Noviembre de 2017

Documentación

Proyecto Final Programación II

Oswaldo Manuel Juárez Rojas Matrícula 160726 Francisco Rafael Lezama Hernández Matrícula 161148

Universidad Politécnica de San Luis Potosí – Ingeniería en Tecnologías de la Información. Programación II

Tabla de contenido

[Manual de Usuario. 2](#_Toc498704953)

[Interfaz 2](#_Toc498704954)

[Administrador 2](#_Toc498704955)

[Personal 4](#_Toc498704956)

[Manual de programador 5](#_Toc498704957)

[Archivos 5](#_Toc498704958)

[Estructuras 5](#_Toc498704959)

[Librerías 6](#_Toc498704960)

[Valores definidos 6](#_Toc498704961)

[Validaciones.h 7](#_Toc498704962)

[Agregar.h y Recuperar.h 7](#_Toc498704963)

[Modificar.h 9](#_Toc498704964)

[Actualizar.h 9](#_Toc498704965)

[Asignar.h 11](#_Toc498704966)

[Administrador.h 12](#_Toc498704967)

[Personal.h 13](#_Toc498704968)

[Creditos.h 13](#_Toc498704969)

# Manual de Usuario.

Bienvenido a su manual de usuario, gracias por adquirir nuestro producto. Se muestran a continuación una serie de instrucciones para llevar a cabo el correcto funcionamiento de la aplicación. Primeramente, se debe de ejecutar el archivo “Proyecto Final.exe” que se encuentra en la carpeta de instalación de la aplicación.

## Interfaz

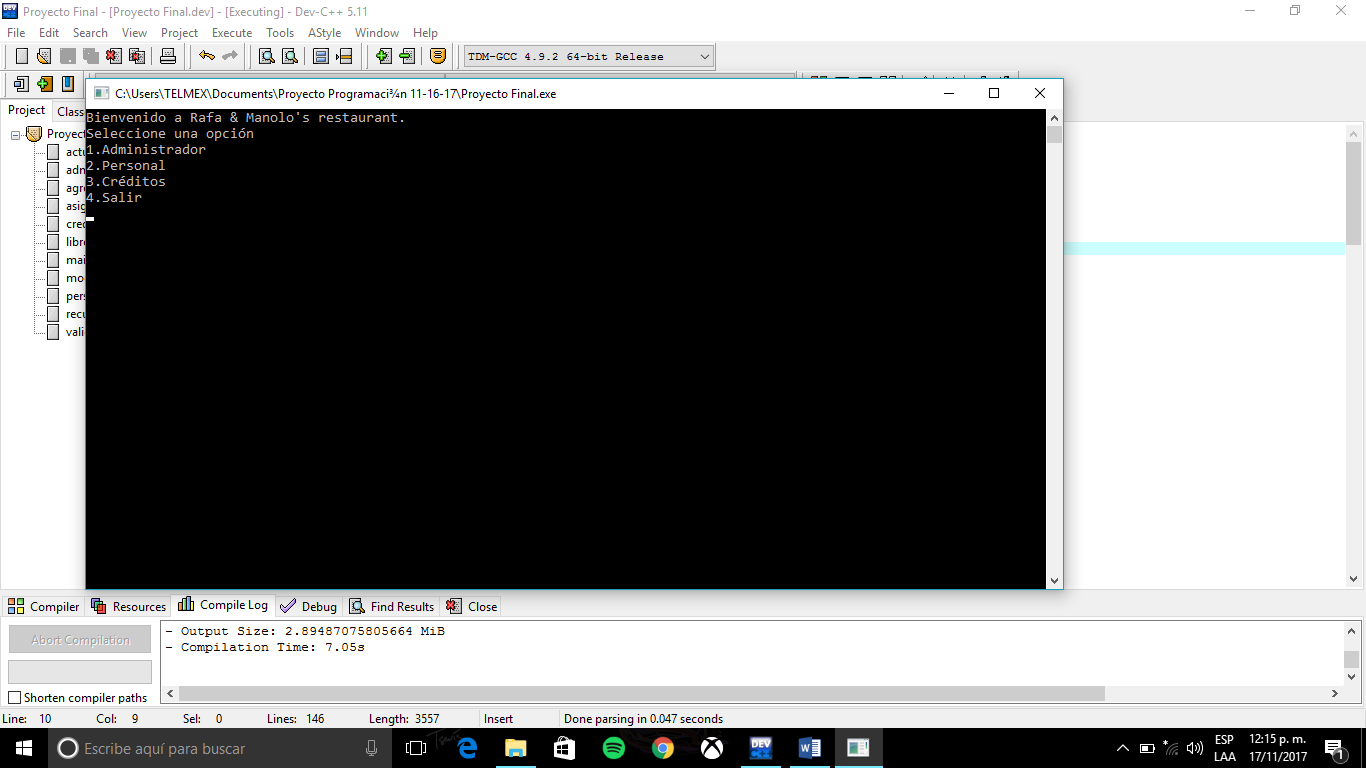
La ejecución del programa muestra una interfaz limpia, en donde se pueden distinguir claramente las distintas opciones que se ofrecen en la aplicación. Como se muestra en la ilustración 1. Para poder acceder a cualquiera de estas opciones, se debe introducir datos mediante el teclado de acuerdo con el número de opción a la que se desea acceder (1,2,3 ó 4).

