº	
1	1-2-3-4-5-7-8-9-11-13-15-16
2	1-2-3-4-6-12-4-5-7-8-9-11-13-15-16
3	1-2-3-4-5-7-8-10-7-8-9-11-13-15-16
4	1-2-3-4-5-7-8-9-11-13-14-2-3-4-5-7-8-9-11-13-15-16

Por lo tanto, nuestras rutas independientes coinciden y el flujo del fragmento es correcto.

5.3 Devoluciones.

5.3.1 Pruebas de caja negra.

Vamos a realizar las pruebas de caja negra con un fragmento de la función “dev_listar”:

```
do{

        system("cls");

        printf("\n|DEVOLUCIONES ESIZON|\n");

    printf("\nIntroduzca el ID del cliente:  ");
    fflush(stdin);
    scanf("%i",&aux_cliente);


    p1=dev_existe_cliente(pe,aux_cliente);

}while((aux_cliente<=0 || aux_cliente>nt_pedidos-1) && p1==-1);
```

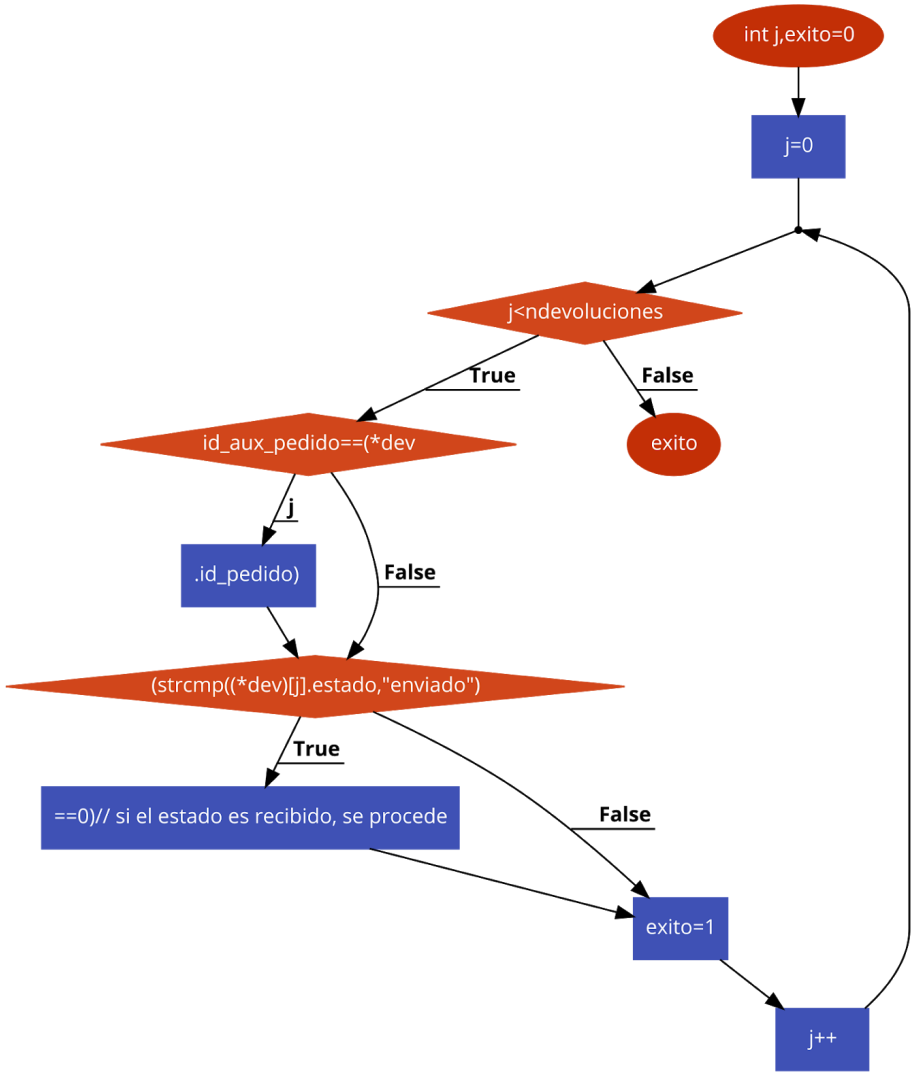
ENTRADA	SALIDA
pi = -1 y el id cliente es menor o igual que cero, mayor al nº max en la estructura o ambos	Se entra en el bucle de nuevo
p1 != -1 o el id cliente está entre 1 y nt_pedidos	id correcto por lo tanto se sale del bucle

Este fragmento cumple con creces las entradas/salidas de acuerdo con las condiciones establecidas. Las variables pi y aux_cliente devuelve lo que le corresponde en función del id introducido luego cumple con los objetivos para los que ha sido diseñada.

