

Ingeniería en Tecnologías de la Información



Universidad Politécnica de San Luis Potosí | 177685
Martínez Lara Santiago de la cruz

Almacén de línea blanca

MANUAL DEL PROGRAMADOR

UNIVERSIDAD
POLITÉCNICA

DE SAN LUIS POTOSÍ

INGENIERÍA EN
TECNOLOGÍAS
DE LA INFORMACIÓN



Introducción	5
Objetivo	5
Definiciones	5
Requerimientos del Sistema	6
Estructura de Cabeceras	6
IO.h	7
File.c	7
getFileLines()	7
Argumentos:	7
IO.c	8
Input()	8
Argumentos:	8
Funciones evaluadoras	9
Str.c	10
int2str()	10
Argumentos	10
double2str()	11
Argumentos	11
Sys.h	11
Carrito.c	11
ImprimirCarrito()	11
Argumentos	11
Pedidos.c	13
getPedidosSize()	13
appendPedido()	13
Argumentos	13
loadPedidoFile()	13
Argumentos	13
savePedidoFile()	14
Argumentos	14
imprimirPedido()	14
Argumentos	14
PedidosLogica.c	15
listarPedidos()	15
Argumentos	15
registrarPedido()	17

mostrarPedidoPor()	17
Argumentos	17
registrarEntrega()	17
modificarPedido()	18
Win.h	18
Menu.c	18
noEcho()	18
echo()	18
setMenuData()	18
Argumentos	18
focusMenu()	19
Argumentos:	19
updateMenu()	19
Argumentos:	19
Table.c	19
tableSetHeaders()	19
Argumentos:	19
tableAppendRow()	20
Argumentos:	20
tablePrepareDataAlign()	20
Argumentos:	20
prepareTableData()	21
Argumentos:	21
printTable()	21
Argumentos:	21
freeTable()	21
Argumentos	21
Win.c	21
innit()	21
winprint()	21
Argumentos:	21
getcolsrows()	22
Argumentos:	22
getcols()	22
Argumentos:	22
getrows()	22
Argumentos:	22
getxy()	22

Argumentos:	22
getx()	23
Argumentos:	23
gety()	23
Argumentos:	23
printlnTheMiddle()	23
Argumentos:	23
printlnTheMiddleSize()	23
Argumentos:	23
printMessage()	24
Argumentos:	24

Introducción

El almacén es un lugar especialmente estructurado y planificado para custodiar, proteger y controlar los bienes de activo fijo o variable de la empresa, antes de ser requeridos para la administración, la producción o la venta de artículos o mercancías.

Objetivo

Que el alumno aplique los conocimientos básicos de programación estructurada para el desarrollo de una aplicación que solucione una situación real.

Definiciones

El desarrollo de sistemas computacionales en la actualidad, han ayudado a facilitar el desarrollo de tareas y llevarlo a cabo de manera más eficiente. Dentro de la programación estructurada, el uso de archivos permite gestionar información de manera ordenada y garantizando la correcta manipulación de memoria.

Recepción de mercancías. Proceso que consiste en dar entrada a las mercancías que envían los proveedores. Se comprueba que la mercancía recibida coincide con la información que figura en la nota de entrega. Se comprueba si las cantidades, la calidad o las características corresponden con el pedido.

Almacenamiento. Consiste en la ubicación de las mercancías en las zonas idóneas con el objetivo de acceder a las mismas y que estén fácilmente localizables. Conservación y mantenimiento. La mercancía está almacenada, tiene que conservarse en perfecto estado. Implica el seguimiento de las normas especiales sobre mantenimiento y cuidado de cada producto.

Gestión y control de existencia. Determinar la cantidad de cada producto que hay que almacenar, calcular la cantidad y la frecuencia con la que se solicitará cada pedido con el objetivo de disminuir al máximo los costes de almacenamiento.

Expedición de mercancías. Comienza desde que el cliente realiza el pedido, inicia el proceso con la selección de mercancía y embalaje, así como la elección del medio de transporte.

Requerimientos del Sistema

Sistema Operativo GNU / Linux, (La versión para MS Windows está en desarrollo).