Ilustración Interfaz de inicio

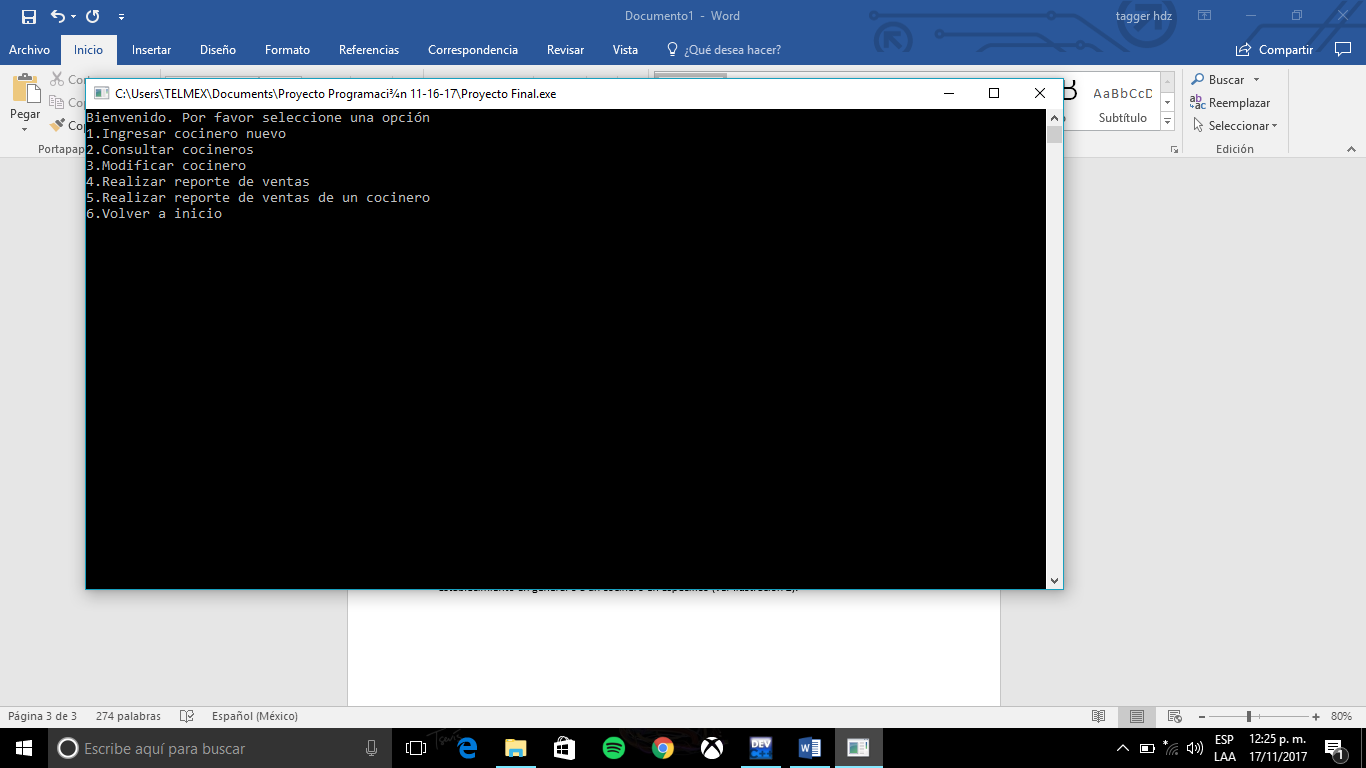
Cada una de las interfaces muestran opciones acordes a la selección que se elija mediante el teclado, el número de opciones dependerá del tipo de usuario que esté ejecutando el programa (Administrador o Personal) y de las modificaciones que se deseen hacer (Agregar/Modificar un cocinero, Realizar/Finalizar pedido).

Una vez seleccionada la opción, el sistema lo direccionará a una nueva interfaz donde podrá realizar lo que se desee.

## Administrador

Para ingresar a las opciones de Administrador se tiene que ingresar una contraseña, o el número cero (0) para regresar a la interfaz principal, la contraseña definida es “upslp123” (sin comillas). Una vez dentro del administrador usted podrá observar cada una de las opciones con respecto a los cocineros que ocupan el establecimiento o generar reportes de ventas ya sea del establecimiento en general o a un cocinero en específico (ver ilustración 2).

Ilustración Administrador

Ingresar la opción 1 lo llevará a un nuevo apartado en el que se generará un nuevo ID para un cocinero de manera aleatoria. Cuando se ha generado, se imprimirá en la pantalla y se pedirá que ingrese en nombre del cocinero a agregar en el historial. Cuando se haya agregado el cocinero, se imprime un mensaje en pantalla confirmando la operación.

