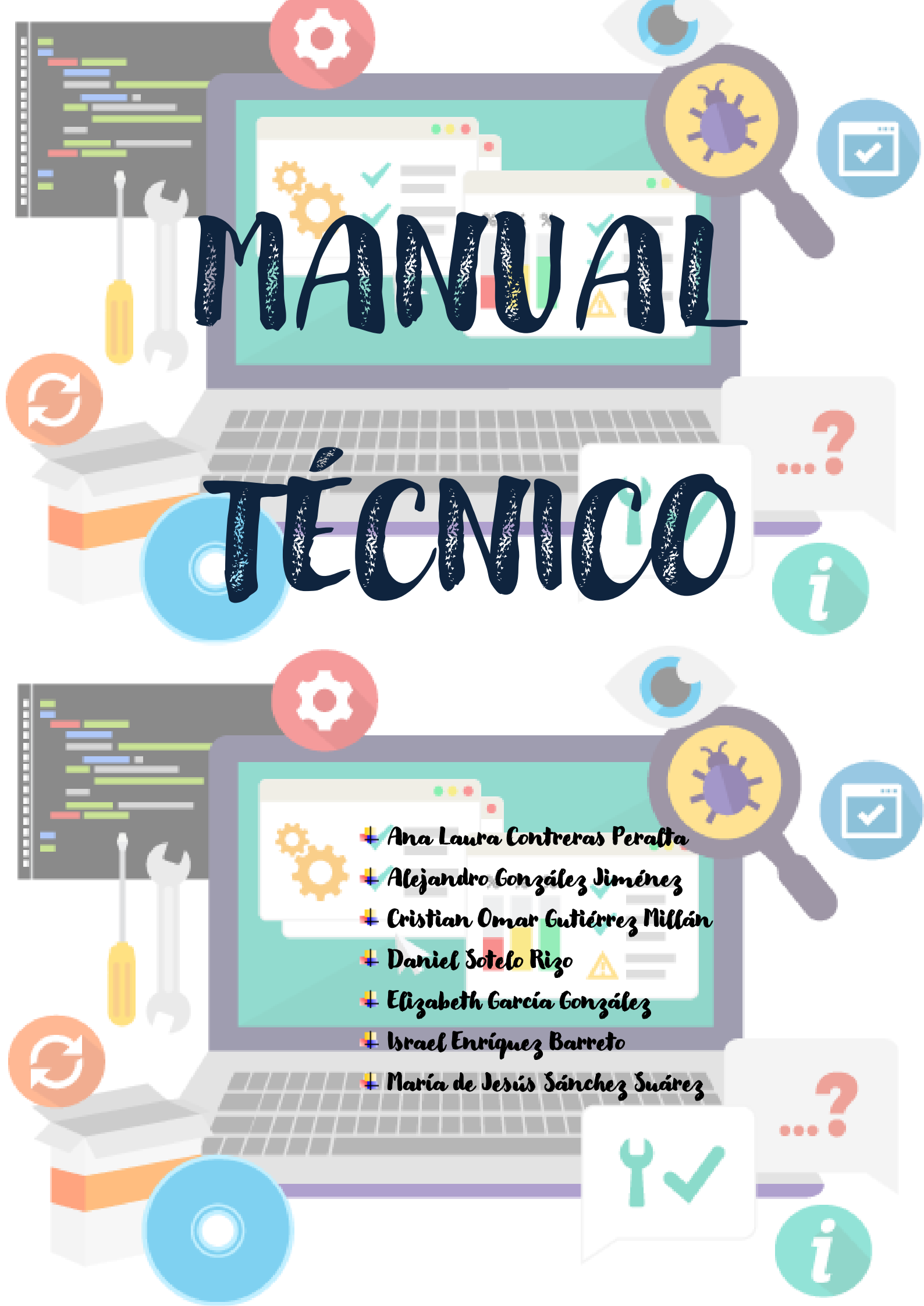


MANUAL TÉCNICO

- Ana Laura Contreras Peralta
- Alejandro González Jiménez
- Cristian Omar Gutiérrez Millán
- Daniel Sotelo Rizo
- Elizabeth García González
- Israel Enríquez Barreto
- María de Jesús Sánchez Suárez

ÍNDICE

1. Descripción del problema

Se debe de tener el registro de inventarios y proveedores de un kiosco para la venta de artículos accesorios de celulares.
El usuario podrá llevar el control, lo cual le permitirá tener acceso a los reportes y ventas, además del control de proveedores, mediante una opción para altas, una para bajas, consultas y además otra para realizar modificaciones.
Para el área de administración se deberán tener 3 opciones, las cuales serán para:

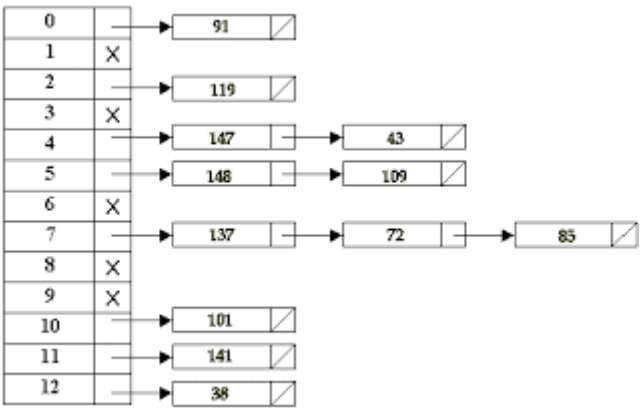
1. Reporte de ventas
2. Reporte de proveedores
3. Reporte de inventario de los accesorios

El programa tendrá opciones para generar dos informes a solicitud del usuario: Reporte de accesorios, reporte de ventas. Según lo que se eligió, el usuario podrá obtener sus informes en pantalla o en archivo texto.

2. Propuesta de solución y limitantes (incluye E y S y fundamentos teóricos)

Como propuesta de solución se realizó un programa el cual en el menú principal tiene las opciones de: 1) Accesorios, 2) Proveedores, 3) Vendedor, 4) Ventas, 5) Informes, 6) Administración, 7) Ayuda y por último 8) Salir.
En las opciones 1, 2 y 3 nos dará la opción de hacer altas, bajas, cambios y consultas de su respectivo registro.
En la opción 4 nos permitirá hacer ventas o reembolsos.
En la opción 5 podremos crear o ver informes sobre los accesorios, proveedores, vendedores y ventas.
En la opción 6 podremos crear un archivo, hacer un respaldo de los archivos del programa, restaurar los respaldos y eliminar los elementos vacíos de los archivos secuenciales.
La opción 7 nos permitirá abrir uno de los 2 manuales.

Para el manejo del archivo de accesorios se optó por usar un diseño directo el cual consiste en ***"una colección de registros de longitud fija almacenados uno al lado del otro en un dispositivo de almacenamiento de acceso directo"*** (Para más información ir a sección 7, referencia 1.0)
Se creó una estructura para los productos la cual nos sirve como nodo y esto nos servirá para manejar las colisiones como listas simplemente enlazadas que se caracterizan por ***"ser una colección de nodos que tienen una sola dirección y que en conjunto forman una estructura de datos lineal. Cada nodo es un objeto compuesto que guarda una referencia a un elemento (dato) y una referencia a otro nodo (dirección)"*** (Para más información ir a la sección 7, referencia 1.1); Una colisión es: ***"una situación que se produce cuando dos entradas distintas a una función de hash producen la misma salida"*** (Para más información ir a la sección 7, referencia 1.2)
Una vez, que hemos cerrado el programa y deseemos volver a abrir, el programa tiene que acomodar su respectivo índice con la función hash que ***"son un algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija y que independientemente de la longitud de los datos de entrada, el valor hash de salida tendrá siempre la misma longitud"***(Para más información ir a sección 7 referencia 1.3) en este caso la función hash retornará los 2 primeros números del código.

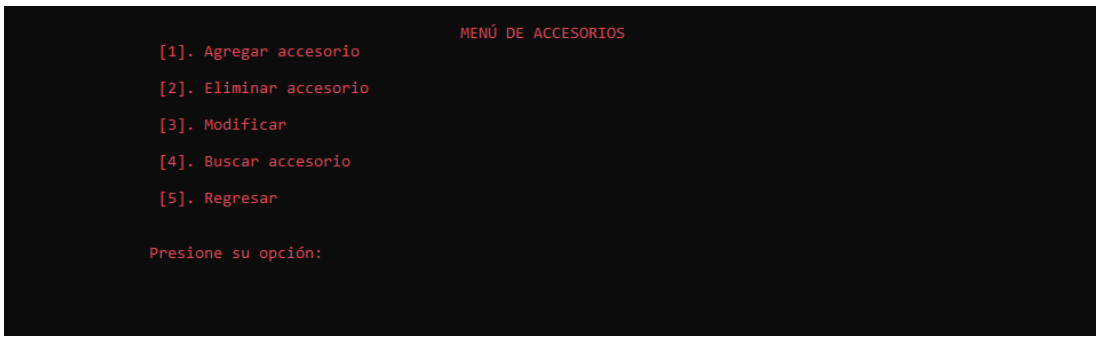


Para el manejo de los archivos de proveedores, vendedores y ventas se optó por usar un diseño secuencial. Ya que ocupa menos espacio en disco y al momento de hacer altas, bajas modificaciones o consultas de estos, la velocidad no es una prioridad, aunque lo haga a una velocidad muy buena.

Entonces se usan registros para accesorios, proveedores, vendedores, ventas y un registro más para manejar las colisiones de los productos.

2.1 Propuesta para accesorios

En el apartado de accesorios se buscó cumplir la necesidad de poder llevar a cabo altas, eliminar, modificar y la de buscar algún accesorio como se puede ver en la siguiente imagen:



Para poder agregar un accesorio se pedirá ingresar un código de producto, el cual será validado por una función dentro del programa la cual es llamada validar Código, esta función lo que hace es checar que se estén ingresando solo 2 letras y 4 números para así poder guardar un código correcto.
Se propuso crear un archivo de texto llamado inventario para poder llevar el control y registro de los productos que han sido ingresados en el kiosko, pero también se propuso una opción dentro de consola en el menú de accesorios para poder checar algún accesorio, este es el apartado número 4 y al ingresar el código de algún producto este nos desplegara sus datos como lo son: código, color, precio al que se compró, precio de venta, productos en existencia, el número de unidades que fueron adquiridas por el dueño del Kiosko, la marca del producto, el modelo y el proveedor del producto.
También se propuso el pedir una clave de proveedor al dar de alta un producto, si la clave que se ingresa no existe, el sistema no nos dejará ingresar el producto ya que el proveedor tiene que darse de alta primero.

En la descripción del problema se menciona que se requiere que se pueda tener un control para el área de administración y que se deberán tener 3 opciones, las cuales tienen que ser para: reporte de ventas, reporte de proveedores y reporte de inventario de los accesorios, por lo cual se propuso que para la opción número 5 del menú principal se pueda entrar a lo que es el inventario, el reporte de ventas y el de proveedores, pero también se decidió hacer un reporte de vendedores para poder llevar un control de vendedores en la tienda también, al ingresar a alguna de estas opciones mencionadas se decidió el poner las opciones de reporte impreso en pantalla y la de reporte en archivo de texto, además de una opción de salida por si es que el usuario se llega a equivocar.

2.2 Limitantes

En cuanto a limitantes tenemos que se usa una función de Windows llamada GetKeyState, por lo cual solo se reconocerán los números del teclado alfanumérico (ver imagen 2.2.1) y no será posible usar el teclado numérico ya que no reconocerá esta parte del teclado (ver imagen 2.2.2).

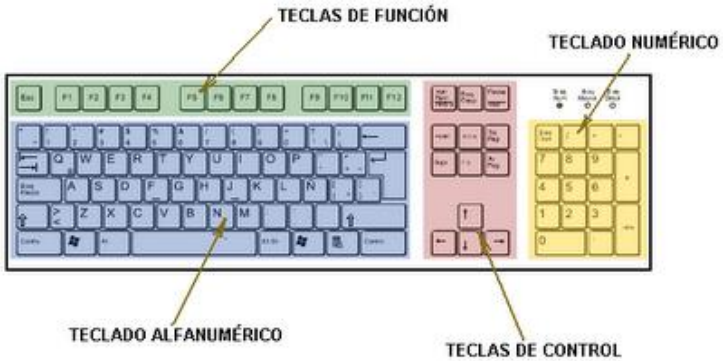


Imagen 2.2.1: Partes del teclado, nótese que el teclado alfanumérico está en color azul.

Imagen 2.2.2: La parte en color rojo no es reconocida por el programa

Otra limitante es que para generar los códigos del producto se debe de cumplir con el formato de 2 letras y 4 numeros, esta parte es validada en una función dentro del programa llamada validarCodigo en la cual dentro de un ciclo “for” se va comparando la cadena, para las primeras 2 posiciones valida que sean letras y para las otras 4 que sean números, en caso de no cumplirse alguna condición se pedirá que se ingrese un código válido.
Otras limitantes son que las claves de vendedores solo se componen de valores numéricos, se tienen que dar de alta los proveedores antes de comenzar a ingresar accesorios ya que de lo contrario no se podrán dar de alta y por lo tanto tampoco se podrán tener ventas, de igual manera se deben de dar de alta los vendedores para hacer algún registro de una venta.