Estructura de Cabeceras

Las cabeceras usadas durante todo el proyecto se encuentran en 3 cabeceras generales:

- IO.h
Usada para la entrada y salida general de la interfaz gráfica basada en texto, así como manejo de archivos, cubre todo lo que conlleva el manejo de entradas y salidas de cualquier tipo de *stream*.
- Sys.h
Colección de funciones específicas para el proyecto, aquí se almacena la lógica del proyecto así como sus estructuras utilizadas.
- Win.h
Colección de funciones para la representación de datos, únicamente para el formato de salidas.

IO.h

File.c

getFileLines()

```
PROYECTO (Workspace) - file.c

1  int getFileLines(char* filename){
2      int rows = 0;
3      FILE* file;
4      file = fopen(filename,"r+");
5      if(file == NULL){
6          return -1;
7      }
8
9      for(char temp; fscanf(file,"%c",&temp) == 1; ){
10         if(temp == '\n') rows++;
11     }
12
13     fclose(file);
14     return rows;
15 }
```

Esta función simplemente devuelve la cantidad de líneas existentes en un archivo, esta función es utilizada únicamente para obtener la cantidad de productos y pedidos existentes.

Argumentos:

- Filename:
Un puntero de tipo char, este es el nombre del archivo a leer.

Ejemplo de uso:

```
PROYECTO (Workspace) -
1 int pedidos = getPedidosSize();
```

```
PROYECTO (Workspace) -
1 int getPedidosSize(){
2     return getFileLines("Pedidos");
3 }
```

IO.c

Input()

Esta función tiene como propósito el recibir una entrada del usuario hasta que esta entrada sea correcta (la evaluación de la entrada es especificada por el programador), está limpia la pantalla, imprime el título de la ventana, imprime la indicación , lee la entrada del usuario, si esta es incorrecta, repite desde la limpieza de la ventana. Esta función devuelve -1 en caso de error (EOF / EOI).

Argumentos:

- Bg_titulo:
Puntero de tipo char, este representa el título de la ventana a imprimir.
- Titulo:
Puntero de tipo char, este representa la indicación del usuario a imprimir.
- Dest
Puntero de tipo void, este puntero será pasado a la función especificada para realizar evaluación de input.
- Funcion
Puntero hacia una función, esta función recibirá como argumento el puntero Dest. Esta función hará la conversión necesaria del puntero recibido, recibirá input del usuario y realizará su evolución, el resultado de esta función será usado para comprobar si se repetirá la input de usuario.

Ejemplo de uso:


```
PROYECTO (Workspace) - PedidosLogica.c

1  if ( input(titulo,
2      BOLD FRGB(185, 251, 192) "Nombre del Cliente",
3      nombre_de_cliente,
4      (void*) &evaluarNombreDelCliente) == -1) return 0;
```

Ejemplo de la función evaluadora.

```
PROYECTO (Workspace) - IO.c

1  int evaluarNombreDelCliente(char* Src){
2      if( evaluarText(Src, MAX_TEXT_LENGTH) == -1) return -1;
3      for(int i = 0; Src[i] != '\0' && Src[i] != '\n'; i++) if(esLetra(Src[i]) == 0) return 0;
4      return 1;
5  }
```

Funciones evaluadoras

Esta es una colección de funciones especificadas y usadas por Input() para evaluar la entrada del usuario, estas depende de la especificación del documento presentado

Ejemplo de estas:

```
PROYECTO (Workspace) - IO.c

1  int evaluarText(char* Dest, int lenght){
2      if(!fgets(Dest, lenght, stdin)){
3          return -1;
4      }
5      Dest[lenght] = '\0';
6      Dest[strcspn(Dest, "\r\n")] = 0;
7
8      return 1;
9  }
```

```
PROYECTO (Workspace) - IO.c

1  int evaluarNumeroTelefonico(char* Src){
2      if( evaluarText(Src, 12) == -1) return -1;
3      for(int i = 0; Src[i] != '\0' && Src[i] != '\n'; i++) if(esNumero(Src[i]) == 0) return 0;
4      if(strlen(Src) < 10 ) return 0;
5      return 1;
6  }
```

Str.c

int2str()

Esta función simplemente convierte tipos de datos *int* a arreglos de caracteres (*str*). Esta función no regresa nada.