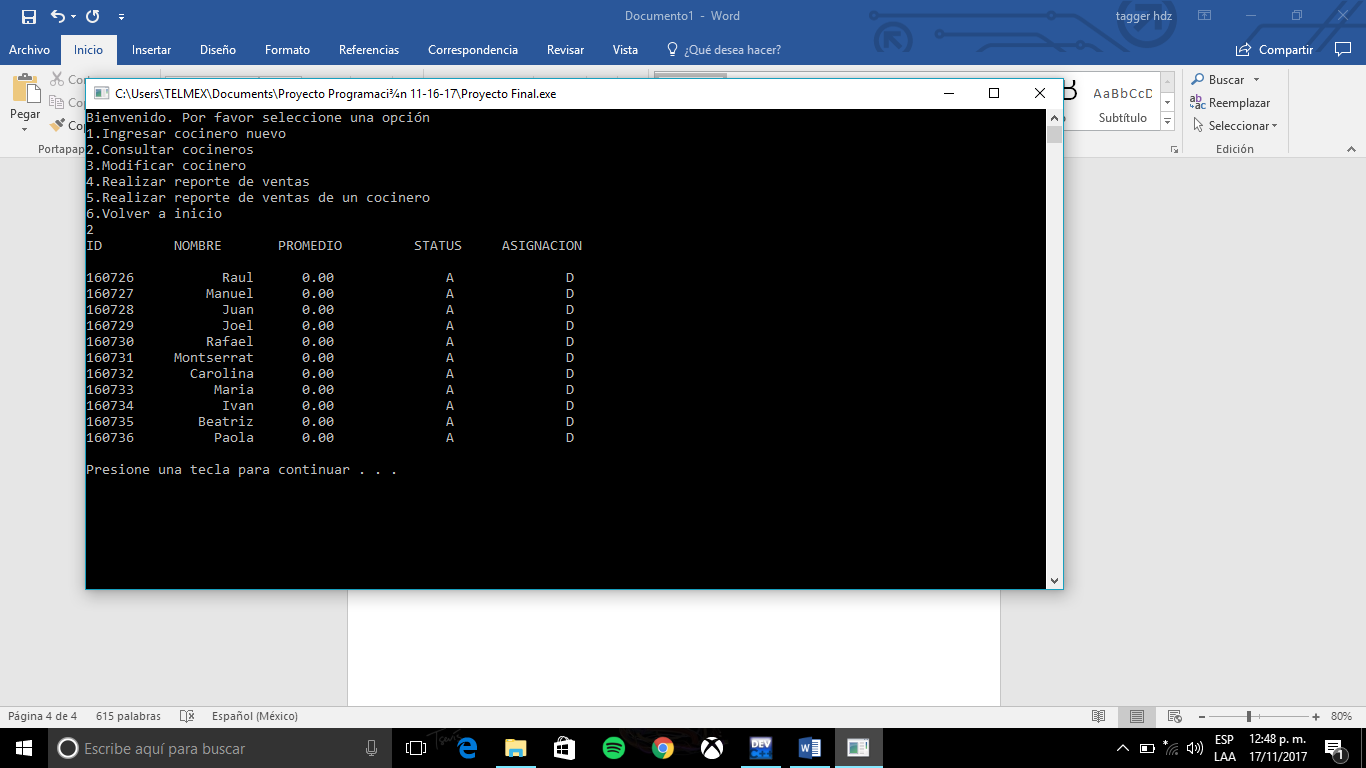
Con la opción número 2, se mostrarán todos los cocineros que se hayan ingresado previamente al establecimiento ordenados de acuerdo a su ID, de manera ascendente (ilustración 3); además se mostrará el nombre del cocinero, el promedio de los pedidos que atiende, su status (A si está activo el cocinero, por lo tanto podrá atender pedidos; I si está inactivo el cocinero, por lo que no podrá atender pedidos) y su asignación (D si está desocupado el cocinero, O si el cocinero está atendiendo un pedido). Si usted ingresa la opción número tres, podrá ver la lista de los cocineros nuevamente, esta vez se le pedirá que ingrese un ID que corresponde al del cocinero que desee modificar. Cuando ingrese el ID, se mostrarán los datos del cocinero seleccionado. Podrá modificar dos cosas, el nombre del cocinero o es status del cocinero. Cuando se modifique satisfactoriamente, el sistema le manda un mensaje de confirmación.

Ilustración Consulta de cocineros

La opción 4 le mostrará el reporte de ventas, esto incluye el número de pedidos que se han atendido hasta ese momento, el total de las ganancias y los datos del establecimiento. Con la opción número 5 se podrá ver el listado de cocineros y nuevamente se le pedirá que ingrese el ID del cocinero que desea consultar su reporte de ventas. El reporte de ventas del cocinero incluye los datos actuales con los que cuenta, además de la cantidad de pedidos que ha atendido y el total de ganancias individuales que ha generado ese cocinero.

Cuando se haya terminado cualquiera de las opciones, el sistema lo redireccionará a la pantalla principal del Administrador al presionar cualquier tecla. Si desea regresar al menú principal del sistema, presione 6.