3.Pseudocódigo

```
//* Incluir librerías
conio.h
locale.h
math.h
stdio.h
stdlib.h
string.h
wchar.h
windows.h
ctime
iomanip
iostream
string
using namespace std;

//* Constantes
MAX <- 1000
NUMBER_OF_SLOTS <- 100

// * Utilerías
//Función que espera a que presiones espacio
SubProceso Llamar escucharEspacio(): tipo vacía
Inicio
Escribir "Presione espacio para continuar..."
BOOLEANA con <- true
Mientras  (con)
//Solo si es la ventana activa
Si  (GetConsoleWindow() == GetForegroundWindow())
Si  (GetKeyState(' ') & 0x8000)
Mientras  (GetKeyState(' ') & 0x8000)
FinMientras
con <- false
FinSi
FinSi
FinMientras
FinSubProceso
SubProceso Llamar escucharEspacio(ENTERO i): tipo vacía
Inicio
Escribir "Presione espacio para continuar..."
BOOLEANA con <- true
Mientras  (con)
//Solo si es la ventana activa
Si  (GetConsoleWindow() == GetForegroundWindow())
```

```
Si  (GetKeyState(' ') & 0x8000)
Mientras  (GetKeyState(' ') & 0x8000)
FinMientras
con <- false
FinSi
FinSi
FinMientras
FinSubProceso

//Función que valida el código de un producto
//Si es válida regresa true, y false si no
SubProceso validarCodigo(CARACTER codigo[]): tipo booleana
Inicio
CARACTER caracter
BOOLEANA esNumero <- false
BOOLEANA esLetra <- false
BOOLEANA codigoValido <- false
//Verifica su longitud
CADENA cadena <- codigo
Si  (cadena.length() > 6 || cadena.length() == 0)
regresar false
FinSi
//Verifica que los primeros sean letras y los siguientes, números
Para  (ENTERO i <- 0 Hasta 6 Con Paso 1
caracter <- codigo[i]
Para  (ENTERO j <- 48 Hasta 5 Con Paso 1)
Si  (caracter == j)
esNumero <- true
esLetra <- false
FinSi
FinPara
Para  (ENTERO j <- 97 Hasta 12 Con Paso 1)
Si  (caracter == j)
esNumero <- false
esLetra <- true
FinSi
FinPara
Para  (ENTERO j <- 65 Hasta 9 Con Paso 1
Si  (caracter == j)
esNumero <- false
esLetra <- true
FinSi
FinPara
Si  (esNumero == false  y  esLetra == false)
regresar false
FinSi
```

```
Si  ((i == 0 || i == 1)  y  esNumero)
regresar false
FinSi
Si  ((i != 0  y  i != 1)  y  esLetra)
regresar false
FinSi
esNumero <- false
esLetra <- false
FinPara
regresar true
FinSubProceso

//Función que pide un número entero y ayuda a prevenir errores
SubProceso pedirEntero(CADENA peticion): tipo entero
Inicio
CADENA str
Escribir peticion
Vaciar Buffer
Leer str
Vaciar Buffer
//Se llama recursivamente hasta que el usuario digite una opción
correcta
try{
regresar stoi(str)
}
catch (...){
Escribir "Debe ingresar un entero!"
regresar Llamar pedirEntero(peticion)
}
FinSubProceso

//Función que pide un número long long sin signo y ayuda a prevenir
errores
SubProceso pedirUnsignedLongLong(CADENA peticion): tipo long long sin
signo
Inicio
CADENA str
Escribir peticion
Vaciar Buffer
Leer str
Vaciar Buffer
//Se llama recursivamente hasta que el usuario digite una opción
correcta
try{
regresar stoull(str)
}
```

```
catch (...){
Escribir "Debe ingresar un entero!"
regresar Llamar pedirUnsignedLongLong(peticion)
}
FinSubProceso

//Función que pide un número flotante y ayuda a prevenir errores
SubProceso pedirFlotante(CADENA peticion): tipo real
Inicio
CADENA str
Escribir peticion
Vaciar Buffer
Leer str
Vaciar Buffer
//Se llama recursivamente hasta que el usuario digite una opción
correcta
try{
regresar stof(str)
}
catch (...){
Escribir "Debe ingresar un flotante!"
regresar Llamar pedirFlotante(peticion)
}
FinSubProceso

//Función que escucha las teclas
SubProceso escucharTecla(ENTERO nOpciones): tipo real
Inicio
CARACTER letras[] = {'U', 'N', 'E', 'I', 'F', 'A', 'Y', 'S'}
//Entra a un bucle que se rompe cuando pulsas una tecla indicada
Mientras  (true)
//Solo si es la ventana activa
Si  (GetConsoleWindow() == GetForegroundWindow())
//Verifiva las opciones que usan ctrl+KEY
Para  (ENTERO i <- 0 Hasta nOpciones Con Paso 1)
Si  ((GetKeyState(VK_CONTROL) & 0x8000)  y  (GetKeyState(letras[i]) &
0x8000))
Mientras  ((GetKeyState(VK_CONTROL) & 0x8000) && (GetKeyState(letras[i])
& 0x8000))
FinMientras
regresar i + 1
FinSi
FinPara
//Verifica las teclas numéricas
Para  (ENTERO i <- 49 Hasta 49+(nOpciones-1) Con Paso 1)
Si  (GetKeyState(i) & 0x8000)
```

```
Mientras  (GetKeyState(i) & 0x8000)
FinMientras
regresar i - 48
FinSi
FinPara
FinSi
FinMientras
FinSubProceso

//Ayuda para centrar el texto
SubProceso substring(CARACTER *cadena, CARACTER *subcadena, ENTERO
inicio, ENTERO longitud): tipo vacía
Inicio
ENTERO i
Para  (i <- 0 Hasta longitud&&inicio+istrlen(cadena) Con Paso 1)
subcadena[i] <- cadena[inicio + i]
subcadena[i] <- '\0'
FinPara
FinSubProceso

SubProceso centrar_linea(CARACTER *linea, ENTERO ancho): tipo vacía
Inicio
ENTERO i, espacios
espacios <- (ancho - strlen(linea)) / 2

Para  (i <- 0 Hasta espacios Con Paso 1)
Escribir " "
Escribir linea
FinPara
FinSubProceso

SubProceso centrar_cadena(char *cadena, int ancho): tipo vacía
Inicio
CARACTER subcadena[MAX]
ENTERO i, total
total <- (ENTERO)ceil((REAL)strlen(cadena) / ancho)

Para  (i <- 0 Hasta total Con Paso 1)
Llamar substring(cadena, subcadena, i * ancho, ancho)
Llamar centrar_linea(subcadena, ancho)
FinPara
FinSubProceso

SubProceso derecha_linea(char *linea, int ancho): tipo vacía
Inicio
ENTERO i, espacios
```

```
espacios <- ancho - strlen(linea)

Para  (i <- 0 Hasta espacios Con Paso 1)
Escribir " "
Escribir linea
FinPara
FinSubProceso

SubProceso Llamar derecha_cadena(char *cadena, int ancho): tipo vacía
Inicio
CARACTER subcadena[MAX]
ENTERO i, total
total <- (ENTERO)ceil((REAL)strlen(cadena) / ancho)

Para  (i <- 0 Hasta total Con Paso 1)
Llamar substring(cadena, subcadena, i * ancho, ancho)
Llamar derecha_linea(subcadena, ancho)
FinPara
FinSubProceso

// * Declaración de los struct
//Productos
typedef struct
{
CARACTER codigo[7]
CARACTER color[20]
CARACTER marca[20]
CARACTER modelo[20]
CARACTER proveedor[20]
REAL costoComprado
REAL costoVendido
ENTERO existencia
ENTERO unidadesCompradas
}tproducto

struct node {
tproducto product
struct node *next
}
typedef struct node *Tlist

//Proveedores
typedef struct
{
CARACTER clave[20]
CARACTER nombre[20]
```

```
LARGO LARGO SIN SIGNO telefono
} Proveedor

//Vendedores
typedef struct
{
ENTERO clave
CARACTER nombre[20]
REAL salario
} Vendedor

//Ventas
typedef struct
{
BOOLEANA esVenta        //Venta o reembolzo
LARGO LARGO SIN SIGNO numero
ENTERO SIN SIGNO dia
ENTERO SIN SIGNO mes
ENTERO SIN SIGNO anho
CARACTER clave[7]
ENTERO cantidad
ENTERO costo
ENTERO vendedor
} Venta

// * Productos
// * Hash
//Función hash
//Toma los primeros dos números del código que se le de
SubProceso hashFunction(CADENA code): tipo entero
Inicio
try{
CADENA number <- code.substr(2, 2)
ENTERO n <- atoi(number.c_str())
regresar n
}
catch (...){
Escribir "Su productos.dat no corresponde con el que se maneja en este
programa"
Escribir "Borrelo o cambielo de lugar"
Llamar escucharEspacio()
Llamar exit(0)
}
regresar 0
FinSubProceso
```

```
//Variable arreglo que contendrá todos los productos
Tlist hashTable[NUMBER_OF_SLOTS]

//inicializa el arreglo con valores nulos
SubProceso initializeHashTable(): tipo vacía
Inicio
Para  (ENTERO i <- 0 Hasta NUMBER_OF_SLOTS Con Paso 1)
hashTable[i] <- NULL
FinPara
FinSubProceso

//Llena el arreglo con los datos de productos.dat
//Si no existe simplemente lo deja con los valores nulos
SubProceso fillHashTable(): tipo vacía
Inicio
FILE *arch
arch <- Abrir Archivo("productos.dat", "rb")
Si  (arch == NULL)
regresar
FinSi
tproducto producto

//Posiciona cada código en su lugar con la función hash
Leer En Archivo(&producto, sizeof(tproducto), 1, arch)
Mientras  (!feof(arch))
ENTERO n <- hashFunction(producto.codigo)
//Usa la lógica de las listas dinámicas
Si  (hashTable[n] == NULL)
Tlist q <- new (struct node)
q->product <- producto
q->next <- NULL
hashTable[n] <- q
SiNo
Tlist t, q <- new (struct node)
q->product <- producto
q->next <- NULL
t <- hashTable[n]
Mientras  (t->next != NULL)
t <- t->next
FinMientras
t->next <- q
FinSi
Leer En Archivo(&producto, sizeof(tproducto), 1, arch)
FinMientras
Cerrar Archivo(arch)
regresar
```

```
FinSubProceso

//Imprime el arreglo
//No es usado en este programa
//Pero es bueno tenerlo por si las moscas
SubProceso printHashTable(): tipo vacía
Inicio
Para  (ENTERO i <- 0 Hasta NUMBER_OF_SLOTS Con Paso 1)
Si  (hashTable[i] == NULL)
Escribir i , ".-"
SiNo
Tlist t <- hashTable[i]
Escribir i , ".- " , t->product.codigo
Mientras  (t->next != NULL)
t <- t->next
Escribir " -> " , t->product.codigo
FinMientras

FinSi
FinPara
FinSubProceso

//Escribe los cambios del arreglo a productos.dat
SubProceso Llamar writeFile(): tipo vacía
Inicio
FILE *arch
arch <- Abrir Archivo("productos.dat", "w+b")
Si  (arch == NULL)
Escribir "Archivo productos.dat no se pudo generar"
Llamar Llamar escucharEspacio()
Llamarexit(1)
FinSi
Tlist list
tproducto producto

//Recorre todo el arreglo
Para  (ENTERO i <- 0 Hasta NUMBER_OF_SLOTS Con Paso 1)
Si  (hashTable[i] == NULL)
SiNo
Tlist t <- hashTable[i]
product <- t->product
//Solo escribe el producto ya que escribir todo el nodo sería un caos
con la memoria
Escribir En Archivo(&product, sizeof(tproducto), 1, arch)
Mientras  (t->next != NULL)
t <- t->next
```

```
product <- t->product
Escribir En Archivo(&product, sizeof(tproducto), 1, arch)
FinMientras
FinSi
FinPara
Cerrar Archivo(arch)
FinSubProceso

// * Archivos de productos
//Función que pide los datos de un producto y los escribe en
productos.dat
SubProceso altaProducto(): tipo vacía
Inicio
tproducto producto
BOOLEANA codigoValido <- 0
////////////////////////////////////////////////////////////////
//Petición de datos
////////////////////////////////////////////////////////////////
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))

Mientras  (!codigoValido)
Escribir "Ingrese el código del producto: "
Vaciar Buffer
Llamar Llamar gets(producto.codigo)
codigoValido <- Llamar validarCodigo(producto.codigo)
Si  (!codigoValido)
Escribir "Usted ingreso un codigo invalido!!!"
Escribir "Un codigo correcto empieza con 2 letras y le siguen 4
numeros."
FinSi
FinMientras
////////////////////////////////////////////////////////////////
//Verifica que ese código no exista
//Si no existe lo posiciona en el arreglo y lo escribe en el archivo
////////////////////////////////////////////////////////////////
ENTERO n <- Llamar hashFunction(producto.codigo)
Tlist t <- new (struct node)
t <- hashTable[n]
Mientras  (t != NULL)
ENTERO ret <- Comparar Cadenas(producto.codigo, t->product.codigo, 6)
Si  (ret == 0)
Escribir "Un producto con ese código ya existe."
Escribir "Se cancela todo."
regresar
FinSi
```

```
t <- t->next
FinMientras

Vaciar Buffer
Escribir "Ingrese el color del producto: "
Leer(producto.color, 20, '')

Vaciar Buffer
Escribir "Ingrese la marca del producto: "
Leer(producto.marca, 20, '')

Vaciar Buffer
Escribir "Ingrese el modelo del producto: "
Leer(producto.modelo, 20, '')

Vaciar Buffer
Escribir "Ingrese el proveedor del producto: "
Leer(producto.proveedor, 20, '')
////////////////////////////////////////////////////////////////////
//Verifica que exista el proveedor
////////////////////////////////////////////////////////////////////
FILE *arch
arch <- Abrir Archivo("proveedores.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo proveedores.dat no encontrado."
Escribir "Agregue proveedores."
Escribir "Se cancela todo."
regresar
FinSi
CADENA cod <- producto.proveedor
Proveedor proveedor
ENTERO existe <- 0
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
Si  (cod == proveedor.clave)
existe <- 1
FinSi
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
Si  (existe == 0)
Escribir "No existe un proveedor con dicha clave."
Escribir "Se cancela todo."
regresar
FinSi
Cerrar Archivo(arch)
```

```
Vaciar Buffer
Escribir "Ingrese el precio al que se compró el producto: "
producto.costoComprado <- Llamar pedirFlotante("")

Vaciar Buffer
Escribir "Ingrese el precio al que se vende el producto: "
producto.costoVendido <- Llamar pedirFlotante("")

Vaciar Buffer
Escribir "Ingrese cuantas unidades se compraron: "
producto.existencia <- Llamar pedirEntero("")

producto.unidadesCompradas <- producto.existencia
////////////////////////////////////////////////////////////////////
//Posiciona el producto en el arreglo
////////////////////////////////////////////////////////////////////
Si  (hashTable[n] == NULL)
Tlist q <- new (struct node)
q->product <- producto
q->next <- NULL
hashTable[n] <- q
SiNo
Tlist q <- new (struct node)
q->product <- producto
q->next <- NULL
t <- hashTable[n]
Mientras  (t->next != NULL)
t <- t->next
FinMientras
t->next <- q
FinSi

Llamar Llamar writeFile()
FinSubProceso

//Función que da de baja un producto
SubProceso bajaProducto(CADENA code): tipo vacía
Inicio
//Lo busca en el arreglo y como es archivo directo se puede buscar
directamente con la función hash
//Verifica que esa posición del arreglo no esté vacía
ENTERO n <- hashFunction(code)
Si  (hashTable[n] == NULL)
Escribir "No existe el producto con ese código."
SiNo
//Si no busca en los nodos hijos
```

```
BOOLEANA existe <- false
Tlist t <- hashTable[n]
ENTERO i <- 1
Si  (t->next == NULL)
Si  (t->product.codigo == code)
Escribir "Código: " , t->product.codigo
Escribir "Color: " , t->product.color
Escribir "Precio al que se compró: " , t->product.costoComprado
Escribir "Precio al que se vende: " , t->product.costoVendido
Escribir "Existencia: " , t->product.existencia
Escribir "Unidades compradas: " , t->product.unidadesCompradas
Escribir "Marca: " , t->product.marca
Escribir "Modelo: " , t->product.modelo
Escribir "Proveedor: " , t->product.proveedor

hashTable[n] <- NULL
Escribir "Producto eliminado."
Llamar Llamar writeFile()
SiNo
Escribir "El producto con ese código no existe."
FinSi
SiNo
Mientras  (t != NULL)
Si  (t->next->product.codigo == code)
Escribir "Código: " , t->product.codigo
Escribir "Color: " , t->product.color
Escribir "Precio al que se compró: " , t->product.costoComprado
Escribir "Precio al que se vende: " , t->product.costoVendido
Escribir "Existencia: " , t->product.existencia
Escribir "Unidades compradas: " , t->product.unidadesCompradas
Escribir "Marca: " , t->product.marca
Escribir "Modelo: " , t->product.modelo
Escribir "Proveedor: " , t->product.proveedor

existe <- true
t->next <- t->next->next
Escribir "Producto eliminado."
Llamar writeFile()
FinSi
t <- t->next
i <- i+1
FinMientras
Si  (!existe)
Escribir "El producto con ese código no existe."
FinSi
```

```
FinSi
FinSi
FinSubProceso

//Función que cambia los datos de un producto
SubProceso cambioProducto(CADENA code): tipo vacía
Inicio
//Lo busca en el arreglo, píde los datos y como es archivo directo se
puede buscar directamente con la función hash
//Verifica que esa posición del arreglo no esté vacía
ENTERO n <- hashFunction(code)
Si  (hashTable[n] == NULL)
Escribir "No existe el producto con ese código."
SiNo
//Si no busca en los nodos hijos
BOOLEANA existe <- false
Tlist t <- hashTable[n]
ENTERO i <- 1
Mientras  (t != NULL)
Si  (t->product.codigo == code)
existe <- true
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Escribir "Código: " , t->product.codigo
Escribir "Color: " , t->product.color
Escribir "Precio al que se compró: " , t->product.costoComprado
Escribir "Precio al que se vende: " , t->product.costoVendido
Escribir "Existencia: " , t->product.existencia
Escribir "Unidades compradas: " , t->product.unidadesCompradas
Escribir "Marca: " , t->product.marca
Escribir "Modelo: " , t->product.modelo
Escribir "Proveedor: " , t->product.proveedor

Vaciar Buffer
Escribir "Ingrese el color del producto: "
Leer(t->product.color, 20, '')

Vaciar Buffer
Escribir "Ingrese la marca del producto: "
Leer(t->product.marca, 20, '')

Vaciar Buffer
Escribir "Ingrese el modelo del producto: "
Leer(t->product.modelo, 20, '')

Vaciar Buffer
```

```
Escribir "Ingrese el proveedor del producto: "
Leer(t->product.proveedor, 20, '')

FILE *arch
arch <- Abrir Archivo("proveedores.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo proveedores.dat no encontrado."
Escribir "Agregue proveedores"
Escribir "Se cancela todo."
regresar
FinSi
CADENA cod <- t->product.proveedor
Proveedor proveedor
ENTERO existe <- 0
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
Si  (cod == proveedor.clave)
existe <- 1
FinSi
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
Si  (existe == 0)
Escribir "No existe un proveedor con dicha clave"
Escribir "Se cancela todo"
regresar
FinSi
Cerrar Archivo(arch)
Vaciar Buffer
Escribir "Ingrese el precio al que se compró el producto: "
t->product.costoComprado <- Llamar pedirFlotante("")

Vaciar Buffer
Escribir "Ingrese el precio al que se vende el producto: "
t->product.costoVendido <- Llamar pedirFlotante("")

Vaciar Buffer
Escribir "Ingrese cuantas unidades se compraron: "
t->product.existencia <- Llamar pedirEntero("")
t->product.unidadesCompradas <- t->product.existencia
Llamar writeFile()
FinSi
t <- t->next
i <- i+1
FinMientras
Si  (!existe)
Escribir "El producto con ese código no existe."
```

```
FinSi

FinSi
FinSubProceso

//Función que busca un producto
SubProceso consultaProducto(CADENA code): tipo vacía
Inicio
//Lo busca en el arreglo y como es archivo directo se puede buscar
directamente con la función hash
//Verifica que esa posición del arreglo no esté vacía
ENTERO n <- Llamar hashFunction(code)
Si  (hashTable[n] == NULL)
Escribir "No existe el producto con ese código."
SiNo
//Si no busca en los nodos hijos
BOOLEANA existe <- false
Tlist t <- hashTable[n]
ENTERO i <- 1
Mientras  (t != NULL)
Si  (t->product.codigo == code)
existe <- true
Escribir "El producto esta en el indice " , n , " de la tabla en la
posicion " , i
Escribir "Código: " , t->product.codigo
Escribir "Color: " , t->product.color
Escribir "Precio al que se compró: " , t->product.costoComprado
Escribir "Precio al que se vende: " , t->product.costoVendido
Escribir "Existencia: " , t->product.existencia
Escribir "Unidades compradas: " , t->product.unidadesCompradas
Escribir "Marca: " , t->product.marca
Escribir "Modelo: " , t->product.modelo
Escribir "Proveedor: " , t->product.proveedor

FinSi
t <- t->next
i <- i+1
FinMientras
Si  (!existe)
Escribir "El producto con ese código no existe."
FinSi

FinSi
FinSubProceso

// * Proveedores
```

```
//Función que da de alta a un proveedor
SubProceso altaProveedor(): tipo vacía
Inicio
Proveedor proveedor
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
////////////////////////////////////////////////////////////////////
//Pide los datos
////////////////////////////////////////////////////////////////////
BOOLEANA codigoValido <- 0
Vaciar Buffer
Escribir "Digite la clave del proveedor: "
Leer(proveedor.clave, 20, '')
////////////////////////////////////////////////////////////////////
//Verifica que no halla un proveedor con esa clave
////////////////////////////////////////////////////////////////////

FILE *arch
Proveedor p
arch <- Abrir Archivo("proveedores.dat", "r+b")
Si  (arch != NULL)
Leer En Archivo(&p, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
ENTERO ret <- Comparar Cadenas(proveedor.clave, p.clave, 20)
Si  (ret == 0)
Escribir "Ya existe un proveedor con esa clave "
Escribir "Se cancela todo"
regresar
FinSi
Leer En Archivo(&p, sizeof(Proveedor), 1, arch)
FinMientras
Cerrar Archivo(arch)
FinSi

Vaciar Buffer
Escribir "Digite el nombre del proveedor: "
Leer(proveedor.nombre, 20, '')

Vaciar Buffer
Escribir "Digite el número telefónico del proveedor: "
proveedor.telefono <- Llamar pedirUnsignedLongLong("")

arch <- Abrir Archivo("proveedores.dat", "ab")
Si  (arch == NULL)
Escribir "Archivo proveedores.dat no se pudo generar."
regresar
```

```
FinSi
Escribir En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Cerrar Archivo(arch)
FinSubProceso

//Función que da de baja a un proveedor
SubProceso bajaProveedor(): tipo vacía
Inicio
FILE *arch
arch <- Abrir Archivo("proveedores.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo proveedores.dat no encontrado."
Escribir "Agregue proveedores."
regresar
FinSi
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Escribir "Ingrese la clave del proveedor a consultar: ")
Vaciar Buffer
CADENA cod
Llamar getline(cin, cod)
Proveedor proveedor
ENTERO existe <- 0
////////////////////////////////////////////////////////////////
//Lo busca en el archivo secuencialmente
////////////////////////////////////////////////////////////////
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
Si  (cod == proveedor.clave)
Escribir "Clave: " , proveedor.clave
Escribir "Nombre: " , proveedor.nombre
Escribir "Teléfono: " , proveedor.telefono

CADENA cadenaVacia <- ""
Copiar Cadena(proveedor.clave, cadenaVacia.c_str())
Copiar Cadena(proveedor.nombre, cadenaVacia.c_str())
proveedor.telefono <- 0

ENTERO pos <- ftell(arch) - sizeof(Proveedor)
Llamar fseek(arch, pos, SEEK_SET)
Escribir En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Escribir "Se borro el proveedor."
existe <- 1
FinSi
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
```

```
Si  (existe == 0)
Escribir "No existe un proveedor con dicha clave."
Cerrar Archivo(arch)
FinSi
FinSunProceso

//Función que cambia los datos de un proveedor
SubProceso cambioProveedor(): tipo vacía
FILE *arch
arch <- Abrir Archivo("proveedores.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo proveedores.dat no encontrado."
Escribir "Agregue proveedores."
regresar
FinSi
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Escribir "Ingrese la clave del proveedor a consultar: ")
Vaciar Buffer
CADENA cod
Llamar getline(cin, cod)
Proveedor proveedor
ENTERO existe <- 0
////////////////////////////////////////////////////////////////////
//Lo busca en el archivo secuencialmente
////////////////////////////////////////////////////////////////////
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
Si  (cod == proveedor.clave)
Escribir "Clave: " , proveedor.clave
Escribir "Nombre: " , proveedor.nombre
Escribir "Teléfono: " , proveedor.telefono
Vaciar Buffer
Escribir "Digite el nombre del proveedor: "
Leer(proveedor.nombre, 20, '')
Vaciar Buffer
Escribir "Digite el número telefónico del proveedor: "
proveedor.telefono <- Llamar pedirUnsignedLongLong("")
ENTERO pos <- ftell(arch) - sizeof(Proveedor)
Llamar fseek(arch, pos, SEEK_SET)
Escribir En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Escribir "Se modifico el proveedor."
existe <- 1
FinSi
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
```

```
Si  (existe == 0)
Escribir "No existe un proveedor con dicha clave."
Cerrar Archivo(arch)
FinSi
FinSubProceso

//Función que consulta los datos de un proveedor
SubProceso consultaProveedor(): tipo vacía
FILE *arch
arch <- Abrir Archivo("proveedores.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo proveedores.dat no encontrado."
Escribir "Agregue proveedores."
regresar
FinSi
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Escribir "Ingrese la clave del proveedor a consultar: "
Vaciar Buffer
CADENA cod
Llamar getline(cin, cod)
Proveedor proveedor
ENTERO existe <- 0
////////////////////////////////////////////////////////////////
//Lo busca en el archivo secuencialmente
////////////////////////////////////////////////////////////////
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
Si  (cod == proveedor.clave)
Escribir "Clave: " , proveedor.clave
Escribir "Nombre: " , proveedor.nombre
Escribir "Teléfono: " , proveedor.telefono
existe <- 1
FinSi
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
Si  (existe == 0)
Escribir "No existe un proveedor con dicha clave.")
Cerrar Archivo(arch)
FinSi
FinSubProceso

// * Vendedores
//Función que da de alta a un vendedor
SubProceso altaVendedor(): tipo vacía
Inicio
```

```
Vendedor vendedor
////////////////////////////////////////////////////////////////////
//Pide los datos
////////////////////////////////////////////////////////////////////
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
BOOLEANA codigoValido <- 0
Vaciar Buffer
vendedor.clave <- Llamar pedirEntero("Digite la clave numérica del
vendedor: ")
////////////////////////////////////////////////////////////////////
//Verifica que no exista vendedor con esa clave
////////////////////////////////////////////////////////////////////
FILE *arch
arch <- Abrir Archivo("vendedores.dat", "r+b")
Si  (arch != NULL)
Vendedor v
Leer En Archivo(&v, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Si  (vendedor.clave == v.clave)
Escribir "Ya existe un vendedor con esa clave"
Escribir "Se cancela todo"
regresar
FinSi
Leer En Archivo(&v, sizeof(Vendedor), 1, arch)
FinMientras
Cerrar Archivo(arch)
FinSi

Vaciar Buffer
Escribir "Digite el nombre del vendedor: "
Leer(vendedor.nombre, 20, '')

Vaciar Buffer
vendedor.salario <- Llamar pedirFlotante("Digite el salario del
vendedor: ")

*arch
arch <- Abrir Archivo("vendedores.dat", "ab")
Si  (arch == NULL)
Escribir "Archivo vendedores.dat no pudo ser generado."
regresar
FinSi
Escribir En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Cerrar Archivo(arch)
FinSubProceso
```

```
//Función que da de baja a un vendedor
SubProceso bajaVendedor(): tipo vacía
Inicio
FILE *arch
arch <- Abrir Archivo("vendedores.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo vendedores.dat no encontrado."
Escribir "Agregue vendedores."
regresar
FinSi
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Vendedor vendedor
Vaciar Buffer
ENTERO cod <- Llamar pedirEntero("Ingrese la clave del vendedor a
consultar: ")

////////////////////////////////////////////////////////////////
//Lo busca en el archivo secuencialmente
////////////////////////////////////////////////////////////////

ENTERO existe <- 0
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Si  (cod == vendedor.clave)
Escribir "Clave: " , vendedor.clave
Escribir "Nombre: " , vendedor.nombre
Escribir "Salario: " , vendedor.salario
CADENA cadenaVacia <- ""
vendedor.clave <- 0
Copiar Cadena(vendedor.nombre, cadenaVacia.c_str())
vendedor.salario <- 0
ENTERO pos <- ftell(arch) - sizeof(Vendedor)
Llamar fseek(arch, pos, SEEK_SET)
Escribir En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Escribir "Se borro el vendedor."
existe <- 1
FinSi
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
FinMientras
Si  (existe == 0)
Escribir "No existe un vendedor con dicha clave."
Cerrar Archivo(arch)
FinSi
FinSubProceso
```

```
//Función que cambia los datos de un vendedor
SubProceso cambioVendedor(): tipo vacía
Inicio
FILE *arch
arch <- Abrir Archivo("vendedores.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo vendedores.dat no encontrado"
Escribir "Agregue vendedores"
regresar
FinSi
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Vendedor vendedor
Vaciar Buffer
ENTERO cod <- Llamar pedirEntero("Ingrese la clave del vendedor a
consultar: ")
////////////////////////////////////////////////////////////////
//Lo busca en el archivo secuencialmente
////////////////////////////////////////////////////////////////
ENTERO existe <- 0
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Si  (cod == vendedor.clave)
Escribir "Clave: " , vendedor.clave
Escribir "Nombre: " , vendedor.nombre
Escribir "Salario: " , vendedor.salario
Vaciar Buffer
Escribir "Digite el nombre del vendedor: "
Leer(vendedor.nombre, 20, '')
Vaciar Buffer
vendedor.salario <- Llamar pedirFlotante("Digite el salario del
vendedor: ")
ENTERO pos <- ftell(arch) - sizeof(Vendedor)
Llamar fseek(arch, pos, SEEK_SET)
Escribir En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Escribir "Se modifico el vendedor.")
existe <- 1
FinSi
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
FinMientras
Si  (existe == 0)
Escribir "No existe un vendedor con dicha clave.")
Cerrar Archivo(arch)
FinSi
FinSubProceso
```

```
//Función que consulta los datos de un vendedor
SubProceso consultaVendedor(): tipo vacía
Inicio
FILE *arch
arch <- Abrir Archivo("vendedores.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo vendedores.dat no encontrado"
Escribir "Agregue vendedores"
regresar
FinSi
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Vendedor vendedor
Vaciar Buffer
ENTERO cod <- Llamar pedirEntero("Ingrese la clave del vendedor a
consultar: ")
////////////////////////////////////////////////////////////////
//Lo busca en el archivo secuencialmente
////////////////////////////////////////////////////////////////

ENTERO existe <- 0
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Si  (cod == vendedor.clave)
Escribir "Clave: " , vendedor.clave
Escribir "Nombre: " , vendedor.nombre
Escribir "Salario: " , vendedor.salario
existe <- 1
FinSi
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
FinMientras
Si  (existe == 0)
Escribir "No existe un vendedor con dicha clave.")
Cerrar Archivo(arch)
FinSi
FinSubProceso

// * Ventas
//Función que da de alta una venta

SubProceso AltaVenta(): tipo vacía
Inicio
FILE *arch
arch <- Abrir Archivo("ventas.dat", "ab")
Si  (arch == NULL)
```

```
Escribir "Archivo ventas.dat no pudo ser generado."
regresar
FinSi
Venta venta
////////////////////////////////////////////////////////////////////
//Pide los datos
////////////////////////////////////////////////////////////////////
venta.esVenta <- true
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
BOOLEANA codigoValido <- 0
Mientras  (!codigoValido)
Escribir "Ingrese el código del producto: "
Vaciar Buffer
Leer(venta.clave, 7, '')
codigoValido <- validarCodigo(venta.clave)
Si  (!codigoValido)
Escribir "Usted ingreso un codigo invalido!!!"
Escribir "Un codigo correcto empieza con 2 letras y le siguen 4
numeros."
FinSi
FinMientras
CADENA code <- venta.clave
Tlist t

////////////////////////////////////////////////////////////////////
//Verifica que exista el código
////////////////////////////////////////////////////////////////////

ENTERO n <- Llamar hashFunction(code)
Si  (hashTable[n] == NULL)
Escribir "No existe el producto con ese código."
Escribir "Se cancela todo."
regresar
SiNo
BOOLEANA existe <- false
t <- hashTable[n]
ENTERO i <- 1
Mientras  (t != NULL)
Si  (t->product.codigo == code)
existe <- true
FinSi
t <- t->next
i <- i+1
FinMientras
Si  (!existe)
```

```
Escribir "El producto con ese código no existe."
Escribir "Se cancela todo."
regresar
FinSi
FinSi

Vaciar Buffer
Escribir "Digite el código numérico del vendedor: "
venta.vendedor <- Llamar pedirEntero("")

////////////////////////////////////////////////////////////////////
//Verifica que el vendedor exista
////////////////////////////////////////////////////////////////////

FILE *archV
archV <- Abrir Archivo("vendedores.dat", "r+b")
Si  (archV == NULL)
Escribir "Archivo vendedores.dat no encontrado."
Escribir "Agregue vendedores."
Escribir "Se cancela todo."
regresar
FinSi
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Vendedor vendedor

Vaciar Buffer
ENTERO cod <- venta.vendedor

ENTERO existe <- 0
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, archV)
Mientras  (!feof(archV))
Si  (cod == vendedor.clave)
existe <- 1
FinSi
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, archV)
FinMientras
Si  (existe == 0)
Escribir "No existe un vendedor con dicha clave.")
Escribir "Se cancela todo."
regresar
FinSi
Cerrar Archivo(archV)

BOOLEANA band <- true
```

```
////////////////////////////////////////////////////////////////
//Verifica que las cantidades sean congruentes
////////////////////////////////////////////////////////////////

Mientras  (band)
Vaciar Buffer
venta.cantidad <- Llamar pedirEntero("Digite la cantidad vendida: ")

Si  (t->product.existencia < venta.cantidad)
Escribir "Se están tratando de comprar" , venta.cantidad , "y solo hay "
, t->product.existencia
regresar
SiNo Si  (venta.cantidad <= 0)
Si  (venta.cantidad == 0)
Escribir "No puede vender 0 productos."
regresar
FinSi
Si  (venta.cantidad < 0)
Escribir "No puede ingresar un número negativo."
regresar
FinSi
SiNo
band <- false
t->product.existencia -= venta.cantidad
Llamar writeFile()
FinSi
FinSi
FinMientras

////////////////////////////////////////////////////////////////
//Genera automáticamente la fecha y el número de la compra
////////////////////////////////////////////////////////////////

time_t now <- time(0)

tm *ltm <- localtime(&now)

venta.dia <- ltm->tm_mday
venta.mes <- 1 + ltm->tm_mon
venta.anho <- 1900 + ltm->tm_year

CADENA str <- to_string(venta.anho - 2000) + to_string(venta.mes) +
to_string(venta.dia) + to_string(1 + ltm->tm_hour) + to_string(1 + ltm-
>tm_min) + to_string(1 + ltm->tm_sec)
venta.numero <- stoull(str)
```

```
Escribir En Archivo(&venta, sizeof(Venta), 1, arch)
Cerrar Archivo(arch)
FinSubProceso

//Función que da de baja una venta
SubProceso bajaVenta(): tipo vacía
Inicio
FILE *arch
arch <- Abrir Archivo("ventas.dat", "ab")
Si  (arch == NULL)
Escribir "Archivo ventas.dat no se pudo generar"
regresar
FinSi
Venta venta

////////////////////////////////////////////////////////////////////
//Pide los datos
////////////////////////////////////////////////////////////////////

venta.esVenta <- false
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
BOOLEANA codigoValido <- 0
Mientras  (!codigoValido)
Escribir "Ingrese el código del producto: ")
Vaciar Buffer
Leer(venta.clave, 7, '')
codigoValido <- Llamar validarCodigo(venta.clave)
Si  (!codigoValido)
Escribir "Usted ingreso un codigo invalido!!!"
Escribir "Un codigo correcto empieza con 2 letras y le siguen 4
numeros."
FinSi
FinMientras
CADENA code <- venta.clave
Tlist t

////////////////////////////////////////////////////////////////////
//Verifica que exista un producto con ese código
////////////////////////////////////////////////////////////////////



ENTERO n <- Llamar hashFunction(code)
Si  (hashTable[n] == NULL)
Escribir "No existe el producto con ese código."
```

```
Escribir "Se cancela todo."
regresar
SiNo
BOOLEANA existe <- false
t <- hashTable[n]
ENTERO i <- 1
Mientras  (t != NULL)
Si  (t->product.codigo == code)
existe <- true
FinSi
t <- t->next
i <- i+1
FinMientras
Si  (!existe)
Escribir "El producto con ese código no existe"
Escribir "Se cancela todo"
regresar
FinSi
FinSi

Vaciar Buffer
Escribir "Digite el código numerico del vendedor: "
venta.vendedor <- Llamar pedirEntero("")

////////////////////////////////////////////////////////////////////
//Verifica que el vendedor exista
////////////////////////////////////////////////////////////////////


FILE *archV
archV <- Abrir Archivo("vendedores.dat", "r+b")
Si  (archV == NULL)
Escribir "Archivo vendedores.dat no encontrado."
Escribir "Se cancela todo."
regresar
FinSi
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Vendedor vendedor

Vaciar Buffer
ENTERO cod <- venta.vendedor

ENTERO existe <- 0
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, archV)
Mientras  (!feof(archV))
```

```
Si  (cod == vendedor.clave)
existe <- 1
FinSi
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, archV)
FinMientras
Si  (existe == 0)
Escribir "No existe un vendedor con dicha clave.")
Escribir "Se cancela todo."
regresar
FinSi

Cerrar Archivo(archV)
BOOLEANA band <- true

////////////////////////////////////////////////////////////////////
//Verifica que las cantidades sean congruentes
////////////////////////////////////////////////////////////////////


Mientras  (band)
Vaciar Buffer
venta.cantidad <- Llamar pedirEntero("Digite la cantidad de productos en
el reembolzo: ")
Si  (venta.cantidad <= 0 || venta.cantidad > t-
>product.unidadesCompradas)
Si  (venta.cantidad == 0)
Escribir "No puede reembolzar 0 productos"
regresar
FinSi
Si  (venta.cantidad < 0)
Escribir "No puede ingresar un número negativo"
regresar
FinSi
Si  (venta.cantidad > t->product.unidadesCompradas)
Escribir "Está tratando de reembolzar más de lo que se compró"
regresar
FinSi
SiNo
band <- false
t->product.existencia += venta.cantidad
Llamar writeFile()
FinSi
FinMientras

////////////////////////////////////////////////////////////////////
//Genera automáticamente la fecha y el número de venta
```

```
////////////////////////////////////////////////////////////////////

time_t now <- time(0)
tm *ltm <- localtime(&now)

venta.dia <- ltm->tm_mday
venta.mes <- 1 + ltm->tm_mon
venta.anho <- 1900 + ltm->tm_year

CADENA str <- to_string(venta.anho - 2000) + to_string(venta.mes) +
to_string(venta.dia) + to_string(1 + ltm->tm_hour) + to_string(1 + ltm-
>tm_min) + to_string(1 + ltm->tm_sec)

venta.numero <- stoull(str)
Escribir En Archivo(&venta, sizeof(Venta), 1, arch)
Cerrar Archivo(arch)
FinSubProceso

//Función que consulta una venta
//No está implementado y no es usado
//Pero lo dejo por si las moscas
SubProceso consultaVenta(): tipo vacía
Inicio
FILE *arch
arch <- Abrir Archivo("ventas.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo ventas.dat no encontrado"
regresar
FinSi
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Vendedor vendedor

Vaciar Buffer
ENTERO cod <- Llamar pedirEntero("Ingrese la clave del vendedor a
consultar: ")

ENTERO existe <- 0
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Si  (cod == vendedor.clave)
Escribir vendedor.clave
Escribir ": %s : ", vendedor.nombre)
Escribir vendedor.salario
existe <- 1
FinSi
```

```
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
FinMientras
Si  (existe == 0)
Escribir "No existe un vendedor con dicha clave")
Cerrar Archivo(arch)
FinSi
FinSubProceso

// * Reportes en pantalla
//Función que imprime el inventario
SubProceso inventarioPantalla(): tipo vacía
Inicio
Escribir "Inventario"
Para  (ENTERO i <- 0 Hasta (10+(20*8)) Con Paso 1)
Escribir "-"
FinPara

Escribir setw(10) , left , "Clave"