Argumentos

- Src
Tipo de dato int, usado para la conversión.
- Dest

Puntero de tipo char, usado para guardar la conversión.

Ejemplos de uso:

```
PROYECTO (Workspace) -  
1 char temp[5] = {0};  
2 int2str(Almacen[i].numero, temp);
```

double2str()

Mismo funcionamiento que *int2str*, solamente que usado para valores double.

Argumentos

- Src
Tipo de dato double, usado para la conversión.
- Dest
Puntero de tipo char, usado para guardar la conversión.

Sys.h

Carrito.c

ImprimirCarrito()

Esta función sólo imprime el carrito dado como argumento.

Argumentos

- Pedido
Tipo de dato *Pedido*, usado para obtener la información a imprimir.
- X
Tipo de dato int, usado como coordenada x en pantalla para imprimir la información. Si este valor es -1, se imprimirá a mitad de pantalla.
- Y

Tipo de dato int, usado como coordenada y en pantalla para imprimir información.

Ejemplo de uso:

```
1 imprimirCarrito(Src, -1, y+2);
```

Ejemplo de salida;

```

MENU PRINCIPAL  MOSTRAR PEDIDO
Pedido: Activo
ID: 13416
NOMBRE DEL CLIENTE: Orlando
TELÉFONO: 1234567890
CORREO: hotmail

      MODELO  CANTIDAD  PRECIO UNITARIO  SUBTOTAL
      EST123      50      4599.00      229950.00
      MOD000      30        0.00         0.00
                        TOTAL      229950.00

↓ anterior pedido
enter salir
  
```

Pedidos.c

getPedidosSize()

Esta función regresa la cantidad de pedidos guardados en el archivo "Pedidos", no recibe ningún argumento.

appendPedido()

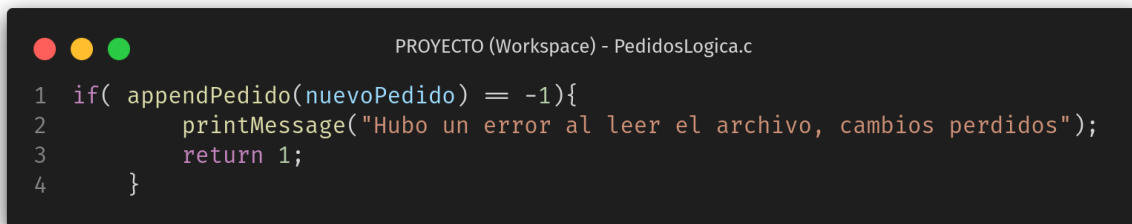
Esta función guarda la información recibida hasta el final del archivo donde los pedidos están guardados.

Argumentos

- Src

Tipo de dato Src, esta es la información que se agregara hasta el final del archivo donde los pedidos se guardan.

Ejemplo de uso:

A screenshot of a code editor window titled "PROYECTO (Workspace) - PedidosLogica.c". The code is as follows:

```
1 if( appendPedido(nuevoPedido) == -1){
2     printMessage("Hubo un error al leer el archivo, cambios perdidos");
3     return 1;
4 }
```

loadPedidoFile()

Esta función guarda todos los pedidos leídos del archivo donde se guardan los pedidos, esta función regresa -1 en caso de algún error

Argumentos

- Destination

Arreglo de datos de tipo Pedido, en este arreglo (cuyo tamaño tiene que ser previamente inicializado), se guardan todos los pedidos leídos del disco.

Ejemplo de uso:

```
PROYECTO (Workspace) - PedidosLogica.c
1  if(productos == -1 || loadAlmacenFile(Almacen) == -1){
2      printMessage("Hubo un error al leer el archivo");
3      return 1;
4  }
```

savePedidoFile()

Esta función guarda todos los datos (recibidos como argumento) en disco. Regresa -1 en caso de error.