## Personal

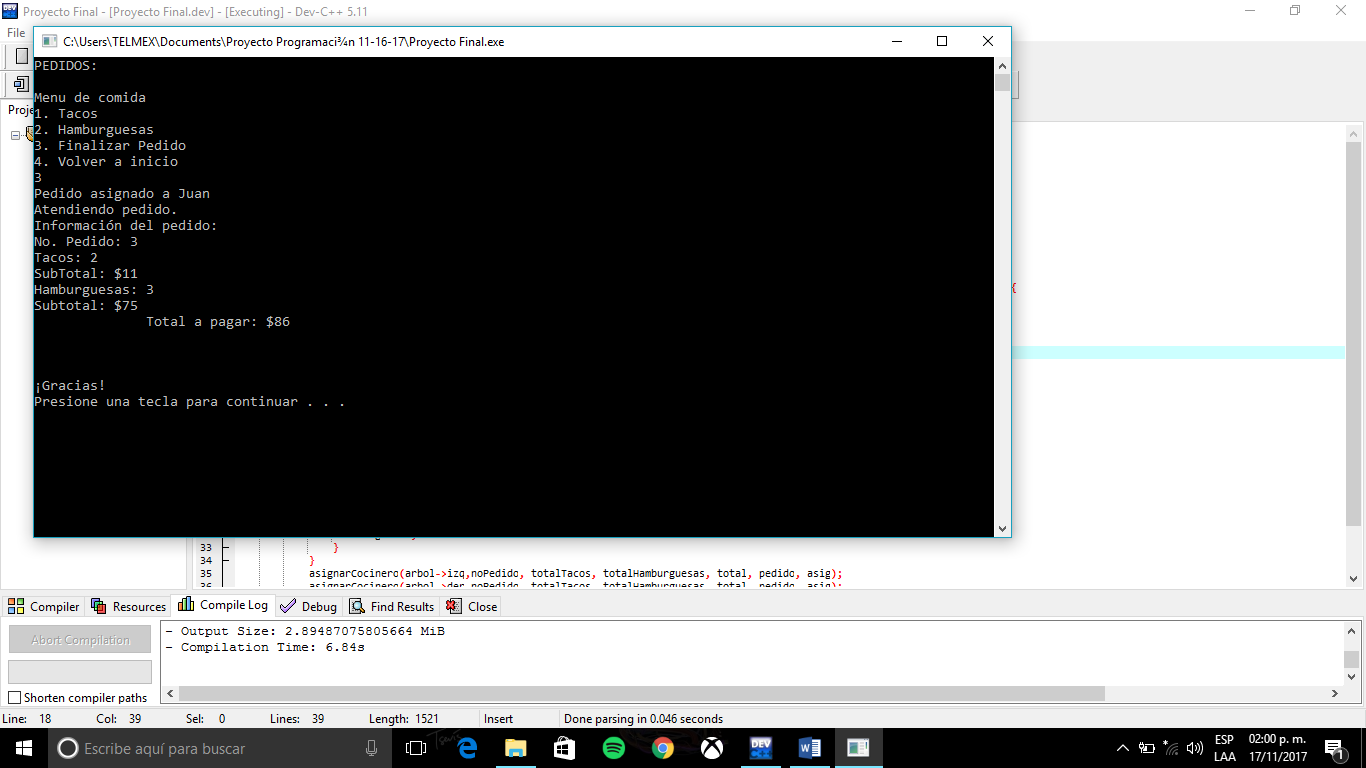
Para acceder a la opción de Personal, no necesita de una contraseña, simplemente se le direccionará a una nueva pantalla en donde se mostrarán un menú para que seleccione la opción de lo que desea pedir en el establecimiento, tacos y/o hamburguesas, si desea agregar más tacos o hamburguesas, presione nuevamente la opción que le permite agregar el producto deseado e inserte la cantidad que desea agregar. Una vez seleccionada la cantidad de tacos y/o hamburguesas que desee, presione 3 para finalizar el pedido. El sistema le asignará el pedido a un cocinero que se encuentre desocupado de manera automática, cuando lo encuentre, se le notificará al momento de finalizar el pedido mostrando el nombre del cocinero y los respectivos datos de la compra (ilustración 4).

