FUNDAMENTOS DE PROGRAMACIÓN Y LABORATORIO
PROYECTO FINAL
SISTEMA BICIRRENTA

Allan Jair Escamilla Hernández, Daniel Logvin Kirchenberg y Carlos Iturralde Punzo

Este documento presenta el desarrollo que se llevó a cabo para crear un programa que simula un sistema de bicirrenta. El programa será capaz de identificar entre usuario y administrador del sistema y, de acuerdo con su función, la persona podrá ejercer las distintas actividades.

I. INTRODUCTION

DENTRO de un sistema de renta de bicicletas, el cual será simulado en el programa que se creará en este trabajo, se permite la creación y administración de una red que consta de distintas bici-estaciones dentro de una zona de la ciudad. El programa es capaz de permitir el acceso a dos tipos de usuarios al sistema.

Al entrar como administrador del servidor, el programa permite dar de alta o baja bici-estaciones, asi como a las bicicletas dentro de una de ellas. También se permite la reasignación entre bici-estaciones y mostrar su estatus. Por último, también se puede dar de alta o baja a usuarios, si así lo requiere el administrador.

Al ingresar como usuaro regular, el programa permite la renta y devolución de biciletas, así como la consulta del saldo.

II. ANÁLISIS

Este programa, como ya se mencionó en el apartado anterior, simula un servicio de bicirrenta y permite al usuario el uso, goce y disfrute del mismo.

A. Entradas

En la línea de comandos se introducirá una forma de ejecución y, dependiendo de la manera en la que ésta se escriba, será la información que brinde. Existen dos formas para ejecutar el programa.

En primer lugar, se puede escribir la opción $ ./bicirenta.exe. Mediante este método, no se le pasa ningun argumento al programa, por lo tanto, sólo se le brindará información general al usuario.

Por otro lado, se puede escribir el mismo enunciado de ejecución, acompañado del argumento -h, -c o -usu. En estos casos, se imprimirá la información que corresponde a cada argumento, misma que se encuentra dentro de los diferentes archivos. Lo mencionado con anterioridad, será explicado a más detalle bajo el subtítulo de salidas.

Además, como entradas serán necesarios los datos de los usuarios, bicicletas y biciestaciones; tanto como para ser creadas como para ser eliminadas. De igual manera será importante considerar como entrada el nombre y passowrd del usuario para el inicio de sesión.

De suma relevancia, para este apartado, es la información que se encuentra dentro de los archivos.

B. Proceso

Mediante desenvolvimientos secuenciales, en los que cada acción lleva a otro proceso o a un menú con distintas opciones a realizar, se irán resolviendo las acciones que desee ejecutar el usario. Así mismo dependiendo de la función de la persona que ingrese al programa, ya sea usuario o administrador, será las acciones que que podrá realizar.

C. Salidas

Como se explicó en el primer subtítulo, del segundo apartado del escrito en cuestión, existen dos formas de ejecutar el programa con sus subdivisiones respectivas; por lo tanto existen el mismo numero de impresiones en la pantalla disponibles.

Bajo la primera forma de ejecución, es decir con el comando $ ./bicirenta.exe, se mostarará el nombre de los desarrolladores del sistema y, al presionar cualquier tecla, se desplegará un menú que cuente con los procesos realizables para el usuario. Dichas posibilidades fueron mencionadas en el apartado introductorio. El proceso se encontrará en un ciclo, hasta que el ususario elija abandonar la plataforma.

Así mismo, y por medio de argumentos específicos, se puede obtener más información respecto al programa. Mediante el enunciado $ ./bicirenta.exe -h, se desplegará el menú de ayuda para el usuario. Con la segundo opción, $ ./bicirenta.exe -c, se imprimirán los créditos de los desarrolladores del programa. Por último, con la opción $ ./bicirenta.exe -usu se mostará un listado de todos los usuaros del servicio.

También, mediante un archivo de texto y por medio de códigos de acción, se podrá visualizar un registro de las acciones realizadas por un usuario administrador durante su

Allan Jair Escamilla Hernández, Ingeniería en Tecnologías de Cómputo y Telecomunicaciones: 208303-4
Daniel Logvin Kirchenberg, Ingeniería en Tecnologías de Cómputo y Telecomunicaciones: 21063-6
Carlos Iturralde Punzo, Ingeniería en Mecatrónica y Producción: 210619-3.

sesión. Los comandos que se utilizaron fueron los que se mencionarán a continuación: 101 para crear una bici-estación y 102 para darla de baja, 201 para dar de alta a una bicicleta y 202 para borrarla, 301 para dar de alta a un usuario y 302 para borrarlo y, el número 203 será utilizado para la reasignación de una bicicleta.

D. Diagrama de entradas y salidas

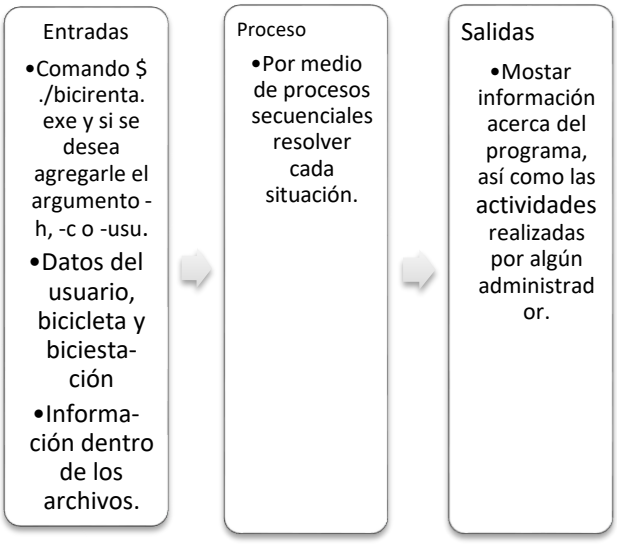
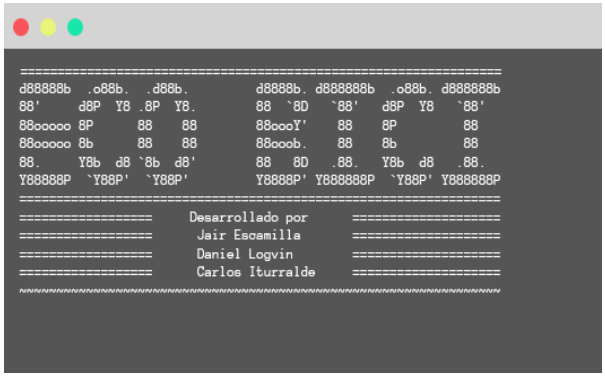


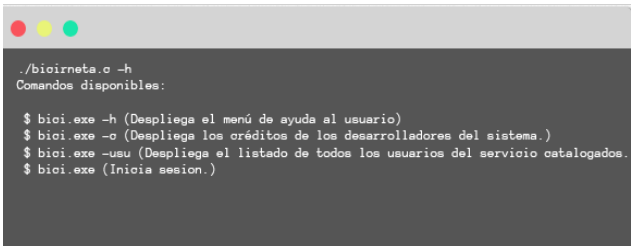
Fig. 1. Diagrama que sintetiza las entradas y salidas, así como el proceso utilizado para la creación del programa.

III. Diseño de Pantallas

Al ingresar se muestra un mensaje el cual establece que es un sistema de ecobici.



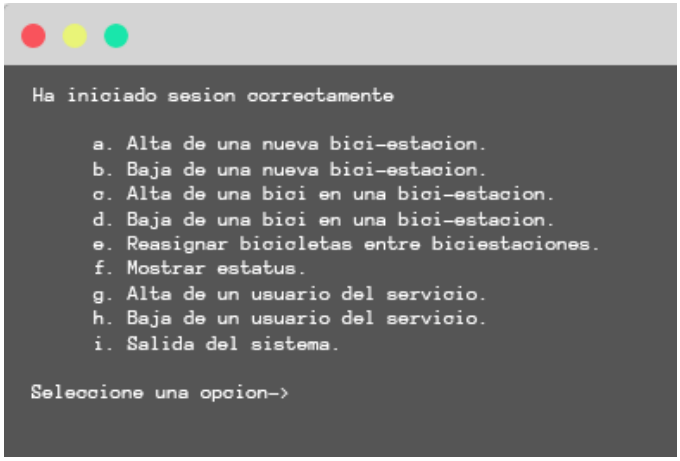
Para el menú de ayuda, se presenta la siguiente información dentro de la pantalla.