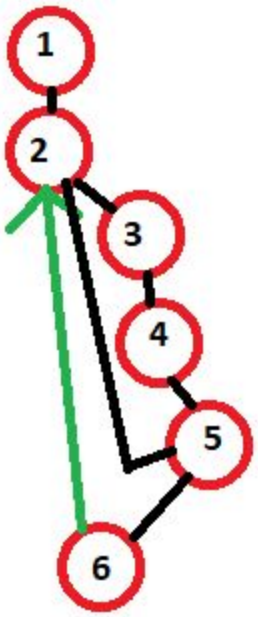
5.1.2 Pruebas de caja blanca.

Prueba de ruta básica.

El diagrama de flujo de la función dev_enviado



El grafo es el siguiente:



Calculemos, ahora pues, la complejidad ciclomática para saber el número de rutas independientes:

- V(G)= NNP+1= 2 + 1 = 3
- V(G)= 7-6+2= 3

Veamos ahora las 3 rutas independientes que salen del grafo:

RUTA Nº	
1	1-2-3-4-5-6
2	1-2-6
3	1-2-3-4-5-2-3-4-5-6

Por lo tanto, nuestras rutas independientes coinciden y el flujo del fragmento es correcto.

5.4 Pedidos.

5.4.1 Pruebas de caja negra.

Vamos a realizar las pruebas de caja negra con un fragmento de la función “modificar_pedidos”:

```
do
{
    printf(" Introduce el numero del dato que desea modificar: ");
    scanf("%i",&op);
    if(op<1 || op>9)
        puts(" Numero no valido.");
}
while(op<1 || op>9);
```

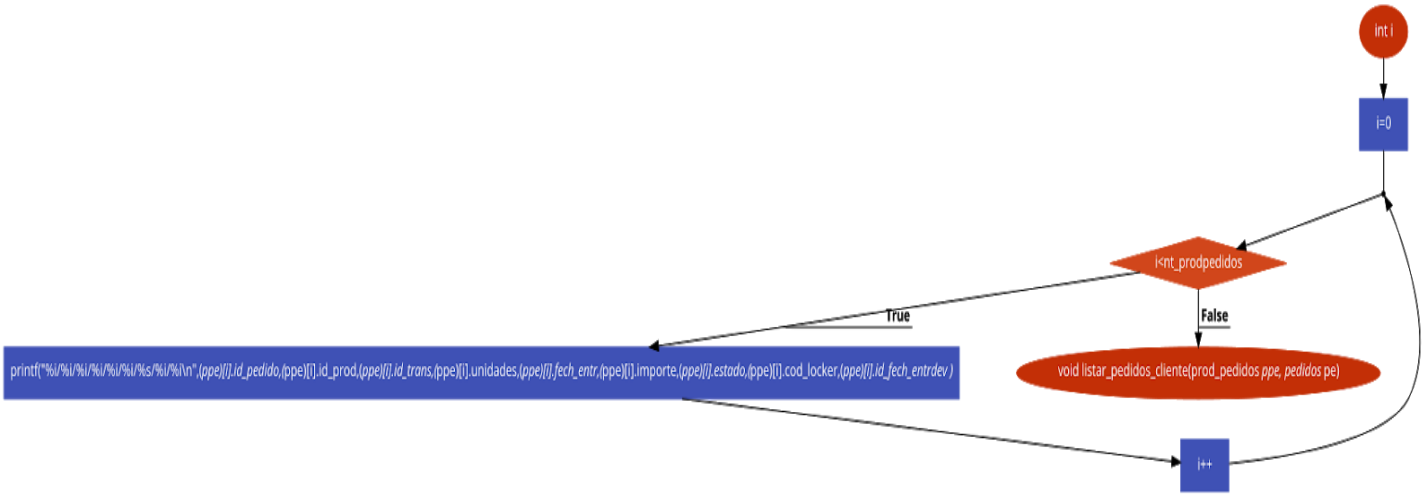
ENTRADA	SALIDA
Introducimos un entero menor que 1 o mayor que 9	No se sale del bucle y vuelve a dar otra vuelta
Introducimos un entero entre 1 y 9	Se guarda el valor en op correctamente y sale del bucle

Este fragmento cumple con creces las entradas/salidas de acuerdo con las condiciones establecidas. La variable op es la encargada de que el flujo del programa funcione correctamente, luego cumple con los objetivos para los que ha sido diseñada.