Escribir setw(20) , left , "Modelo"
Escribir setw(20) , left , "Marca"
Escribir setw(20) , left , "Color"
Escribir setw(20) , left , "Precio Venta"
Escribir setw(20) , left , "Precio Compra"
Escribir setw(20) , left , "Existencia"
Escribir setw(20) , left , "Unidades compradas"
Escribir setw(20) , left , "Proveedor"

Para  (ENTERO i <- 0 Hasta (10+(20*8)) Con Paso 1)
Escribir "-"
FinPara

////////////////////////////////////////////////////////////////
//Imprime directamente
////////////////////////////////////////////////////////////////

Para  (ENTERO i <- 0 Hasta NUMBER_OF_SLOTS Con Paso 1)
Si  (hashTable[i] == NULL)
SiNo
Tlist t <- hashTable[i]
Escribir setw(10) , left , t->product.codigo
Escribir setw(20) , left , t->product.modelo
Escribir setw(20) , left , t->product.marca
Escribir setw(20) , left , t->product.color
Escribir setw(20) , left , t->product.costoVendido
Escribir setw(20) , left , t->product.costoComprado
Escribir setw(20) , left , t->product.existencia
```

```
Escribir setw(20) , left , t->product.unidadesCompradas
Escribir setw(20) , left , t->product.proveedor

Mientras  (t->next != NULL)
t <- t->next
Escribir setw(10) , left , t->product.codigo
Escribir setw(20) , left , t->product.modelo
Escribir setw(20) , left , t->product.marca
Escribir setw(20) , left , t->product.color
Escribir setw(20) , left , t->product.costoVendido
Escribir setw(20) , left , t->product.costoComprado
Escribir setw(20) , left , t->product.existencia
Escribir setw(20) , left , t->product.unidadesCompradas
Escribir setw(20) , left , t->product.proveedor

FinMientras
FinSi
FinPara
Para  (ENTERO i <- 0 Hasta (10+(20*8)) Con Paso 1)
Escribir "-"
FinPara
FinSubProceso

//Función que imprime el reporte de ventas
SubProceso reporteDeVentasPantalla(): tipo vacía
Inicio
Escribir "Reporte de ventas"
FILE *arch

REAL pc <- 0
REAL pv <- 0

arch <- Abrir Archivo("ventas.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo ventas.dat no encontrado"
regresar
FinSi
Para  (ENTERO i <- 0 Hasta ((10*5)+15+(20*2)) Con Paso 1)
Escribir "-"
FinPara

Escribir setw(10) , left , "Tipo"
Escribir setw(13) , left , "No."
Escribir setw(15) , left , "Fecha"
Escribir setw(10) , left , "Clave"
Escribir setw(10) , left , "Cantidad"
```

```
Escribir setw(20) , left , "Precio Comprado"
Escribir setw(20) , left , "Precio Vendido"
Escribir setw(10) , left , "Vendedor"

Para  (ENTERO i <- 0 Hasta ((10*5)+15+(20*2)) Con Paso 1)
Escribir "-"
FinPara

////////////////////////////////////////////////////////////////////
//Imprime secuencialmente
////////////////////////////////////////////////////////////////////

Venta venta
Leer En Archivo(&venta, sizeof(Venta), 1, arch)
Mientras  (!feof(arch))
Si  (venta.esVenta)
Escribir setw(10) , left , "Venta"
SiNo
Escribir setw(10) , left , "Reembolzo"
FinSi

CADENA fecha <- to_string(venta.dia) + "/" + to_string(venta.mes) + "/"
+ to_string(venta.año)

Escribir setw(13) , left , venta.numero
Escribir setw(15) , left , fecha
Escribir setw(10) , left , venta.clave
Escribir setw(10) , left , venta.cantidad

////////////////////////////////////////////////////////////////////
//Busca el producto con esa clave
//Para obtener datos como el precio
////////////////////////////////////////////////////////////////////

CADENA code <- venta.clave
Tlist t
ENTERO n <- Llamar hashFunction(code)
Si  (hashTable[n] == NULL)
Escribir "No existe el producto con ese código."
SiNo
BOOLEANA existe <- false
t <- hashTable[n]
ENTERO i <- 1
Mientras  (t != NULL)
existe <- true
Si  (t->product.codigo == code)
```

```
FinSi
t <- t->next
i <- i+1
FinMientras
Si  (!existe)
Escribir "El producto con ese código no existe."
FinSi
FinSi

Escribir setw(20) , left , t->product.costoComprado
Escribir setw(20) , left , t->product.costoVendido

Si  (venta.esVenta)
pc += (t->product.costoComprado * venta.cantidad)
pv += (t->product.costoVendido * venta.cantidad)
SiNo
pc -= (t->product.costoComprado * venta.cantidad)
pv -= (t->product.costoVendido * venta.cantidad)
FinSi
Escribir setw(10) , left , venta.vendedor

Leer En Archivo(&venta, sizeof(Venta), 1, arch)
FinMientras
Cerrar Archivo(arch)
Para  (ENTERO i <- 0 Hasta ((10*5)+15+(20*2)) Con Paso 1)
Escribir "-"
FinPara
Escribir "Inversión: %.2f ", pc
Escribir "Dinero total recaudado de las ventas: %.2f ", pv
Escribir "Ganancia: %.2f ", pv - pc
FinSubProceso

//Función que imprime el reporte de vendedores
SubProceso reporteDeVendedoresPantalla(): tipo vacía
Inicio
Escribir "Reporte de vendedores"
FILE *arch
arch <- Abrir Archivo("vendedores.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo vendedores.dat no encontrado."
regresar
FinSi
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir "-"
FinPara
```

```
Escribir setw(20) , left , "Clave"
Escribir setw(20) , left , "Nombre"
Escribir setw(20) , left , "Salario"

Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir "-"
FinPara

////////////////////////////////////////////////////////////////////
//Imprime secuencialmente
////////////////////////////////////////////////////////////////////

Vendedor vendedor
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Escribir setw(20) , left , vendedor.clave
Escribir setw(20) , left , vendedor.nombre
Escribir setw(20) , left , vendedor.salario

Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
FinMientras
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir "-"
FinPara

Cerrar Archivo(arch)
FinSubProceso

//Función que imprime el reporte de proveedores
SubProceso reporteDeProveedoresPantalla(): tipo vacía
Inicio
Escribir "Reporte de proveedores"
FILE *arch
arch <- Abrir Archivo("proveedores.dat", "r+b")
Si  (arch == NULL)
Escribir "Archivo proveedores.dat no encontrado."
regresar
FinSi
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir "-"
FinPara

Escribir setw(20) , left , "Clave"
Escribir setw(20) , left , "Nombre"
Escribir setw(20) , left , "Teléfono"
```

```
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir "-"
FinPara

////////////////////////////////////////////////////////////////////
//Imprime secuencialmente
////////////////////////////////////////////////////////////////////
Proveedor proveedor
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
Escribir setw(20) , left , proveedor.clave
Escribir setw(20) , left , proveedor.nombre
Escribir setw(20) , left , proveedor.telefono

Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir "-"
FinPara

Cerrar Archivo(arch)
FinSubProceso

// * Reportes en archivo de texto
//Función que genera un archivo con el inventario

SubProceso inventarioArchivo(): tipo vacía
Inicio
FILE *arch
arch <- Abrir Archivo("Inventario.txt", "w")
Escribir en Archivo(arch, "Inventario ")
Para  (ENTERO i <- 0 Hasta (10+(20*8)) Con Paso 1)
Escribir en Archivo(arch, "-")
FinPara
Escribir en Archivo(arch, "")

Escribir en Archivo(arch, "%-10s %-20s %-20s %-20s %-20s %-20s %-20s %-
20s %-20s", "Clave", "Modelo", "Marca", "Color", "Precio Venta", "Precio
Compra", "Existencia", "Unidades compradas", "Proveedor")
Para  (ENTERO i <- 0 Hasta (10+(20*8)) Con Paso 1)
Escribir en Archivo(arch, "-")
FinPara
Escribir en Archivo(arch, "")
////////////////////////////////////////////////////////////////////
//Escribe directamente
////////////////////////////////////////////////////////////////////
```

```
Para  (ENTERO i <- 0 Hasta NUMBER_OF_SLOTS Con Paso 1)
Si  (hashTable[i] == NULL)
SiNo
Tlist t <- hashTable[i]
Escribir en Archivo(arch, "%-10s %-20s %-20s %-20.2f %-20.2f %-20d
%-20d %-20s", t->product.codigo, t->product.modelo, t->product.marca, t-
>product.color, t->product.costoVendido, t->product.costoComprado, t-
>product.existencia, t->product.unidadesCompradas, t->product.proveedor)
Mientras  (t->next != NULL)
t <- t->next
Escribir en Archivo(arch, "%-10s %-20s %-20s %-20.2f %-20.2f %-20d
%-20d %-20s", t->product.codigo, t->product.modelo, t->product.marca, t-
>product.color, t->product.costoVendido, t->product.costoComprado, t-
>product.existencia, t->product.unidadesCompradas, t->product.proveedor)
FinMientras
FinSi
FinPara
Para  (ENTERO i <- 0 Hasta (10+(20*8)) Con Paso 1)
Escribir en Archivo(arch, "-")
FinPara
Escribir en Archivo(arch, "")

Cerrar Archivo(arch)
FinSubProceso

//Función que genera un archivo con el reporte de ventas
SubProceso reporteDeVentasArchivo(): tipo vacía
Inicio
FILE *archd, *archt
archt <- Abrir Archivo("Reporte de ventas.txt", "w")
archd <- Abrir Archivo("ventas.dat", "r+b")
Escribir en Archivo(archt, "Reporte de ventas")
REAL pc <- 0
REAL pv <- 0
Si  (archd == NULL)
Escribir en Archivo(archt, "Archivo ventas.dat no encontrado.")
regresar
FinSi
Para  (ENTERO i <- 0 Hasta ((10*5)+15+(20*2)) Con Paso 1)
Escribir en Archivo(archt, "-")
FinPara
Escribir en Archivo(archt, "")
Escribir en Archivo(archt, "%-10s %-13s %-15s %-10s %-10s %-20s %-20s %-
10s", "Tipo", "No.", "Fecha", "Clave", "Cantidad", "Precio Comprado",
"Precio Vendido", "Vendedor")
```

```
Para  (ENTERO i <- 0 Hasta ((10*5)+15+(20*2)) Con Paso 1)
Escribir en Archivo(archt, "-")
FinPara
Escribir en Archivo(archt, "")

////////////////////////////////////////////////////////////////////
//Escribe secuencialmente
////////////////////////////////////////////////////////////////////

Venta venta
Leer En Archivo(&venta, sizeof(Venta), 1, archd)
Mientras  (!feof(archd))
Si  (venta.esVenta)
Escribir en Archivo(archt, "%-10s ", "Venta")
SiNo
Escribir en Archivo(archt, "%-10s ", "Reembolzo")
FinSi

CADENA fecha <- to_string(venta.dia) + "/" + to_string(venta.mes) + "/"
+ to_string(venta.anho)
ENTERO k <- fecha.length()
CARACTER fechav[k + 1]
Copiar Cadena(fechav, fecha.c_str())
Escribir en Archivo(archt, "%-13d %-15s %-10s %-10d ", venta.numero,
fechav, venta.clave, venta.cantidad)
CADENA code <- venta.clave
Tlist t
ENTERO n <- Llamar hashFunction(code)
Si  (hashTable[n] == NULL)
Escribir en Archivo(archt, "No existe el producto con ese código")
SiNo
BOOLEANA existe <- false
t <- hashTable[n]
ENTERO i <- 1
Mientras  (t != NULL)
existe <- true
Si  (t->product.codigo == code)
FinSi
t <- t->next
i <- i+1
FinMientras
Si  (!existe)
Escribir en Archivo(archt, "El producto con ese código no existe")
FinSi
FinSi
```

```
Escribir en Archivo(archt, "%-20.2f %-20.2f ", t->product.costoComprado,
t->product.costoVendido)

Si  (venta.esVenta)
pc += (t->product.costoComprado * venta.cantidad)
pv += (t->product.costoVendido * venta.cantidad)
SiNo
pc -= (t->product.costoComprado * venta.cantidad)
pv -= (t->product.costoVendido * venta.cantidad)
FinSi
Escribir en Archivo(archt, "%-10d", venta.vendedor)
Leer En Archivo(&venta, sizeof(Venta), 1, archd)
FinMientras
Cerrar Archivo(archd)

Para  (ENTERO i <- 0 Hasta ((10*5)+15+(20*2)) Con Paso 1)
Escribir en Archivo(archt, "-")
FinPara

Escribir en Archivo(archt, "Inversión: %.2f ", pc)
Escribir en Archivo(archt, "Dinero total recaudado de las ventas: %.2f
", pv)
Escribir en Archivo(archt, "Ganancia: %.2f ", pv - pc)

Cerrar Archivo(archt)
FinSubProceso

//Función que genera un archivo con el reporte de vendedores
SubProceso reporteDeVendedoresArchivo(): tipo vacía
Inicio
FILE *archd, *archt
archt <- Abrir Archivo("Reporte de vendedores.txt", "w")
archd <- Abrir Archivo("vendedores.dat", "r+b")
Escribir en Archivo(archt, "Reporte de vendedores")
Si  (archd == NULL)
Escribir en Archivo(archt, "Archivo vendedores.dat no encontrado")
regresar
FinSi
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir en Archivo(archt, "-")
FinPara
Escribir en Archivo(archt, "")

Escribir en Archivo(archt, "%-20s %-20s %-20s", "Clave", "Nombre",
"Salario")
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
```

```
Escribir en Archivo(archt, "-")
FinPara
Escribir en Archivo(archt, "")

////////////////////////////////////////////////////////////////////
//Escribe secuencialmente
////////////////////////////////////////////////////////////////////

Vendedor vendedor
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, archd)
Mientras  (!feof(archd))
Escribir en Archivo(archt, "%-20d %-20s %-20f", vendedor.clave,
vendedor.nombre, vendedor.salario)
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, archd)
FinMientras
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir en Archivo(archt, "-")
FinPara
Escribir en Archivo(archt, "")
Cerrar Archivo(archd)
Cerrar Archivo(archt)
FinSubProceso

//Función que genera un archivo con el reporte de proveedores
SubProceso reporteDeProveedoresArchivo(): tipo vacía
Inicio
FILE *archd, *archt
archt <- Abrir Archivo("Reporte de proveedores.txt", "w")
archd <- Abrir Archivo("proveedores.dat", "r+b")
Escribir en Archivo(archt, "Reporte de proveedores")
Si  (archd == NULL)
Escribir en Archivo(archt, "Archivo proveedores.dat no encontrado")
regresar
FinSi
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir en Archivo(archt, "-")
FinPara
Escribir en Archivo(archt, "")

Escribir en Archivo(archt, "%-20s %-20s %-20s", "Clave", "Nombre",
"Teléfono")
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir en Archivo(archt, "-")
FinPara
Escribir en Archivo(archt, "")
```

```
////////////////////////////////////////////////////////////////
//Escribe secuencialmente
////////////////////////////////////////////////////////////////

Proveedor proveedor
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, archd)
Mientras  (!feof(archd))
Escribir en Archivo(archt, "%-20s %-20s %-20d", proveedor.clave,
proveedor.nombre, proveedor.telefono)
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, archd)
FinMientras
Para  (ENTERO i <- 0 Hasta ((20*3)) Con Paso 1)
Escribir en Archivo(archt, "-")
FinPara
Escribir en Archivo(archt, "")
Cerrar Archivo(archd)
Cerrar Archivo(archt)
FinSubProceso




// * Administración
//Función que crea un archivo
SubProceso crear(): tipo vacía
CADENA str
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Escribir "Dijite el nombre del archivo que quiere crear, junto con su
extensión: "
Leer str
CADENA comando <- "type nul > " + str
Llamar system(comando.c_str())
FinSubProceso

//Función que respalda productos.dat, proveedores.dat, vendedores.dat y
ventas.dat
SubProceso respaldar(): tipo vacía
Inicio
FILE *arch

//Genera el respaldo y copia los datos
//Verifica que no exista antes un respaldo, si existe, lo borra
arch <- Abrir Archivo("productos.dat", "r+b")
Si  (arch == NULL)
SiNo
```

```
FILE *arch2
arch2 <- Abrir Archivo("productos respaldo.dat", "r+b")
Si  (arch2 == NULL)
arch2 <- Abrir Archivo("productos respaldo.dat", "ab")
Si  (arch2 == NULL)
SiNo
tproducto producto
Leer En Archivo(&producto, sizeof(tproducto), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&producto, sizeof(tproducto), 1, arch2)
Leer En Archivo(&producto, sizeof(tproducto), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
SiNo
Cerrar Archivo(arch2)
Llamar systemLlamar system(del \productos respaldo.dat\)
arch2 <- Abrir Archivo("productos respaldo.dat", "ab")
Si  (arch2 == NULL)
SiNo
tproducto producto
Leer En Archivo(&producto, sizeof(tproducto), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&producto, sizeof(tproducto), 1, arch2)
Leer En Archivo(&producto, sizeof(tproducto), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
FinSi
FinSi
Cerrar Archivo(arch)

//Genera el respaldo y copia los datos
//Verifica que no exista antes un respaldo, si existe, lo borra

arch <- Abrir Archivo("proveedores.dat", "r+b")
Si  (arch == NULL)
SiNo
FILE *arch2
arch2 <- Abrir Archivo("proveedores respaldo.dat", "r+b")
Si  (arch2 == NULL)
arch2 <- Abrir Archivo("proveedores respaldo.dat", "ab")
Si  (arch2 == NULL)
SiNo
Proveedor proveedor
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
```

```
Mientras  (!feof(arch))
Escribir En Archivo(&proveedor, sizeof(Proveedor), 1, arch2)
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
SiNo
Cerrar Archivo(arch2)
Llamar systemLlamar system(del \proveedores respaldo.dat\)
arch2 <- Abrir Archivo("proveedores respaldo.dat", "ab")
Si  (arch2 == NULL)
SiNo
Proveedor proveedor
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&proveedor, sizeof(Proveedor), 1, arch2)
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
FinSi
FinSi
Cerrar Archivo(arch)

//Genera el respaldo y copia los datos
//Verifica que no exista antes un respaldo, si existe, lo borra

arch <- Abrir Archivo("vendedores.dat", "r+b")
Si  (arch == NULL)
SiNo
FILE *arch2
arch2 <- Abrir Archivo("vendedores respaldo.dat", "r+b")
Si  (arch2 == NULL)
arch2 <- Abrir Archivo("vendedores respaldo.dat", "ab")
Si  (arch2 == NULL)
SiNo
Vendedor vendedor
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&vendedor, sizeof(Vendedor), 1, arch2)
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
SiNo
Cerrar Archivo(arch2)
```

```
Llamar systemLlamar system(del \vendedores respaldo.dat\)
arch2 <- Abrir Archivo("vendedores respaldo.dat", "ab")
Si  (arch2 == NULL)
SiNo
Vendedor vendedor
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&vendedor, sizeof(Vendedor), 1, arch2)
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
FinSi
FinSi
Cerrar Archivo(arch)
//Genera el respaldo y copia los datos
//Verifica que no exista antes un respaldo, si existe, lo borra
arch <- Abrir Archivo("ventas.dat", "r+b")
Si  (arch == NULL)
SiNo
FILE *arch2
arch2 <- Abrir Archivo("ventas respaldo.dat", "r+b")
Si  (arch2 == NULL)
arch2 <- Abrir Archivo("ventas respaldo.dat", "ab")
Si  (arch2 == NULL)
SiNo
Venta venta
Leer En Archivo(&venta, sizeof(Venta), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&venta, sizeof(Venta), 1, arch2)
Leer En Archivo(&venta, sizeof(Venta), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
SiNo
Cerrar Archivo(arch2)
Llamar systemLlamar system(del \ventas respaldo.dat\)
arch2 <- Abrir Archivo("ventas respaldo.dat", "ab")
Si  (arch2 == NULL)
SiNo
Venta venta
Leer En Archivo(&venta, sizeof(Venta), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&venta, sizeof(Venta), 1, arch2)
Leer En Archivo(&venta, sizeof(Venta), 1, arch)
FinMientras
```

```
Cerrar Archivo(arch2)
FinSi
FinSi
FinSi
Cerrar Archivo(arch)
FinSubProceso

//Función que recupera productos.dat, proveedores.dat, vendedores.dat y
ventas.dat desde sus respaldos
//Básicamente es la función inversa del respaldar

SubProceso restaurar(): tipo vacía
Inicio
FILE *arch
//Copia los datos del respaldo a otro archivo nuevo y luego lo renombra
//Borra el archivo original si existe
arch <- Abrir Archivo("productos respaldo.dat", "r+b")
Si  (arch == NULL)
SiNo
FILE *arch2
arch2 <- Abrir Archivo("productos.dat", "r+b")
Si  (arch2 == NULL)
arch2 <- Abrir Archivo("productos.dat", "ab")
Si  (arch2 == NULL)
SiNo
tproducto producto
Leer En Archivo(&producto, sizeof(tproducto), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&producto, sizeof(tproducto), 1, arch2)
Leer En Archivo(&producto, sizeof(tproducto), 1, arch)
FinMientras
Cerrar Archivo(arch2)
//Inicializa al arreglo en valores nulos
Llamar initializeHashTable();
//Rellena con productos.dat
Llamar fillHashTable();
FinSi
SiNo
Cerrar Archivo(arch2)
Llamar systemLlamar system(del \productos.dat\)
arch2 <- Abrir Archivo("productos.dat", "ab")
Si  (arch2 == NULL)
SiNo
tproducto producto
Leer En Archivo(&producto, sizeof(tproducto), 1, arch)
Mientras  (!feof(arch))
```

```
Escribir En Archivo(&producto, sizeof(tproducto), 1, arch2)
Leer En Archivo(&producto, sizeof(tproducto), 1, arch)
FinMientras
Cerrar Archivo(arch2)
//Inicializa al arreglo en valores nulos
Llamar initializeHashTable();
//Rellena con productos.dat
Llamar fillHashTable();
FinSi
FinSi
FinSi
Cerrar Archivo(arch)

//Copia los datos del respaldo a otro archivo nuevo y luego lo renombra
//Borra el archivo original si existe
arch <- Abrir Archivo("proveedores respaldo.dat", "r+b")
Si  (arch == NULL)
SiNo
FILE *arch2
arch2 <- Abrir Archivo("proveedores.dat", "r+b")
Si  (arch2 == NULL)
arch2 <- Abrir Archivo("proveedores.dat", "ab")
Si  (arch2 == NULL)
SiNo
Proveedor proveedor
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&proveedor, sizeof(Proveedor), 1, arch2)
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
SiNo
Cerrar Archivo(arch2)
Llamar system(del \proveedores.dat\)
arch2 <- Abrir Archivo("proveedores.dat", "ab")
Si  (arch2 == NULL)
SiNo
Proveedor proveedor
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&proveedor, sizeof(Proveedor), 1, arch2)
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
```

```
FinSi
FinSi
Cerrar Archivo(arch)

//Copia los datos del respaldo a otro archivo nuevo y luego lo renombra
//Borra el archivo original si existe

arch <- Abrir Archivo("vendedores respaldo.dat", "r+b")
Si  (arch == NULL)
SiNo
FILE *arch2
arch2 <- Abrir Archivo("vendedores.dat", "r+b")
Si  (arch2 == NULL)
arch2 <- Abrir Archivo("vendedores.dat", "ab")
Si  (arch2 == NULL)
SiNo
Vendedor vendedor
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&vendedor, sizeof(Vendedor), 1, arch2)
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
SiNo
Cerrar Archivo(arch2)
Llamar system(del \vendedores.dat\)
arch2 <- Abrir Archivo("vendedores.dat", "ab")
Si  (arch2 == NULL)
SiNo
Vendedor vendedor
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&vendedor, sizeof(Vendedor), 1, arch2)
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
FinSi
FinSi
Cerrar Archivo(arch)
//Copia los datos del respaldo a otro archivo nuevo y luego lo renombra
//Borra el archivo original si existe

arch <- Abrir Archivo("ventas respaldo.dat", "r+b")
Si  (arch == NULL)
```

```
SiNo
FILE *arch2
arch2 <- Abrir Archivo("ventas.dat", "r+b")
Si  (arch2 == NULL)
arch2 <- Abrir Archivo("ventas.dat", "ab")
Si  (arch2 == NULL)
SiNo
Venta venta
Leer En Archivo(&venta, sizeof(Venta), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&venta, sizeof(Venta), 1, arch2)
Leer En Archivo(&venta, sizeof(Venta), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
SiNo
Cerrar Archivo(arch2)
Llamar system(del \ventas.dat\)
arch2 <- Abrir Archivo("ventas.dat", "ab")
Si  (arch2 == NULL)
SiNo
Venta venta
Leer En Archivo(&venta, sizeof(Venta), 1, arch)
Mientras  (!feof(arch))
Escribir En Archivo(&venta, sizeof(Venta), 1, arch2)
Leer En Archivo(&venta, sizeof(Venta), 1, arch)
FinMientras
Cerrar Archivo(arch2)
FinSi
FinSi
FinSi
Cerrar Archivo(arch)
FinSubProceso

//Función que elimina completamente los registros vacíos
SubProceso compactar(): tipo vacía
Inicio
FILE *arch
//Solo puede eliminar de proveedores.dat y vendedores.dat porque son
secuenciales
//Aunque ventas.dat también es secuencial, las bajas no se manejan de la
misma manera
//En ventas.dat se da un reembolso y se tiene registro de ello, por eso
no se borra
//Funcionamiento
//En el archivo busca los registros con clave=""
```

```
arch <- Abrir Archivo("proveedores.dat", "r+b")
Si  (arch == NULL)
Cerrar Archivo(arch)
SiNo
FILE *arch2
arch2 <- Abrir Archivo("proveedores temporal.dat", "ab")
Proveedor proveedor
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
Mientras  (!feof(arch))
ENTERO ret <- Comparar Cadenas(proveedor.clave, "", 20)
Si  (ret != 0)
Escribir En Archivo(&proveedor, sizeof(Proveedor), 1, arch2)
FinSi
Leer En Archivo(&proveedor, sizeof(Proveedor), 1, arch)
FinMientras
Cerrar Archivo(arch)
Cerrar Archivo(arch2)
Llamar system(del proveedores.dat)
Llamar system(ren \proveedores temporal.dat\ proveedores.dat)
FinSi

arch <- Abrir Archivo("vendedores.dat", "r+b")
Si  (arch == NULL)
Cerrar Archivo(arch)
SiNo
FILE *arch2
arch2 <- Abrir Archivo("vendedores temporal.dat", "ab")
Vendedor vendedor
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
Mientras  (!feof(arch))
Si  (vendedor.clave != 0)
Escribir En Archivo(&vendedor, sizeof(Vendedor), 1, arch2)
FinSi
Leer En Archivo(&vendedor, sizeof(Vendedor), 1, arch)
FinMientras
Cerrar Archivo(arch)
Cerrar Archivo(arch2)
Llamar system(del vendedores.dat)
Llamar system(ren \vendedores temporal.dat\ vendedores.dat)
FinSi
FinSubProceso

// * Main
//El main solo se encarga del menú
Algoritmo MENÚ PRINCIPAL
// Establecer el idioma a español
```

```
setlocale(LC_ALL, "es_ES")          // Cambiar locale - Suficiente para
máquinas Linux
SetConsoleCP(65001)                 // Cambiar STDIN -  Para máquinas
Windows
SetConsoleOutputCP(65001)           // Cambiar STDOUT - Para máquinas
Windows
Vaciar Buffer

//Inicializa al arreglo en valores nulos
Llamar initializeHashTable()
//Rellena con productos.dat
Llamar fillHashTable()

//Variables para controlar los menús y submenús
ENTERO opcion1,
opcion2, opcion3

CADENA objetivo

CARACTER c[] <- "123"
CARACTER Intento[20]

BOOLEANA con <- true

//Obtiene el número de columnas y filas de la consola
CONSOLE_SCREEN_BUFFER_INFO csbi
ENTERO columns, rows

////////////////////////////////////////////////////
//Primer mensaje
////////////////////////////////////////////////////

Limpiar Pantalla
//system("color 89")




//Obtiene el número de columnas y filas de la consola
Llamar GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&csbi)
columns <- csbi.srWindow.Right - csbi.srWindow.Left + 1
rows <- csbi.srWindow.Bottom - csbi.srWindow.Top + 1
Llamar centrar_cadena("UNIVERsiDAD AUTÓNOMA DEL ESTADO DE MÉXICO",
columns)
Llamar centrar_cadena("FACULTAD DE INGENIERÍA", columns)
```

```
Llamar centrar_cadena("INGENIERÍA EN COMPUTACIÓN", columns)

Llamar centrar_cadena("ORGANIZACIÓN DE ARCHIVOS", columns)

Llamar centrar_cadena("PROYECTO FINAL", columns)

Escribir "Grupo: CO02"

Escribir "Estudiantes: Ana Laura Contreras Peralta"
Escribir "              Alejandro González Jiménez"
Escribir "              Cristian Omar Gutiérrez Millán"
Escribir "              Daniel Sotelo Rizo"
Escribir "              Elizabeth García González"
Escribir "              Israel Enríquez Barreto"
Escribir "              María de Jesús Sánchez Suárez"


Escribir "     SEMESTRE 2020-A"
Llamar derecha_cadena("JUNIO DE 2020      ", columns - 15 - 5)

Llamar escucharEspacio(1)
//Switch anidados dentro de whiles para navegar dentro del menú
Mientras  (opcion1 != 8)
//Obtiene el número de columnas y filas de la consola
Llamar GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&csbi)
columns <- csbi.srWindow.Right - csbi.srWindow.Left + 1
rows <- csbi.srWindow.Bottom - csbi.srWindow.Top + 1

Limpiar Pantalla
Llamar system(color 0B)

Llamar centrar_cadena("MENÚ PRINCIPAL", columns)

Escribir " [1]. Accesorios"

Escribir " [2]. Proveedores"

Escribir " [3]. Vendedor"

Escribir " [4]. Ventas"

Escribir " [5]. Informes"

Escribir " [6]. Administración"
```

```
Escribir " [7]. Ayuda"

Escribir " [8]. Salir"

Escribir "Presione su opción: "
opcion1 <- Llamar escucharTecla(8)
opcion2 <- 0
Segun  (opcion1) Hacer
1:
Mientras  (opcion2 != 5)
objetivo <- "accesorio"
BOOLEANA codigoValido
CADENA code
CARACTER code_CARACTER[6 + 1]
Limpiar Pantalla
Llamar system(color 0C)
//Obtiene el número de columnas y filas de la consola
Llamar GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&csbi)
columns <- csbi.srWindow.Right - csbi.srWindow.Left + 1
rows <- csbi.srWindow.Bottom - csbi.srWindow.Top + 1

Llamar centrar_cadena("MENÚ DE ACCESORIOS", columns)

Escribir " [1]. Agregar " + objetivo
Escribir " [2]. Eliminar " + objetivo
Escribir " [3]. Modificar "
Escribir " [4]. Buscar " + objetivo
Escribir " [5]. Regresar"
Escribir "Presione su opción: "
opcion2 <- Llamar escucharTecla(5)
Segun  (opcion2) Hacer
1:
Llamar system(color 0A)
Llamar altaProducto()

FinSegun
2:
codigoValido <- 0
Llamar system(color 0A)
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Mientras  (!codigoValido)
Escribir "Ingrese el codigo del producto a eliminar: ")
Vaciar Buffer
Llamar getline(cin, code)
```

```
Copiar Cadena(code_CARACTER, code.c_str())
codigoValido <- Llamar validarCodigo(code_CARACTER)
Si  (!codigoValido)
Escribir "Usted ingreso un codigo invalido!!!"
Escribir "Un codigo correcto empieza con 2 letras y le siguen 4
numeros."

FinSi
FinMientras
Llamar bajaProducto(code)
FinMientras
3:
Llamar system(color 0A)
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
codigoValido <- 0
Mientras  (!codigoValido)
Llamar system(color 0A)
Escribir "Ingrese el codigo del producto a modificar: ")
Vaciar Buffer
Llamar getline(cin, code)
Copiar Cadena(code_CARACTER, code.c_str())
codigoValido <- Llamar validarCodigo(code_CARACTER)
Si  (!codigoValido)
Escribir "Usted ingreso un codigo invalido!!!"
Escribir "Un codigo correcto empieza con 2 letras y le siguen 4
numeros."

FinSi
FinMientras
Llamar cambioProducto(code)
FinSegun
4:{
Llamar system(color 0A)
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
codigoValido <- 0
Mientras  (!codigoValido)
Escribir "Ingrese el codigo del producto a buscar: ")
Vaciar Buffer
Llamar getline(cin, code)
Copiar Cadena(code_CARACTER, code.c_str())
codigoValido <- Llamar validarCodigo(code_CARACTER)
Si  (!codigoValido)
Escribir "Usted ingreso un codigo invalido!!!"
Escribir "Un codigo correcto empieza con 2 letras y le siguen 4
```

```
numeros."

FinSi
FinMientras
Llamar consultaProducto(code)
FinMientras
5:{
Llamar system(color B)
Escribir "Regresando al menú principal..."

FinAlgoritmo
De Otro Modo:{
Llamar system(color 0A)
Escribir "Digite una opción correcta."
}
}
Llamar escucharEspacio()
}
}
2:{
Mientras  (opcion2 != 5)
objetivo <- "proveedor"
Limpiar Pantalla
Llamar system(color 0C)
//Obtiene el número de columnas y filas de la consola
Llamar GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&csbi)
columns <- csbi.srWindow.Right - csbi.srWindow.Left + 1
rows <- csbi.srWindow.Bottom - csbi.srWindow.Top + 1

Llamar centrar_cadena("MENÚ DE PROVEEDORES", columns)
Escribir " [1]. Agregar " + objetivo
Escribir " [2]. Eliminar " + objetivo
Escribir " [3]. Modificar "
Escribir " [4]. Buscar " + objetivo
Escribir " [5]. Regresar"
Escribir "Presione su opción: "
opcion2 <- Llamar escucharTecla(5)
Segun  (opcion2) Hacer
1:
Llamar system(color 0A)
Llamar altaProveedor()
FinSegun
2:
Llamar system(color 0A)
Llamar bajaProveedor()
```

```
FinMientras
3:
Llamar system(color 0A)
Llamar cambioProveedor()
4:
Llamar system(color 0A)
Llamar consultaProveedor()
5:
Llamar system(color 0B)
Escribir "Regresando al menú principal..."

De Otro Modo:
Escribir "Digite una opción correcta"
}
}
Llamar escucharEspacio()
}
}
3:{
Mientras  (opcion2 != 5)
objetivo <- "vendedor"
Limpiar Pantalla
Llamar system(color 0C)
//Obtiene el número de columnas y filas de la consola
Llamar GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&csbi)
columns <- csbi.srWindow.Right - csbi.srWindow.Left + 1
rows <- csbi.srWindow.Bottom - csbi.srWindow.Top + 1

Llamar centrar_cadena("MENÚ DE VENDEDOR", columns)
Escribir " [1]. Agregar " + objetivo
Escribir " [2]. Eliminar " + objetivo
Escribir " [3]. Modificar "
Escribir " [4]. Buscar " + objetivo
Escribir " [5]. Regresar"
Escribir "Presione su opción: "
opcion2 <- Llamar escucharTecla(5)
Segun  (opcion2) Hacer
1:
Llamar system(color 0A)
Llamar altaVendedor()
FinSegun
2:
Llamar system(color 0A)
Llamar bajaVendedor()
FinMientras
```

```
3:
Llamar system(color 0A)
Llamar cambioVendedor()
4:
Llamar system(color 0A)
Llamar consultaVendedor()
5:
Llamar system(color 0B)
Escribir "Regresando al menú principal..."

De Otro Modo:
Escribir "Digite una opción correcta"
}
Llamar escucharEspacio()
}
}
4:{
Mientras  (opcion2 != 3)
BOOLEANA codigoValido
CADENA code
CARACTER code_CARACTER[6 + 1]
Limpiar Pantalla
Llamar system(color 0C)
//Obtiene el número de columnas y filas de la consola
Llamar GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&csbi)
columns <- csbi.srWindow.Right - csbi.srWindow.Left + 1
rows <- csbi.srWindow.Bottom - csbi.srWindow.Top + 1

Llamar centrar_cadena("MENÚ DE VENTAS", columns)
Escribir " [1]. Venta de un producto "
Escribir " [2]. Reembolso de un producto "
Escribir " [3]. Regresar"
Escribir "Presione su opción: "
opcion2 <- Llamar escucharTecla(3)
Segun  (opcion2) Hacer
1:
Llamar system(color 0A)
Llamar altaVenta()
FinSegun
2:
Llamar system(color 0A)
Llamar bajaVenta()
FinMientras
3:
Llamar system(color 0B)
```

```
Escribir "Regresando al menú principal..."

De Otro Modo:
Escribir "Digite una opción correcta"
Llamar escucharEspacio()
}
}
5:{
Mientras  (opcion2 != 5)
Limpiar Pantalla
Llamar system(color 0C)
//Obtiene el número de columnas y filas de la consola
Llamar GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&csbi)
columns <- csbi.srWindow.Right - csbi.srWindow.Left + 1
rows <- csbi.srWindow.Bottom - csbi.srWindow.Top + 1

Llamar centrar_cadena("MENÚ DE INFORMES", columns)
Escribir " [1]. Inventario"
Escribir " [2]. Reporte de ventas"
Escribir " [3]. Reporte de vendedores"
Escribir " [4]. Reporte de proveedores"
Escribir " [5]. Regresar"
Escribir "Presione su opción: "
opcion2 <- Llamar escucharTecla(5)
opcion3 <- 0
Segun  (opcion2) Hacer
1:
Mientras  (opcion3 != 3)
Limpiar Pantalla
Llamar system(color 0A)
Llamar centrar_cadena("MENÚ DE INVENTARIO", columns)
Escribir " [1]. Reporte impreso en pantalla"
Escribir " [2]. Reporte en archivo de texto"
Escribir " [3]. Regresar"
Escribir "Presione su opción: "
opcion3 <- Llamar escucharTecla(3)
Segun  (opcion3) Hacer
1:
Llamar system(color 0F)
Llamar inventarioPantalla()
FinSegun
2:
Llamar system(color 0F)
Llamar inventarioArchivo()
FinMientras
```

```
3:
Llamar system(color 0C)
Escribir "Regresando al menú de informes..."
FinSegun
De Otro Modo:
Escribir "Digite una opción correcta"

FinMientras
}
Llamar escucharEspacio()
}
}
2:{
Mientras  (opcion3 != 3)
Limpiar Pantalla
Llamar system(color 0A)
Llamar centrar_cadena("MENÚ DE REPORTE DE VENTAS", columns)
Escribir " [1]. Reporte impreso en pantalla"
Escribir " [2]. Reporte en archivo de texto"
Escribir " [3]. Regresar"
Escribir "Presione su opción: "
opcion3 <- Llamar escucharTecla(3)
Segun  (opcion3) Hacer
1:
Llamar system(color 0F)
Llamar reporteDeVentasPantalla()
FinSegun
2:
Llamar system(color 0F)
Llamar reporteDeVentasArchivo()
FinMientras
3:
Llamar system(color 0C)
Escribir "Regresando al menú de informes..."


De Otro Modo:
Escribir "Digite una opción correcta"
}
}
Llamar escucharEspacio()
}
}
3:{
Mientras  (opcion3 != 3)
Limpiar Pantalla
```

```
Llamar system(color 0A)
Llamar centrar_cadena("MENÚ DE VENDEDORES", columns)
Escribir " [1]. Reporte impreso en pantalla"
Escribir " [2]. Reporte en archivo de texto"
Escribir " [3]. Regresar"
Escribir "Presione su opción: "
opcion3 <- Llamar escucharTecla(3)
Segun  (opcion3) Hacer
1:
Llamar system(color 0F)
Llamar reporteDeVendedoresPantalla()
FinSegun
2:
Llamar system(color 0F)
Llamar reporteDeVendedoresArchivo()
FinMientras
3:
Llamar system(color 0C)
Escribir "Regresando al menú de informes..."

De Otro Modo:
Escribir "Digite una opción correcta"

}
}
Llamar escucharEspacio()
}
}
4:{
Mientras  (opcion3 != 3)
Limpiar Pantalla
Llamar system(color 0A)
Llamar centrar_cadena("MENÚ DE REPORTE DE PROVEEDORES", columns)
Escribir " [1]. Reporte impreso en pantalla"
Escribir " [2]. Reporte en archivo de texto"
Escribir " [3]. Regresar"
Escribir "Presione su opción: "
opcion3 <- Llamar escucharTecla(3)
Segun  (opcion3) Hacer
1:
Llamar system(color 0F)
Llamar reporteDeProveedoresPantalla()
FinSegun
2:
Llamar system(color 0F)
Llamar reporteDeProveedoresArchivo()
```

```
FinMientras
3:
Llamar system(color 0C)
Escribir "Regresando al menú de informes..."
De Otro Modo:
Escribir "Digite una opción correcta"
}
}
Llamar escucharEspacio()
}
}
5:{
Llamar system(color 0B)
Escribir "Regresando al menú principal..."

Llamar escucharEspacio()
De Otro Modo:
Escribir "Digite una opción correcta"
Llamar escucharEspacio()
}
}
}
}
6:{
//Borra lo que escribimos escuchando las teclas
Llamar FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))
Limpiar Pantalla
Llamar system(color 0C)
//Obtiene el número de columnas y filas de la consola
Llamar GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&csbi)
columns <- csbi.srWindow.Right - csbi.srWindow.Left + 1
rows <- csbi.srWindow.Bottom - csbi.srWindow.Top + 1

Llamar centrar_cadena("MENÚ DE ADMINISTRACIÓN", columns)
BOOLEANA hayCaracterIncorrecto <- false
Vaciar Buffer
Escribir "Digite la contraseña: "
Vaciar Buffer
Llamar gets(Intento)
Para  (ENTERO j <- 0 Hasta strlen(Intento Con Paso 1)
Si  (c[j] != Intento[j])
hayCaracterIncorrecto <- true
FinSi
FinPara
Si  (!hayCaracterIncorrecto)
```

```
Mientras  (opcion2 != 5)
Limpiar Pantalla
Llamar system(color 0A)
Llamar centrar_cadena("ADMINISTRACIÓN", columns)
Escribir " [1]. Crear archivos"
Escribir " [2]. Respaldar"
Escribir " [3]. Restaurar"
Escribir " [4]. Compactar archivos"
Escribir " [5]. Regresar"
opcion2 <- Llamar escucharTecla(5)
Segun  (opcion2) Hacer
1:
Llamar system(color 0F)
Llamar crear()
FinSegun
2:
Llamar system(color 0F)
Llamar respaldar()
FinMientras
3:
Llamar system(color 0F)
Llamar restaurar()
FinSi
4:
Llamar system(color 0F)
Llamar compactar()
5:
Llamar system(color 0B)
Escribir "Regresando al menú principal..."
De Otro Modo:
Escribir "Digite una opción correcta"
}
}
Llamar escucharEspacio()
}
}
SiNo
Llamar system(color 0A)
Escribir "Contraseña incorrecta"

Llamar system(color 0B)
Escribir "Regresando al menú principal..."

Llamar escucharEspacio()
FinSi
}
```

```
7:
      Abrir Manual de Usuario / Técnico
8:{
Llamar system(color 0E)
Llamar GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&csbi)
columns <- csbi.srWindow.Right - csbi.srWindow.Left + 1
rows <- csbi.srWindow.Bottom - csbi.srWindow.Top + 1
Llamar centrar_cadena("¡Hasta luego!", columns)


Llamar escucharEspacio()
}
De Otro Modo:
Escribir "¡Digite una opción correcta! "
Llamar escucharEspacio()
}
}
}
regresar 0
}
FinAlgoritmo
```