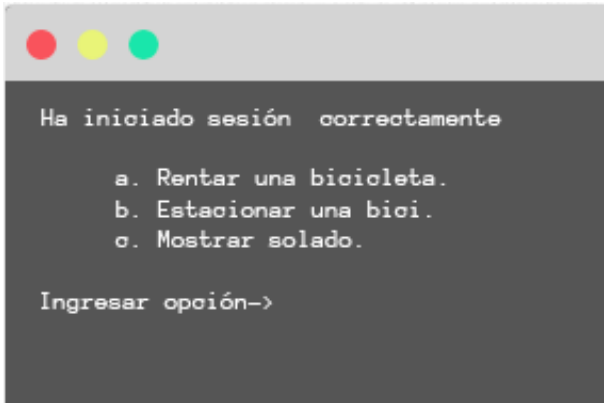
A continuación, se le pide a la persona que introduzca su nombre de usuario, así como su contraseña. Esto, por medio de las siguientes líneas:

Ingresar nombre:
Ingresar password:

Seguido de esto, si la persona que es un administrador, se le desplegará un menú con las siguiente información:



Si la persona ingresa al programa como usuario normal, se le desplegará un menú con las siguiente información:



IV. Diseño del Sistema

Diagrama de bloques

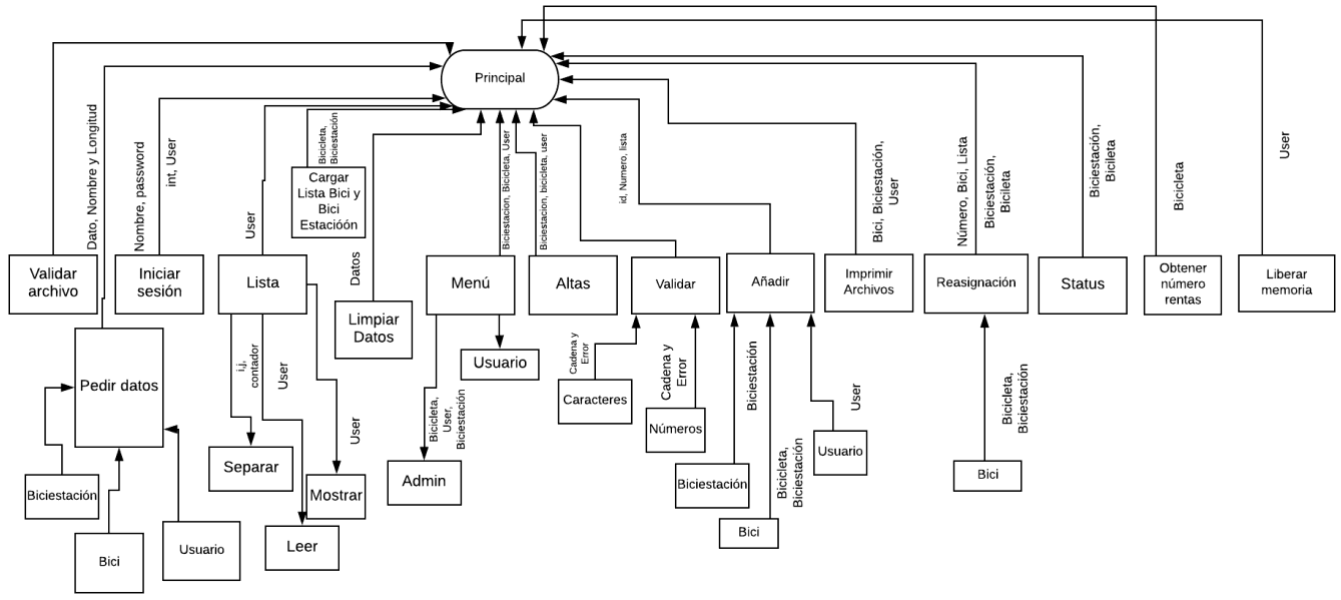


Fig. 2. Diagrama de bloques que representa las partes que fueron utilizadas para la resolución del programa.