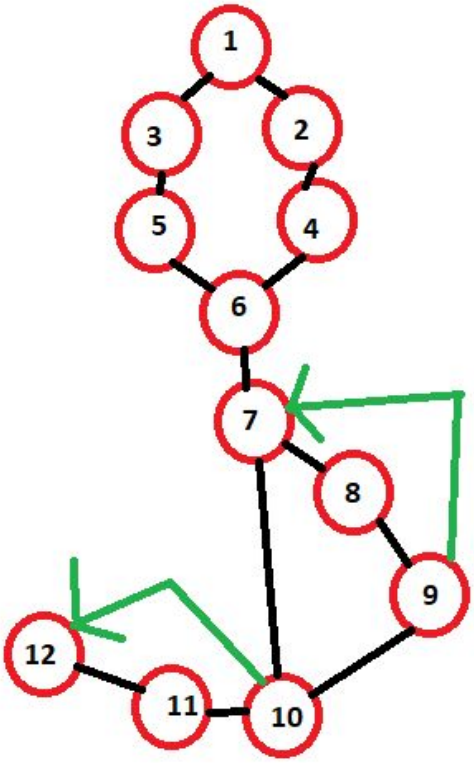
5.5.2 Pruebas de caja blanca.

Prueba de ruta básica.

El diagrama de flujo de la función listar_pedidos_cliente



El grafo es el siguiente:



Calculemos, ahora pues, la complejidad ciclomática para saber el número de rutas independientes:

- V(G)= NNP+1= 2 + 1 = 3
- V(G)= 13-12+2= 3

Veamos ahora las 3 rutas independientes que salen del grafo:

RUTA Nº	
1	1-2-4-6-7-8-9-10-11-12
2	1-3-5-6-7-10-11-12
3	1-2-4-6-7-8-9-10-12

Por lo tanto, nuestras rutas independientes coinciden y el flujo del fragmento es correcto.

5.5 Datos/Usuarios.

5.5.1 Pruebas de caja negra.

Vamos a realizar las pruebas de caja negra con un fragmento de la función “comprobar_pass”:

```
do
{
    printf("1.- Reintentar\n2.- Cerrar\nElija una opcion: ");
    fflush(stdin);
    scanf("%i", &op);
    system("cls");
}
while(0>=op || op>2);
```