Ilustración Datos de la compra

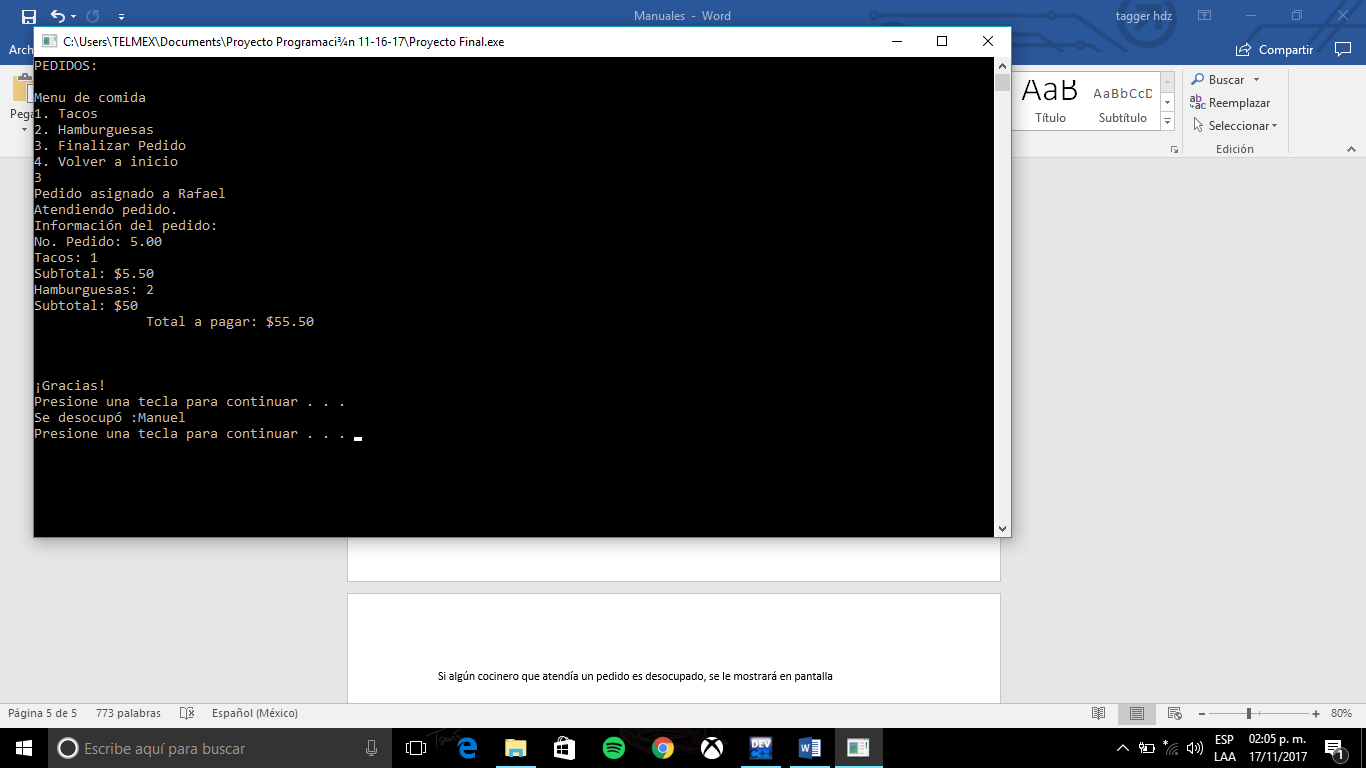
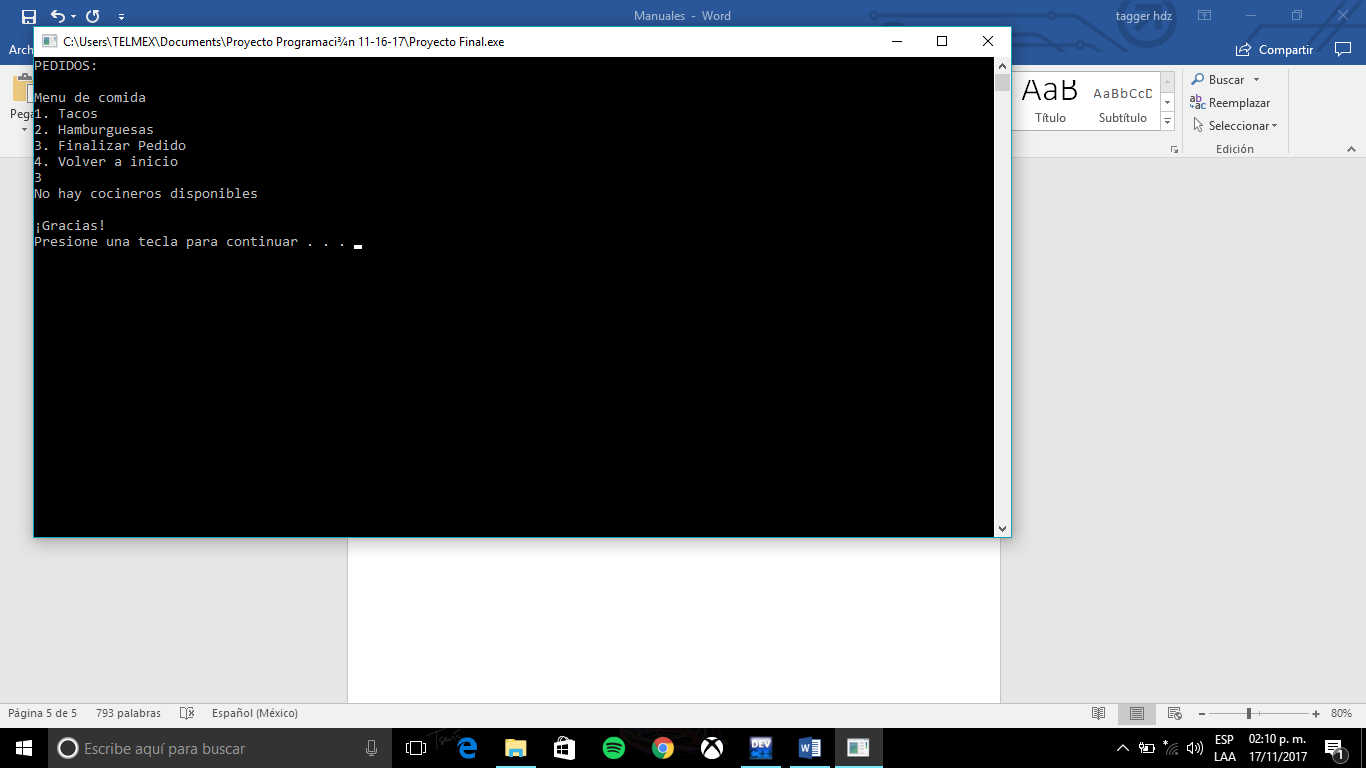
Si algún cocinero que atendía un pedido es desocupado, se le mostrará en pantalla. El cocinero que se desocupe estará disponible la próxima vez que se haga un pedido (ilustración 5). 

Ilustración Disponibilidad del cocinero

Si se desea agregar un nuevo pedido y todos los cocineros están ocupados, el sistema le mandará un mensaje de alerta, por lo tanto, no se podrán realizar pedidos hasta que alguno de los cocineros se desocupe (ilustración 6).

Si se ha elegido la opción de realizar pedido sin antes haber agregado tacos y/o hamburguesas, el sistema no deja realizar dicho pedido, se tiene que agregar forzosamente una cantidad de cualquiera de los productos que se venden.

Ilustración Disponibilidad de cocineros

Para volver al menú principal del sistema, presione 4.