4. Requerimientos de HW Y SW para el desarrollo y mantenimiento

4.1 Hardware

Computadora o Laptop con:

- ✓ Procesador mayor a 1.2 GHz
- ✓ Ram 2GB
- ✓ Almacenamiento 1GB disponible

Además de que debe de contar con teclado y mouse o trackpad (ver imagen 4.1.1) para el desplazamiento durante la ejecución del programa.

Imagen 4.1.1: De lado izquierdo tenemos lo que es un trackpad y de lado derecho un mouse

4.2 Software

- Windows 7 o superior (Aunque de preferencia se le recomienda usar Windows 10).
- Tener instalado algún compilador de lenguaje C, C++.

En caso de tener instalado Dev-C++ se recomienda hacer las siguientes configuraciones:

1. Ir al apartado de herramientas (ver imagen 4.2.1).
2. Seleccionar opciones de compilador (ver imagen 4.2.2).
3. Escoger TDM-GCC 4.9.2 64-bit Debug(ver imagen 4.2.3).
4. Ir a la parte de Generación/Optimización de Código y luego a Generación de código.
5. En el apartado de lenguaje standard escoger GNU C++11(ver imagen 4.2.4).
6. Dar click en aceptar.

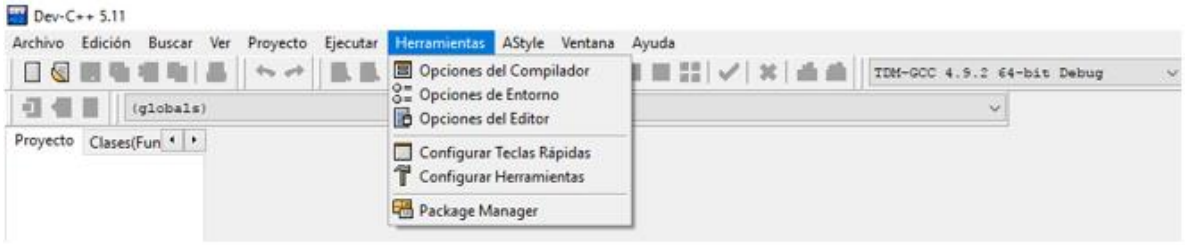


Imagen 4.2.1

Imagen 4.2.2

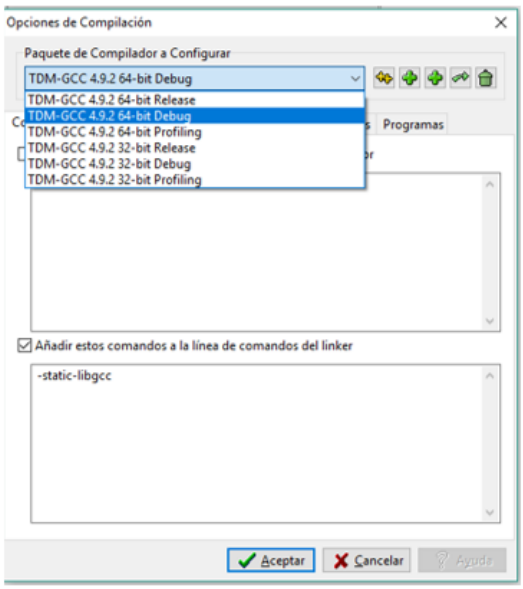


Imagen 4.2.3

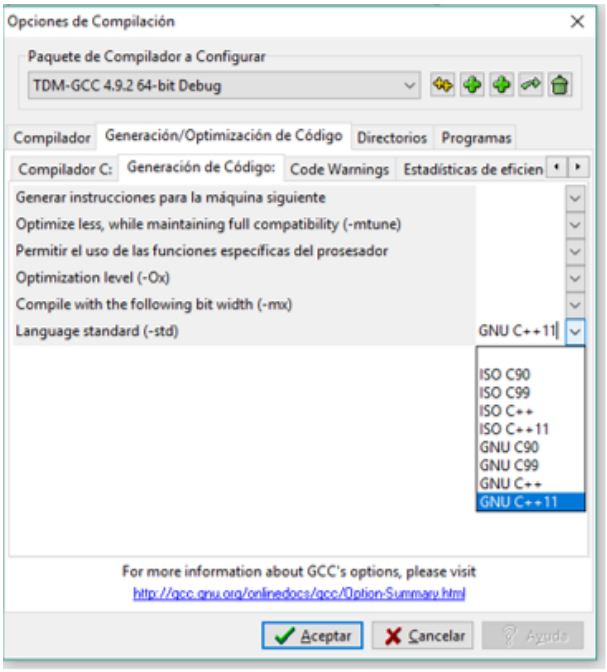


Imagen 4.2.4

5. Resultados de las Pruebas y validación, y recomendaciones.

Este apartado permitirá reconocer tipos de errores que debe de tratar evitar al realizar la ejecución del programa, dicho así se muestran algunas pruebas, donde se reafirma lo que se ha mostrado anteriormente con las limitaciones que tiene el programa.
El primer error que se puede presentar es el uso de las teclas numéricas del lado derecho del teclado, como muestra a continuación de color rojo (ver Imagen 6.1.) ya que todo es posible gracias a una función que está implementada en el programa llamada GetKeyState recuerda que puedes usar sólo las teclas que se muestran de color verde (ver Imagen 6.1.)

Imagen 6.1. Teclas numéricas permitidas y denegadas.

Es importante reconocer que el programa permite reaccionar ante cualquier número, como respuesta a una opción dentro de los menú y submenús; sin dar enter. A continuación, se muestra la prueba como un ejemplo claro, ya que una vez que se corre el programa se desea introducir un número que reaccione cómodamente.

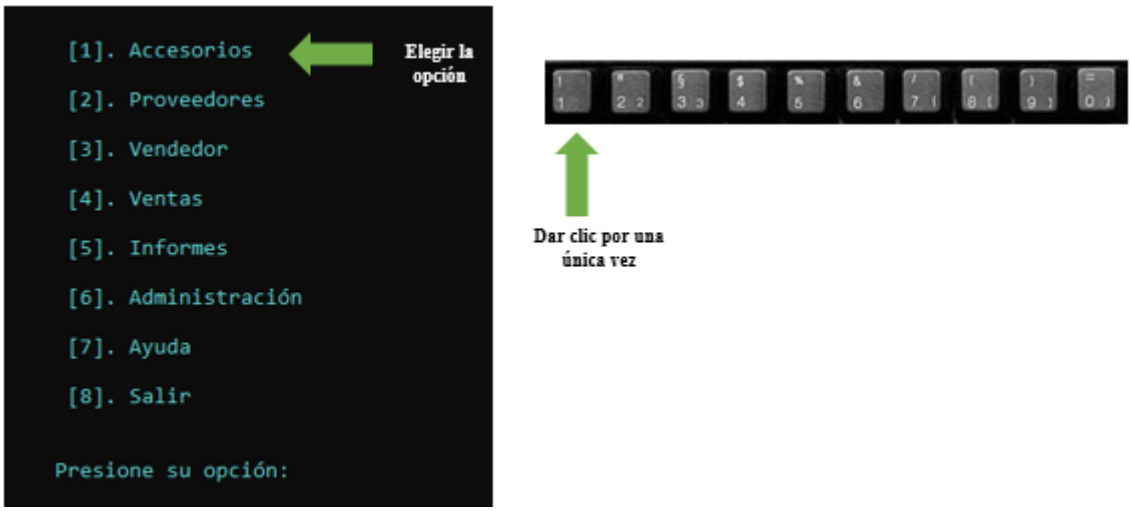


Imagen 6.1.1. Ingresar con una sola tecla la opción que deseamos, para Menú Principal.

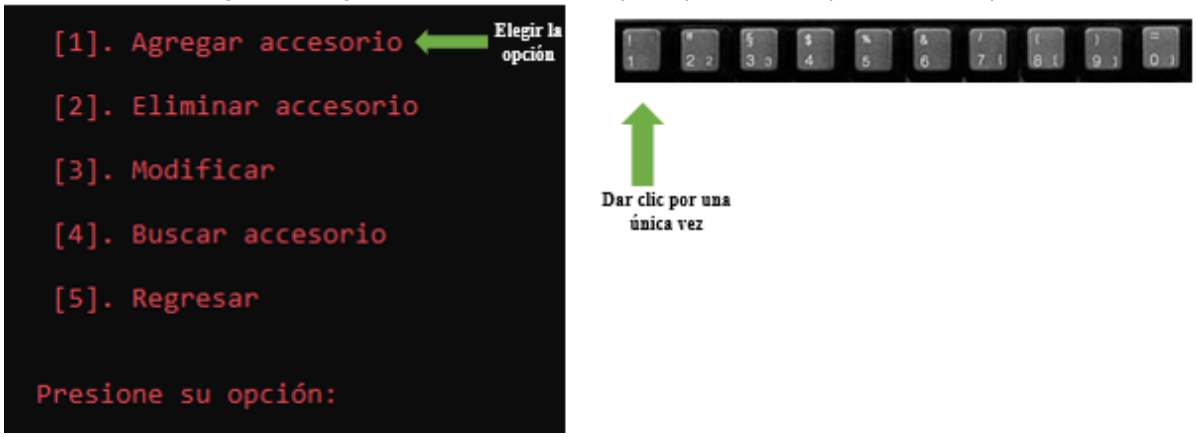


Imagen 6.1.2. Ingresar con una sola tecla la opción que deseamos, para Submenús

Como se mostró anteriormente, esta función de las teclas será válida para cualquier menú y submenú dentro del programa.
En la parte de altas, es posible contar con la restricción de ingresar un código inadecuado en el programa como en el tamaño de la cadena al ingresar datos de tipo char, el siguiente ejemplo puede ser un error que podría presentarse en código de accesorio (ver Imagen 6.2.1.)