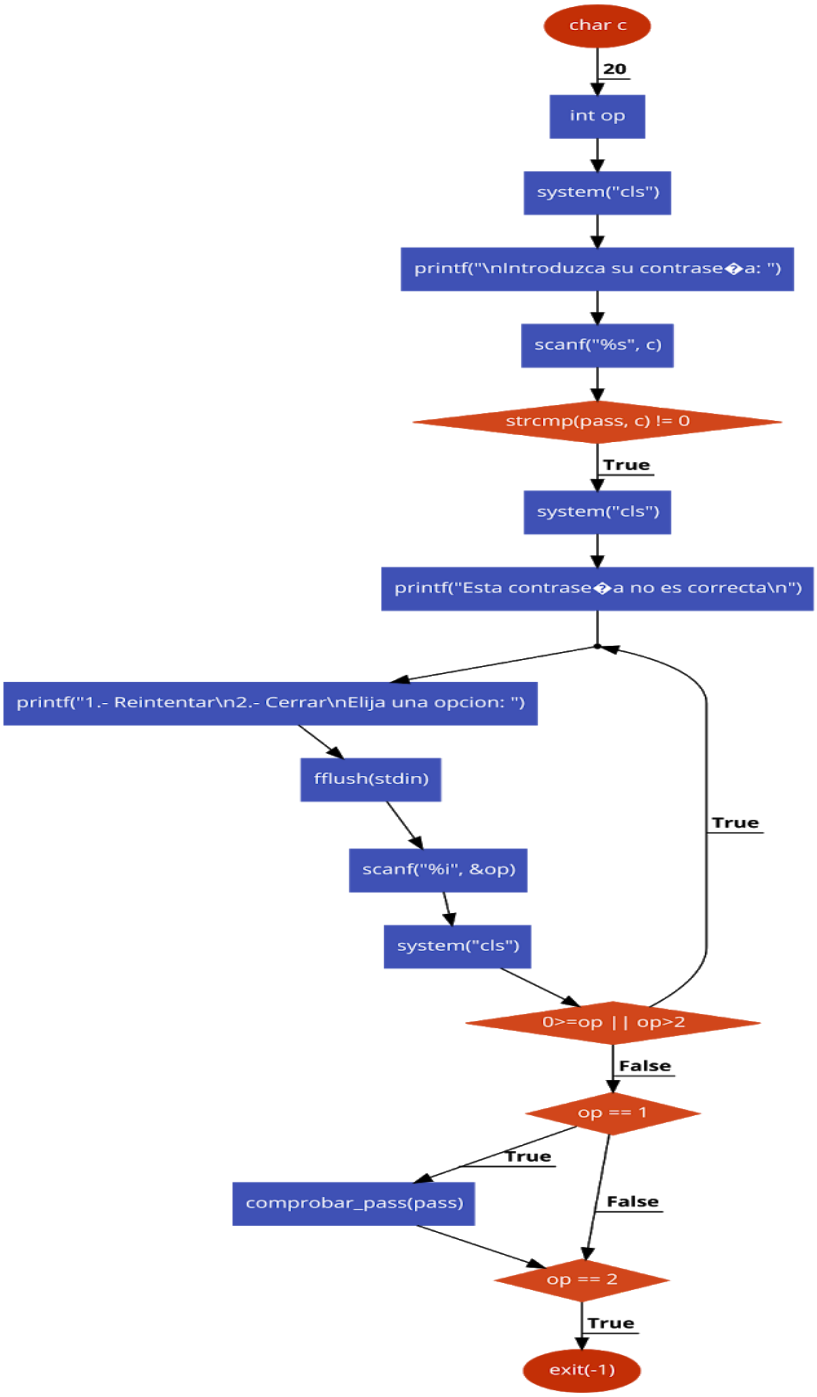
ENTRADA	SALIDA
Introducimos un entero menor o igual que 0 o mayor a 2	No sale del bucle hasta que se introduzca un número dentro del rango
Introducimos 1	Se llama de nuevo a la función y se vuelve a solicitar la contraseña para su comprobación
Introducimos 2	El programa finaliza

Este fragmento cumple con creces las entradas/salidas de acuerdo con las condiciones establecidas. La variable op es la encargada de que el flujo del programa funcione correctamente, luego cumple con los objetivos para los que ha sido diseñada.

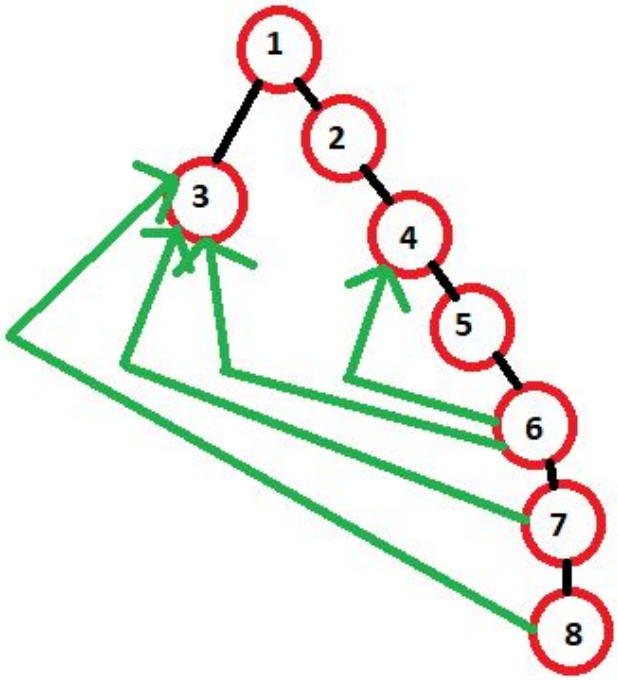
5.5.2 Pruebas de caja blanca.

