Ingeniería en Tecnologías de la Información

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

Rutas de Transporte

MANUAL DEL PROGRAMADOR

UNIVERSIDAD POLITÉCNICA DE SAN LUIS POTOSÍ

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

⚠	**Reutilización de Código**

Debido a que se trabajó en una librería previamente utilizada, la única diferencia actual es que se *porteo* para que fuera posible trabajar ***wchar_t*** en lugar de ***char***, permitiendo trabajar con un mayor rango en cuanto a texto se refiere (e.g tildes, caracteres especiales, emojis, etc.).

Introducción

De acuerdo con las estadísticas de transporte de San Luis Potosí el promedio del tiempo que las personas pasan en transporte es de 42 minutos. Más del 43% de esos pasajeros pasan más de 2 horas en el transporte público todos los días.

Existen algunas aplicaciones que ofrecen información sobre las rutas actuales en el estado, brindando ayuda al pasajero sobre, cuál ruta tomar, tiempos de llegadas, paradas establecidas, etc

Objetivo

Lograr una administración satisfactoria de todo el sistema de rutas de transporte propuestas por el usuario.

Requerimientos del Sistema

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

Uso del Software

A continuación encontrará una guía acerca del uso del software, tenga en cuenta que la salida mostrada en las ilustraciones varia conforme el uso del mismo software,

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*.

- logic.h, core.h
  Colección de funciones específicas para el proyecto, aquí se almacena la lógica del proyecto así como sus estructuras utilizadas.
- auth.h
  Colección de funciones necesarias para el manejo de los usuarios.
- routes.h
  Colección de funciones específicas para el manejo de las rutas.
- llist.h
  Colección de funciones específicas para el manejo de las listas vinculadas.
- 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

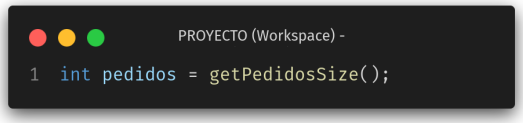
```c
int getFileLines(char* filename){
    int rows = 0;
    FILE* file;
    file = fopen(filename,"r+");
    if(file == NULL){
        return -1;
    }

    for(char temp; fscanf(file,"%c",&temp) == 1;  ){
        if(temp == '\n') rows++;
    }

    fclose(file);
    return rows;
}
```

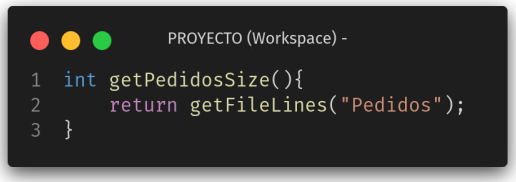
Esta función simplemente devuelve la cantidad de líneas existentes en un archivo, esta función es utilizada únicamente para obtener la cantidad de productos y pedidos existentes.

Argumentos:

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

Ejemplo de uso:





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 wchar_t, este representa el título de la ventana a imprimir.
- Titulo:
  Puntero de tipo wchar_t, 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(tituto,
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

```c
int evaluarText(char* Dest, int lenght){
    if(!fgets(Dest, lenght, stdin)){
        return -1;
    }
    Dest[lenght] = '\0';
    Dest[strcspn(Dest, "\r\n")] = 0;

    return 1;
}
```

PROYECTO (Workspace) - IO.c

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

.