Argumentos

- Source
Arreglo de datos de tipo Pedido que se guardaran en disco.
- Filas
Tipo de dato int, longitud del arreglo previamente recibido.

Ejemplo de uso:

```
PROYECTO (Workspace) - PedidosLogica.c
1  if(savePedidoFile(Almacen, productos) == -1){
2      printMessage("Hubo un error al guardar el archivo, el progreso se perdio");
3      return 0;
4  }
```

imprimirPedido()

Esta función imprime el pedido recibido como argumento, no devuelve nada.

Argumentos

- Src
Tipo de dato Pedido, mismo dato que se usará para imprimir la información.
- X

Tipo de dato int, usado como coordenada x para la impresión de la información.

- Y

Tipo de dato int, usado como coordenada y para la impresión de la información.

Ejemplo de salida:

The screenshot shows a terminal window titled "PROYECTO : main -- Konsole". Inside, there is a menu with "MENU PRINCIPAL" and "MOSTRAR PEDIDO" (highlighted in green). Below the menu, the order status is "Pedido: Activo" and the ID is "ID: 13416". The customer information is "NOMBRE DEL CLIENTE: Orlando", "TELÉFONO: 1234567890", and "CORREO: hotmail". A table displays the order items:

MODELO	CANTIDAD	PRECIO UNITARIO	SUBTOTAL
EST123	50	4599.00	229950.00
MOD000	30	0.00	0.00
TOTAL			229950.00

At the bottom, there are navigation instructions: "↓ anterior pedido" and "enter salir".

PedidosLogica.c

listarPedidos()

Esta función imprime los pedidos recibidos como argumento, esta impresión se hace de manera interactiva para que el usuario puede seleccionar el pedido por medio de las teclas de navegación, esta función regresa el index de el pedido seleccionado.

Argumentos

- Titulo
Puntero de tipo char, usado para imprimir el título de la ventana.
- Almacen

Arreglo de tipo de dato Pedido, usado para mostrar cada pedido dentro de este arreglo.

- Productos

Tipo de dato int. Cantidad de productos dentro del arreglo Almacén, longitud del arreglo.

Ejemplo de uso:

```
PROYECTO (Workspace) - PedidosLogica.c
1 index = listarPedidos(titulo, Pedidos, pedidos);
```

Ejemplo de salida:

```
PROYECTO : main - Konsole
MENU PRINCIPAL  ACTUALIZACIÓN DE ALMACEN  MODIFICAR PRODUCTO
EST123
Existentes: 850
EST456
Existentes: 900
EST789
Existentes: 4000
EST932
Existentes: 100
HOR550
Existentes: 100
HOR103
Existentes: 40
HOR943
Existentes: 90
HOR636
Existentes: 27
EST682
Existentes: 50
EST916
Existentes: 50
↓↑ Moverse / Borrar
← Regresar al inicio
```


registrarPedido()

Esta función no toma ni devuelve ningún valor. Esta pide el nombre del cliente, número de teléfono del cliente, correo electrónico del cliente y posteriormente de cada producto seleccionado, la cantidad de este.

mostrarPedidoPor()

Esta función muestra los pedidos leídos desde disco que sean igual al tipo seleccionado.

Argumentos

- Tipo

Un caracter cuyo valor sera buscado entre cada pedido para comprobar el estado de este, [A]ctivo, [C]ancelado, [E]ntregado.

Ejemplo de salida:

The screenshot shows a terminal window titled 'PROYECTO : main - Konsole'. It displays a menu with two options: 'MENU PRINCIPAL' and 'MOSTRAR PEDIDO'. The 'MOSTRAR PEDIDO' option is highlighted in green. Below the menu, the following information is displayed:

```
Pedido: Activo
ID: 54213
NOMBRE DEL CLIENTE: Jose
TELÉFONO: 1234567890
CORREO: gmail
```

Below this information is a table with four columns: MODELO, CANTIDAD, PRECIO UNITARIO, and SUBTOTAL. The table contains the following data:

MODELO	CANTIDAD	PRECIO UNITARIO	SUBTOTAL
EST123	1	4599.00	4599.00
EST456	2	7000.00	14000.00
EST789	3	5699.00	17097.00
EST932	4	34.00	136.00
EST932	5	34.00	170.00
HOR550	6	700.00	4200.00
HOR103	7	500.00	3500.00
TOTAL			43702.00

At the bottom of the terminal, there are instructions for navigation: '↑ siguiente pedido', '↓ anterior pedido', and 'enter salir'.

registrarEntrega()

Esta función despliega una lista de pedidos a escoger, cuyo pedido escogido es marcado como [E]ntregado y los productos almacenados son restados del total en el Almacén.