Prueba de ruta básica.

El diagrama de flujo de la función comprobar_pass


char c
20
int op
system("cls")
printf("\nIntroduzca su contrase&#65533;a: ")
scanf("%s", c)
strcmp(pass, c) != 0
True
system("cls")
printf("Esta contrase&#65533;a no es correcta\n")
printf("1.- Reintentar\n2.- Cerrar\nElija una opcion: ")
fflush(stdin)
scanf("%i", &op)
system("cls")
0>=op || op>2
True
False
op == 1
True
comprobar_pass(pass)
False
op == 2
True
exit(-1)

El grafo es el siguiente:



Calculemos, ahora pues, la complejidad ciclomática para saber el número de rutas independientes:

- V(G)= NNP+1= 4 + 1 = 5
- V(G)= 11-8+2= 5

Veamos ahora las 5 rutas independientes que salen del grafo:

RUTA Nº	
1	1-2-4-5-6-7-8-3
2	1-3

3	1-2-4-5-6-4-5-6-7-8-3
4	1-2-4-5-6-3
5	1-2-4-5-6-7-3

Por lo tanto, nuestras rutas independientes coinciden y el flujo del fragmento es correcto.

5.6 Productos/Categorías.

5.6.1 Pruebas de caja negra.

Vamos a realizar las pruebas de caja negra con un fragmento de la función “modificar_categoria”:

```
do
{
    printf("1.- Reintentar\n2.- Cerrar\nElija una opcion: ");
    fflush(stdin);
    scanf("%i", &op);
    system("cls");
}
while(0>=op|| op>2);
```

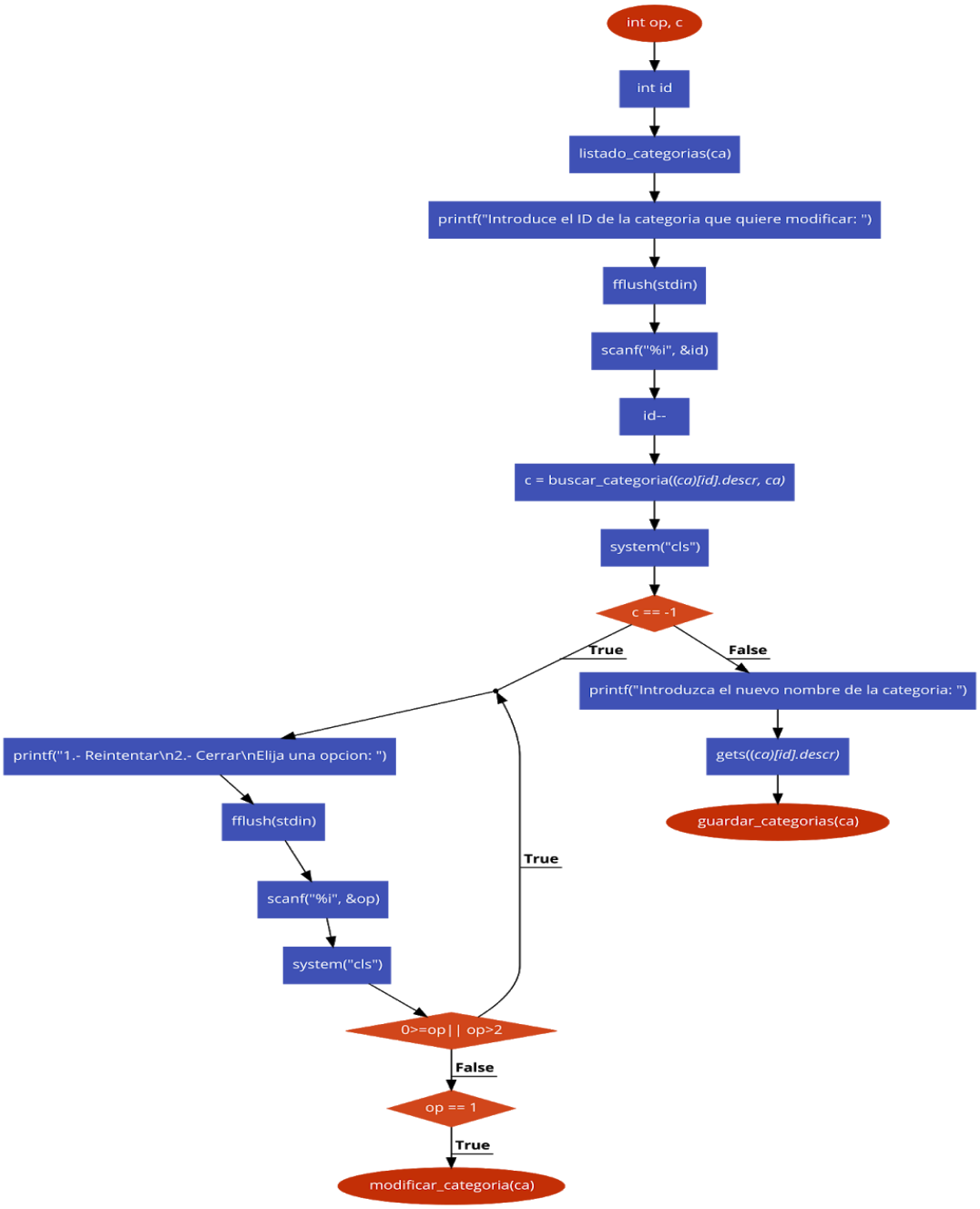
ENTRADA	SALIDA
Introducimos un entero menor que 0 o mayor a 2	No sale del bucle hasta que se introduzca un número dentro del rango
Introducimos 1	Se llama de nuevo a la función y se vuelve a solicitar el id pedido para su comprobación
Introducimos 2	El programa finaliza