# Manual de programador

Para la realización de este proyecto se utilizó el software libre Dev-C++, en su versión 5.6.1.

## Archivos

Dentro de la elaboración del proyecto, se emplearon dos archivos con extensión .xls, “cocineros.xls” y “pedidos.xls”. El archivo “cocineros.xls” se utilizó para guardar los datos relacionados con los cocineros que se registran por el administrador a ejecutar el programa. Los datos que se guardaron en el archivo son:

* ID del cocinero, generado de manera aleatoria al momento de elegir la opción de Agregar cocinero.
* Nombre del cocinero que puede ser cualquier nombre.
* Un promedio que se asigna de manera automática con un valor inicial de 0, mismo que cambiará conforme el cocinero atienda pedidos.
* Un status, que es introducido por medio del teclado por el usuario al momento de agregar el cocinero, validado para que solamente se puedan agregar “A” o “I” (A – activo; I – inactivo).
* Número de pedidos atendidos por el cocinero y ganancias generadas por el cocinero, ambos datos inicializados con 0, al igual que el promedio, estos irán cambiando conforme el cocinero atienda pedidos.

El archivo “pedidos.xls” se utilizó para guardar los datos referentes a los pedidos que se generan en la tienda, esto con la finalidad de poder realizar un reporte de ventas y obtener cuántos pedidos se generaron en el día y el total de las ganancias. Los datos que guarda el archivo son:

* Número de pedido. Utilizando un contador para cada vez que se realice un pedido.
* Cantidad de tacos y hamburguesas (cada uno en una columna), para poder realizar operaciones y determinar el total a pagar.
* Total de la compra. Con la suma del subtotal de los tacos y el subtotal de las hamburguesas.

## Estructuras

Para el registro y guardado de los cocineros se utilizó un ABB (árbol binario de búsqueda), donde el id generado al momento de realizar un registro es el que permite que se ordenen los registro en el árbol. A continuación se muestran fragmentos de código en donde se puede observar la declaración del ABB, junto con sus variables.

struct nodo{

float promedio;

int id\_cocinero;

char nombre[20];

char status[10];

char turno[10];

float pedidosA;

float ganancias;

struct nodo \*izq, \*der;

};

typedef struct nodo \*ARBOLB;

ARBOLB arbol = NULL;

Se declara un nuevo tipo de dato ARBOLB y una nueva variable de tipo ARBOLB llamada árbol. Ésta variable será el árbol que contendrá la información de los cocineros conforme se agreguen por el administrador.

* float promedio guarda el promedio del cocinero para atender pedidos. Inicializado en 0 y actualizado cada que el cocinero atienda una orden.
* int id\_cocinero guarda el ID que identifica al cocinero y variable que permite la movilidad en el ABB para guardar los nodos.
* char nombre guarda el nombre del cocinero.
* char status almacena el status del cocinero, ya sea “A” (activo) o “I” (inactivo).
* char turno guarda si el cocinero se encuentra ocupado “O” o desocupado “D”. Se inicializa con “D”.
* float pedidosA es el número de pedidos que atiende el cocinero. Inicializa en 0.
* float ganancias representa el total de ganancias que genera el cocinero, inicializado en 0

## Librerías

Se utilizaron un total de 6 librerías predefinidas por Dev-C++, las cuales son:

* #include<iostream> Para utilizer los métodos “cout” y “cin”, escritura y lectura, respectivamente
* #include<stdlib.h> Utilizado para generar una semilla que inicializa los números aleatorios al momento de asignar el ID del cocinero. También para reservar memoria en el nodo mediante la función malloc, contenida en esta librería. Atoi() devuelve el numero de una cadena *char* en entero *int.*
* #include<iomanip> Se utilizan los métodos “setw()”, “fixed”, “setprecision()” gracias a esta librería.
* #include<string.h> Utilizada para poder realizer validaciones de introducción de datos por el usuario.
* #include<time.h> Inicializa la semilla de acuerdo al horario del sistema.
* #include<windows.h> Función Sleep().

Además de estas librerías, se utilizaron librerías creadas por el usuario de Dev-C++, el desarrollo de la aplicación se llevó a cabo mediante proyecto, lo cual permite crear las librerías que el usuario crea pertinentes. Las librerías son:

* #include"validaciones.h"
* #include"agregar.h"
* #include"recuperar.h"
* #include"modificar.h"
* #include"actualizar.h"
* #include"asignar.h"
* #include"administrador.h"
* #include"personal.h"
* #include"creditos.h"

### Valores definidos

Se crearon 3 *#define*:

* #define CONTRASENIA "upslp123"
  + Que guarda la contraseña para poder acceder a las funciones de administrador
* #define precioTacos 5.5 y #define precioHamburguesa 25
  + Precios predefinidos para los tacos y las hamburguesas