V. Código Fuente

```
// Incluimos las librerias
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
// Definimos las estructuras
//*****************************************************************************
// Estructura de un Usuario
typedef struct defLogin{
 char Nombre[50];
 char Direccion[200];
 char Contrasenia[50];
 char TarjetaCredito[17];
 long UserNumber;
 int Flag;
 struct defLogin* siguiente;
}User;
// Estructura de una biciestacion
typedef struct defBiciestacion{
 long NumBiciestacion;
 char NombreGenerico[100];
 char Calle[50];
 int Numero;
```

```
 char CP[6];
 char Ciudad[50];
 struct defBiciestacion* siguiente;
}Biciestacion;

// Estructura de una biciestacion
typedef struct defBici{
 long NumeroBici;
 long Biciestacion;
 long rentas;
 char Timestamp[100];
 int esrentada;
 long esrentadapor;
 struct defBici* siguiente;
}Bicicleta;
//*********************************************************************************


// Prototipos de las funciones
//*********************************************************************************
void validar_archivo_login();
int Pedir_datos(char[], char[], int);
int iniciar_sesion(int*, char[], char[], User*, long*);
void leerListaUsuarios(User**);
void cargarListaBiciestacion(Biciestacion**);
void cargarListaBicis(Bicicleta**);
void separarListaUsuarios(int*, int*, int*, char[], char[6][200]);
void limpiarDatos(char[6][200]);
void MenuAdministrador(Biciestacion**, Bicicleta**, User**, long);
void MenuUsuario(Biciestacion**, Bicicleta**, User**, long);
void altaBiciestacion(Biciestacion**);
void altaBici(Bicicleta**, Biciestacion**);
void altaUsuarios(User**);
int ValidarCaracteres(char[], char[]);
int validarNumeros(char[], char[]);
void anadirBiciestacion(int, char[], char[], char[], char[], char[], Biciestacion**);
void anadirBici(int, char[], Bicicleta**, Biciestacion**);
void anadirUsuario(int, char[], char[], char[], char[], char[], User**);
void MostrarLista(User*);
void imprimirArchivos(Biciestacion**, Bicicleta**, User**);
void reasignarBicis(Bicicleta**, Biciestacion**);
void reasignar(char[], char[], Bicicleta**, Biciestacion**);
void status(Biciestacion**, Bicicleta**);
int obtenerNumerorentas(int, Bicicleta**);
void eliminarBiciestacion(Biciestacion**, Bicicleta**);
void deleteBiciestacion(Biciestacion**, Bicicleta**, char[]);
void eliminarBici(Bicicleta**);
void deleteBici(Bicicleta**, char[]);
void eliminarUsuario(User**, Bicicleta**);
void deleteUsuario(User**, Bicicleta**, char[]);
void pedirDatosRenta(long, Biciestacion**, Bicicleta**);
void rentar(Bicicleta**, char[], long);
void devolverBici(long, Biciestacion**, Bicicleta**);
```

```
void devolver(long, Bicicleta**, Biciestacion**, char[]);
void bitacora(char[], int, int, long);
void Timestamp(char[]);
int restarHoras(char[]);
void mostrarSaldo(long);
void agregarMulta(long);
void creditos();
void ayuda();
void liberarMemoria(User**, Bicicleta**, Biciestacion**);
//*****************************************************************************


// Funcion principal
//*****************************************************************************
int main(int argc, char *argv[]) {
 // Declaracion de las variables a utilizar
 int TipoUsuario = 0;
 char Nombre[50];
 char Password[50];
 long IdUsuario;
 User* ListaUsuarios = NULL;
 Biciestacion* ListaBiciestacion = NULL;
 Bicicleta* ListaBicis = NULL;
 validar_archivo_login(); // Validamos la existencia del archivo de usuarios
 // Cargamos las listas dinamicas desde los archivos
 cargarListaBiciestacion(&ListaBiciestacion);
 cargarListaBicis(&ListaBicis);
 leerListaUsuarios(&ListaUsuarios);
 if (argc != 1) { // Verificamos la cantidad de parametros pasados por terminal
   if(argc == 2){
     if(strcmp(argv[1], "-c") == 0) // En caso de obtener como parametro -c
       creditos();
     else{
       if(strcmp(argv[1], "-h") == 0) // Mostrar la ayuda
         ayuda();
       else{
         if(strcmp(argv[1], "-usu") == 0) // Despliega la lista de usuarios
           MostrarLista(ListaUsuarios);
         else{
           printf("Comando no reconocido\n"); // En caso de no encontrar ninguna
opcion
         }
       }
     }
   }else
     printf("Solo puede introducir un parametro!\n"); // En caso de introducir mas de
un parametros
 }else{ // Ejecucion sin parametros del programa
   system("clear");
   while(Pedir_datos(Nombre, "nombre", 49)); // Pedimos nombre
   system("clear");
   while(Pedir_datos(Password, "password", 49)); // Pedimos password
   if(iniciar_sesion(&TipoUsuario, Nombre, Password, ListaUsuarios, &IdUsuario)){ //
```

```
Iniciamos sesion
     if (TipoUsuario == 1) { // Inicio para administrador
       bitacora("Login", 0, 0, IdUsuario); // Generamos la bitacora
       MenuAdministrador(&ListaBiciestacion, &ListaBicis, &ListaUsuarios, IdUsuario);
     }
     if (TipoUsuario == 0) { // Inicio para usuario normal
       MenuUsuario(&ListaBiciestacion, &ListaBicis, &ListaUsuarios, IdUsuario);
     }
   }
   else
     printf("Fallo en la autenticacion\n"); // En caso de no haber encontrado ningun
usuario con los datos introducidos
 }

 liberarMemoria(&ListaUsuarios, &ListaBicis, &ListaBiciestacion); // Liberamos la
memoria
 return 0;
}
//*****************************************************************************

// Desarrollo de las funciones
//*****************************************************************************
void validar_archivo_login(){ // Validamos que exista el archivo de login
 User Usuario;
 FILE* Arch;
 FILE* Archivo;
 Arch = fopen("login.txt", "rt");
 if (Arch == NULL) { // En caso de no existir, se crea con datos predefinidos
   Archivo = fopen("login.txt", "wt");
   strcpy(Usuario.Nombre, "Ibero");
   strcpy(Usuario.Direccion, "Prolongacion Paseo de la Reforma");
   strcpy(Usuario.Contrasenia, "c123");
   strcpy(Usuario.TarjetaCredito, "1234567891156489");
   Usuario.UserNumber = 1;
   Usuario.Flag = 1;
   fprintf(Archivo, "%s/%s/%s/%s/%ld/%d/\n", Usuario.Nombre, Usuario.Direccion,
Usuario.Contrasenia, Usuario.TarjetaCredito, Usuario.UserNumber, Usuario.Flag);
   fclose(Archivo);
 }
}
int Pedir_datos(char Dato[], char NombreDato[], int longitud){ // Funcion que pide
los datos y valida la extension de los mismos
 __fpurge(stdin);
 int i = 0;
 int status = 0;
 printf("Ingresar %s: ", NombreDato);
 i = 0;
 while ((Dato[i] = getchar()) != '\n' && status == 0) {
   if (i > longitud) {
    printf("Haz sobrepasado el limite de caracteres!\n"); // En caso de haber
sobrepasado el limite
     Dato[i+1] = '\n';
     status = 1;
```

```
    }
    i++;
  }
  if(i == 0){ // En caso de no haber escrito nada
    printf("Asegurate de escribir algo!\n");
    status = 1;
  }
  Dato[i] = '\0';
  return status;
}
void liberarMemoria(User** Lista, Bicicleta** Lista2, Biciestacion** Lista3){ //
Funcion que libera la memoria al finalizar el programa
  User* Proximo;
  Bicicleta* Proximo2;
  Biciestacion* Proximo3;
  while (*Lista !=  NULL) {
    Proximo = (*Lista)->siguiente;
    free(*Lista);
    *Lista = Proximo;
  }
  while (*Lista2 !=  NULL) {
    Proximo2 = (*Lista2)->siguiente;
    free(*Lista2);
    *Lista2 = Proximo2;
  }
  while (*Lista3 !=  NULL) {
    Proximo3 = (*Lista3)->siguiente;
    free(*Lista3);
    *Lista3 = Proximo3;
  }
}
void MostrarLista(User* Lista){ // Mostramos la lista de usuarios
  system("clear");
  User* aux = Lista;
  printf("\t\tListado de todos los usuarios catalogados en el servicio\n\n");
  while (aux != NULL) {
    printf("Numero de usuario: %ld\n", aux->UserNumber);
    printf("\tNombre del usuario: %s\n", aux->Nombre);
    printf("\tDireccion del usuario: %s\n", aux->Direccion);
    printf("\tNumero de tarjeta de credito: %s\n", aux->TarjetaCredito);
    printf("\tTipo de cuenta: %d\n", aux->Flag);
    printf("\n\n");
    aux = aux->siguiente;
  }
}
int iniciar_sesion(int* TipoUsuario, char Nombre[], char Password[], User* Lista,
long* IdUsuario){ // Funcionque inicia sesion
  User* aux = Lista;
  int inicio = 0;
  do{
    if ((strcmp(Nombre, aux->Nombre) == 0) && (strcmp(Password, aux->Contrasenia) ==
0)) { // En caso de encontrar el usuario, inicia sesion
      inicio = 1;
```

```
      *TipoUsuario = aux->Flag;
      *IdUsuario = aux->UserNumber;
    }
    aux = aux->siguiente;
  }while ((aux != NULL));
  return inicio;
}
void MenuAdministrador(Biciestacion** ListaBiciestaciones, Bicicleta** ListaBicis,
User** ListaUsuarios, long User) { // Menu para el administrador
  system("clear");
  char Opcion;
  printf("Ha iniciado sesion correctamente\n\n");
  printf("\ta. Alta de una nueva bici-estacion.\n");
  printf("\tb. Baja de una nueva bici-estacion.\n");
  printf("\tc. Alta de una bici en una bici-estacion.\n");