Este fragmento cumple con creces las entradas/salidas de acuerdo con las condiciones establecidas. La variable op es la encargada de que el flujo del programa funcione correctamente, luego cumple con los objetivos para los que ha sido diseñada.

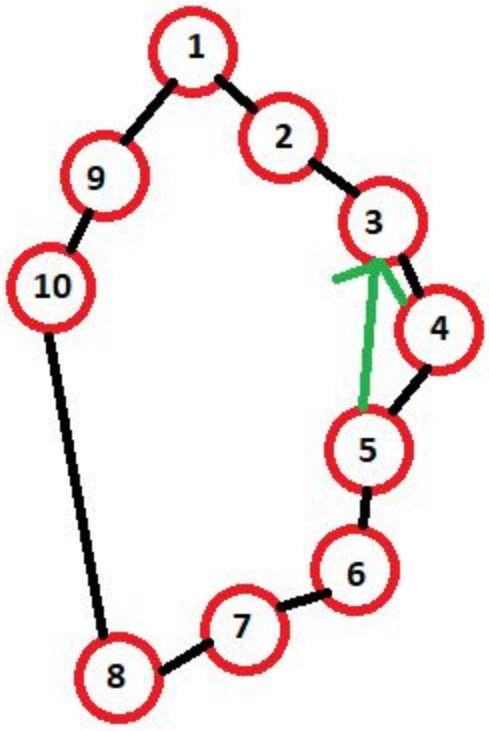
5.5.2 Pruebas de caja blanca.

Prueba de ruta básica.

El diagrama de flujo de la función modificar_categoria


int op, c
int id
listado_categorias(ca)
printf("Introduce el ID de la categoria que quiere modificar: ")
fflush(stdin)
scanf("%i", &id)
id--
c = buscar_categoria((ca)[id].descr, ca)
system("cls")
c == -1
True
False
printf("Introduzca el nuevo nombre de la categoria: ")
gets((ca)[id].descr)
guardar_categorias(ca)
printf("1.- Reintentar\n2.- Cerrar\nElija una opcion: ")
fflush(stdin)
scanf("%i", &op)
system("cls")
True
0>=op|| op>2
False
op == 1
True
modificar_categoria(ca)

El grafo es el siguiente:



Calculemos, ahora pues, la complejidad ciclomática para saber el número de rutas independientes:

- V(G)= NNP+1= 2 + 1 = 3
- V(G)= 11-10+2= 3

Veamos ahora las 3 rutas independientes que salen del grafo:

RUTA Nº	
1	1-2-3-4-5-6-7-8
2	1-2-3-4-5-3-4-5-6-7-8
3	1-9-10-8

Por lo tanto, nuestras rutas independientes coinciden y el flujo del fragmento es correcto.