### Validaciones.h

* int validarNum(char []); int validarModif(char []); int validarAdmin(char []);
  + En estas funciones se validan los datos que introdujo el usuario al momento de seleccionar una acción en la aplicación, es decir, que lo introducido sea un número no mayor de un dígito, que la cadena ingresada sea un dígito y no una letra o un carácter, finalmente, si la cadena introducida pasa por estos filtros, se convierte el número de tipo *char* a uno tipo *int* y se devuelve este número a donde fue llamada la función.
* bool logIn();
  + Esta función permite validar que el usuario ingrese correctamente la contraseña definida como *#define CONTRASENIA "upslp123"* mediante la función strcmp(), contenida en la librería string.h. También se valida que, si el usuario no conoce la contraseña de administrador, al presionar 0 se devuelva al menú principal.
* bool validarStatus(char []);
  + Función que permite validar que la cadena introducida por el usuario sea solamente de un carácter, y que este sea una letra “A” o “I”.
* bool validarId(char []);
  + Dado que los ID’s generados son de 6 dígitos, se valida que la cadena que introduce el usuario sea de 6 caracteres, posteriormente con una función *for* se verifica caracter por carácter que no haya una letra, si encuentra una letra, retorna false. Mediante la función atoi se comprueba que el ID se encuentre entre el rango de ID’s generados, si cumple esta condición, se devuelve un valor de true.
* bool validaTH(char []);
  + Comprueba que el usuario no introduzca una cadena de más de tres caracteres, posteriormente, comprueba que esta cadena sean dígitos. Retorna true si cumple estas condiciones.

### Agregar.h y Recuperar.h

Se observa que en la función “*insertar”* se ejecuta de manera recursiva cada que el árbol sea diferente de nulo, es decir, cada vez que se encuentren datos en el nodo. Cuando encuentre un nodo vacío o NULL, se llama a la función “*crearNodo”* con argumento del id que se generó aleatoriamente, donde se registrarán los datos del cocinero. Dentro de la función “*crearNodo”* se llenan los datos del nuevo nodo creado del árbol. Se observa que una vez que los datos se guardan en el árbol, de la misma manera se guardan en el archivo “cocineros.xls”. La variable de tipo FILE sirve como puntero para poder abrir el archivo con el nombre previamente dicho, permitiendo que se guarden los datos. Para guardar el status en el árbol se utiliza una validación para que el usuario solamente pueda introducir “A” o “I”.

bool insertar (ARBOLB &arbol, int id){ //insertar nuevo nodo

if(arbol == NULL){

arbol=crearNodo(id);

return true;

}

else

if(id < arbol->id\_cocinero)

insertar(arbol->izq, id);

else

if(id > arbol->id\_cocinero)

insertar(arbol->der, id);

}

Cabe recordar que un árbol tiene dos punteros “izq” y “der”, a estos se les debe de asignar el valor de NULL, indicando que el nodo actual es una hoja del árbol, cada que se agregue un nuevo nodo partirán de estos punteros, reasignando los punteros “izq” y “der” del nuevo nodo asignado.

ARBOLB crearNodo(int id){ //crear nodo y guardar en el archivo

bool rep;

char promedio[10];

ARBOLB nuevoNodo = new(struct nodo);

FILE \*archivo;

char status[10];

archivo=fopen("cocineros.xls", "a");

if(archivo!=NULL){

nuevoNodo->id\_cocinero=id;

cout<<"ID generado: "<<nuevoNodo->id\_cocinero<<endl;

cout<<"Ingresa nombre: ";

cin>>nuevoNodo->nombre;

fflush(stdin);

nuevoNodo->promedio=0;

strcpy(nuevoNodo->turno, "D");

nuevoNodo->pedidosA=0;

nuevoNodo->ganancias=0;

do{

rep=true;

cout<<"Status cocinero: ";

cin>>status;

rep=validaStatus(status);

}while(rep==false);

strcpy(nuevoNodo->status,status);

fprintf(archivo, "%d\t", nuevoNodo->id\_cocinero);

fprintf(archivo, "%s\t", nuevoNodo->nombre);

fprintf(archivo, "%.2f\t", nuevoNodo->promedio);

fprintf(archivo, "%s\t", nuevoNodo->status);

fprintf(archivo, "%s\t", nuevoNodo->turno);

fprintf(archivo, "%.2f\t", nuevoNodo->pedidosA);

fprintf(archivo, "%.2f\n", nuevoNodo->ganancias);

}

fclose(archivo);

nuevoNodo->izq = NULL;

nuevoNodo->der = NULL;

cout<<"Cocinero agregado exitosamente."<<endl;

system("pause");

return nuevoNodo;

}

Para recuperar los datos de un archivo y guardarlos en el árbol en caso de que el programa se haya cerrado, se utilizan las funciones *void recuperarDatos(), void recuperando()* y *ARBOLB recuperarNodo()* contenidas en la librería *recuperar.h* empleando el mismo algoritmo que en las funciones void *insertar()* y *ARBOLB crearNodo()*. La diferencia consiste en que en la función *void recuperarDatos()*primeramente se crea una variable de tipo FILE con un apuntador llamado *recuperarArchivo*.

void recuperarDatos(ARBOLB &arbol){//recuperar datos de cocineros de archivo

FILE \*recuperarArchivo;

int id;

char nombre[20], status[10], turno[10];

float promedio, ganancias,pedidosA;

recuperarArchivo=fopen("cocineros.xls", "r");

if(recuperarArchivo!=NULL){

while(!feof(recuperarArchivo)){

fscanf(recuperarArchivo, "%d\t", &id);

fscanf(recuperarArchivo, "%s\t", &nombre);

fscanf(recuperarArchivo, "%f\t", &promedio);

fscanf(recuperarArchivo, "%s\t", &status);

fscanf(recuperarArchivo, "%s\t", &turno);

fscanf(recuperarArchivo, "%f\t", &pedidosA);

fscanf(recuperarArchivo, "%f\t", &ganancias);

recuperando(arbol, nombre, id, promedio, status, turno, pedidosA, ganancias);