  printf("\td. Baja de una bici en una bici-estacion.\n");
  printf("\te. Reasignar bicicletas entre biciestaciones.\n");
  printf("\tf. Mostrar estatus.\n");
  printf("\tg. Alta de un usuario del servicio.\n");
  printf("\th. Baja de un usuario del servicio.\n");
  printf("\ti. Salida del sistema.\n\n");
  printf("Seleccione una opcion-> ");
  scanf("%c", &Opcion);
  switch (Opcion) {
    case 'a':
      altaBiciestacion(ListaBiciestaciones);
      break;
    case 'b':
      eliminarBiciestacion(ListaBiciestaciones, ListaBicis);
      break;
    case 'c':
      altaBici(ListaBicis, ListaBiciestaciones);
      break;
    case 'd':
      eliminarBici(ListaBicis);
      break;
    case 'e':
      reasignarBicis(ListaBicis, ListaBiciestaciones);
      break;
    case 'f':
      status(ListaBiciestaciones, ListaBicis);
      break;
    case 'g':
      altaUsuarios(ListaUsuarios);
      break;
    case 'h':
      eliminarUsuario(ListaUsuarios, ListaBicis);
      break;
    case 'i':
      printf("Hasta pronto\nVuelva pronto\n");
      bitacora("Logout", 0, 0, User);
      imprimirArchivos(ListaBiciestaciones, ListaBicis, ListaUsuarios);
      exit(0);
```

```
      break;
    default:
      printf("Opcion incorrecta\nSelecciona una opcion correcta.\n");
      break;
  }
  printf("Presiona enter para volver al menu...");
  __fpurge(stdin);
  getchar();
  MenuAdministrador(ListaBiciestaciones, ListaBicis, ListaUsuarios, User);
}
void MenuUsuario(Biciestacion** ListaBiciestaciones, Bicicleta** ListaBicis, User**
ListaUsuarios, long Usuario){ // Menu para el usuario
  system("clear");
  char opcion;
  printf("Ha iniciado sesion correctamente\n\n");
  printf("\ta. Rentar una bicicleta.\n");
  printf("\tb. Estacionar una bicicleta.\n");
  printf("\tc. Mostrar saldo.\n");
  printf("\td. Salir del sistema.\n\n");
  printf("Ingresar una opcion: ");
  scanf("%c", &opcion);
  switch (opcion) {
    case 'a':
      pedirDatosRenta(Usuario, ListaBiciestaciones, ListaBicis);
      break;
    case 'b':
      devolverBici(Usuario, ListaBiciestaciones, ListaBicis);
      break;
    case 'c':
      mostrarSaldo(Usuario);
      break;
    case 'd':
      printf("Hasta pronto\nVuelva pronto\n");
      imprimirArchivos(ListaBiciestaciones, ListaBicis, ListaUsuarios);
      exit(0);
      break;
    default:
      printf("Opcion incorrecta\nSelecciona una opcion correcta.\n");
      break;
  }
  printf("Presiona enter para volver al menu...");
  __fpurge(stdin);
  getchar();
  MenuUsuario(ListaBiciestaciones, ListaBicis, ListaUsuarios, Usuario);
}
void leerListaUsuarios(User** Lista){ // Funcion que lee la lista de usuarios
  char linea[500], Datos[6][200];
  int i, j = 0, contador = 0;
  FILE* Archivo = fopen("login.txt", "rt");
  if (Archivo == NULL) { // Verificamos que exista el archivo
    printf("Ha ocurrido un error, vuelva a intentar\n");
    exit(0);
  }
```

```
 while (fgets(linea, 500, Archivo) != NULL) { // Leemos el archivo
   User* Usuario = (User*)malloc(sizeof(User));
   i = 0;
   contador = 0;
   separarListaUsuarios(&i, &j, &contador, linea, Datos); // Separamos la linea leida
   strcpy(Usuario->Nombre, Datos[0]);
   strcpy(Usuario->Direccion, Datos[1]);
   strcpy(Usuario->Contrasenia, Datos[2]);
   strcpy(Usuario->TarjetaCredito, Datos[3]);
   Usuario->UserNumber = atoi(Datos[4]);
   Usuario->Flag = atoi(Datos[5]);
   Usuario->siguiente = NULL;
   if (*Lista == NULL) { // Se agrega el nodo
     *Lista = Usuario;
   }else{
     User* aux = *Lista;
     while (aux->siguiente != NULL) {
       aux = aux->siguiente;
     }
     aux->siguiente = Usuario;
   }
 }
}
void separarListaUsuarios(int* i, int* j, int* contador, char linea[], char
Datos[6][200]){
 while(linea[*i] != '\0' && linea[*i] != '\n'){
   Datos[*contador][*j] = linea[*i];

   if(linea[*i+1] == '/'){
     Datos[(*contador)][(*j)+1] = '\0';
     (*contador)++;
     (*j) = -1;
     (*i)++;
   }
   (*i)++;
   (*j)++;
 }
} // Funcion que separa una linea
void limpiarDatos(char Datos[6][200]){
 for(int i = 0; i < 5; i++){
   Datos[i][0] = '\0';
 }
} // Funcion que limpia los datos de un array de cadenas
void altaBiciestacion(Biciestacion** Lista){
 FILE* Archivo;
 Biciestacion Estacion, *auxiliar = *Lista;
 char numero[5], numeroTotal[3], cp[5], error[50], renglon[500], basura[100];
 int validacion = 1, validacion2 = 1, id = 0;
 error[0] = '\0';
 numero[0] = '\0';
 cp[0] = '\0';
 Archivo = fopen("biciestaciones.txt", "rt");
 if (Archivo == NULL) { // Obtenemos el identificador
```

```
   id = 1;
 }else{
   while(auxiliar->siguiente != NULL)
     auxiliar = auxiliar->siguiente;
   id = auxiliar->NumBiciestacion +1;
 }
 // Pedimos datos
 Archivo = fopen("biciestaciones.txt", "at");
 system("clear");
 printf("\t\tDar de alta una nueva biciestacion\n");
 while(Pedir_datos(Estacion.NombreGenerico, "nombre generico de biciestacion", 100));
 while(Pedir_datos(Estacion.Calle, "calle de la biciestacion", 50));
 while(validacion || validacion2){
   validacion = Pedir_datos(numero, "numero (numeracion de la calle)", 3);
   validacion2 = validarNumeros(numero, error);
   if(strlen(error) != 0)
     puts(error);
   error[0] = '\0';
 }
 validacion = 1;
 validacion2 = 1;
 while(validacion || validacion2){
   validacion = Pedir_datos(cp, "codigo postal", 5);
   validacion2 = validarNumeros(cp, error);
   if(strlen(cp) != 5){
     printf("Este campo se compone de solo 5 caracteres\n");
     validacion = 1;
   }
   if(strlen(error) != 0)
     printf("Este campo solo puede contener numeros\n");
   error[0] = '\0';
 }

 validacion = 1;
 validacion2 = 1;
 while(validacion || validacion2){
   validacion = Pedir_datos(Estacion.Ciudad, "ciudad", 50);
   validacion2 = ValidarCaracteres(Estacion.Ciudad, error);
   if(strlen(error) != 0)
     puts(error);
   error[0] = '\0';
 }
 anadirBiciestacion(id, Estacion.NombreGenerico, Estacion.Calle, numero, cp,
Estacion.Ciudad, Lista); //  Se anade el nodo a la lista de biciestaciones
 printf("\nSe ha agregado correctamente la biciestacion\n");
 bitacora("101", id, 0, 0); // Bitacora
 fclose(Archivo);
} // Funcion para dar de alta una biciestacion// Funcion que da de alta una
biciestacion // Funcion para dar de alta la funcion
void altaBici(Bicicleta** Lista, Biciestacion** ListaBiciestaciones){ // Funcion que
da de alta una bici a una bieciestacion
 FILE* Archivo;
 Bicicleta *auxiliar = *Lista;
```

```
 char NumBiciestacion[4], error[100];
 //Archivo = fopen("bicis.txt", "rt");
 int id = 0;
 Biciestacion* LBiciestaciones = *ListaBiciestaciones;
 int validacion = 1, validacion2 = 1;
 //if (Archivo == NULL) { // Verificamos el archivo

 //}else{
 if(*Lista == NULL)
   id = 1;
 else{
   while(auxiliar->siguiente != NULL)
     auxiliar = auxiliar->siguiente;
   printf("%ld\n", auxiliar->NumeroBici);
   id = auxiliar->NumeroBici +1;
 }
   //fclose(Archivo);
 //}

 system("clear");
 printf("\t\tDar de alta una nueva bicicleta\n");
 printf("\n");
 if(*ListaBiciestaciones == NULL)
   printf("No puedes dar de alta una bici, ya que no hay biciestaciones\n");
 // Pedimos los datos
 else{
   while(LBiciestaciones != NULL){
     printf("\t%ld-> %s\n", LBiciestaciones->NumBiciestacion, LBiciestaciones-
>NombreGenerico);
     LBiciestaciones = LBiciestaciones->siguiente;
   }
   printf("\n");
   while(validacion || validacion2){
     validacion = Pedir_datos(NumBiciestacion, "de la lista anterior, el numero de
biciestacion", 3);
     validacion2 = validarNumeros(NumBiciestacion, error);
     if(strlen(error) != 0)
       puts(error);
     error[0] = '\0';
   }
   anadirBici(id, NumBiciestacion, Lista, ListaBiciestaciones); // Agregamos el nodo
a la lista
   bitacora("202", id,  0, 0);
 }
} // Funcion que da de alta una bicicleta en una biciestacion
void altaUsuarios(User** Lista){ // Funcion que da de alta a un usuario
 FILE* Archivo;
 User Usuario, *auxiliar = *Lista, *auxiliar2 = *Lista;
 char tarjeta[17], error[100], TipoUsuario[2];
 int validacion = 1, validacion2 = 1, validacion3 = 1, id = 0;
 Archivo = fopen("login.txt", "rt");
 if (Archivo == NULL) { // Obtenemos su identificador
   id = 1;
```

```
}else{
  while(auxiliar->siguiente != NULL)
    auxiliar = auxiliar->siguiente;
  id = auxiliar->UserNumber +1;
}
Archivo = fopen("login.txt", "at");
system("clear");
// Pedimos datos
printf("\t\tDar de alta un nuevo usuario\n");
while(validacion || validacion2){
  validacion = Pedir_datos(Usuario.Nombre, "nombre de usuario", 49);
  validacion2 = ValidarCaracteres(Usuario.Nombre, error);
  if(strlen(error) != 0)
    puts(error);
  error[0] = '\0';
}
validacion = 1;
while(validacion){
  validacion = Pedir_datos(Usuario.Direccion, "direccion del usuario", 199);
  if(strlen(error) != 0)
    puts(error);
  error[0] = '\0';