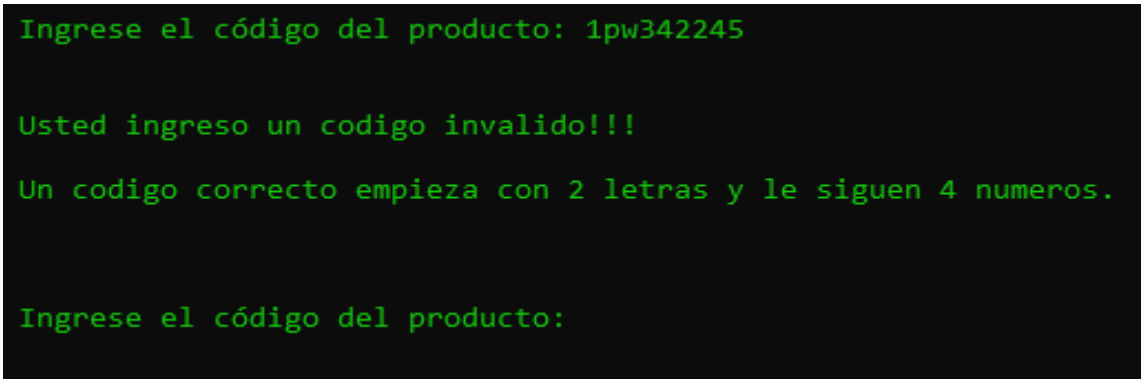


Imagen 6.2.1. El código no cumple con el tamaño de la cadena rebasando los 6 caracteres sin cumplir con el número de números.

Podemos notar que no sólo nos muestra un mensaje que nos dice que el código es inválido, sino que también el programa nos seguirá pidiendo un código hasta que sea correcto. Cada vez que se presente este caso se mostrará un mensaje que especifica las características para

que el código a ingresar sea correcto, esto permite que el usuario note cual es el error que está cometiendo y verifique su código.

A continuación, se muestra un ejemplo donde el código es ingresado correctamente (ver Imagen 6.2.2.).

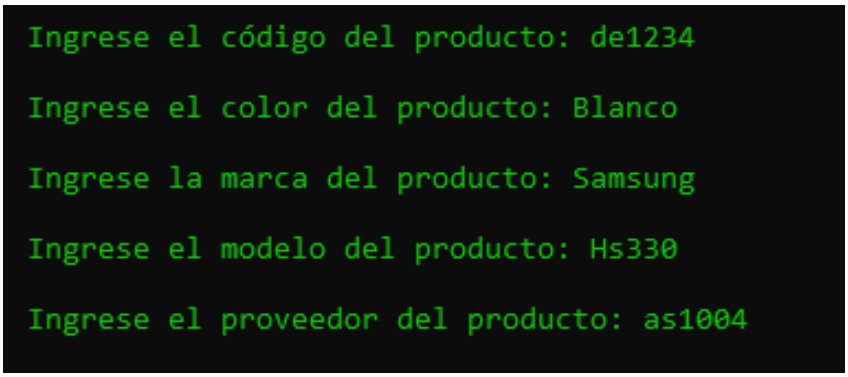


Imagen 6.2.2. El código cumple con las expectativas del programa, entonces permite seguir llenando los campos.

Sin embargo, es recomendable dar de alta primero el campo de proveedores para poder agregar un accesorio, siguiendo las indicaciones que se mostraron en un principio (ver Imagen 6.2.3.).

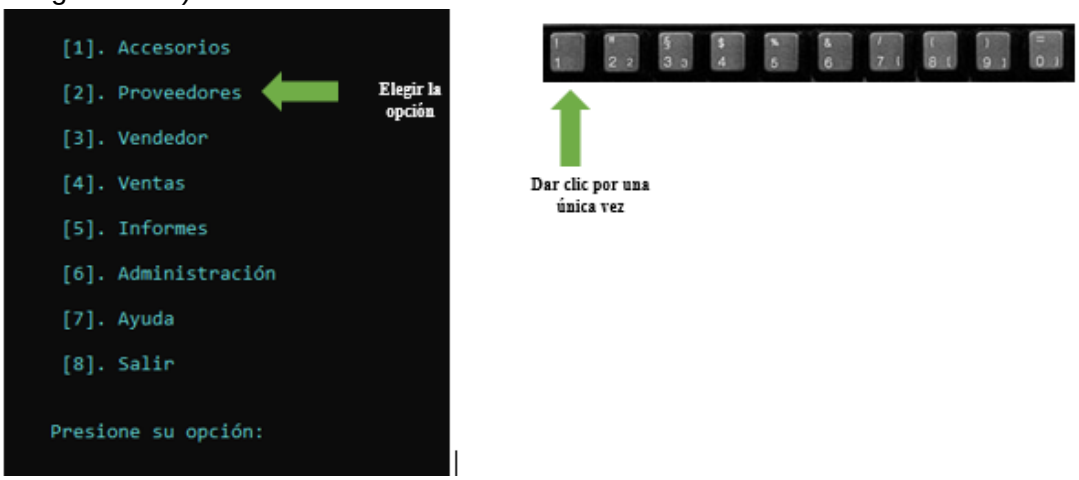


Imagen 6.2.3. Menú de proveedores para que se realicen las altas.

Y lo mismo pasa para ventas, es necesario primero agregar un vendedor y llenar los campos de los datos que pide para proceder a ventas (ver Imagen 6.2.4.).

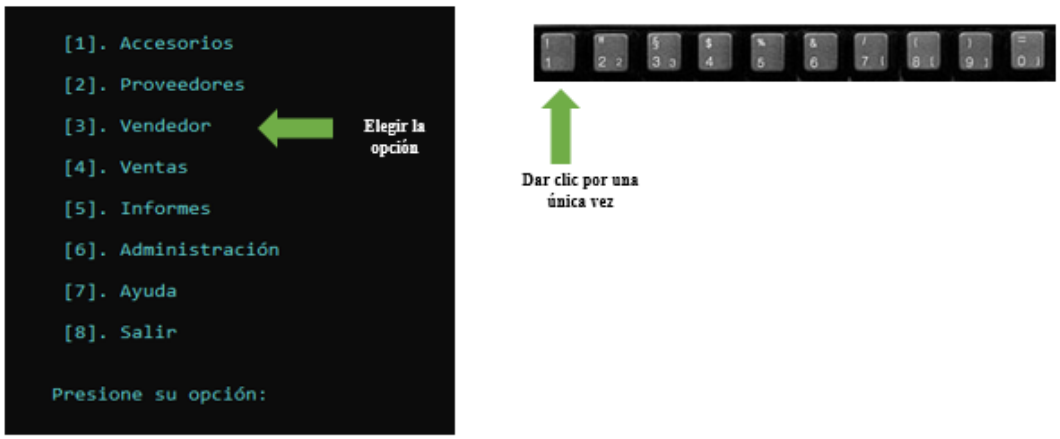


Imagen 6.2.4. Menú de vendedor para iniciar con altas.