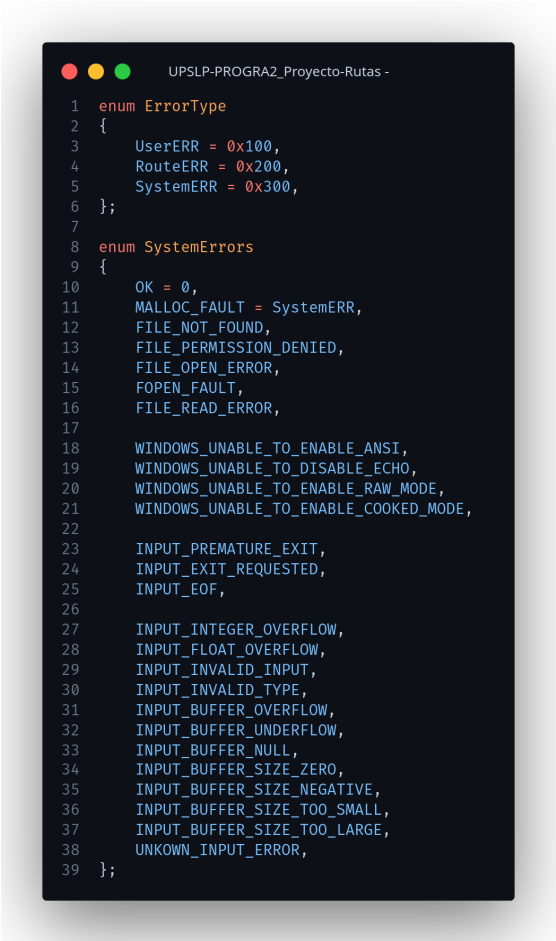
logic.h / core.h

En el archivo logic.h se define los rangos de los errores usados durante el proyecto, estos rangos son ampliamente usados entre las funciones para no solo identificar la presencia de un error, sino también del tipo de error.

Estos rangos están abiertos al programador de turno para la implementación de más errores.

⚠ Obsolencia

Se tiene planeado apegarse totalmente al estándar y usar ***errno*** como método de identificación de errores. Se espera que el sistema actual de manejo de errores cambie totalmente.



En el archivo *logic.c* se lleva a cabo toda la lógica especial del proyecto, esta se espera que cambie únicamente cuando cambien los requerimientos esenciales del proyecto.

En el archivo *core.h* simplemente se define la estructura ***HandShake*** la cual se encarga de únicamente recibir entrada *buffered* para posteriormente evaluar esa entrada por alguna función ajena.

Además de definir un tipo de resultado usado durante todo el proyecto: inspirado en el *result* propuesto por *Rust*.

UPSLP-PROGRA2_Proyecto-Rutas - core.h

```c
typedef struct _Result
{
    // If an error occurs, this will be set in the error field
    int Error_state;
    // if not, the pointer may be safe to use
    void *Result;
} Result;
```

Routes.h

En este archivo se identifican las funciones y estructuras primas con las cuales se manejan las rutas.

La definición de las rutas es la siguiente:

```c
typedef enum Weekday
{
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
} Weekday;

typedef struct
```

```
{
    int hour;
    int minute;
} ClockedTime;

typedef struct Time
{
    Weekday day;
    ClockedTime time;
} Time;

typedef enum _RouteState{
    REQUESTED,
    TAKEN,
    COMPLETED
} RouteState;

typedef struct Route
{
    wchar_t* name;
    wchar_t* destination;
    RouteState state;
    LList scheduled_times;
} Route;
```

Siendo la variable *state* utilizada únicamente cuando la ruta está referenciada en el listado de rutas del usuario, puesto que al momento de seleccionar un horario para la ruta, solo se deja esa ruta con ese horario, usando únicamente un espacio en la lista; por esa razón se puede hablar de un estado atómico, puesto que por el diseño del código en ese momento se habla únicamente de una ruta.

Ademas de declarar algunas contantes como ***ROUTE_NAME_MAX_LENGTH*** y ***ROUTE_DESCRIPTION_MAX_LENGTH*** que se tienen que usar durante todo el proyecto, puesto que hace referencia al espacio máximo posible a guardar en las entradas de texto de los nombre y descripciones de las rutas. Cualquier violación de este tamaño deja al programador con *comportamiento indefinido* dentro del programa.

Conforme a las funciones se refiere, se hace la declaración de 5 funciones esenciales:

```
Result loadAllRoutes();
```

Esta función simplemente carga todas las rutas en memoria desde el archivo interno, no es necesario un cambio. Se sugiere que la llamada de esta función se realice antes de siquiera dar la salida al usuario. Esta función tiene su contraparte:

```
void freeRoutes();
```

Libera las rutas de la memoria.

```
Result add_route(wchar_t* name, wchar_t* destination, int state);
```

Agrega la ruta cuyo nombre y descripción es especificado por el programador, el estado es el mismo mencionado anteriormente. Esta función guarda los cambios automáticamente en el archivo.

```
Result number_of_routes();
```

Regresa el número total de rutas.

```
Result query_route_by_id(int id);
```

Regresa la ruta cuyo index es especificado por el usuario.