}

validacion = 1;
while(validacion){
  validacion = Pedir_datos(Usuario.Contrasenia, "contrasenia", 49);
  if(strlen(error) != 0)
    puts(error);
  error[0] = '\0';
}
validacion = 1;
validacion2 = 1;
while(validacion || validacion2 || validacion3){
  validacion = Pedir_datos(tarjeta, "numero de tarjeta de credito", 16);
  validacion2= validarNumeros(tarjeta, error);
  validacion3 = 0;
  while(auxiliar2 != NULL && validacion3 == 0){
    if(strcmp(auxiliar2->TarjetaCredito, tarjeta) == 0){
      __fpurge(stdin);
      printf("Esta tarjeta ya esta registrada\n");
      validacion3 = 1;
    }
    auxiliar2 = auxiliar2->siguiente;
  }
  if(strlen(tarjeta) != 16){
    validacion = 1;
    printf("Este campo solo admite 16 caracteres\n");
  }
  if(validacion2)
    printf("Este campo solo admite numeros\n");
  error[0] = '\0';
}
```

```
 validacion = 1;
 while(validacion){
   __fpurge(stdin);
   printf("Ingresar tipo de usuario (1 para administrador o 0 para usuario normal):
");
   TipoUsuario[0] = getchar();
   if(TipoUsuario[0] != '1' && TipoUsuario[0] != '0'){
     validacion = 1;
     printf("Este campo solo admite 1 o 0!\n");
   }else{
     validacion = 0;
   }
 }
 TipoUsuario[1] = '\0';
 anadirUsuario(id, Usuario.Nombre, Usuario.Direccion, Usuario.Contrasenia, tarjeta,
TipoUsuario, Lista);
 bitacora("301", id, 0, 0);
 fclose(Archivo);
}
int ValidarCaracteres(char Cadena[], char Error[]){ // Funcion que valida el los
caracteres
 int i = 0;
 int Status = 0;
 while(Cadena[i] != '\0' && Status == 0){
   if(!(Cadena[i] >= 'a' && Cadena[i] <= 'z') && !(Cadena[i] >= 'A' && Cadena[i] <=
'Z')){
     Status = 1;
     strcpy(Error, "Este campo solamente admite letras\n");
   }
   i++;
 }
 return Status;
}
int validarNumeros(char Cadena[], char Error[]){ // Funcion que valida numeros
 int i = 0;
 int Status = 0;
 while(Cadena[i] != '\0' && Status == 0){
   if(!(Cadena[i] >= '0' && Cadena[i] <= '9')){
     Status = 1;
     strcpy(Error, "Este campo solo puede contener numeros\n");
   }
   i++;
 }
 return Status;
}
void cargarListaBiciestacion(Biciestacion** Lista){ // Cargamos la lista de
biciestaciones
 char linea[500], Datos[6][200];
 int i, j = 0, contador = 0, fail;
 FILE* Archivo = fopen("biciestaciones.txt", "rt");
 if (Archivo == NULL) {
   fail = 1;
 }else{
```

```
   while (fgets(linea, 500, Archivo) != NULL) {
     Biciestacion* Nueva = (Biciestacion*)malloc(sizeof(Biciestacion));
     i = 0;
     contador = 0;
     separarListaUsuarios(&i, &j, &contador, linea, Datos);
     Nueva->NumBiciestacion = atoi(Datos[0]);
     strcpy(Nueva->NombreGenerico, Datos[1]);
     strcpy(Nueva->Calle, Datos[2]);
     Nueva->Numero = atoi(Datos[3]);
     strcpy(Nueva->CP, Datos[4]);
     strcpy(Nueva->Ciudad, Datos[5]);
     Nueva->siguiente = NULL;
     if (*Lista == NULL) {
       *Lista = Nueva;
     }else{
       Biciestacion* aux = *Lista;
       while (aux->siguiente != NULL) {
         aux = aux->siguiente;
       }
       aux->siguiente = Nueva;
     }
   }
   fclose(Archivo);
 }
}
void cargarListaBicis(Bicicleta** Lista){
 char linea[500], Datos[6][200];
 int i, j = 0, contador = 0;
 FILE* Archivo = fopen("bicis.txt", "rt");
 if (Archivo == NULL) {
   printf("Ha ocurrido un error, vuelva a intentar\n");
 }else{
   while (fgets(linea, 500, Archivo) != NULL) {
     Bicicleta* Nueva = (Bicicleta*)malloc(sizeof(Bicicleta));
     i = 0;
     contador = 0;
     separarListaUsuarios(&i, &j, &contador, linea, Datos);
     Nueva->NumeroBici = atoi(Datos[0]);
     Nueva->Biciestacion = atoi(Datos[1]);
     Nueva->rentas = atoi(Datos[2]);
     strcpy(Nueva->Timestamp, Datos[3]);
     Nueva->esrentada = atoi(Datos[4]);
     Nueva->esrentadapor = atoi(Datos[5]);
     Nueva->siguiente = NULL;
     if (*Lista == NULL) {
       *Lista = Nueva;
     }else{
       Bicicleta* aux = *Lista;
       while (aux->siguiente != NULL) {
         aux = aux->siguiente;
       }
       aux->siguiente = Nueva;
     }
```

```
   }
 }
}// Funcion para cargar la lista de bicis
void anadirBici(int id, char NumeroBici[], Bicicleta** Lista, Biciestacion**
ListaBiciestaciones){

 Bicicleta* Nueva = (Bicicleta*)malloc(sizeof(Bicicleta));
 Biciestacion* aux = *ListaBiciestaciones;
 Bicicleta* ListaBicis = *Lista;
 int found = 0, numeroBiciestacion = atoi(NumeroBici), CuentaBici = 0;
 Nueva->NumeroBici = id;
 Nueva->rentas = 0;
 strcpy(Nueva->Timestamp, "NULL");
 Nueva->Biciestacion = numeroBiciestacion;
 Nueva->esrentada = 0;
 Nueva->esrentadapor = 0;
 Nueva->siguiente = NULL;
 while(aux != NULL){
   if(Nueva->Biciestacion == aux->NumBiciestacion)
     found = 1;
   aux = aux->siguiente;
 }
 if(found == 0)
   printf("No se pudo añadir porque no se encontro la biciestacion introducida\n");
 else{
   while(ListaBicis != NULL){
     if(ListaBicis->Biciestacion == numeroBiciestacion)
       CuentaBici++;
     ListaBicis = ListaBicis->siguiente;
   }
   if(CuentaBici >= 10)
     printf("No se puede agregar la Bici a la biciestacion seleccionada, ya que se
encuentra llena\n");
   else{
     if (*Lista == NULL) {
       *Lista = Nueva;
     }else{
       ListaBicis = *Lista;
       while (ListaBicis->siguiente != NULL) {
         ListaBicis = ListaBicis->siguiente;
       }
       ListaBicis->siguiente = Nueva;
     }
     printf("La bicicleta se ha añadido correctamente\n");
   }
 }
} // Agrega un nodo a la lista bicis
void anadirBiciestacion(int id, char NombreGenerico[], char Calle[], char numero[],
char cp[], char Ciudad[], Biciestacion** Lista){
 Biciestacion* Nueva = (Biciestacion*)malloc(sizeof(Biciestacion));
 Biciestacion* aux;
 Nueva->NumBiciestacion = id;
 strcpy(Nueva->NombreGenerico, NombreGenerico);
```

```
 strcpy(Nueva->Calle, Calle);
 Nueva->Numero = atoi(numero);
 strcpy(Nueva->CP, cp);
 strcpy(Nueva->Ciudad, Ciudad);
 Nueva->siguiente = NULL;
 if (*Lista == NULL) {
   *Lista = Nueva;
 }else{
   aux = *Lista;
   while (aux->siguiente != NULL) {
     aux = aux->siguiente;
   }
   aux->siguiente = Nueva;
 }
}// Agrega un nodo a la lista biciestaciones
void anadirUsuario(int id, char Nombre[], char Direccion[], char Contrasenia[], char
tarjeta[], char TipoUsuario[], User** Lista){
 User* Nuevo = (User*)malloc(sizeof(User));
 User* aux;
 strcpy(Nuevo->Nombre, Nombre);
 strcpy(Nuevo->Direccion, Direccion);
 strcpy(Nuevo->Contrasenia, Contrasenia);
 strcpy(Nuevo->TarjetaCredito, tarjeta);
 Nuevo->UserNumber = id;
 Nuevo->Flag = atoi(TipoUsuario);
 Nuevo->siguiente = NULL;
 if (*Lista == NULL) {
   *Lista = Nuevo;
 }else{
   aux = *Lista;
   while (aux->siguiente != NULL) {
     aux = aux->siguiente;
   }
   aux->siguiente = Nuevo;
 }
 printf("Se ha añadido con exito el usuario\n");
}// Agrega un nodo a la lista usuarios
void reasignarBicis(Bicicleta** ListaBicis, Biciestacion** ListaBiciestaciones){
 system("clear");
 printf("\t\tReasignar Bicis entre Biciestaciones\n\n");
 Biciestacion* aux = *ListaBiciestaciones;
 Bicicleta* aux2 = *ListaBicis;
 char Numero[4], NumBici[4], error[100];
 int validacion = 1, validacion2 = 1;
 if(*ListaBicis == NULL)
   printf("Actualmente no hay bicicletas registradas\n");
 else{
   printf("Listado de bicicletas:\n");
   while(aux2 != NULL){
     if(aux2->Biciestacion != 0)
       printf("\tNumero de bici: %ld. Pertenece a biciestacion numero: %ld\n", aux2-
>NumeroBici, aux2->Biciestacion);
     aux2 = aux2->siguiente;
```

```
   }
   printf("\n");
   while(validacion || validacion2){
     validacion = Pedir_datos(Numero, "numero de bicicleta", 3);
     validacion2= validarNumeros(Numero, error);
     if(strlen(error) != 0)
       puts(error);
     error[0] = '\0';
   }
   validacion = 1;
   validacion2 = 1;
   while(validacion || validacion2){
     validacion = Pedir_datos(NumBici, "numero de biciestacion a la que se desea
reasignar la bicicleta", 3);
     validacion2= validarNumeros(NumBici, error);
     if(strlen(error) != 0)
       puts(error);
     error[0] = '\0';
   }
   reasignar(Numero, NumBici, ListaBicis, ListaBiciestaciones);
 }
}// Pide los datos para reasignar las bicis
void reasignar(char Numero[],char NumBici[],Bicicleta** ListaBicis,Biciestacion**
ListaBiciestaciones){
 Biciestacion* aux = *ListaBiciestaciones;
 Bicicleta* aux2 = *ListaBicis, *aux3 = *ListaBicis;
 int found = 0, cantidadBicis = 0, contador = 0;
 while(aux != NULL){
   if(aux->NumBiciestacion == atoi(NumBici))
     found = 1;
   aux = aux->siguiente;
 }
 if(found == 0)
   printf("No se puede realizar la reasignacion debido a que el numero de
biciestacion seleccionada, no existe\n");
 else{
   while(aux2 != NULL){
     if(aux2->Biciestacion == atoi(NumBici))
       cantidadBicis++;
     aux2 = aux2->siguiente;
   }
   if(cantidadBicis >= 10){