6.Codificación comentada coincidente con pseudocódigo.

```
#include <conio.h>
#include <locale.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <wchar.h>
#include <windows.h>

#include <ctime>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>

#define MAX 1000
#define NUMBER_OF_SLOTS 100

using namespace std;
```

```
/**************************************
 * Utilerías
 **************************************/

//Función que espera a que presiones espacio
void escucharEspacio() {
    cout << "\t\tPresione espacio para continuar...";
    bool con = true;
    while (con) {
        //Solo si es la ventana activa
        if (GetConsoleWindow() == GetForegroundWindow()) {
            if (GetKeyState(' ') & 0x8000) {
                while (GetKeyState(' ') & 0x8000) {
                }
                con = false;
            }
        }
    }
}
```

```
void escucharEspacio(int i) {
```

```
    cout << "Presione espacio para continuar...";
    bool con = true;
    while (con) {
        //Solo si es la ventana activa
        if (GetConsoleWindow() == GetForegroundWindow()) {
            if (GetKeyState(' ') & 0x8000) {
                while (GetKeyState(' ') & 0x8000) {
                }
                con = false;
            }
        }
    }
}
```

```
//Función que valida el código de un producto
//Si es valida regresa true, y false si no
bool validarCodigo(char codigo[]) {
    char caracter;
    bool esNumero = false;
    bool esLetra = false;

    bool codigoValido = false;

    //Verifica su longitud
    string cadena = codigo;
    if (cadena.length() > 6 || cadena.length() == 0) {
        return false;
    }

    //Verifica que los primeros sean letras y los siguientes, números
    for (int i = 0; i < 6; i++) {
        caracter = codigo[i];
        for (int j = 48; j <= 57; j++) {
            if (caracter == j) {
                esNumero = true;
                esLetra = false;
            }
        }
        for (int j = 97; j <= 122; j++) {
            if (caracter == j) {
                esNumero = false;
                esLetra = true;
            }
        }
        for (int j = 65; j <= 90; j++) {
```

```
            if (caracter == j) {
                esNumero = false;
                esLetra = true;
            }
        }

        if (esNumero == false && esLetra == false) {
            return false;
        }
        if ((i == 0 || i == 1) && esNumero) {
            return false;
        }
        if ((i != 0 && i != 1) && esLetra) {
            return false;
        }

        esNumero = false;
        esLetra = false;
    }
    return true;
}
```

```
//Función que pide un número entero y ayuda a prevenir errores
int pedirEntero(string peticion) {
    string str;
    cout << peticion;

    fflush(stdin);
    cin >> str;
    fflush(stdin);

    //Se llama recursivamente hasta que el usuario digite una opción
correcta
    try {
        return stoi(str);
    }
    catch (...) {
        cout << "\n\t\tDebe ingresar un entero!" << endl;
        return pedirEntero(peticion);
    }
}
```

```
//Función que pide un número long long sin signo y ayuda a prevenir
errores
unsigned long long pedirUnsignedLongLong(string peticion) {
```

```
    string str;
    cout << peticion;

    fflush(stdin);
    cin >> str;
    fflush(stdin);

    //Se llama recursivamente hasta que el usuario digite una opción
correcta
    try {
        return stoull(str);
    }
    catch (...) {
        cout << "\n\n\t\tDebe ingresar un entero!" << endl;
        return pedirUnsignedLongLong(peticion);
    }
}
```

```
//Función que pide un número flotante y ayuda a prevenir errores
float pedirFlotante(string peticion) {
    string str;
    cout << peticion;

    fflush(stdin);
    cin >> str;
    fflush(stdin);

    //Se llama recursivamente hasta que el usuario digite una opción
correcta
    try {
        return stof(str);
    }
    catch (...) {
        cout << "\n\t\tDebe ingresar un flotante!" << endl;
        return pedirFlotante(peticion);
    }
}
```

```
//Función que escucha las teclas
int escucharTecla(int nOpciones) {
    char letras[] = {'U', 'N', 'E', 'I', 'F', 'A', 'Y', 'S'};
    //Entra a un bucle que se rompe cuando pulsas una tecla indicada
    while (true) {
        //Solo si es la ventana activa
```

```
        if (GetConsoleWindow() == GetForegroundWindow()) {
            //Verifiva las opciones que usan ctrl+KEY
            for (int i = 0; i < nOpciones; i++) {
                if ((GetKeyState(VK_CONTROL) & 0x8000) &&
(GetKeyState(letras[i]) & 0x8000)) {
                    while ((GetKeyState(VK_CONTROL) & 0x8000) &&
(GetKeyState(letras[i]) & 0x8000)) {
                    }
                    return i + 1;
                }
            }
            //Verifica las teclas numéricas
            for (int i = 49; i <= 49 + (nOpciones - 1); i++) {
                if (GetKeyState(i) & 0x8000) {
                    while (GetKeyState(i) & 0x8000) {
                    }
                    return i - 48;
                }
            }
        }
    }
}
```

```
//Ayuda para centrar el texto
void substring(char *cadena, char *subcadena, int inicio, int longitud)
{
    int i;

    for (i = 0; i < longitud && inicio + i < strlen(cadena); i++)
        subcadena[i] = cadena[inicio + i];

    subcadena[i] = '\0';
}
```

```
void centrar_linea(char *linea, int ancho) {
    int i, espacios;
    espacios = (ancho - strlen(linea)) / 2;

    for (i = 0; i < espacios; i++)
        printf(" ");

    printf("%s", linea);
}
```

```c
void centrar_cadena(char *cadena, int ancho) {
    char subcadena[MAX];
    int i, total;

    total = (int)ceil((float)strlen(cadena) / ancho);

    for (i = 0; i < total; i++) {
        substring(cadena, subcadena, i * ancho, ancho);
        centrar_linea(subcadena, ancho);
    }
}
```

```c
void derecha_linea(char *linea, int ancho) {
    int i, espacios;
    espacios = ancho - strlen(linea);

    for (i = 0; i < espacios; i++)
        printf(" ");

    printf("%s", linea);
}
```

```c
void derecha_cadena(char *cadena, int ancho) {
    char subcadena[MAX];
    int i, total;

    total = (int)ceil((float)strlen(cadena) / ancho);

    for (i = 0; i < total; i++) {
        substring(cadena, subcadena, i * ancho, ancho);
        derecha_linea(subcadena, ancho);
    }
}
```

```c
/**************************************
 * Declaración de los struct
 **************************************/
//Productos
typedef struct
{
    char codigo[7];
    char color[20];
    char marca[20];
    char modelo[20];
```

```
    char proveedor[20];
    float costoComprado;
    float costoVendido;
    int existencia;
    int unidadesCompradas;
} tproducto;

struct node {
    tproducto product;
    struct node *next;
};
typedef struct node *Tlist;
```

```
//Proveedores
typedef struct {
    char clave[20];
    char nombre[20];
    unsigned long long telefono;
} Proveedor;
```

```
//Vendedores
typedef struct {
    int clave;
    char nombre[20];
    float salario;
} Vendedor;
```

```
//Ventas
typedef struct {
    bool esVenta;  //Venta o reembolzo
    unsigned long long numero;
    unsigned int dia;
    unsigned int mes;
    unsigned int anho;
    char clave[7];
    int cantidad;
    int costo;
    int vendedor;
} Venta;
```

```
/*****************************************
 * Productos
 *****************************************/
```

```
/************
 * Hash
 ************/

//Función hash
//Toma los primeros dos números del código que se le de
int hashFunction(string code) {
    try {
        string number = code.substr(2, 2);
        int n = atoi(number.c_str());
        return n;
    }
    catch (...) {
        cout << "Su productos.dat no corresponde con el que se maneja en
este programa" << endl;
        cout << "Borrelo o cambielo de lugar" << endl;
        escucharEspacio();
        exit(0);
    }
    return 0;
}
```

```
//Variable arreglo que contendrá todos los productos
Tlist hashTable[NUMBER_OF_SLOTS];
```

```
//inicializa el arreglo con valores nulos
void initializeHashTable() {
    for (int i = 0; i < NUMBER_OF_SLOTS; i++) {
        hashTable[i] = NULL;
    }
}
```

```
//Llena el arreglo con los datos de productos.dat
//Si no existe simplemente lo deja con los valores nulos
void fillHashTable() {
    FILE *arch;

    arch = fopen("productos.dat", "rb");

    if (arch == NULL) {
        return;
    }
```

```
    tproducto producto;

    //Posiciona cada código en su lugar con la función hash
    fread(&producto, sizeof(tproducto), 1, arch);
    while (!feof(arch)) {
        int n = hashFunction(producto.codigo);

        //Usa la lógica de las listas dinámicas
        if (hashTable[n] == NULL) {
            Tlist q = new (struct node);
            q->product = producto;
            q->next = NULL;
            hashTable[n] = q;
        }
        else {
            Tlist t, q = new (struct node);
            q->product = producto;
            q->next = NULL;
            t = hashTable[n];
            while (t->next != NULL) {
                t = t->next;
            }
            t->next = q;
        }

        fread(&producto, sizeof(tproducto), 1, arch);
    }
    fclose(arch);
    return;
}
```

```
//Imprime el arreglo
//No es usado en este programa
//Pero es bueno tenerlo por si las moscas
void printHashTable() {
    for (int i = 0; i < NUMBER_OF_SLOTS; i++) {
        if (hashTable[i] == NULL) {
            cout << i << ".-" << endl;
        }
        else {
            Tlist t = hashTable[i];
            cout << i << ".- " << t->product.codigo;
            while (t->next != NULL) {
                t = t->next;
                cout << " -> " << t->product.codigo;
```

```
            }
            cout << endl;
        }
    }
}
```

```
//Escribe los cambios de el arreglo a productos.dat
void writeFile() {
    FILE *arch;
    arch = fopen("productos.dat", "w+b");
    if (arch == NULL) {
        cout << "Archivo productos.dat no se pudo generar" << endl;
        escucharEspacio();
        exit(1);
    }

    Tlist list;
    tproducto product;

    //Recorre todo el arreglo
    for (int i = 0; i < NUMBER_OF_SLOTS; i++) {
        if (hashTable[i] == NULL) {
        }
        else {
            Tlist t = hashTable[i];
            product = t->product;
            //Solo escribe el producto ya que escribir todo el nodo
sería un caos con la memoria
            fwrite(&product, sizeof(tproducto), 1, arch);
            while (t->next != NULL) {
                t = t->next;
                product = t->product;
                fwrite(&product, sizeof(tproducto), 1, arch);
            }
        }
    }
    fclose(arch);
}
```

```
/************
 * Archivos de productos
 ************/

//Función que pide los datos de un producto y los escribe en
productos.dat
```

```
void altaProducto() {
    tproducto producto;
    bool codigoValido = 0;


////////////////////////////////////////////////////////////////////////
/////////////
    //Petición de datos

////////////////////////////////////////////////////////////////////////
/////////////

    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));

    while (!codigoValido) {
        printf("\n\n\t\tIngrese el código del producto: ");
        fflush(stdin);
        gets(producto.codigo);
        codigoValido = validarCodigo(producto.codigo);
        if (!codigoValido) {
            cout << "\n\n\t\tUsted ingreso un codigo invalido!!!" <<
endl;
            cout << "\n\t\tUn codigo correcto empieza con 2 letras y le
siguen 4 numeros.\n"
                 << endl;
        }
    }


////////////////////////////////////////////////////////////////////////
/////////////
    //Verifica que ese código no exista
    //Si no existe lo posiciona en el arreglo y lo escribe en el archivo

////////////////////////////////////////////////////////////////////////
/////////////

    int n = hashFunction(producto.codigo);
    Tlist t = new (struct node);
    t = hashTable[n];
    while (t != NULL) {
        int ret = strncmp(producto.codigo, t->product.codigo, 6);
        if (ret == 0) {
            cout << "\n\n\t\tUn producto con ese código ya existe." <<
endl;
```

```
            cout << "\n\t\tSe cancela todo.";
            return;
        }
        t = t->next;
    }

    fflush(stdin);
    cout << "\n\t\tIngrese el color del producto: ";
    cin.getline(producto.color, 20, '\n');

    fflush(stdin);
    cout << "\n\t\tIngrese la marca del producto: ";
    cin.getline(producto.marca, 20, '\n');

    fflush(stdin);
    cout << "\n\t\tIngrese el modelo del producto: ";
    cin.getline(producto.modelo, 20, '\n');

    fflush(stdin);
    cout << "\n\t\tIngrese el proveedor del producto: ";
    cin.getline(producto.proveedor, 20, '\n');


////////////////////////////////////////////////////////////////////////
/////////////
    //Verifica que exista el proveedor

////////////////////////////////////////////////////////////////////////
/////////////

    FILE *arch;
    arch = fopen("proveedores.dat", "r+b");
    if (arch == NULL) {
        cout << "\n\n\t\tArchivo proveedores.dat no encontrado." <<
endl;
        cout << "\n\t\tAgregue proveedores.\n"
             << endl;
        cout << "\t\tSe cancela todo.\n"
             << endl;
        return;
    }

    string cod = producto.proveedor;
    Proveedor proveedor;
    int existe = 0;
    fread(&proveedor, sizeof(Proveedor), 1, arch);
```

```
    while (!feof(arch)) {
        if (cod == proveedor.clave) {
            existe = 1;
            break;
        }
        fread(&proveedor, sizeof(Proveedor), 1, arch);
    }
    if (existe == 0) {
        printf("\n\n\t\tNo existe un proveedor con dicha clave.");
        cout << "\n\t\tSe cancela todo.\n"
             << endl;
        return;
    }

    fclose(arch);

    fflush(stdin);
    cout << "\n\t\tIngrese el precio al que se compró el producto: ";
    producto.costoComprado = pedirFlotante("");

    fflush(stdin);
    cout << "\n\t\tIngrese el precio al que se vende el producto: ";
    producto.costoVendido = pedirFlotante("");

    fflush(stdin);
    cout << "\n\t\tIngrese cuantas unidades se compraron: ";
    producto.existencia = pedirEntero("");

    producto.unidadesCompradas = producto.existencia;


////////////////////////////////////////////////////////////////////////
////////////
    //Posiciona el producto en el arreglo

////////////////////////////////////////////////////////////////////////
////////////

    if (hashTable[n] == NULL) {
        Tlist q = new (struct node);
        q->product = producto;
        q->next = NULL;
        hashTable[n] = q;
    }
    else {
        Tlist q = new (struct node);
```

```
        q->product = producto;
        q->next = NULL;
        t = hashTable[n];
        while (t->next != NULL) {
            t = t->next;
        }
        t->next = q;
    }
    cout << endl;

    writeFile();
}
```

```
//Función que da de baja un producto
void bajaProducto(string code) {
    //Lo busca en el arreglo y como es archivo directo se puede buscar
directamente con la función hash

    //Verifica que esa posición del arreglo no esté vacía
    int n = hashFunction(code);
    if (hashTable[n] == NULL) {
        cout << "\n\t\tNo existe el producto con ese código." << endl;
    }
    else {
        //Si no busca en los nodos hijos
        bool existe = false;
        Tlist t = hashTable[n];
        int i = 1;
        if (t->next == NULL) {
            if (t->product.codigo == code) {
                cout << "\n\t\tCódigo: " << t->product.codigo << endl;
                cout << "\n\t\tColor: " << t->product.color << endl;
                cout << "\n\t\tPrecio al que se compró: " << t-
>product.costoComprado << endl;
                cout << "\n\t\tPrecio al que se vende: " << t-
>product.costoVendido << endl;
                cout << "\n\t\tExistencia: " << t->product.existencia <<
endl;
                cout << "\n\t\tUnidades compradas: " << t-
>product.unidadesCompradas << endl;
                cout << "\n\t\tMarca: " << t->product.marca << endl;
                cout << "\n\t\tModelo: " << t->product.modelo << endl;
                cout << "\n\t\tProveedor: " << t->product.proveedor <<
endl;
                cout << endl;
```

```
                hashTable[n] = NULL;
                cout << "\n\t\tProducto eliminado." << endl;
                writeFile();
            }
            else {
                cout << "\n\t\tEl producto con ese código no existe." <<
endl;
            }
        }
        else {
            while (t != NULL) {
                if (t->next->product.codigo == code) {
                    cout << "\n\t\tCódigo: " << t->product.codigo <<
endl;
                    cout << "\n\t\tColor: " << t->product.color << endl;
                    cout << "\n\t\tPrecio al que se compró: " << t-
>product.costoComprado << endl;
                    cout << "\n\t\tPrecio al que se vende: " << t-
>product.costoVendido << endl;
                    cout << "\n\t\tExistencia: " << t-
>product.existencia << endl;
                    cout << "\n\t\tUnidades compradas: " << t-
>product.unidadesCompradas << endl;
                    cout << "\n\t\tMarca: " << t->product.marca << endl;
                    cout << "\n\t\tModelo: " << t->product.modelo <<
endl;
                    cout << "\n\t\tProveedor: " << t->product.proveedor
<< endl;
                    cout << endl;

                    existe = true;
                    t->next = t->next->next;
                    cout << "\n\t\tProducto eliminado." << endl;
                    writeFile();
                }
                t = t->next;
                i++;
            }
            if (!existe) {
                cout << "\n\t\tEl producto con ese código no existe." <<
endl;
            }
            cout << endl;
        }
    }
```

```
//Función que cambia los datos de un producto
void cambioProducto(string code) {
    //Lo busca en el arreglo, píde los datos y como es archivo directo
se puede buscar directamente con la función hash

    //Verifica que esa posición del arreglo no esté vacía
    int n = hashFunction(code);
    if (hashTable[n] == NULL) {
        cout << "\n\t\tNo existe el producto con ese código." << endl;
    }
    else {
        //Si no busca en los nodos hijos
        bool existe = false;
        Tlist t = hashTable[n];
        int i = 1;
        while (t != NULL) {
            if (t->product.codigo == code) {
                existe = true;

                //Borra lo que escribimos escuchando las teclas
                FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));

                cout << "\n\t\tCódigo: " << t->product.codigo << endl;
                cout << "\t\tColor: " << t->product.color << endl;
                cout << "\t\tPrecio al que se compró: " << t-
>product.costoComprado << endl;
                cout << "\t\tPrecio al que se vende: " << t-
>product.costoVendido << endl;
                cout << "\t\tExistencia: " << t->product.existencia <<
endl;
                cout << "\t\tUnidades compradas: " << t-
>product.unidadesCompradas << endl;
                cout << "\t\tMarca: " << t->product.marca << endl;
                cout << "\t\tModelo: " << t->product.modelo << endl;
                cout << "\t\tProveedor: " << t->product.proveedor <<
endl;
                cout << endl;

                fflush(stdin);
                cout << "\n\t\tIngrese el color del producto: ";
                cin.getline(t->product.color, 20, '\n');

                fflush(stdin);
                cout << "\n\t\tIngrese la marca del producto: ";
                cin.getline(t->product.marca, 20, '\n');
```

```
                fflush(stdin);
                cout << "\n\t\tIngrese el modelo del producto: ";
                cin.getline(t->product.modelo, 20, '\n');

                fflush(stdin);
                cout << "\n\t\tIngrese el proveedor del producto: ";
                cin.getline(t->product.proveedor, 20, '\n');

                FILE *arch;
                arch = fopen("proveedores.dat", "r+b");
                if (arch == NULL) {
                    cout << "\n\t\tArchivo proveedores.dat no
encontrado." << endl;
                    cout << "\t\tAgregue proveedores" << endl;
                    cout << "\t\tSe cancela todo." << endl;
                    return;
                }

                string cod = t->product.proveedor;
                Proveedor proveedor;
                int existe = 0;
                fread(&proveedor, sizeof(Proveedor), 1, arch);
                while (!feof(arch)) {
                    if (cod == proveedor.clave) {
                        existe = 1;
                        break;
                    }
                    fread(&proveedor, sizeof(Proveedor), 1, arch);
                }
                if (existe == 0) {
                    printf("\n\n\t\tNo existe un proveedor con dicha
clave\n");
                    cout << "\n\t\tSe cancela todo" << endl;
                    return;
                }

                fclose(arch);

                fflush(stdin);
                cout << "\n\t\tIngrese el precio al que se compró el
producto: ";
                t->product.costoComprado = pedirFlotante("");

                fflush(stdin);
                cout << "\n\t\tIngrese el precio al que se vende el
```

```
producto: ";
                t->product.costoVendido = pedirFlotante("");

                fflush(stdin);
                cout << "\n\t\tIngrese cuantas unidades se compraron: ";
                t->product.existencia = pedirEntero("");

                t->product.unidadesCompradas = t->product.existencia;

                writeFile();
            }
            t = t->next;
            i++;
        }
        if (!existe) {
            cout << "\n\t\tEl producto con ese código no existe." <<
endl;
        }
        cout << endl;
    }
}
}
```

```
//Función que busca un producto
void consultaProducto(string code) {
    //Lo busca en el arreglo y como es archivo directo se puede buscar
directamente con la función hash

    //Verifica que esa posición del arreglo no esté vacía
    int n = hashFunction(code);
    if (hashTable[n] == NULL) {
        cout << "\n\t\tNo existe el producto con ese código." << endl;
    }
    else {
        //Si no busca en los nodos hijos
        bool existe = false;
        Tlist t = hashTable[n];
        int i = 1;
        while (t != NULL) {
            if (t->product.codigo == code) {
                existe = true;
                cout << "\n\t\tEl producto esta en el indice " << n << "
de la tabla en la posicion " << i << endl;
                cout << "\t\tCódigo: " << t->product.codigo << endl;
                cout << "\t\tColor: " << t->product.color << endl;
```

```
                cout << "\t\tPrecio al que se compró: " << t-
>product.costoComprado << endl;
                cout << "\t\tPrecio al que se vende: " << t-
>product.costoVendido << endl;
                cout << "\t\tExistencia: " << t->product.existencia <<
endl;
                cout << "\t\tUnidades compradas: " << t-
>product.unidadesCompradas << endl;
                cout << "\t\tMarca: " << t->product.marca << endl;
                cout << "\t\tModelo: " << t->product.modelo << endl;
                cout << "\t\tProveedor: " << t->product.proveedor <<
endl;
                cout << endl;
            }
            t = t->next;
            i++;
        }
        if (!existe) {
            cout << "\n\t\tEl producto con ese código no existe." <<
endl;
        }
        cout << endl;
    }
}
```

```
/**************************************
 * Proveedores
 **************************************/

//Función que da de alta a un proveedor
void altaProveedor() {
    Proveedor proveedor;

    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));


////////////////////////////////////////////////////////////////////////
/////////////
    //Pide los datos

////////////////////////////////////////////////////////////////////////
/////////////

    bool codigoValido = 0;
```

```
    fflush(stdin);
    cout << "\n\n\t\tDigite la clave del proveedor: ";
    cin.getline(proveedor.clave, 20, '\n');


////////////////////////////////////////////////////////////////////////////
//////////////
    //Verifica que no halla un proveedor con esa clave

////////////////////////////////////////////////////////////////////////////
//////////////

    FILE *arch;
    Proveedor p;
    arch = fopen("proveedores.dat", "r+b");
    if (arch != NULL) {
        fread(&p, sizeof(Proveedor), 1, arch);
        while (!feof(arch)) {
            int ret = strncmp(proveedor.clave, p.clave, 20);
            if (ret == 0) {
                cout << "\n\n\t\tYa existe un proveedor con esa clave "
<< endl;
                cout << "\t\tSe cancela todo" << endl;
                return;
            }
            fread(&p, sizeof(Proveedor), 1, arch);
        }
        fclose(arch);
    }

    fflush(stdin);
    cout << "\n\t\tDigite el nombre del proveedor: ";
    cin.getline(proveedor.nombre, 20, '\n');

    fflush(stdin);
    cout << "\n\t\tDigite el número telefónico del proveedor: ";
    proveedor.telefono = pedirUnsignedLongLong("");

    *arch;
    arch = fopen("proveedores.dat", "ab");
    if (arch == NULL) {
        cout << "\n\t\tArchivo proveedores.dat no se pudo generar." <<
endl;
        return;
    }
```

```
    fwrite(&proveedor, sizeof(Proveedor), 1, arch);
    fclose(arch);
}
```

```
//Función que da de baja a un proveedor
void bajaProveedor() {
    FILE *arch;
    arch = fopen("proveedores.dat", "r+b");
    if (arch == NULL) {
        cout << "\n\n\t\tArchivo proveedores.dat no encontrado." <<
endl;
        cout << "\t\tAgregue proveedores." << endl;
        return;
    }
    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));

    printf("\n\n\t\tIngrese la clave del proveedor a consultar: ");
    fflush(stdin);
    string cod;
    getline(cin, cod);
    Proveedor proveedor;
    int existe = 0;

////////////////////////////////////////////////////////////////////////
/////////////
    //Lo busca en el archivo secuencialmente

////////////////////////////////////////////////////////////////////////
/////////////
    fread(&proveedor, sizeof(Proveedor), 1, arch);
    while (!feof(arch)) {
        if (cod == proveedor.clave) {
            cout << "\n\t\tClave: " << proveedor.clave << endl;
            cout << "\t\tNombre: " << proveedor.nombre << endl;
            cout << "\t\tTeléfono: " << proveedor.telefono << endl;

            string cadenaVacia = "";

            strcpy(proveedor.clave, cadenaVacia.c_str());
            strcpy(proveedor.nombre, cadenaVacia.c_str());
            proveedor.telefono = 0;

            int pos = ftell(arch) - sizeof(Proveedor);
```

```
            fseek(arch, pos, SEEK_SET);
            fwrite(&proveedor, sizeof(Proveedor), 1, arch);
            printf("\n\n\t\tSe borro el proveedor.\n");
            existe = 1;
            break;
        }
        fread(&proveedor, sizeof(Proveedor), 1, arch);
    }
    if (existe == 0)
        printf("\n\n\t\tNo existe un proveedor con dicha clave.\n");
    fclose(arch);
}
```

```
//Función que cambia los datos de un proveedor
void cambioProveedor() {
    FILE *arch;
    arch = fopen("proveedores.dat", "r+b");
    if (arch == NULL) {
        cout << "n\n\t\tArchivo proveedores.dat no encontrado." << endl;
        cout << "\t\tAgregue proveedores." << endl;
        return;
    }
    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));

    printf("\n\t\tIngrese la clave del proveedor a consultar: ");
    fflush(stdin);
    string cod;
    getline(cin, cod);
    Proveedor proveedor;
    int existe = 0;

////////////////////////////////////////////////////////////////////////
/////////////
    //Lo busca en el archivo secuencialmente

////////////////////////////////////////////////////////////////////////
/////////////
    fread(&proveedor, sizeof(Proveedor), 1, arch);
    while (!feof(arch)) {
        if (cod == proveedor.clave) {
            cout << "\n\t\tClave: " << proveedor.clave << endl;
            cout << "\t\tNombre: " << proveedor.nombre << endl;
            cout << "\t\tTeléfono: " << proveedor.telefono << endl;
```

```
            fflush(stdin);
            cout << "\n\t\tDigite el nombre del proveedor: ";
            cin.getline(proveedor.nombre, 20, '\n');

            fflush(stdin);
            cout << "\n\t\tDigite el número telefónico del proveedor: ";
            proveedor.telefono = pedirUnsignedLongLong("");

            int pos = ftell(arch) - sizeof(Proveedor);
            fseek(arch, pos, SEEK_SET);
            fwrite(&proveedor, sizeof(Proveedor), 1, arch);
            printf("\n\n\t\tSe modifico el proveedor.\n");
            existe = 1;
            break;
        }
        fread(&proveedor, sizeof(Proveedor), 1, arch);
    }
    if (existe == 0)
        printf("\n\n\t\tNo existe un proveedor con dicha clave.\n");
    fclose(arch);
}
```

```
//Función que consulta los datos de un proveedor
void consultaProveedor() {
    FILE *arch;
    arch = fopen("proveedores.dat", "r+b");
    if (arch == NULL) {
        cout << "\n\n\t\tArchivo proveedores.dat no encontrado." <<
endl;
        cout << "\t\tAgregue proveedores." << endl;
        return;
    }
    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));

    printf("\n\t\tIngrese la clave del proveedor a consultar: ");
    fflush(stdin);
    string cod;
    getline(cin, cod);
    Proveedor proveedor;
    int existe = 0;

////////////////////////////////////////////////////////////////////////
/////////////
    //Lo busca en el archivo secuencialmente
```

```
////////////////////////////////////////////////////////////////////////
/////////////
    fread(&proveedor, sizeof(Proveedor), 1, arch);
    while (!feof(arch)) {
        if (cod == proveedor.clave) {
            cout << "\n\t\tClave: " << proveedor.clave << endl;
            cout << "\t\tNombre: " << proveedor.nombre << endl;
            cout << "\t\tTeléfono: " << proveedor.telefono << endl;

            existe = 1;
            break;
        }
        fread(&proveedor, sizeof(Proveedor), 1, arch);
    }
    if (existe == 0)
        printf("\n\t\tNo existe un proveedor con dicha clave.\n");
    fclose(arch);
}
```

```
/**************************************
 * Vendedores
 **************************************/

//Función que da de alta a un vendedor
void altaVendedor() {
    Vendedor vendedor;


////////////////////////////////////////////////////////////////////////
/////////////
    //Pide los datos

////////////////////////////////////////////////////////////////////////
/////////////

    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));

    bool codigoValido = 0;

    fflush(stdin);
    vendedor.clave = pedirEntero("\n\n\t\tDigite la clave numérica del
vendedor: ");
```

```
////////////////////////////////////////////////////////////////////////////
/////////////
    //Verifica que no exista vendedor con esa clave

////////////////////////////////////////////////////////////////////////////
/////////////

    FILE *arch;
    arch = fopen("vendedores.dat", "r+b");
    if (arch != NULL) {
        Vendedor v;
        fread(&v, sizeof(Vendedor), 1, arch);
        while (!feof(arch)) {
            if (vendedor.clave == v.clave) {
                cout << "\n\t\tYa existe un vendedor con esa clave" <<
endl;
                cout << "\t\tSe cancela todo" << endl;
                return;
            }
            fread(&v, sizeof(Vendedor), 1, arch);
        }
        fclose(arch);
    }

    fflush(stdin);
    cout << "\n\t\tDigite el nombre del vendedor: ";
    cin.getline(vendedor.nombre, 20, '\n');

    fflush(stdin);
    vendedor.salario = pedirFlotante("\n\t\tDigite el salario del
vendedor: ");

    *arch;
    arch = fopen("vendedores.dat", "ab");
    if (arch == NULL) {
        cout << "\n\n\t\tArchivo vendedores.dat no pudo ser generado."
<< endl;
        return;
    }
    fwrite(&vendedor, sizeof(Vendedor), 1, arch);
    fclose(arch);
}
```

```
//Función que da de baja a un vendedor
```

```
void bajaVendedor() {
    FILE *arch;
    arch = fopen("vendedores.dat", "r+b");
    if (arch == NULL) {
        cout << "\n\n\t\tArchivo vendedores.dat no encontrado." << endl;
        cout << "\t\tAgregue vendedores." << endl;
        return;
    }
    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));

    Vendedor vendedor;

    fflush(stdin);
    int cod = pedirEntero("\n\t\tIngrese la clave del vendedor a
consultar: ");


////////////////////////////////////////////////////////////////////////
/////////////
    //Lo busca en el archivo secuencialmente

////////////////////////////////////////////////////////////////////////
/////////////

    int existe = 0;
    fread(&vendedor, sizeof(Vendedor), 1, arch);
    while (!feof(arch)) {
        if (cod == vendedor.clave) {
            cout << "\n\t\tClave: " << vendedor.clave << endl;
            cout << "\t\tNombre: " << vendedor.nombre << endl;
            cout << "\t\tSalario: " << vendedor.salario << endl;

            string cadenaVacia = "";

            vendedor.clave = 0;
            strcpy(vendedor.nombre, cadenaVacia.c_str());
            vendedor.salario = 0;

            int pos = ftell(arch) - sizeof(Vendedor);
            fseek(arch, pos, SEEK_SET);
            fwrite(&vendedor, sizeof(Vendedor), 1, arch);
            printf("\n\n\t\tSe borro el vendedor.\n");
            existe = 1;
            break;
        }
```

```
        fread(&vendedor, sizeof(Vendedor), 1, arch);
    }
    if (existe == 0)
        printf("\n\n\t\tNo existe un vendedor con dicha clave.\n");
    fclose(arch);
}
```

```
//Función que cambia los datos de un vendedor
void cambioVendedor() {
    FILE *arch;
    arch = fopen("vendedores.dat", "r+b");
    if (arch == NULL) {
        cout << "\n\n\t\tArchivo vendedores.dat no encontrado" << endl;
        cout << "\t\tAgregue vendedores" << endl;
        return;
    }
    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
    Vendedor vendedor;

    fflush(stdin);
    int cod = pedirEntero("\n\n\t\tIngrese la clave del vendedor a
consultar: ");


////////////////////////////////////////////////////////////////////////
/////////////
    //Lo busca en el archivo secuencialmente

////////////////////////////////////////////////////////////////////////
/////////////

    int existe = 0;
    fread(&vendedor, sizeof(Vendedor), 1, arch);
    while (!feof(arch)) {
        if (cod == vendedor.clave) {
            cout << "\n\t\tClave: " << vendedor.clave << endl;
            cout << "\t\tNombre: " << vendedor.nombre << endl;
            cout << "\t\tSalario: " << vendedor.salario << endl;

            fflush(stdin);
            cout << "\t\tDigite el nombre del vendedor: ";
            cin.getline(vendedor.nombre, 20, '\n');

            fflush(stdin);
```

```
            vendedor.salario = pedirFlotante("\n\t\tDigite el salario
del vendedor: ");

            int pos = ftell(arch) - sizeof(Vendedor);
            fseek(arch, pos, SEEK_SET);
            fwrite(&vendedor, sizeof(Vendedor), 1, arch);
            printf("\n\t\tSe modifico el vendedor.\n");
            existe = 1;
            break;
        }
        fread(&vendedor, sizeof(Vendedor), 1, arch);
    }
    if (existe == 0)
        printf("\n\t\tNo existe un vendedor con dicha clave.\n");
    fclose(arch);
}
```

```
//Función que consulta los datos de un vendedor
void consultaVendedor() {
    FILE *arch;
    arch = fopen("vendedores.dat", "r+b");
    if (arch == NULL) {
        cout << "\n\t\tArchivo vendedores.dat no encontrado" << endl;
        cout << "\t\tAgregue vendedores" << endl;
        return;
    }
    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
    Vendedor vendedor;

    fflush(stdin);
    int cod = pedirEntero("\n\n\t\tIngrese la clave del vendedor a
consultar: ");


////////////////////////////////////////////////////////////////////////
/////////////
    //Lo busca en el archivo secuencialmente

////////////////////////////////////////////////////////////////////////
/////////////

    int existe = 0;
    fread(&vendedor, sizeof(Vendedor), 1, arch);
    while (!feof(arch)) {
```

```
        if (cod == vendedor.clave) {
            cout << "\n\t\tClave: " << vendedor.clave << endl;
            cout << "\t\tNombre: " << vendedor.nombre << endl;
            cout << "\t\tSalario: " << vendedor.salario << endl;

            existe = 1;
            break;
        }
        fread(&vendedor, sizeof(Vendedor), 1, arch);
    }
    if (existe == 0)
        printf("\n\t\tNo existe un vendedor con dicha clave.\n");
    fclose(arch);
}
```

```
/****************************************
 * Ventas
 ****************************************/
//Función que da de alta una venta
void altaVenta() {
    FILE *arch;
    arch = fopen("ventas.dat", "ab");
    if (arch == NULL) {
        cout << "\n\n\t\tArchivo ventas.dat no pudo ser generado." <<
endl;
        return;
    }

    Venta venta;


////////////////////////////////////////////////////////////////////////
/////////////
    //Pide los datos

////////////////////////////////////////////////////////////////////////
/////////////

    venta.esVenta = true;

    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));

    bool codigoValido = 0;
```

```
    while (!codigoValido) {
        printf("\n\n\t\tIngrese el código del producto: ");
        fflush(stdin);
        cin.getline(venta.clave, 7, '\n');
        codigoValido = validarCodigo(venta.clave);
        if (!codigoValido) {
            cout << "\n\t\tUsted ingreso un codigo invalido!!!" << endl;
            cout << "\n\t\tUn codigo correcto empieza con 2 letras y le
siguen 4 numeros.\n"
                 << endl;
        }
    }

    string code = venta.clave;

    Tlist t;


////////////////////////////////////////////////////////////////////////
/////////////
    //Verifica que exista el código

////////////////////////////////////////////////////////////////////////
/////////////

    int n = hashFunction(code);
    if (hashTable[n] == NULL) {
        cout << "\n\t\tNo existe el producto con ese código." << endl;
        cout << "\t\tSe cancela todo." << endl;
        return;
    }
    else {
        bool existe = false;
        t = hashTable[n];
        int i = 1;
        while (t != NULL) {
            if (t->product.codigo == code) {
                existe = true;
                break;
            }
            t = t->next;
            i++;
        }
        if (!existe) {
            cout << "\n\n\t\tEl producto con ese código no existe." <<
endl;
```

```
            cout << "\t\tSe cancela todo." << endl;
            return;
        }
    }

    fflush(stdin);
    cout << "\n\n\t\tDigite el código numerico del vendedor: ";
    venta.vendedor = pedirEntero("");


////////////////////////////////////////////////////////////////////////////
/////////////
    //Verifica que el vendedor exista

////////////////////////////////////////////////////////////////////////////
/////////////

    FILE *archV;
    archV = fopen("vendedores.dat", "r+b");
    if (archV == NULL) {
        cout << "\n\n\t\tArchivo vendedores.dat no encontrado." << endl;
        cout << "\t\tAgregue vendedores." << endl;
        cout << "\t\tSe cancela todo." << endl;
        return;
    }
    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
    Vendedor vendedor;

    fflush(stdin);
    int cod = venta.vendedor;

    int existe = 0;
    fread(&vendedor, sizeof(Vendedor), 1, archV);
    while (!feof(archV)) {
        if (cod == vendedor.clave) {
            existe = 1;
            break;
        }
        fread(&vendedor, sizeof(Vendedor), 1, archV);
    }
    if (existe == 0) {
        printf("\n\n\t\tNo existe un vendedor con dicha clave.\n");
        cout << "\t\tSe cancela todo." << endl;
        return;
    }
```

```
    fclose(archV);

    bool band = true;


////////////////////////////////////////////////////////////////////////////
/////////////
    //Verifica que las cantidades sean congruentes

////////////////////////////////////////////////////////////////////////////
/////////////

    while (band) {
        fflush(stdin);
        venta.cantidad = pedirEntero("\n\n\t\tDigite la cantidad
vendida: ");

        if (t->product.existencia < venta.cantidad) {
            cout << "\n\t\tSe están tratando de comprar" <<
venta.cantidad << "y solo hay " << t->product.existencia;
            return;
        }
        else if (venta.cantidad <= 0) {
            if (venta.cantidad == 0) {
                cout << "\n\n\t\tNo puede vender 0 productos." << endl;
                return;
            }
            if (venta.cantidad < 0) {
                cout << "\n\n\t\tNo puede ingresar un número negativo."
<< endl;
                return;
            }
        }
        else {
            band = false;
            t->product.existencia -= venta.cantidad;
            writeFile();
        }
    }


////////////////////////////////////////////////////////////////////////////
/////////////
    //Genera automáticamente la fecha y el número de la compra
```

```
////////////////////////////////////////////////////////////////////////
////////////

    time_t now = time(0);

    tm *ltm = localtime(&now);

    venta.dia = ltm->tm_mday;
    venta.mes = 1 + ltm->tm_mon;
    venta.anho = 1900 + ltm->tm_year;

    string str = to_string(venta.anho - 2000) + to_string(venta.mes) +
to_string(venta.dia) + to_string(1 + ltm->tm_hour) + to_string(1 + ltm-
>tm_min) + to_string(1 + ltm->tm_sec);

    venta.numero = stoull(str);

    fwrite(&venta, sizeof(Venta), 1, arch);
    fclose(arch);
}

//Función que da de alta una venta
void bajaVenta() {
    FILE *arch;
    arch = fopen("ventas.dat", "ab");
    if (arch == NULL) {
        cout << "\n\n\t\tArchivo ventas.dat no se pudo generar" << endl;
        return;
    }

    Venta venta;


////////////////////////////////////////////////////////////////////////
////////////
    //Pide los datos

////////////////////////////////////////////////////////////////////////
////////////

    venta.esVenta = false;

    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));

    bool codigoValido = 0;
```

```
    while (!codigoValido) {
        printf("\n\n\t\tIngrese el código del producto: ");
        fflush(stdin);
        cin.getline(venta.clave, 7, '\n');
        codigoValido = validarCodigo(venta.clave);
        if (!codigoValido) {
            cout << "\n\n\t\tUsted ingreso un codigo invalido!!!" <<
endl;
            cout << "\n\t\tUn codigo correcto empieza con 2 letras y le
siguen 4 numeros.\n"
                 << endl;
        }
    }

    string code = venta.clave;

    Tlist t;


////////////////////////////////////////////////////////////////////////
/////////////
    //Verifica que exista un producto con ese código

////////////////////////////////////////////////////////////////////////
/////////////

    int n = hashFunction(code);
    if (hashTable[n] == NULL) {
        cout << "\n\n\t\tNo existe el producto con ese código." << endl;
        cout << "\t\tSe cancela todo." << endl;
        return;
    }
    else {
        bool existe = false;
        t = hashTable[n];
        int i = 1;
        while (t != NULL) {
            if (t->product.codigo == code) {
                existe = true;
                break;
            }
            t = t->next;
            i++;
        }
        if (!existe) {
```

```
            cout << "\n\t\tEl producto con ese código no existe" <<
endl;
            cout << "\t\tSe cancela todo" << endl;
            return;
        }
    }

    fflush(stdin);
    cout << "\n\n\t\tDigite el código numerico del vendedor: ";
    venta.vendedor = pedirEntero("");


////////////////////////////////////////////////////////////////////////
/////////////
    //Verifica que el vendedor exista

////////////////////////////////////////////////////////////////////////
/////////////

    FILE *archV;
    archV = fopen("vendedores.dat", "r+b");
    if (archV == NULL) {
        cout << "\n\n\t\tArchivo vendedores.dat no encontrado." << endl;
        cout << "\t\tSe cancela todo." << endl;
        return;
    }
    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
    Vendedor vendedor;

    fflush(stdin);
    int cod = venta.vendedor;

    int existe = 0;
    fread(&vendedor, sizeof(Vendedor), 1, archV);
    while (!feof(archV)) {
        if (cod == vendedor.clave) {
            existe = 1;
            break;
        }
        fread(&vendedor, sizeof(Vendedor), 1, archV);
    }
    if (existe == 0) {
        printf("\n\n\t\tNo existe un vendedor con dicha clave.\n");
        cout << "\t\tSe cancela todo." << endl;
        return;
```

```
    }

    fclose(archV);

    bool band = true;


////////////////////////////////////////////////////////////////////////
/////////////
    //Verifica que las cantidades sean congruentes

////////////////////////////////////////////////////////////////////////
/////////////

    while (band) {
        fflush(stdin);
        venta.cantidad = pedirEntero("\n\n\t\tDigite la cantidad de
productos en el reembolzo: ");

        if (venta.cantidad <= 0 || venta.cantidad > t-
>product.unidadesCompradas) {
            if (venta.cantidad == 0) {
                cout << "\n\t\tNo puede reembolzar 0 productos" << endl;
                return;
            }
            if (venta.cantidad < 0) {
                cout << "\n\t\tNo puede ingresar un número negativo" <<
endl;
                return;
            }
            if (venta.cantidad > t->product.unidadesCompradas) {
                cout << "\n\t\tEstá tratando de reembolzar más de lo que
se compró" << endl;
                return;
            }
        }
        else {
            band = false;
            t->product.existencia += venta.cantidad;
            writeFile();
        }
    }


////////////////////////////////////////////////////////////////////////
/////////////
```

```
    //Genera automáticamente la fecha y el número de venta

////////////////////////////////////////////////////////////////////////
////////////

    time_t now = time(0);

    tm *ltm = localtime(&now);

    venta.dia = ltm->tm_mday;
    venta.mes = 1 + ltm->tm_mon;
    venta.anho = 1900 + ltm->tm_year;

    string str = to_string(venta.anho - 2000) + to_string(venta.mes) +
to_string(venta.dia) + to_string(1 + ltm->tm_hour) + to_string(1 + ltm-
>tm_min) + to_string(1 + ltm->tm_sec);

    venta.numero = stoull(str);

    fwrite(&venta, sizeof(Venta), 1, arch);
    fclose(arch);
}
```

```
//Función que consulta una venta
//No está implementado y no es usado
void consultaVenta() {
    FILE *arch;
    arch = fopen("ventas.dat", "r+b");
    if (arch == NULL) {
        cout << "Archivo ventas.dat no encontrado" << endl;
        return;
    }
    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
    Vendedor vendedor;

    fflush(stdin);
    int cod = pedirEntero("Ingrese la clave del vendedor a consultar:
");

    int existe = 0;
    fread(&vendedor, sizeof(Vendedor), 1, arch);
    while (!feof(arch)) {
        if (cod == vendedor.clave) {
```

```
            cout << vendedor.clave;
            printf(": %s : ", vendedor.nombre);
            cout << vendedor.salario << endl;

            existe = 1;
            break;
        }
        fread(&vendedor, sizeof(Vendedor), 1, arch);
    }
    if (existe == 0)
        printf("No existe un vendedor con dicha clave\n");
    fclose(arch);
}
```

```
/**************************************
 * Reportes en pantalla
 **************************************/
//Función que imprime el inventario
void inventarioPantalla() {
    double totalInvertido = 0;
    double gananciaEsperada = 0;

    cout << "Inventario" << endl;
    for (int i = 0; i < (10 + (20 * 8) - 2); i++) {
        cout << "-";
    }
    cout << endl;

    cout << setw(10) << left << "Clave";
    cout << setw(20) << left << "Modelo";
    cout << setw(20) << left << "Marca";
    cout << setw(20) << left << "Color";
    cout << setw(20) << left << "Precio Venta";
    cout << setw(20) << left << "Precio Compra";
    cout << setw(20) << left << "Existencia";
    cout << setw(20) << left << "Unidades compradas";
    cout << setw(20) << left << "Proveedor";
    cout << endl;

    for (int i = 0; i < (10 + (20 * 8) - 2); i++) {
        cout << "-";
    }
    cout << endl;
```

```
////////////////////////////////////////////////////////////////////////
/////////////
    //Imprime directamente

////////////////////////////////////////////////////////////////////////
/////////////

    for (int i = 0; i < NUMBER_OF_SLOTS; i++) {
        if (hashTable[i] == NULL) {
        }
        else {
            Tlist t = hashTable[i];
            cout << setw(10) << left << t->product.codigo;
            cout << setw(20) << left << t->product.modelo;
            cout << setw(20) << left << t->product.marca;
            cout << setw(20) << left << t->product.color;
            cout << setw(20) << left << t->product.costoVendido;
            cout << setw(20) << left << t->product.costoComprado;
            cout << setw(20) << left << t->product.existencia;
            cout << setw(20) << left << t->product.unidadesCompradas;
            cout << setw(20) << left << t->product.proveedor;
            totalInvertido += t->product.costoComprado;
            gananciaEsperada += t->product.costoVendido;
            cout << endl;
            while (t->next != NULL) {
                t = t->next;
                cout << setw(10) << left << t->product.codigo;
                cout << setw(20) << left << t->product.modelo;
                cout << setw(20) << left << t->product.marca;
                cout << setw(20) << left << t->product.color;
                cout << setw(20) << left << t->product.costoVendido;
                cout << setw(20) << left << t->product.costoComprado;
                cout << setw(20) << left << t->product.existencia;
                cout << setw(20) << left << t-
>product.unidadesCompradas;
                cout << setw(20) << left << t->product.proveedor;
                totalInvertido += t->product.costoComprado;
                gananciaEsperada += t->product.costoVendido;
                cout << endl;
            }
        }
    }
    for (int i = 0; i < (10 + (20 * 8) - 2); i++) {
        cout << "-";
    }
    cout << "Total invertido: " << totalInvertido << endl;
```

```
    cout << "Ganancia total esperada: " << gananciaEsperada << endl;
    cout << "Superavit: " << gananciaEsperada - totalInvertido << endl;

    for (int i = 0; i < (10 + (20 * 8) - 2); i++) {
        cout << "-";
    }

    cout << endl;
}
```

```
//Función que imprime el reporte de ventas
void reporteDeVentasPantalla() {
    cout << "\n\n\t\tReporte de ventas\n"
         << endl;
    FILE *arch;

    float pc = 0;
    float pv = 0;

    arch = fopen("ventas.dat", "r+b");
    if (arch == NULL) {
        cout << "Archivo ventas.dat no encontrado" << endl;
        return;
    }
    for (int i = 0; i < ((10 * 5) + 15 + (20 * 2)); i++) {
        cout << "-";
    }
    cout << endl;

    cout << setw(10) << left << "Tipo";
    cout << setw(13) << left << "No.";
    cout << setw(15) << left << "Fecha";
    cout << setw(10) << left << "Clave";
    cout << setw(10) << left << "Cantidad";
    cout << setw(20) << left << "Precio Comprado";
    cout << setw(20) << left << "Precio Vendido";
    cout << setw(10) << left << "Vendedor";
    cout << endl;

    for (int i = 0; i < ((10 * 5) + 15 + (20 * 2)); i++) {
        cout << "-";
    }
    cout << endl;
```

```
////////////////////////////////////////////////////////////////////////
/////////////
    //Imprime secuencialmente

////////////////////////////////////////////////////////////////////////
/////////////

    Venta venta;
    fread(&venta, sizeof(Venta), 1, arch);
    while (!feof(arch)) {
        if (venta.esVenta) {
            cout << setw(10) << left << "Venta";
        }
        else {
            cout << setw(10) << left << "Reembolzo";
        }

        string fecha = to_string(venta.dia) + "/" + to_string(venta.mes)
+ "/" + to_string(venta.anho);

        cout << setw(13) << left << venta.numero;
        cout << setw(15) << left << fecha;
        cout << setw(10) << left << venta.clave;
        cout << setw(10) << left << venta.cantidad;


////////////////////////////////////////////////////////////////////////
/////////////
        //Busca el producto con esa clave
        //Para obtener datos como el precio

////////////////////////////////////////////////////////////////////////
/////////////

        string code = venta.clave;
        Tlist t;
        int n = hashFunction(code);
        if (hashTable[n] == NULL) {
            cout << "\n\n\t\tNo existe el producto con ese código." <<
endl;
        }
        else {
            bool existe = false;
            t = hashTable[n];
```

```
            int i = 1;
            while (t != NULL) {
                existe = true;
                if (t->product.codigo == code) {
                    break;
                }
                t = t->next;
                i++;
            }
            if (!existe) {
                cout << "\n\n\t\tEl producto con ese código no existe."
<< endl;
            }
        }
        cout << setw(20) << left << t->product.costoComprado;

        cout << setw(20) << left << t->product.costoVendido;

        if (venta.esVenta) {
            pc += (t->product.costoComprado * venta.cantidad);
            pv += (t->product.costoVendido * venta.cantidad);
        }
        else {
            pc -= (t->product.costoComprado * venta.cantidad);
            pv -= (t->product.costoVendido * venta.cantidad);
        }
        cout << setw(10) << left << venta.vendedor;
        cout << endl;

        fread(&venta, sizeof(Venta), 1, arch);
    }
    fclose(arch);

    for (int i = 0; i < ((10 * 5) + 15 + (20 * 2)); i++) {
        cout << "-";
    }

    printf("\n\n\t\tInversión: %.2f \n", pc);
    printf("\n\t\tDinero total recaudado de las ventas: %.2f \n", pv);
    printf("\n\t\tGanancia: %.2f \n", pv - pc);
    printf("\n\n");
}
```

```
//Función que imprime el reporte de vendedores
void reporteDeVendedoresPantalla() {
```

```
    cout << "\n\n\t\tReporte de vendedores" << endl;
    FILE *arch;
    arch = fopen("vendedores.dat", "r+b");
    if (arch == NULL) {
        cout << "\n\t\tArchivo vendedores.dat no encontrado.\n\n"
             << endl;
        return;
    }
    for (int i = 0; i < ((20 * 3)); i++) {