Llist.h

Se define todo el comportamiento utilizado para el manejo de listas.

```
int llist_append(LList* list, void* data);
```

Agrega un elemento cualquiera hasta el final de la lista *list*. Regresa -1 en caso de que ocurra un error con el manejo de la memoria.

```
void* llist_remove(LList* list, int index);
```

Elimina el elemento cuyo index es proporcionado en la lista especificada.

```
int llist_insert(LList* list, int index, void* data);
```

Inserta un elemento cualquiera en la lista especificada después del index específico. Regresa -1 en caso de ocurrir un error con la memoria.

```
void* llist_get(LList* list, int index);
```

Regresa el puntero hacia la información guardada en el index especificado. Regresa *NULL* en caso de cualquier error.

```
int llist_set(LList* list, int index, void* data);
```

Cambia el puntero del elemento en el index indicado con el nuevo puntero.

```
int llist_size(LList* list);
```

Regresa el tamaño de la lista

Auth.h

Además de definirse el tamaño máximo soportado por las cadenas, se definen los rangos de errores usados por esta librería.

```
enum UserErrors
{
    USER_NOT_FOUND = UserERR,
    INCORRECT_PASSWORD,
    USER_DISABLED,
    USER_ALREADY_EXISTS,
    USER_NOT_ALLOWED,
    UNKNOWN_USER_ERROR,
```

```
} UserErrors;
```

Y la definición de la estructura del usuario:

```
typedef enum State
{
    ENABLED,
    DISABLED
} State;

typedef enum Type
{
    PASSANGER,
    ADMIN
} Type;

typedef struct
{
    wchar_t *name;
    wchar_t *pass;
    Type type;
    State state;
    LList queued_routes;
} User;
```

Al igual que las rutas, tiene una función para cargar y liberar los usuarios de la memoria.

```
Result loadAllUsers();
```

```
void freeUsers();
```

```
Result login(const wchar_t *name, const wchar_t *pass);
```

Trata de obtener el usuario solicitado segun la contraseña y el nombre de usuario, en caso de éxito se regresa un puntero hacia el usuario *wrapped* dentro de *Result.*

```
Result add_user(const User Requester, const wchar_t *NewUserName, const
wchar_t *NewUserPass, const Type NewUserType);
```

Añade un nuevo usuario a la lista de usuarios, es obligatorio llenar cada argumento, de lo contrario nos encontramos con comportamiento indefinido. En caso de que el usuario que este solicitando la acción no cuente conn los suficientes privilegios, no se llevará a cabo la acción y se notificará al usuario por medio del mensaje de error previamente definido.

```
Result modify_user(User Requester, wchar_t *UserName, wchar_t *NewUserName,
wchar_t *NewUserPass, Type NewUserType);
```

Mismo comportamiento que con `add_user()`, pero no es necesario llenar cada elemento de nuevo, es posible dejar un campo con el valor *NULL* y se interpretara como que no se desea cambiar ese campo.

```
Result remove_user(const User Requester, const wchar_t *UserName);
```

Elimina el usuario especificado por su nombre de la lista de usuarios.

```
Result query_user(const User Requester, const wchar_t *UserName);
```

Solicita un puntero hacia algún usuario por medio de su nombre.

```
Result query_user_by_id(const User Requester, const int id);
```

Solicita un puntero hacia algún usuario por medio de su index en la lista de usuarios.

```
Result number_of_users();
```

Solicita la cantidad de usuarios totales registrados en el sistema.

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 wchar_t, este arreglo será el título de cada opción a mostrar.
- Descripciones
  Doble puntero de tipo de dato wchar_t, 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, imprimiendolas, resaltando con un color distinto las mismas que tengan el index a enfocar.

Win.c

innit()

Esta función crea una nueva pantalla para no sobreescribir 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 wchar_t, 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 Xe 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.

printInTheMiddle()

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 wchar_t, arreglo a imprimir.

printInTheMiddleSize()

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 wchar_t, arreglo a imprimir.
- Tam

Tipo de dato inr, 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 wchar_t, texto a imprimir.