     printf("No se puede realizar la reasignacion debido a que la biciestacion
seleccionada se encuentra llena\n");
   }else{
     while(aux3 != NULL && contador == 0){
       if(aux3->NumeroBici == atoi(Numero) && aux3->esrentada == 0)
         contador++;
       else
         aux3 = aux3->siguiente;
     }
     if(contador == 1){
       aux3->Biciestacion = atoi(NumBici);
```

```
       bitacora("203", atoi(Numero), atoi(NumBici), 0);
       printf("Reasignacion completada con exito\n");
     }else{
       printf("La bicicleta no puede ser reasignada porque actualmente esta siendo
rentada\n");
     }
   }
 }
}// Realiza la reasignacion // Funcion para reasignar Bicicletas
void status(Biciestacion** ListaBiciestaciones, Bicicleta** ListaBicis){
 Biciestacion* aux = *ListaBiciestaciones;
 int Numrenta = 0, lugaresDisponibles = 0;
 system("clear");
 printf("\t\tEstatus de biciestaciones\n");
 for(int i = 0; i < 74; i++){
   printf("*");
 }
 printf("\n");
 printf("N.Biciestacion*                Nombre generico  *B.D*L.D*\n");
 for(int i = 0; i < 74; i++){
   printf("*");
 }
 printf("\n");
 while(aux != NULL){
   Numrenta = obtenerNumerorentas(aux->NumBiciestacion, ListaBicis);
   lugaresDisponibles = 10 - Numrenta;
   printf("%8ld      *%-50s*%2d *%3d*\n", aux->NumBiciestacion, aux->NombreGenerico,
Numrenta, lugaresDisponibles);
   for(int i = 0; i < 74; i++){
     printf("*");
   }
   printf("\n");
   aux = aux->siguiente;
 }
 printf("\n\nB.D = Bicis Disponibles para Renta\nL.D = Lugares Disponible para
Estacionarse\n");
}// Muestra el estatus de una biciestacion
int obtenerNumerorentas(int numeroBiciestacion, Bicicleta** ListaBicis){
 Bicicleta* aux = *ListaBicis;
 int numeroRentas = 0;
 while(aux != NULL){
   if(aux->Biciestacion == numeroBiciestacion && aux->esrentada == 0)
     numeroRentas++;
   aux = aux->siguiente;
 }
 return numeroRentas;
} // Obtiene las rentas de una bicicleta
void eliminarBiciestacion(Biciestacion** ListaBiciestaciones, Bicicleta**
ListaBicis){
 system("clear");
 char numero[4], error[100];
 int validacion = 1, validacion2 = 1;
 long Nbitacora;
```

```
 printf("\t\tBaja de una biciestacion\n");
 if(*ListaBiciestaciones == NULL)
   printf("Actualmente no hay biciestaciones registradas\n");
 else{
   while(validacion || validacion2){
     validacion = Pedir_datos(numero, "biciestacion a eliminar", 3);
     validacion2 = validarNumeros(numero, error);
     if(strlen(error) != 0)
       puts(error);
     error[0] = '\0';
   }
   deleteBiciestacion(ListaBiciestaciones, ListaBicis, numero);
 }
}// Pide datos para eliminar biciestacion
void deleteBiciestacion(Biciestacion** ListaBiciestaciones, Bicicleta** ListaBicis,
char numero[]){
 int found = 0, i = 0;
 Biciestacion* aux = *ListaBiciestaciones, *aux3 = *ListaBiciestaciones, *ant = NULL;
 Bicicleta* aux2 = *ListaBicis;
 int cuentaBicis = 0;
 while(aux != NULL){
   if(aux->NumBiciestacion == atoi(numero))
     found = 1;
   aux = aux->siguiente;
 }
 if(found == 0)
   printf("No se puede dar de baja la biciestacion porque la biciestacion introducida
no existe\n");
 else{
   while(aux2 != NULL){
     if(aux2->Biciestacion == atoi(numero))
       cuentaBicis++;
     aux2 = aux2->siguiente;
   }
   if(cuentaBicis != 0)
     printf("No se puede dar de baja la biciestacion seleccionada debido a que aun
tiene bicis relacionadas a ella\n");
   else{
     while(aux3 != NULL && aux3->NumBiciestacion != atoi(numero)) {
       i++;
       ant = aux3;
       aux3 = aux3->siguiente;
     }
     if(i == 0 && ant == NULL)
       remove("biciestaciones.txt");
     if(aux3 != NULL){
       if(ant != NULL){
         ant->siguiente = aux3->siguiente;
       }else{
         *ListaBiciestaciones = aux3->siguiente;
       }
       free(aux3);
     }
```

```
      printf("Se ha eliminado correctamente la biciestacion\n");
      bitacora("102", atoi(numero), 0, 0);
    }
  }
} // Elimina un nodo de la lista de biciestaciones
void eliminarBici(Bicicleta** Lista){
 system("clear");
 char numero[4], error[100];
 int validacion = 1, validacion2 = 1;
 Bicicleta* aux = *Lista;
 printf("\t\tBaja de una bicicleta de una biciestacion\n");
 if(*Lista != NULL){
   printf("Lista de bicicletas: \n\n");
   while(aux != NULL){
     if(aux->Biciestacion == 0)
       printf("\t\tNumero de bici: %ld. Esta siendo rentada\n", aux->NumeroBici);
     else
       printf("\t\tNumero de bici: %ld. Pertenece a biciestacion numero: %ld\n", aux-
>NumeroBici, aux->Biciestacion);
     aux = aux->siguiente;
   }
   printf("\n");

   while(validacion || validacion2){
     validacion = Pedir_datos(numero, "bicicleta a eliminar", 3);
     validacion2 = validarNumeros(numero, error);
     if(strlen(error) != 0)
       puts(error);
     error[0] = '\0';
   }
   deleteBici(Lista, numero);
 }else
   printf("Actualmente no hay bicicletas dadas de alta\n");
} // Pide datos para dar de baja una bici
void deleteBici(Bicicleta** ListaBicis, char numero[]){
 int found = 0, i = 0;
 Bicicleta* aux = *ListaBicis, *aux2 = *ListaBicis, *aux3, *ant = NULL;
 while(aux != NULL){
   if(aux->NumeroBici == atoi(numero)){
     found = 1;
     aux3 = aux;
   }
   aux = aux->siguiente;
 }
 if(found == 0)
   printf("No se puede dar de baja la bicicleta porque la bicicleta introducida no
existe\n");
 else{
   if(aux3->esrentada == 1)
     printf("La bicicleta no se puede dar de baja debido a que en este momento esta
siendo rentada\n");
   else{
     while(aux2 != NULL && aux2->NumeroBici != atoi(numero)) {
```

```
      i++;
      ant = aux2;
      aux2 = aux2->siguiente;
    }
    if(i == 0 && ant == NULL)
      remove("bicis.txt");
    if(aux2 != NULL){
      if(ant != NULL){
        ant->siguiente = aux2->siguiente;
      }else{
        *ListaBicis = aux2->siguiente;
      }
      free(aux2);
    }
    printf("Se ha eliminado correctamente la bicicleta\n");
    bitacora("202", atoi(numero), 0, 0);
  }
 }
} // Elimina un nodo de la lista de bicis
void eliminarUsuario(User** ListaUsuarios, Bicicleta** ListaBicis){
 system("clear");
 char numero[4], error[100];
 int validacion = 1, validacion2 = 1;
 printf("\t\tBaja de un usuario\n");
 while(validacion || validacion2){
   validacion = Pedir_datos(numero, "numero de usuario a eliminar", 3);
   validacion2 = validarNumeros(numero, error);
   if(strlen(error) != 0)
     puts(error);
   error[0] = '\0';
 }
 deleteUsuario(ListaUsuarios, ListaBicis, numero);
}// Pide datos para dar de baja un usuario
void deleteUsuario(User** ListaUsuarios, Bicicleta** ListaBicis, char numero[]){
 int found = 0, i = 0;
 User* aux = *ListaUsuarios, *aux3 = *ListaUsuarios, *ant = NULL;
 Bicicleta* aux2 = *ListaBicis;
 int cuenta = 0;
 while(aux != NULL){
   if(aux->UserNumber == atoi(numero))
     found = 1;
   aux = aux->siguiente;
 }
 if(found == 0)
   printf("No se puede dar de baja el usuario, ya que el usuario introducido no
existe\n");
 else{
   while(aux2 != NULL){
     if(aux2->esrentadapor == atoi(numero))
       cuenta++;
     aux2 = aux2->siguiente;
   }
   if(cuenta != 0)
```

```
     printf("No se puede dar de baja el usuario, ya que esta rentando una
bicicleta\n");
   else{
     while(aux3 != NULL && aux3->UserNumber != atoi(numero)) {
       i++;
       ant = aux3;
       aux3 = aux3->siguiente;
     }
     if(i == 0 && ant == NULL)
       remove("login.txt");
     if(aux3 != NULL){
       if(ant != NULL){
         ant->siguiente = aux3->siguiente;
       }else{
         *ListaUsuarios = aux3->siguiente;
       }
       free(aux3);
     }
     printf("Se ha eliminado correctamente el usuario seleccionado\n");
     bitacora("302", atoi(numero), 0, 0);
   }
 }
}// Elimina un nodo de la lista de usuarios
void Timestamp(char Cadena[]){
 time_t rawtime;
 struct tm *timeinfo;
 time(&rawtime);
 timeinfo = localtime(&rawtime);
 sprintf(Cadena, "%d%d%d-%d:%d:%d\n", timeinfo->tm_year+1900, timeinfo->tm_mon+1,
timeinfo->tm_mday, timeinfo->tm_hour, timeinfo->tm_min, timeinfo->tm_sec);
 Cadena[strlen(Cadena)-1] = '\0';
} // Obtiene el Timestamp
void pedirDatosRenta(long Usuario, Biciestacion** ListaBiciestaciones, Bicicleta**
ListaBicicletas){
 system("clear");
 char numero[4], error[100];
 int validacion = 1, validacion2 = 1, validacion3 = 0;
 Bicicleta* aux = *ListaBicicletas, *aux2 = *ListaBicicletas;
 printf("\t\tRenta de una bicicleta\n");

 while(aux2 != NULL){
   if(aux2->esrentadapor == Usuario)
     validacion3 = 1;
   aux2 = aux2->siguiente;
 }

 if(validacion3 == 1)
   printf("Actualmente ya tienes rentada una bicicleta\n");
 else{
   if(*ListaBicicletas != NULL){
     printf("Lista de bicicletas disponibles para renta: \n\n");
     while(aux != NULL){
       if(aux->esrentada == 0)
```

```
        printf("\t\tNumero de bici: %ld. Pertenece a biciestacion numero: %ld\n",