        cout << "-";
    }
    cout << endl;

    cout << setw(20) << left << "Clave";
    cout << setw(20) << left << "Nombre";
    cout << setw(20) << left << "Salario";
    cout << endl;

    for (int i = 0; i < ((20 * 3)); i++) {
        cout << "-";
    }
    cout << endl;


////////////////////////////////////////////////////////////////////////////
/////////////
    //Imprime secuencialmente

////////////////////////////////////////////////////////////////////////////
/////////////

    Vendedor vendedor;
    fread(&vendedor, sizeof(Vendedor), 1, arch);
    while (!feof(arch)) {
        cout << setw(20) << left << vendedor.clave;
        cout << setw(20) << left << vendedor.nombre;
        cout << setw(20) << left << vendedor.salario;
        cout << endl;

        fread(&vendedor, sizeof(Vendedor), 1, arch);
    }
    for (int i = 0; i < ((20 * 3)); i++) {
        cout << "-";
    }
    cout << endl;
    fclose(arch);
```

```
}
```

```
//Función que imprime el reporte de proveedores
void reporteDeProveedoresPantalla() {
    cout << "\n\n\t\tReporte de proveedores" << endl;
    FILE *arch;
    arch = fopen("proveedores.dat", "r+b");
    if (arch == NULL) {
        cout << "\n\t\tArchivo proveedores.dat no encontrado.\n\n"
             << endl;
        return;
    }
    for (int i = 0; i < ((20 * 3)); i++) {
        cout << "-";
    }
    cout << endl;

    cout << setw(20) << left << "Clave";
    cout << setw(20) << left << "Nombre";
    cout << setw(20) << left << "Teléfono";
    cout << endl;

    for (int i = 0; i < ((20 * 3)); i++) {
        cout << "-";
    }
    cout << endl;


////////////////////////////////////////////////////////////////////////
/////////////
    //Imprime secuencialmente

////////////////////////////////////////////////////////////////////////
/////////////

    Proveedor proveedor;
    fread(&proveedor, sizeof(Proveedor), 1, arch);
    while (!feof(arch)) {
        cout << setw(20) << left << proveedor.clave;
        cout << setw(20) << left << proveedor.nombre;
        cout << setw(20) << left << proveedor.telefono;
        cout << endl;

        fread(&proveedor, sizeof(Proveedor), 1, arch);
    }
```

```
    for (int i = 0; i < ((20 * 3)); i++) {
        cout << "-";
    }
    cout << endl;
    fclose(arch);
}
```

```
/*************************************
 * Reportes en archivo de texto
 *************************************/
//Función que genera un archivo con el inventario
void inventarioArchivo() {
    double totalInvertido = 0;
    double gananciaEsperada = 0;

    FILE *arch;
    arch = fopen("Inventario.txt", "w");

    fprintf(arch, "Inventario \n");
    for (int i = 0; i < (10 + (20 * 8)); i++) {
        fprintf(arch, "-");
    }
    fprintf(arch, "\n");

    fprintf(arch, "%-10s %-20s %-20s %-20s %-20s %-20s %-20s %-
20s\n", "Clave", "Modelo", "Marca", "Color", "Precio Venta", "Precio
Compra", "Existencia", "Unidades compradas", "Proveedor");
    for (int i = 0; i < (10 + (20 * 8)); i++) {
        fprintf(arch, "-");
    }
    fprintf(arch, "\n");


////////////////////////////////////////////////////////////////////////
/////////////
    //Escribe directamente

////////////////////////////////////////////////////////////////////////
/////////////

    for (int i = 0; i < NUMBER_OF_SLOTS; i++) {
        if (hashTable[i] == NULL) {
        }
        else {
            Tlist t = hashTable[i];
```

```
            fprintf(arch, "%-10s %-20s %-20s %-20.2f %-20.2f %-20d
%-20d %-20s\n", t->product.codigo, t->product.modelo, t->product.marca,
t->product.color, t->product.costoVendido, t->product.costoComprado, t-
>product.existencia, t->product.unidadesCompradas, t-
>product.proveedor);
            totalInvertido += t->product.costoComprado;
            gananciaEsperada += t->product.costoVendido;
            while (t->next != NULL) {
                t = t->next;
                fprintf(arch, "%-10s %-20s %-20s %-20s %-20.2f %-20.2f
%-20d %-20d %-20s\n", t->product.codigo, t->product.modelo, t-
>product.marca, t->product.color, t->product.costoVendido, t-
>product.costoComprado, t->product.existencia, t-
>product.unidadesCompradas, t->product.proveedor);
                totalInvertido += t->product.costoComprado;
                gananciaEsperada += t->product.costoVendido;
            }
        }
    }
    for (int i = 0; i < (10 + (20 * 8)); i++) {
        fprintf(arch, "-");
    }
    fprintf(arch, "\n");
    fprintf(arch, "Total invertido: %f\n", totalInvertido);
    fprintf(arch, "Ganancia total esperada: %f\n", gananciaEsperada);
    fprintf(arch, "Superavit: %f\n", gananciaEsperada - totalInvertido);
    for (int i = 0; i < (10 + (20 * 8)); i++) {
        fprintf(arch, "-");
    }
    fprintf(arch, "\n");

    fclose(arch);
}
```

```
//Función que genera un archivo con el reporte de ventas
void reporteDeVentasArchivo() {
    FILE *archd, *archt;

    archt = fopen("Reporte de ventas.txt", "w");
    archd = fopen("ventas.dat", "r+b");

    fprintf(archt, "Reporte de ventas\n");

    float pc = 0;
```

```
    float pv = 0;

    if (archd == NULL) {
        fprintf(archt, "\n\n\t\tArchivo ventas.dat no encontrado.\n");
        return;
    }
    for (int i = 0; i < ((10 * 5) + 15 + (20 * 2)); i++) {
        fprintf(archt, "-");
    }
    fprintf(archt, "\n");

    fprintf(archt, "%-10s %-13s %-15s %-10s %-10s %-20s %-20s %-10s\n",
"Tipo", "No.", "Fecha", "Clave", "Cantidad", "Precio Comprado", "Precio
Vendido", "Vendedor");

    for (int i = 0; i < ((10 * 5) + 15 + (20 * 2)); i++) {
        fprintf(archt, "-");
    }
    fprintf(archt, "\n");


////////////////////////////////////////////////////////////////////////
/////////////
    //Escribe secuencialmente

////////////////////////////////////////////////////////////////////////
/////////////

    Venta venta;
    fread(&venta, sizeof(Venta), 1, archd);
    while (!feof(archd)) {
        if (venta.esVenta) {
            fprintf(archt, "%-10s ", "Venta");
        }
        else {
            fprintf(archt, "%-10s ", "Reembolzo");
        }

        string fecha = to_string(venta.dia) + "/" + to_string(venta.mes)
+ "/" + to_string(venta.anho);
        int k = fecha.length();
        char fechav[k + 1];

        strcpy(fechav, fecha.c_str());

        fprintf(archt, "%-13d %-15s %-10s %-10d ", venta.numero, fechav,
```

```
venta.clave, venta.cantidad);

        string code = venta.clave;
        Tlist t;
        int n = hashFunction(code);
        if (hashTable[n] == NULL) {
            fprintf(archt, "No existe el producto con ese código\n");
        }
        else {
            bool existe = false;
            t = hashTable[n];
            int i = 1;
            while (t != NULL) {
                existe = true;
                if (t->product.codigo == code) {
                    break;
                }
                t = t->next;
                i++;
            }
            if (!existe) {
                fprintf(archt, "El producto con ese código no
existe\n");
            }
        }
        fprintf(archt, "%-20.2f %-20.2f ", t->product.costoComprado, t-
>product.costoVendido);

        if (venta.esVenta) {
            pc += (t->product.costoComprado * venta.cantidad);
            pv += (t->product.costoVendido * venta.cantidad);
        }
        else {
            pc -= (t->product.costoComprado * venta.cantidad);
            pv -= (t->product.costoVendido * venta.cantidad);
        }
        fprintf(archt, "%-10d\n", venta.vendedor);

        fread(&venta, sizeof(Venta), 1, archd);
    }

    fclose(archd);

    for (int i = 0; i < ((10 * 5) + 15 + (20 * 2)); i++) {
        fprintf(archt, "-");
    }
```

```
    fprintf(archt, "\nInversión: %.2f \n", pc);
    fprintf(archt, "Dinero total recaudado de las ventas: %.2f \n", pv);
    fprintf(archt, "Ganancia: %.2f \n", pv - pc);

    fclose(archt);
}
```

```
//Función que genera un archivo con el reporte de vendedores
void reporteDeVendedoresArchivo() {
    FILE *archd, *archt;

    archt = fopen("Reporte de vendedores.txt", "w");
    archd = fopen("vendedores.dat", "r+b");

    fprintf(archt, "Reporte de vendedores\n");

    if (archd == NULL) {
        fprintf(archt, "Archivo vendedores.dat no encontrado\n");
        return;
    }
    for (int i = 0; i < ((20 * 3)); i++) {
        fprintf(archt, "-");
    }
    fprintf(archt, "\n");

    fprintf(archt, "%-20s %-20s %-20s\n", "Clave", "Nombre", "Salario");

    for (int i = 0; i < ((20 * 3)); i++) {
        fprintf(archt, "-");
    }
    fprintf(archt, "\n");


////////////////////////////////////////////////////////////////////////
/////////////
    //Escribe secuencialmente

////////////////////////////////////////////////////////////////////////
/////////////

    Vendedor vendedor;
    fread(&vendedor, sizeof(Vendedor), 1, archd);
    while (!feof(archd)) {
        fprintf(archt, "%-20d %-20s %-20f\n", vendedor.clave,
```

```
vendedor.nombre, vendedor.salario);

        fread(&vendedor, sizeof(Vendedor), 1, archd);
    }
    for (int i = 0; i < ((20 * 3)); i++) {
        fprintf(archt, "-");
    }
    fprintf(archt, "\n");
    fclose(archd);
    fclose(archt);
}
```

```
//Función que genera un archivo con el reporte de proveedores
void reporteDeProveedoresArchivo() {
    FILE *archd, *archt;

    archt = fopen("Reporte de proveedores.txt", "w");
    archd = fopen("proveedores.dat", "r+b");

    fprintf(archt, "Reporte de proveedores\n");

    if (archd == NULL) {
        fprintf(archt, "Archivo proveedores.dat no encontrado\n");
        return;
    }
    for (int i = 0; i < ((20 * 3)); i++) {
        fprintf(archt, "-");
    }
    fprintf(archt, "\n");

    fprintf(archt, "%-20s %-20s %-20s\n", "Clave", "Nombre",
"Teléfono");

    for (int i = 0; i < ((20 * 3)); i++) {
        fprintf(archt, "-");
    }
    fprintf(archt, "\n");


////////////////////////////////////////////////////////////////////////
/////////////
    //Escribe secuencialmente

////////////////////////////////////////////////////////////////////////
/////////////
```

```
    Proveedor proveedor;
    fread(&proveedor, sizeof(Proveedor), 1, archd);
    while (!feof(archd)) {
        fprintf(archt, "%-20s %-20s %-20d\n", proveedor.clave,
proveedor.nombre, proveedor.telefono);

        fread(&proveedor, sizeof(Proveedor), 1, archd);
    }
    for (int i = 0; i < ((20 * 3)); i++) {
        fprintf(archt, "-");
    }
    fprintf(archt, "\n");
    fclose(archd);
    fclose(archt);
}
```

```
/**************************************
 * Administración
 **************************************/

//Función que crea un archivo
void crear() {
    string str;

    //Borra lo que escribimos escuchando las teclas
    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));

    cout << "\n\n\t\tDijite el nombre del archivo que quiere crear,
junto con su extensión: ";
    cin >> str;

    string comando = "type nul > " + str;

    system(comando.c_str());
}
```

```
//Función que respalda productos.dat, proveedores.dat, vendedores.dat y
ventas.dat
void respaldar() {
    FILE *arch;

    //Genera el respaldo y copia los datos
    //Verifica que no exista antes un respaldo, si existe, lo borra
    arch = fopen("productos.dat", "r+b");
```

```
    if (arch == NULL) {
    }
    else {
        FILE *arch2;
        arch2 = fopen("productos respaldo.dat", "r+b");
        if (arch2 == NULL) {
            arch2 = fopen("productos respaldo.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                tproducto producto;
                fread(&producto, sizeof(tproducto), 1, arch);
                while (!feof(arch)) {
                    fwrite(&producto, sizeof(tproducto), 1, arch2);
                    fread(&producto, sizeof(tproducto), 1, arch);
                }
                fclose(arch2);
            }
        }
        else {
            fclose(arch2);
            system("del \"productos respaldo.dat\"");

            arch2 = fopen("productos respaldo.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                tproducto producto;
                fread(&producto, sizeof(tproducto), 1, arch);
                while (!feof(arch)) {
                    fwrite(&producto, sizeof(tproducto), 1, arch2);
                    fread(&producto, sizeof(tproducto), 1, arch);
                }
                fclose(arch2);
            }
        }
    }
    fclose(arch);

    //Genera el respaldo y copia los datos
    //Verifica que no exista antes un respaldo, si existe, lo borr
    arch = fopen("proveedores.dat", "r+b");
    if (arch == NULL) {
    }
    else {
        FILE *arch2;
```

```
        arch2 = fopen("proveedores respaldo.dat", "r+b");
        if (arch2 == NULL) {
            arch2 = fopen("proveedores respaldo.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                Proveedor proveedor;
                fread(&proveedor, sizeof(Proveedor), 1, arch);
                while (!feof(arch)) {
                    fwrite(&proveedor, sizeof(Proveedor), 1, arch2);
                    fread(&proveedor, sizeof(Proveedor), 1, arch);
                }
                fclose(arch2);
            }
        }
        else {
            fclose(arch2);
            system("del \"proveedores respaldo.dat\"");

            arch2 = fopen("proveedores respaldo.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                Proveedor proveedor;
                fread(&proveedor, sizeof(Proveedor), 1, arch);
                while (!feof(arch)) {
                    fwrite(&proveedor, sizeof(Proveedor), 1, arch2);
                    fread(&proveedor, sizeof(Proveedor), 1, arch);
                }
                fclose(arch2);
            }
        }
    }
    fclose(arch);

    //Genera el respaldo y copia los datos
    //Verifica que no exista antes un respaldo, si existe, lo borr
    arch = fopen("vendedores.dat", "r+b");
    if (arch == NULL) {
    }
    else {
        FILE *arch2;
        arch2 = fopen("vendedores respaldo.dat", "r+b");
        if (arch2 == NULL) {
            arch2 = fopen("vendedores respaldo.dat", "ab");
            if (arch2 == NULL) {
```

```
            }
            else {
                Vendedor vendedor;
                fread(&vendedor, sizeof(Vendedor), 1, arch);
                while (!feof(arch)) {
                    fwrite(&vendedor, sizeof(Vendedor), 1, arch2);
                    fread(&vendedor, sizeof(Vendedor), 1, arch);
                }
                fclose(arch2);
            }
        }
        else {
            fclose(arch2);
            system("del \"vendedores respaldo.dat\"");

            arch2 = fopen("vendedores respaldo.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                Vendedor vendedor;
                fread(&vendedor, sizeof(Vendedor), 1, arch);
                while (!feof(arch)) {
                    fwrite(&vendedor, sizeof(Vendedor), 1, arch2);
                    fread(&vendedor, sizeof(Vendedor), 1, arch);
                }
                fclose(arch2);
            }
        }
    }
    fclose(arch);

    //Genera el respaldo y copia los datos
    //Verifica que no exista antes un respaldo, si existe, lo borr
    arch = fopen("ventas.dat", "r+b");
    if (arch == NULL) {
    }
    else {
        FILE *arch2;
        arch2 = fopen("ventas respaldo.dat", "r+b");
        if (arch2 == NULL) {
            arch2 = fopen("ventas respaldo.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                Venta venta;
                fread(&venta, sizeof(Venta), 1, arch);
```

```
                while (!feof(arch)) {
                    fwrite(&venta, sizeof(Venta), 1, arch2);
                    fread(&venta, sizeof(Venta), 1, arch);
                }
                fclose(arch2);
            }
        }
        else {
            fclose(arch2);
            system("del \"ventas respaldo.dat\"");

            arch2 = fopen("ventas respaldo.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                Venta venta;
                fread(&venta, sizeof(Venta), 1, arch);
                while (!feof(arch)) {
                    fwrite(&venta, sizeof(Venta), 1, arch2);
                    fread(&venta, sizeof(Venta), 1, arch);
                }
                fclose(arch2);
            }
        }
    }
    fclose(arch);
}
```

```
//Función que recupera productos.dat, proveedores.dat, vendedores.dat y
ventas.dat desde sus respaldos
//Básicamente es la función inversa del respaldar
void restaurar() {
    FILE *arch;

    //Copia los datos del respalo a otro archivo nuevo y luego lo
renombra
    //Borra el archivo original si existe
    arch = fopen("productos respaldo.dat", "r+b");
    if (arch == NULL) {
    }
    else {
        FILE *arch2;
        arch2 = fopen("productos.dat", "r+b");
        if (arch2 == NULL) {
            arch2 = fopen("productos.dat", "ab");
```

```
            if (arch2 == NULL) {
            }
            else {
                tproducto producto;
                fread(&producto, sizeof(tproducto), 1, arch);
                while (!feof(arch)) {
                    fwrite(&producto, sizeof(tproducto), 1, arch2);
                    fread(&producto, sizeof(tproducto), 1, arch);
                }
                fclose(arch2);
                //Inicializa al arreglo en valores nulos
                initializeHashTable();
                //Rellena con productos.dat
                fillHashTable();
            }
        }
        else {
            fclose(arch2);
            system("del \"productos.dat\"");

            arch2 = fopen("productos.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                tproducto producto;
                fread(&producto, sizeof(tproducto), 1, arch);
                while (!feof(arch)) {
                    fwrite(&producto, sizeof(tproducto), 1, arch2);
                    fread(&producto, sizeof(tproducto), 1, arch);
                }
                fclose(arch2);
                //Inicializa al arreglo en valores nulos
                initializeHashTable();
                //Rellena con productos.dat
                fillHashTable();
            }
        }
    }
    fclose(arch);

    //Copia los datos del respalo a otro archivo nuevo y luego lo
renombra
    //Borra el archivo original si existe
    arch = fopen("proveedores respaldo.dat", "r+b");
    if (arch == NULL) {
    }
```

```
    else {
        FILE *arch2;
        arch2 = fopen("proveedores.dat", "r+b");
        if (arch2 == NULL) {
            arch2 = fopen("proveedores.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                Proveedor proveedor;
                fread(&proveedor, sizeof(Proveedor), 1, arch);
                while (!feof(arch)) {
                    fwrite(&proveedor, sizeof(Proveedor), 1, arch2);
                    fread(&proveedor, sizeof(Proveedor), 1, arch);
                }
                fclose(arch2);
            }
        }
        else {
            fclose(arch2);
            system("del \"proveedores.dat\"");

            arch2 = fopen("proveedores.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                Proveedor proveedor;
                fread(&proveedor, sizeof(Proveedor), 1, arch);
                while (!feof(arch)) {
                    fwrite(&proveedor, sizeof(Proveedor), 1, arch2);
                    fread(&proveedor, sizeof(Proveedor), 1, arch);
                }
                fclose(arch2);
            }
        }
    }
    fclose(arch);

    //Copia los datos del respalo a otro archivo nuevo y luego lo
renombra
    //Borra el archivo original si existe
    arch = fopen("vendedores respaldo.dat", "r+b");
    if (arch == NULL) {
    }
    else {
        FILE *arch2;
        arch2 = fopen("vendedores.dat", "r+b");
```

```
        if (arch2 == NULL) {
            arch2 = fopen("vendedores.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                Vendedor vendedor;
                fread(&vendedor, sizeof(Vendedor), 1, arch);
                while (!feof(arch)) {
                    fwrite(&vendedor, sizeof(Vendedor), 1, arch2);
                    fread(&vendedor, sizeof(Vendedor), 1, arch);
                }
                fclose(arch2);
            }
        }
        else {
            fclose(arch2);
            system("del \"vendedores.dat\"");

            arch2 = fopen("vendedores.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                Vendedor vendedor;
                fread(&vendedor, sizeof(Vendedor), 1, arch);
                while (!feof(arch)) {
                    fwrite(&vendedor, sizeof(Vendedor), 1, arch2);
                    fread(&vendedor, sizeof(Vendedor), 1, arch);
                }
                fclose(arch2);
            }
        }
    }
    fclose(arch);

    //Copia los datos del respalo a otro archivo nuevo y luego lo
renombra
    //Borra el archivo original si existe
    arch = fopen("ventas respaldo.dat", "r+b");
    if (arch == NULL) {
    }
    else {
        FILE *arch2;
        arch2 = fopen("ventas.dat", "r+b");
        if (arch2 == NULL) {
            arch2 = fopen("ventas.dat", "ab");
            if (arch2 == NULL) {
```

```
            }
            else {
                Venta venta;
                fread(&venta, sizeof(Venta), 1, arch);
                while (!feof(arch)) {
                    fwrite(&venta, sizeof(Venta), 1, arch2);
                    fread(&venta, sizeof(Venta), 1, arch);
                }
                fclose(arch2);
            }
        }
        else {
            fclose(arch2);
            system("del \"ventas.dat\"");

            arch2 = fopen("ventas.dat", "ab");
            if (arch2 == NULL) {
            }
            else {
                Venta venta;
                fread(&venta, sizeof(Venta), 1, arch);
                while (!feof(arch)) {
                    fwrite(&venta, sizeof(Venta), 1, arch2);
                    fread(&venta, sizeof(Venta), 1, arch);
                }
                fclose(arch2);
            }
        }
    }
    fclose(arch);
}
```

```
//Función que elimina completamente los registros vacíos
void compactar() {
    FILE *arch;
    //Solo puede eliminar de proveedores.dat y vendedores.dat porque son
secuenciales
    //Aunque ventas.dat también es secuencial, lsa bajas no se manejan
de la misma manera
    //En ventas.dat se da un reembolzo y se tiene registro de ello, por
eso no se borra

    //Funcionamiento
    //En el archivo busca los registros con clave=""
    arch = fopen("proveedores.dat", "r+b");
```

```
    if (arch == NULL) {
        fclose(arch);
    }
    else {
        FILE *arch2;
        arch2 = fopen("proveedores temporal.dat", "ab");
        Proveedor proveedor;
        fread(&proveedor, sizeof(Proveedor), 1, arch);
        while (!feof(arch)) {
            int ret = strncmp(proveedor.clave, "", 20);
            if (ret != 0) {
                fwrite(&proveedor, sizeof(Proveedor), 1, arch2);
            }
            fread(&proveedor, sizeof(Proveedor), 1, arch);
        }
        fclose(arch);
        fclose(arch2);
        system("del proveedores.dat");
        system("ren \"proveedores temporal.dat\" proveedores.dat");
    }

    arch = fopen("vendedores.dat", "r+b");
    if (arch == NULL) {
        fclose(arch);
    }
    else {
        FILE *arch2;
        arch2 = fopen("vendedores temporal.dat", "ab");
        Vendedor vendedor;
        fread(&vendedor, sizeof(Vendedor), 1, arch);
        while (!feof(arch)) {
            if (vendedor.clave != 0) {
                fwrite(&vendedor, sizeof(Vendedor), 1, arch2);
            }
            fread(&vendedor, sizeof(Vendedor), 1, arch);
        }
        fclose(arch);
        fclose(arch2);
        system("del vendedores.dat");
        system("ren \"vendedores temporal.dat\" vendedores.dat");
    }
}
```

```
/*****************************************
 * Main
```

```
 ************************************/
//El main solo se encarga del menÃº
int main() {
    // Establecer el idioma a espaÃ±ol
    setlocale(LC_ALL, "es_ES");  // Cambiar locale - Suficiente para
mÃ¡quinas Linux
    SetConsoleCP(65001);         // Cambiar STDIN -  Para mÃ¡quinas
Windows
    SetConsoleOutputCP(65001);   // Cambiar STDOUT - Para mÃ¡quinas
Windows
    fflush(stdin);

    //Inicializa al arreglo en valores nulos
    initializeHashTable();
    //Rellena con productos.dat
    fillHashTable();

    //Variables para controlar los menÃºs y submenÃºs
    int opcion1,
        opcion2, opcion3;

    string objetivo;

    char c[] = "123";
    char intento[20];

    bool con = true;

    //Obtiene el nÃºmero de columnas y filas de la consola
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    int columns, rows;

    ////////////////////////////////////////////////////////
    //Primer mensaje
    ////////////////////////////////////////////////////////
    system("cls");
    //system("color 89");

    //Obtiene el nÃºmero de columnas y filas de la consola
    GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
    columns = csbi.srWindow.Right - csbi.srWindow.Left + 1;
    rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;

    centrar_cadena("UNIVERSIDAD AUTÃ"NOMA DEL ESTADO DE MÃ‰XICO\n",
columns);
    centrar_cadena("FACULTAD DE INGENIERÃA\n", columns);
```

```
    centrar_cadena("INGENIERÃA EN COMPUTACIÃ"N\n", columns);
    cout << endl;
    centrar_cadena("ORGANIZACIÃ"N DE ARCHIVOS\n", columns);
    cout << endl;
    centrar_cadena("PROYECTO FINAL\n", columns);
    cout << endl;

    cout << "\n\tGrupo: CO02" << endl
         << endl;
    cout << "\n\tEstudiantes: Ana Laura Contreras Peralta";
    cout << "\n\t             Alejandro GonzÃ¡lez JimÃ(c)nez";
    cout << "\n\t             Cristian Omar GutiÃ(c)rrez MillÃ¡n";
    cout << "\n\t             Daniel Sotelo Rizo";
    cout << "\n\t             Elizabeth GarcÃ-a GonzÃ¡lez";
    cout << "\n\t             Israel EnrÃ-quez Barreto";
    cout << "\n\t             MarÃ-a de JesÃºs SÃ¡nchez SuÃ¡rez";

    cout << endl
         << endl;

    cout << "     SEMESTRE 2020-A";
    derecha_cadena("JUNIO DE 2020     \n", columns - 15 /*SEMESTRE 2020-
A*/ - 5 /*Margen de 5 espacios*/);
    cout << endl;
    escucharEspacio(1);

    //Switch anidados dentro de whiles para navegar dentro del menÃº
    while (opcion1 != 8) {
        //Obtiene el nÃºmero de columnas y filas de la consola
        GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
&csbi);
        columns = csbi.srWindow.Right - csbi.srWindow.Left + 1;
        rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;

        system("cls");
        system("color 0B");
        cout << endl;
        centrar_cadena("MENÃš PRINCIPAL\n", columns);
        cout << endl;
        cout << "\t\t [1]. Accesorios\n"
             << endl;
        cout << "\t\t [2]. Proveedores\n"
             << endl;
        cout << "\t\t [3]. Vendedor\n"
             << endl;
        cout << "\t\t [4]. Ventas\n"
```

```
            << endl;
        cout << "\t\t [5]. Informes\n"
            << endl;
        cout << "\t\t [6]. AdministraciÃ³n\n"
            << endl;
        cout << "\t\t [7]. Ayuda\n"
            << endl;
        cout << "\t\t [8]. Salir\n\n"
            << endl;
        cout << "\t\tPresione su opciÃ³n: " << endl;
        opcion1 = escucharTecla(8);
        opcion2 = 0;
        switch (opcion1) {
            case 1: {
                while (opcion2 != 5) {
                    objetivo = "accesorio";

                    bool codigoValido;
                    string code;
                    char code_char[6 + 1];

                    system("cls");
                    system("color 0C");

                    //Obtiene el nÃºmero de columnas y filas de la
consola

GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
                    columns = csbi.srWindow.Right - csbi.srWindow.Left +
1;
                    rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;
                    cout << endl;
                    centrar_cadena("MENÃš DE ACCESORIOS\n", columns);

                    cout << "\t\t [1]. Agregar " + objetivo << endl;
                    cout << "\n\t\t [2]. Eliminar " + objetivo << endl;
                    cout << "\n\t\t [3]. Modificar " << endl;
                    cout << "\n\t\t [4]. Buscar " + objetivo << endl;
                    cout << "\n\t\t [5]. Regresar" << endl;
                    cout << "\n\n\t\tPresione su opciÃ³n: " << endl;
                    opcion2 = escucharTecla(5);
                    switch (opcion2) {
                        case 1: {
                            system("color 0A");
                            altaProducto();
                            cout << endl;
```

```
                            break;
                        }
                        case 2: {
                            codigoValido = 0;
                            system("color 0A");
                            //Borra lo que escribimos escuchando las
teclas

FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
                            while (!codigoValido) {
                                printf("\n\n\t\tIngrese el codigo del
producto a eliminar: ");
                                fflush(stdin);
                                getline(cin, code);
                                strcpy(code_char, code.c_str());
                                codigoValido = validarCodigo(code_char);
                                if (!codigoValido) {
                                    cout << "\n\n\t\tUsted ingreso un
codigo invalido!!!" << endl;
                                    cout << "\n\t\tUn codigo correcto
empieza con 2 letras y le siguen 4 numeros.\n"
                                         << endl;
                                }
                            }
                            bajaProducto(code);
                            break;
                        }
                        case 3: {
                            system("color 0A");
                            //Borra lo que escribimos escuchando las
teclas

FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
                            codigoValido = 0;
                            while (!codigoValido) {
                                system("color 0A");
                                printf("\n\n\t\tIngrese el codigo del
producto a modificar: ");
                                fflush(stdin);
                                getline(cin, code);
                                strcpy(code_char, code.c_str());
                                codigoValido = validarCodigo(code_char);
                                if (!codigoValido) {
                                    cout << "\n\n\t\tUsted ingreso un
codigo invalido!!!" << endl;
                                    cout << "\n\t\tUn codigo correcto
```

```
empieza con 2 letras y le siguen 4 numeros.\n"
                                        << endl;
                                }
                            }
                            cambioProducto(code);
                            break;
                        }
                        case 4: {
                            system("color 0A");
                            //Borra lo que escribimos escuchando las
teclas

FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
                            codigoValido = 0;
                            while (!codigoValido) {
                                printf("\n\n\t\tIngrese el codigo del
producto a buscar: ");
                                fflush(stdin);
                                getline(cin, code);
                                strcpy(code_char, code.c_str());
                                codigoValido = validarCodigo(code_char);
                                if (!codigoValido) {
                                    cout << "\n\n\t\tUsted ingreso un
codigo invalido!!!" << endl;
                                    cout << "\n\t\tUn codigo correcto
empieza con 2 letras y le siguen 4 numeros.\n"
                                        << endl;
                                }
                            }
                            consultaProducto(code);
                            break;
                        }
                        case 5: {
                            system("color B");
                            cout << "\n\n\n\t\tRegresando al menÃº
principal..." << endl;
                            cout << endl;
                            break;
                        }
                        default: {
                            system("color 0A");
                            cout << "\n\t\tDigite una opciÃ³n correcta."
<< endl;
                            break;
                        }
                    }
```

```
                    escucharEspacio();
                }
                break;
            }
            case 2: {
                while (opcion2 != 5) {
                    objetivo = "proveedor";
                    system("cls");
                    system("color 0C");
                    //Obtiene el nÃºmero de columnas y filas de la
consola

GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
                    columns = csbi.srWindow.Right - csbi.srWindow.Left +
1;
                    rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;
                    cout << endl;
                    centrar_cadena("MENÃš DE PROVEEDORES\n", columns);
                    cout << "\t\t [1]. Agregar " + objetivo << endl;
                    cout << "\n\t\t [2]. Eliminar " + objetivo << endl;
                    cout << "\n\t\t [3]. Modificar " << endl;
                    cout << "\n\t\t [4]. Buscar " + objetivo << endl;
                    cout << "\n\t\t [5]. Regresar" << endl;
                    cout << "\n\n\t\tPresione su opciÃ³n: " << endl;
                    opcion2 = escucharTecla(5);
                    switch (opcion2) {
                        case 1: {
                            system("color 0A");
                            altaProveedor();
                            break;
                        }
                        case 2: {
                            system("color 0A");
                            bajaProveedor();
                            break;
                        }
                        case 3: {
                            system("color 0A");
                            cambioProveedor();
                            break;
                        }
                        case 4: {
                            system("color 0A");
                            consultaProveedor();
                            break;
                        }
```

```
                        case 5: {
                            system("color 0B");
                            cout << "\n\n\n\t\tRegresando al menÃº
principal..." << endl;
                            cout << endl;
                            break;
                        }
                        default: {
                            cout << "\n\t\tDigite una opciÃ³n correcta"
<< endl;
                            break;
                        }
                    }
                    escucharEspacio();
                }
                break;
            }
            case 3: {
                while (opcion2 != 5) {
                    objetivo = "vendedor";
                    system("cls");
                    system("color 0C");
                    //Obtiene el nÃºmero de columnas y filas de la
consola

GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
                    columns = csbi.srWindow.Right - csbi.srWindow.Left +
1;
                    rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;
                    cout << endl;
                    centrar_cadena("MENÃš DE VENDEDOR\n", columns);
                    cout << "\t\t [1]. Agregar " + objetivo << endl;
                    cout << "\n\t\t [2]. Eliminar " + objetivo << endl;
                    cout << "\n\t\t [3]. Modificar " << endl;
                    cout << "\n\t\t [4]. Buscar " + objetivo << endl;
                    cout << "\n\t\t [5]. Regresar" << endl;
                    cout << "\n\n\t\tPresione su opciÃ³n: " << endl;
                    opcion2 = escucharTecla(5);
                    switch (opcion2) {
                        case 1: {
                            system("color 0A");
                            altaVendedor();
                            break;
                        }
                        case 2: {
                            system("color 0A");
```

```
                            bajaVendedor();
                            break;
                        }
                        case 3: {
                            system("color 0A");
                            cambioVendedor();
                            break;
                        }
                        case 4: {
                            system("color 0A");
                            consultaVendedor();
                            break;
                        }
                        case 5: {
                            system("color 0B");
                            cout << "\n\n\n\t\tRegresando al menÃº
principal..." << endl;
                            cout << endl;
                            break;
                        }
                        default: {
                            cout << "Digite una opciÃ³n correcta" <<
endl;
                            break;
                        }
                    }
                    escucharEspacio();
                }
                break;
            }
            case 4: {
                while (opcion2 != 3) {
                    bool codigoValido;
                    string code;
                    char code_char[6 + 1];

                    system("cls");
                    system("color 0C");
                    //Obtiene el nÃºmero de columnas y filas de la
consola

GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
                    columns = csbi.srWindow.Right - csbi.srWindow.Left +
1;
                    rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;
                    cout << endl;
```

```
                    centrar_cadena("MENÃš DE VENTAS\n", columns);
                    cout << "\t\t [1]. Venta de un producto " << endl;
                    cout << "\n\t\t [2]. Reembolso de un producto " <<
endl;
                    cout << "\n\t\t [3]. Regresar" << endl;
                    cout << "\n\n\t\tPresione su opciÃ³n: " << endl;
                    opcion2 = escucharTecla(3);
                    switch (opcion2) {
                        case 1: {
                            system("color 0A");
                            altaVenta();
                            break;
                        }
                        case 2: {
                            system("color 0A");
                            bajaVenta();
                            break;
                        }
                        case 3: {
                            system("color 0B");
                            cout << "\n\n\n\t\tRegresando al menÃº
principal..." << endl;
                            cout << endl;
                            break;
                        }
                        default: {
                            cout << "Digite una opciÃ³n correcta" <<
endl;
                            break;
                        }
                    }
                    escucharEspacio();
                }
                break;
            }
            case 5: {
                while (opcion2 != 5) {
                    system("cls");
                    system("color 0C");
                    //Obtiene el nÃºmero de columnas y filas de la
consola

GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
                    columns = csbi.srWindow.Right - csbi.srWindow.Left +
1;
                    rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;
```

```
                    cout << endl;
                    centrar_cadena("MENÃš DE INFORMES\n", columns);
                    cout << "\t\t [1]. Inventario" << endl;
                    cout << "\n\t\t [2]. Reporte de ventas" << endl;
                    cout << "\n\t\t [3]. Reporte de vendedores" << endl;
                    cout << "\n\t\t [4]. Reporte de proveedores" <<
endl;
                    cout << "\n\t\t [5]. Regresar" << endl;
                    cout << "\n\n\t\tPresione su opciÃ³n: " << endl;
                    opcion2 = escucharTecla(5);
                    opcion3 = 0;
                    switch (opcion2) {
                        case 1: {
                            while (opcion3 != 3) {
                                system("cls");
                                system("color 0A");
                                centrar_cadena("MENÃš DE
INVENTARIO\n\n", columns);
                                cout << "\t\t [1]. Reporte impreso en
pantalla" << endl;
                                cout << "\n\t\t [2]. Reporte en archivo
de texto" << endl;
                                cout << "\n\t\t [3]. Regresar" << endl;
                                cout << "\n\n\t\tPresione su opciÃ³n: "
<< endl;
                                opcion3 = escucharTecla(3);
                                switch (opcion3) {
                                    case 1: {
                                        system("color 0F");
                                        inventarioPantalla();
                                        break;
                                    }
                                    case 2: {
                                        system("color 0F");
                                        inventarioArchivo();
                                        break;
                                    }
                                    case 3: {
                                        system("color 0C");
                                        cout << "\n\n\t\tRegresando al
menÃº de informes...\n"
                                             << endl;
                                        break;
                                    }
                                    default: {
                                        cout << "\n\t\tDigite una
```

```
opciÃ³n correcta\n"
                                        << endl;
                                    break;
                                }
                            }
                            escucharEspacio();
                        }
                        break;
                    }
                    case 2: {
                        while (opcion3 != 3) {
                            system("cls");
                            system("color 0A");
                            centrar_cadena("MENÃš DE REPORTE DE
VENTAS\n\n", columns);
                            cout << "\t\t [1]. Reporte impreso en
pantalla" << endl;
                            cout << "\n\t\t [2]. Reporte en archivo
de texto" << endl;
                            cout << "\n\t\t [3]. Regresar" << endl;
                            cout << "\n\n\t\tPresione su opciÃ³n: "
<< endl;
                            opcion3 = escucharTecla(3);
                            switch (opcion3) {
                                case 1: {
                                    system("color 0F");
                                    reporteDeVentasPantalla();
                                    break;
                                }
                                case 2: {
                                    system("color 0F");
                                    reporteDeVentasArchivo();
                                    break;
                                }
                                case 3: {
                                    system("color 0C");
                                    cout << "\n\n\t\tRegresando al
menÃº de informes...\n"
                                        << endl;
                                    break;
                                }
                                default: {
                                    cout << "Digite una opciÃ³n
correcta" << endl;
                                    break;
                                }
```

```
                                }
                                escucharEspacio();
                            }
                            break;
                        }
                        case 3: {
                            while (opcion3 != 3) {
                                system("cls");
                                system("color 0A");
                                centrar_cadena("MENÃš DE
VENDEDORES\n\n", columns);
                                cout << "\t\t [1]. Reporte impreso en
pantalla" << endl;
                                cout << "\n\t\t [2]. Reporte en archivo
de texto" << endl;
                                cout << "\n\t\t [3]. Regresar" << endl;
                                cout << "\n\n\t\tPresione su opciÃ³n: "
<< endl;
                                opcion3 = escucharTecla(3);
                                switch (opcion3) {
                                    case 1: {
                                        system("color 0F");
                                        reporteDeVendedoresPantalla();
                                        break;
                                    }
                                    case 2: {
                                        system("color 0F");
                                        reporteDeVendedoresArchivo();
                                        break;
                                    }
                                    case 3: {
                                        system("color 0C");
                                        cout << "\n\n\t\tRegresando al
menÃº de informes...\n"
                                             << endl;
                                        break;
                                    }
                                    default: {
                                        cout << "\n\t\tDigite una
opciÃ³n correcta\n"
                                             << endl;
                                        break;
                                    }
                                }
                                escucharEspacio();
                            }
```

```
                            break;
                        }
                        case 4: {
                            while (opcion3 != 3) {
                                system("cls");
                                system("color 0A");
                                centrar_cadena("MENÃš DE REPORTE DE
PROVEEDORES\n\n", columns);
                                cout << "\t\t [1]. Reporte impreso en
pantalla" << endl;
                                cout << "\n\t\t [2]. Reporte en archivo
de texto" << endl;
                                cout << "\n\t\t [3]. Regresar" << endl;
                                cout << "\n\n\t\tPresione su opciÃ³n: "
<< endl;
                                opcion3 = escucharTecla(3);
                                switch (opcion3) {
                                    case 1: {
                                        system("color 0F");
                                        reporteDeProveedoresPantalla();
                                        break;
                                    }
                                    case 2: {
                                        system("color 0F");
                                        reporteDeProveedoresArchivo();
                                        break;
                                    }
                                    case 3: {
                                        system("color 0C");
                                        cout << "\n\n\t\tRegresando al
menÃº de informes...\n"
                                             << endl;
                                        break;
                                    }
                                    default: {
                                        cout << "Digite una opciÃ³n
correcta" << endl;
                                        break;
                                    }
                                }
                                escucharEspacio();
                            }
                            break;
                        }
                        case 5: {
                            system("color 0B");
```

```
                            cout << "\n\n\n\t\tRegresando al menÃº
principal..." << endl;
                            cout << endl;
                            escucharEspacio();
                            break;
                        }
                        default: {
                            cout << "Digite una opciÃ³n correcta" <<
endl;
                            escucharEspacio();
                            break;
                        }
                    }
                }
                break;
            }

            case 6: {
                //Borra lo que escribimos escuchando las teclas
                FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
                system("cls");
                system("color 0C");
                //Obtiene el nÃºmero de columnas y filas de la consola

GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
                columns = csbi.srWindow.Right - csbi.srWindow.Left + 1;
                rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;
                cout << endl;
                centrar_cadena("MENÃš DE ADMINISTRACIÃ"N\n", columns);
                bool hayCaracterIncorrecto = false;
                fflush(stdin);
                cout << "\n\t\tDigite la contraseÃ±a: ";
                fflush(stdin);
                gets(intento);
                for (int j = 0; j <= strlen(intento); j++) {
                    if (c[j] != intento[j]) {
                        hayCaracterIncorrecto = true;
                    }
                }
                if (!hayCaracterIncorrecto) {
                    while (opcion2 != 5) {
                        system("cls");
                        system("color 0A");
                        centrar_cadena("ADMINISTRACIÃ"N\n", columns);
                        cout << "\t\t [1]. Crear archivos" << endl;
                        cout << "\n\t\t [2]. Respaldar" << endl;
```

```
                        cout << "\n\t\t [3]. Restaurar" << endl;
                        cout << "\n\t\t [4]. Compactar archivos" <<
endl;
                        cout << "\n\t\t [5]. Regresar" << endl;
                        opcion2 = escucharTecla(5);
                        switch (opcion2) {
                            case 1: {
                                system("color 0F");
                                crear();
                                break;
                            }
                            case 2: {
                                system("color 0F");
                                respaldar();
                                break;
                            }
                            case 3: {
                                system("color 0F");
                                restaurar();
                                break;
                            }
                            case 4: {
                                system("color 0F");
                                compactar();
                                break;
                            }
                            case 5: {
                                system("color 0B");
                                cout << "\n\n\n\t\tRegresando al menÃº
principal..." << endl;
                                cout << endl;
                                break;
                            }
                            default: {
                                cout << "Digite una opciÃ³n correcta" <<
endl;
                                break;
                            }
                        }
                        escucharEspacio();
                    }
                }
                else {
                    system("color 0A");
                    cout << "\n\n\t\tContraseÃ±a incorrecta" << endl;
                    cout << endl;
```

```
                    system("color 0B");
                    cout << "\n\n\t\tRegresando al menÃº principal..."
<< endl;
                    cout << endl;
                    escucharEspacio();
                }
                break;
            }
            case 7: {
                system("cls");
                system("color 0A");
                centrar_cadena("AYUDA\n", columns);
                cout << "\t\t [1]. Manual de usuario" << endl;
                cout << "\n\t\t [2]. Manual tÃ(c)cnico" << endl;
                cout << "\n\t\t [3]. Regresar al menÃº principal" <<
endl;
                opcion2 = escucharTecla(3);
                switch (opcion2) {
                    case 1: {
                        ShellExecute(NULL, "open", "Manual de
Usuario.pdf",
                                     NULL, NULL, SW_SHOWNORMAL);
                        break;
                    }
                    case 2: {
                        ShellExecute(NULL, "open", "Manual tecnico.pdf",
                                     NULL, NULL, SW_SHOWNORMAL);
                        break;
                    }
                    case 3: {
                        system("color 0B");
                        cout << "\n\n\n\t\tRegresando al menÃº
principal..." << endl;
                        cout << endl;
                        break;
                    }
                    default: {
                        cout << "Digite una opciÃ³n correcta" << endl;
                        break;
                    }
                }

                break;
            }
            case 8: {
                system("color 0E");
```

```
GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
                columns = csbi.srWindow.Right - csbi.srWindow.Left + 1;
                rows = csbi.srWindow.Bottom - csbi.srWindow.Top + 1;
                centrar_cadena("Â¡Hasta luego!\n", columns);
                cout << endl;
                cout << endl;
                escucharEspacio();
                break;
            }
            default: {
                cout << "\n\t\tÂ¡Digite una opciÃ³n correcta! " << endl;
                escucharEspacio();
                break;
            }
        }
    }
    return 0;
}
```