No recibe ni devuelve argumentos.

`modificarPedido()`

Esta función despliega una lista de pedidos a escoger, cuyo pedido escogido es marcado como [C]ancelado si este está [A]ctivo y viceversa, si el pedido está marcado como [E]ntregado, se le pide al usuario de nuevo un pedido.

Win.h

Menu.c

`noEcho()`

Ordena a la terminal el no escribir lo que el usuario mete como input

`echo()`

Ordena a la terminal el escribir lo que el usuario mete como input.

`setMenuData()`

Inicializa la variable pasada como argumento con la información necesaria para el correcto despliegue del menú.

Argumentos

- Destination
Puntero de tipo de dato MENU, esta variable es la que será inicializada.
- Parent
Puntero de tipo de dato WINDOW, esta variable será utilizada para heredar coordenadas-
- X
Tipo de dato int, cuyo total será sumado con la coordenada X de Parent.
- Y
Tipo de dato int, cuyo total será sumado con la coordenada Y de Parent.
- Rows
Tipo de dato int, cantidad de filas máximas a usar, esencial para saber la cantidad de opciones a usar.
- Opciones
Doble puntero de tipo de dato char, este arreglo será el título de cada opción a mostrar.
- Descripciones

Doble puntero de tipo de dato char, este arreglo será la descripción mostrada a cada opción.

focusMenu()

Esta función es un driver para la correcta lectura de la opción seleccionada al momento de desplegar el menú.

Argumentos:

- Menu

Puntero de tipo de dato MENU, que sera utilizado para obtener toda la información.

Esta función maneja las entradas del usuario, leyendo únicamente las teclas de dirección UP & DOWN, además de la tecla ENTER, siendo el evento de esta última la asignación del index actual seleccionado y fin de la función.

updateMenu()

Esta función es esencial para el mostrado del menú, puesto que esta solo sabe imprimir el menú y enfocar la opción que está actualmente enfocada.

Argumentos:

- Menu

Puntero de tipo de dato MENU, mismo que proporcionará la información para que la función sepa que imprimir.

Esta función recorre cada opción y descripción, imprimiéndolas, resaltando con un color distinto las mismas que tengan el index a enfocar.

Table.c

tableSetHeaders()

Esta función copia los headers heredados hacia la tabla destinada.

Argumentos:

- Src

Puntero de tipo TABLE, tabla destino.

- Headers

Doble puntero de tipo char, headers que serán copiados a los headers de la tabla destino.

tableAppendRow()

Función que agrega la información hacia la tabla destino.

Argumentos:

- Src
Puntero de tipo TABLE, tabla destino.
- ...
Cantidad incierta de (esperados) puntero de tipo char, este arreglo de caracteres será copiado a la columna cuyo index es el mismo al del puntero en la última fila sin rellenar.

Ejemplo de uso:

```
PROYECTO (Workspace) - Carrito.c
1  for(int fila = 0; fila < Pedido.productos; fila++){
2      Producto temp = getProductoByName(Pedido.Detalles[fila].nombre);
3      total += temp.precioUnitario * Pedido.Detalles[fila].cantidad;
4
5      precio = malloc(30);
6      sub = malloc(30);
7      cant = malloc(30);
8
9      double2str(temp.precioUnitario, precio);
10     double2str(temp.precioUnitario * (double)Pedido.Detalles[fila].cantidad, sub);
11     int2str(Pedido.Detalles[fila].cantidad, cant);
12
13     tableAppendRow(&carritoTable,
14         Pedido.Detalles[fila].nombre,
15         cant,
16         precio,
17         sub
18     );
19 }
```

tablePrepareDataAlign()

Esta función es la responsable de la alineación de la información, gracias a esta función es posible generar una tabla cuyas columnas son del mismo ancho.