aux->NumeroBici, aux->Biciestacion);
      aux = aux->siguiente;
    }
    printf("\n");

    while(validacion || validacion2){
      validacion = Pedir_datos(numero, "numero de bicicleta a rentar", 3);
      validacion2 = validarNumeros(numero, error);
      if(strlen(error) != 0)
        puts(error);
      error[0] = '\0';
    }
    rentar(ListaBicicletas, numero, Usuario);
  }else
    printf("Actualmente no hay bicicletas en el sistema\n");
 }
} // Pide datos de renta
void rentar(Bicicleta** ListaBicicletas, char numero[], long Usuario){
 Bicicleta* aux = *ListaBicicletas;
 int found = 0;
 while(aux != NULL && found == 0){
   if(aux->esrentada == 0 && aux->NumeroBici == atoi(numero))
     found = 1;
   else
     aux = aux->siguiente;
 }
 if(found == 0)
   printf("La bici seleccionada no se encuentra disponible para ser rentada\n");
 else{
   aux->esrentada = 1;
   aux->esrentadapor = Usuario;
   aux->rentas = aux->rentas+1;
   //aux->Biciestacion = 0;
   Timestamp(aux->Timestamp);
   printf("La bicicleta esta siendo rentada apartir de ahora\n");
 }
} // Pone el estatus de una bici a rentada
void devolverBici(long Usuario, Biciestacion** ListaBiciestaciones, Bicicleta**
ListaBicis){
 Biciestacion* aux = *ListaBiciestaciones;
 Bicicleta* aux2 = *ListaBicis;
 int NumeroD = 0;
 char numero[4], error[100];
 int validacion = 1, validacion2 = 1, found = 0;
 system("clear");
 printf("\t\tEstacionar bicicleta\n");
 while(aux2 != NULL){
   if(aux2->esrentadapor == Usuario)
     found = 1;
   aux2 = aux2->siguiente;
 }
```

```
 if(found == 1){
   printf("Lista de biciestaciones con lugares disponibles: \n\n");
   while(aux != NULL){
     NumeroD = 10 - obtenerNumerorentas(aux->NumBiciestacion, ListaBicis);
     if(NumeroD > 0)
       printf("N. biciestacion: %ld-> Nombre biciestacion: %s-> Lugares disponibles:
%d\n", aux->NumBiciestacion, aux->NombreGenerico, NumeroD);
     aux = aux->siguiente;
   }
   printf("\n");
   while(validacion || validacion2){
     validacion = Pedir_datos(numero, "biciestacion donde se desea devolver", 3);
     validacion2 = validarNumeros(numero, error);
     if(strlen(error) != 0)
       puts(error);
     error[0] = '\0';
   }
   devolver(Usuario, ListaBicis, ListaBiciestaciones, numero);
 }else
   printf("Actualmente usted no tiene bicis rentadas\n");
} // Pide datos para estacionar una bici
void devolver(long Usuario, Bicicleta** ListaBicis, Biciestacion**
ListaBiciestaciones, char numero[]){
 Biciestacion* aux = *ListaBiciestaciones;
 Bicicleta* aux2 = *ListaBicis;
 int Multa = 0;
 int NumeroD = 0, found = 0, found2 = 0;
 while(aux != NULL && found == 0){
   NumeroD = 10 - obtenerNumerorentas(aux->NumBiciestacion, ListaBicis);
   if(NumeroD > 0 && aux->NumBiciestacion == atoi(numero))
     found = 1;
   else
     aux = aux->siguiente;
 }
 if(found == 0)
   printf("No se puede devolver la bicicleta a la biciestacion seleccionada porque ya
no hay lugares o porque no introdujo una biciestacion invalida\n");
 else{
   while(aux2 != NULL && found2 == 0){
     if(aux2->esrentadapor == Usuario)
       found2 = 1;
     else
       aux2 = aux2->siguiente;
   }
   Multa = restarHoras(aux2->Timestamp);
   strcpy(aux2->Timestamp, "NULL");
   aux2->esrentada = 0;
   aux2->esrentadapor = 0;
   aux2->Biciestacion = atoi(numero);
   if(Multa){
     printf("Tienes una multa extra de $65 por pasarte del tiempo\n");
     agregarMulta(Usuario);
   }
```

```
    printf("Se ha devuelto de manera correcta la bicicleta\n");
  }
} // Devuelve la bici a una biciestacion
int restarHoras(char Horainicial[]){
 int horas, minutos, segundos;
 int horasF, minutosF, segundosF, Dhoras, Dminutos;
 int Haymulta = 0;
 long fecha;
 time_t rawtime;
 struct tm *timeinfo;
 char Timestamp[100];
 time(&rawtime);
 timeinfo = localtime(&rawtime);
 horasF = timeinfo->tm_hour;
 minutosF = timeinfo->tm_min;
 segundosF = timeinfo->tm_sec;
 sscanf(Horainicial, "%ld-%d:%d:%d", &fecha, &horas, &minutos, &segundos);
 if(minutosF < minutos){
   horasF--;
   minutosF+= 60;
 }
 Dminutos = minutosF-minutos;
 Dhoras = horasF - horas;
 if(Dhoras > 0 || Dminutos > 30)
   Haymulta = 1;
 else{
   if(Dminutos == 29 && (segundos + segundosF) > 60)
     Haymulta = 1;
 }
 return Haymulta;
} // Obtiene la diferencia de horas
void agregarMulta(long Usuario){
 FILE* Archivo = fopen("multas.txt", "at");
 fprintf(Archivo, "%ld\n", Usuario);
 fclose(Archivo);
} // Agrega una multa
void mostrarSaldo(long Usuario){
 FILE* Archivo = fopen("multas.txt", "rt");
 int Cuentamulta = 0;
 int NumUser;
 system("clear");
 printf("\t\tMostrar saldo\n");
 if(Archivo == NULL)
   printf("Actualmente no tienes ninguna multa :)\n");
 else{
   while(!feof(Archivo)){
     fscanf(Archivo, "%d\n", &NumUser);
     if(NumUser == Usuario)
       Cuentamulta += 65;
   }
   if (Cuentamulta == 0) {
     printf("Actualmente no tienes ninguna multa :)\n");
   }else{
```

```
     printf("Actualmente cuentas con una multa de $%d\n", Cuentamulta);
    }
  }
} // Muestra el saldo
void bitacora(char Accion[], int Adicional1, int Adicional2, long User){
 FILE* Archivo = fopen("bitacora.txt", "at");
 char Time[100];
 Timestamp(Time);
 if(strcmp(Accion, "Login") == 0 || strcmp(Accion, "Logout") == 0){
   fprintf(Archivo, "%s %s %ld\n", Time, Accion, User);
 }else{
   if(strcmp(Accion, "203") != 0)
     fprintf(Archivo, "%s %s %d\n", Time, Accion, Adicional1);
   else
     fprintf(Archivo, "%s %s %d %d\n", Time, Accion, Adicional1, Adicional2);
 }
 fclose(Archivo);
}// Funcion que imprime la bitacora
void creditos(){
 system("clear");
   system("sleep 0.1");
   printf("========================================================================\n");
   system("sleep 0.1");
   printf("d88888b  .o88b. .d88b.         d8888b. d888888b .o88b. d888888b\n");
   system("sleep 0.1");
   printf("88'     d8P Y8 .8P  Y8. 88  `8D `88' d8P  Y8 `88' \n");
   system("sleep 0.1");
   printf("88ooooo 8P      88 88 88oooY'    88 8P 88 \n");
   system("sleep 0.1");
   printf("88ooooo 8b      88 88 88ooob.    88 8b 88 \n");
   system("sleep 0.1");
   printf("88.     Y8b d8 `8b  d8' 88   8D .88. Y8b d8   .88. \n");
   system("sleep 0.1");
   printf("Y88888P  `Y88P' `Y88P'         Y8888P' Y888888P `Y88P' Y888888P\n");
   system("sleep 0.1");
   printf("========================================================================\n");
   system("sleep 0.1");
   printf("==================     Desarrollado por ====================\n");
   system("sleep 0.1");
   printf("==================      Jair Escamilla ====================\n");
   system("sleep 0.1");
   printf("==================      Daniel Logvin ====================\n");
   system("sleep 0.1");
   printf("==================      Carlos Iturralde ====================\n");
   system("sleep 0.1");
   printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
   system("sleep 0.1");
} // Despliega los creditos
void ayuda(){
  system("clear");
  printf("Comandos disponibles: \n");
  printf(" $ bici.exe -h (Despliega el menú de ayuda al usuario)\n");
  printf(" $ bici.exe -c (Despliega los créditos de los desarrolladores del
```

```
sistema.)\n");
  printf(" $ bici.exe -usu (Despliega el listado de todos los usuarios del servicio
catalogados.)\n");
  printf(" $ bici.exe (Inicia sesion.)\n");
} // Despliega el menu de ayuda
void imprimirArchivos(Biciestacion** ListaBicis, Bicicleta** ListaBicicletas, User**
ListaUsuarios){
 Biciestacion* aux = *ListaBicis;
 Bicicleta* aux2 = *ListaBicicletas;
 User* aux3 = *ListaUsuarios;
 FILE* Archivo = fopen("biciestaciones.txt", "wt");
 if(*ListaBicis == NULL){
   fclose(Archivo);
   remove("biciestaciones.txt");
 }else{
   while (aux != NULL) {
     fprintf(Archivo, "%ld/%s/%s/%d/%s/%s/\n", aux->NumBiciestacion, aux-
>NombreGenerico, aux->Calle, aux->Numero, aux->CP, aux->Ciudad);
     aux = aux->siguiente;
   }
   fclose(Archivo);
 }

 Archivo = fopen("bicis.txt", "wt");
 if(*ListaBicicletas == NULL){
   fclose(Archivo);
   remove("bicis.txt");
 }else{
   while(aux2 != NULL){
     fprintf(Archivo, "%ld/%ld/%ld/%s/%d/%ld/\n", aux2->NumeroBici, aux2-
>Biciestacion, aux2->rentas, aux2->Timestamp, aux2->esrentada, aux2->esrentadapor);
     aux2 = aux2->siguiente;
   }
   fclose(Archivo);
 }
 Archivo = fopen("login.txt", "wt");
 if(*ListaUsuarios == NULL){
   fclose(Archivo);
   remove("login.txt");
 }else{
   while (aux3 != NULL) {
     fprintf(Archivo, "%s/%s/%s/%s/%ld/%d/\n", aux3->Nombre, aux3->Direccion, aux3-
>Contrasenia, aux3->TarjetaCredito, aux3->UserNumber, aux3->Flag);
     aux3 = aux3->siguiente;
   }
   fclose(Archivo);
 }
}// Imprime al final los archivos
//*************************************************************************
```