7.Referencias biblio-hemerográficas, web, … (para aspectos teóricos citados).

Referencia 1.0:

Brian Donohue. (2014). ¿Qué Es Un Hash Y Cómo Funciona? 25/06/2020, de Kaspersky daily Sitio web: https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/#:~:text=10%20Abr%202014-,Una%20funci%C3%B3n%20criptogr%C3%A1fica%20hash%2D%20usualmente%20conocida%20como%20%E2%80%9Chash%E2%80%9D%2D,tendr%C3%A1%20siempre%20la%20misma%20longitud.

Referencia 1.1:

Unidad III Listas Enlazadas - Estructura de Datos Informática. (s. f.). Estructura de Datos Informática. Recuperado 27 de junio de 2020, de https://sites.google.com/site/estdatinfjiq/unidad-iii-listas-enlazadas

Referencia 1.2:

Colisión (hash). (2019, 31 agosto). Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Colisi%C3%B3n_(hash)

Referencia 1.3:

¿Qué Es Un Hash Y Cómo Funciona?. (2014).
Brian Donohue 25/06/2020
https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/#:~:text=10%20Abr%202014-,Una%20funci%C3%B3n%20criptogr%C3%A1fica%20hash%2D%20usualmente%20conocida%20como%20%E2%80%9Chash%E2%80%9D%2D,tendr%C3%A1%20siempre%20la%20misma%20longitud.