Argumentos:

- Src
Un puntero de tipo TABLE, esta tabla será la que se alinee la información.

`prepareTableData()`

Esta función es la que inicializa la tabla.

Argumentos:

- Table
Puntero de tipo TABLE, esta será la tabla a trabajar.
- Columnas
Tipo de dato int, esta sera la cantidad de columnas que usará la tabla.
- Filas
Tipo de dato int, esta será la cantidad de filas que usará la tabla.

Esta función mayormente prepara la memoria necesaria para cubrir la toda la información necesitada.

`printTable()`

Esta función es la responsable de imprimir la tabla

Argumentos:

- Table
Puntero de tipo de dato TABLE, será la tabla a imprimir.
- X, Y
Tipos de datos int, serán las coordenadas a usar.

`freeTable()`

Esta función limpia la memoria utilizada para la creación de la tabla.

Argumentos

- Src
Puntero de tipo de dato TABLE, será la tabla a limpiar.

Win.c

`innit()`

Esta función crea una nueva pantalla para no sobrescribir en la actual, además de dar la posibilidad de escribir con unicode e inicializar la semilla para que `rand()` sea ciertamente aleatorio.

`winprint()`

Imprime el texto en las coordenadas seleccionadas.

Argumentos:

- Window
Puntero de tipo WINDOWS, cuyas coordenadas X e Y serán heredadas.
- X, Y
Tipos de dato int, que serán usados para imprimir en estas coordenadas.
- Text
Puntero de tipo de dato char, texto a imprimir.

getcolsrows()

Establece la cantidad de columnas y filas en las variables especificadas y de la ventana especificada.

Argumentos:

- Window
Puntero de tipo WINDOW, cuyos valores de columnas y filas serán escritos en las variables especificadas.
- Cols, Rows
Punteros de tipos de datos int, cuyos valores serán reemplazados con los de Window.

getcols()

Retorna el valor de cols.

Argumentos:

- Window
Puntero de tipo WINDOW, cuyo objeto será utilizado para retornar el valor de cols.

getrows()

Retorna el valor de rows.

Argumentos:

- Window
Puntero de tipo WINDOW, cuyo objeto será utilizado para retornar el valor de rows.

getxy()

Establece la cantidad de X e Y en las variables especificadas y de la ventana especificada.

Argumentos:

- Window
Puntero de tipo WINDOW, cuyos valores de columnas y filas serán escritos en las variables especificadas.
- X, Y
Punteros de tipos de datos int, cuyos valores serán reemplazados con los de Window.

getx()

Retorna el valor de X.

Argumentos:

- Window
Puntero de tipo WINDOW, cuyo objeto será utilizado para retornar el valor de X.

gety()

Retorna el valor de Y.

Argumentos:

- Window
Puntero de tipo WINDOW, cuyo objeto será utilizado para retornar el valor de Y.

printlnTheMiddle()

Función que imprime en medio de la terminal el texto especificado.

Argumentos:

- Window
Puntero de tipo WINDOW, cuya coordenada Y será sumada al valor especificado.
- Y
Tipo de dato int, coordenada de la altura a imprimir el texto.
- Text
Puntero de tipo de dato char, arreglo a imprimir.

printlnTheMiddleSize()

Función que imprime en la terminal el texto especificado cuyo ancho es especificado (necesario para texto que pueda contener caracteres de *0 width space*).

Argumentos:

- Window
Puntero de tipo WINDOW, cuya coordenada Y será sumada al valor especificado.
- Y
Tipo de dato int, coordenada de la altura a imprimir el texto.
- Text
Puntero de tipo de dato char, arreglo a imprimir.
- Tam
Tipo de dato int, tamaño del texto establecido.

`printMessage()`

Función que imprime en medio de la terminal el texto establecido, espera para la confirmación de lectura del usuario e imprime texto de ayuda (presiona cualquier tecla para continuar).

Argumentos:

- Msg
Puntero de tipo char, texto a imprimir.