VI. Plan de pruebas

El plan de pruebas se realizó con el objetivo de organizar las actividades necesarias para encontrar errores y defectos dentro del programa. Esto se hace con el fin de asegurar la calidad del producto. Para corroborar el adecuado funcionamiento del programa, se probó cada aspecto de dicha tarea. Para ello, se separó este apartado en dos grandes bloques: la Prueba de Análisis y la Prueba del Código.

Dentro de la Prueba de Análisis, se revisó el manejo del diagrama de bloques, que fue presentado con anterioridad. También, paso a paso, se fue revisando el desarrollo de la lógica, y en caso de estar erroneo, se corrigió.

Por otra parte, dentro de la Prueba del Código, se corrió a mano el código y una vez que el programa estaba finalizado, se introdujeron datos al azar, con el fin de corroborar el adecuado funcionamiento del programa.

Durante este proceso surgieron ciertas fallas que fueron corregidas. Dentro de éstas, mencionaré las que resultaron más relevantes. En un principio, se llegaba a eliminar información relevante de los usuarios, debido a que se eliminaban archivos, cuando no debían. Así mismo, el erróneo modo de apertura de los datos occasionaba que se duplicaran los datos que se guardaban dentro de ellos. Ambos casos fueron corregidos por medio de una revisión a la lógica utilizada dentro de dicha función.

VII. Conclusiones

El proyecto final, de la materia de Fundamentos de Programación sirve para reforzar los temas y conceptos aprendidos a lo largo del semestre en curso. Así mismo, permite ver nuestras fortalezas y áreas de oportunidad dentro de la materia.

A su vez, es una práctica que nos permite ver el uso de la programación para la solución de problemas del día a día. También ayuda a comprender el mejor funcionamiento de la programación estructurada, la cual ayuda a resolver los problemas de un mundo envuelto en las tecnologías emergentes.

En el proyecto, documentado en el texto en cuestión, se desarrolla un sistema parecido al de las ECOBICI, el cual presta bicicletas a usuarios. Esto, por medio de estaciones que cuentan, en el caso de este programa, con bicicletas abiertas a usuarios registrados dentro del sistema. Dichos vehículos serán concedidos por un lapso de 30 minutos y, pasado este tiempo deberán de ser regresadas

El proyecto cuenta con dos beneficios principales:

En primer lugar, establece las bases de un sistema de renta de bicicletas, que permite el goce y disfrute de los objetos. Esto, por medio de las tecnologías emergentes del mundo actual.

Así mismo, fomenta un desarrollo más sustentable y saludable. Esto, debido a que por medio de las bicicletas se busca disminuir la producción excesiva de CO_2 que se crea por medio de los vehículos que utilizan combustible. De la misma manera, fomenta el ejercicio, ya que al usar las bicicletas se fortalecen no sólo los músculos de una persona, sino que también la salud.

Referencias

Libros consultados:

[1] B. K. “The C Programming Language,” *(1998), 2nd* ed. Prentice Hall