}

}else{

fclose(recuperarArchivo);

remove("cocineros.xls");

}

fclose(recuperarArchivo);

}

Esta variable se encarga de abrir el archivo “cocineros.xls” y va leyendo cada tupla que contiene el archivo, estos datos son guardados en variables que una vez leídos del archivo, se mandan como argumentos en la función *void recuperando()*, una vez en esta función, se ejecuta la misma lógica y algoritmo para guardar datos en el árbol tanto en esta función como en *ARBOLB recuperarNodo()* cuando el nodo del árbol sea nulo en *void recuperando().*

void inorModificar(ARBOLB arbol, int id){//modificar datos de un nodo del arbol

char opc[20], status[10];

int validado;

bool rep;

if(arbol!= NULL){

inorModificar(arbol->izq, id);

if(id==arbol->id\_cocinero){

//impresión de los datos del cocinero y opción a elegir por el usuario

switch(validado){

case 1:

cout<<"Ingrese nuevo nombre: ";

cin>>arbol->nombre;

break;

case 2:

cout<<"Ingrese cambio de status: ";

do{

rep=true;

cin>>status;

rep=validaStatus(status);

}while(rep==false);

strcpy(arbol->status,status);

break;

}

cout<<"Cocinero modificado."<<endl;

system("pause");

}

inorModificar(arbol->der, id);

}

}

### Modificar.h

*void inorModificar(ARBOLB, int);* realiza un recorrido “inOrden” por el árbol a partir del ID introducido por el usuario buscando que halle una coincidencia, una vez encontrado el nodo con el ID del cocinero, imprime los datos. Se pide que seleccione una opción, modificar nombre del cocinero o modificar status del cocinero. Dependiendo la opción elegida los valores del nodo del árbol son sustituidos por los ingresados por el usuario.

### Actualizar.h

La librería contiene dos funciones: *void preActualizar(ARBOLB);* y *void actualizarDatos();*

void preActualizar(ARBOLB arbol){//actualizar los datos en un archivo auxiliar

FILE \*archivo;

if(arbol!= NULL){

archivo=fopen("auxiliar.xls", "a");

if(archivo!=NULL){

fprintf(archivo, "%d\t", arbol->id\_cocinero);

fprintf(archivo, "%s\t", arbol->nombre);

fprintf(archivo, "%f\t", arbol->promedio);

fprintf(archivo, "%s\t", arbol->status);

fprintf(archivo, "%s\t", arbol->turno);

fprintf(archivo, "%f\t", arbol->pedidosA);

fprintf(archivo, "%f\n", arbol->ganancias);

}

fclose(archivo);

preActualizar(arbol->izq);

preActualizar(arbol->der);

}

}

*void preActualizar(ARBOLB);* permite guardar en un archivo auxiliar los datos del árbol hasta el momento en que es llamada la función mediante el método de búsqueda preOrden. La variable de tipo FILE *archivo* permite abrir el archivo “auxiliar.xls” para ir agregando los datos del nodo del árbol.

*void actualizarDatos();* utilizar dos punteros de tipo FILE, *archivo y auxiliar*. El primero se utiliza para abrir el archivo en el que se actualizarán los datos, en este caso “cocineros.xls” mientras que *auxiliar* se utilizará para abrir y leer los datos de “auxiliar.xls” creado en la función *void preActualizar(ARBOLB);* es por esto que se debe de llamar a esta función antes de abrir los punteros de tipo FILE. Con el archivo “auxiliar.xls” creado, se abre nuevamente el archivo con el puntero *auxiliar* y el puntero *archivo* en la función *void actualizarDatos();* para leer los datos y, por cada tupla, se imprimen en el archivo original “cocineros.xls”. Es importante que *archivo* que abre el documento “cocineros.xls” se abra en modo de escritura (w), para que borre los datos anteriores y los sustituya con los datos que se extraigan de “auxiliar.xls”. Finalmente, cuando se terminen de copiar los datos, se cierran ambos archivos y mediante la función remove() se elimina el archivo “auxiliar.xls”.

void actualizarDatos(){//pasar los datos del archivo auxiliar al archivo principal

FILE \*archivo, \*auxiliar;

int id;

float promedio, ganancias,pedidosA;

char nombre[20], status[10], turno[10];

preActualizar(arbol);

archivo=fopen("cocineros.xls", "w");

auxiliar=fopen("auxiliar.xls", "r");

if(auxiliar!=NULL){

while(!feof(auxiliar)){

fscanf(auxiliar, "%d\t", &id);

fscanf(auxiliar, "%s\t", &nombre);

fscanf(auxiliar, "%f\t", &promedio);

fscanf(auxiliar, "%s\t", &status);

fscanf(auxiliar, "%s\t", &turno);

fscanf(auxiliar, "%f\t", &pedidosA);

fscanf(auxiliar, "%f\n", &ganancias);

if(archivo!=NULL){

fprintf(archivo, "%d\t", id);

fprintf(archivo, "%s\t", nombre);

fprintf(archivo, "%f\t", promedio);

fprintf(archivo, "%s\t", status);

fprintf(archivo, "%s\t", turno);

fprintf(archivo, "%f\t", pedidosA);

fprintf(archivo, "%f\n", ganancias);

}

}

}

fclose(auxiliar);

fclose(archivo);

remove("auxiliar.xls");

}

### Asignar.h

Para asignar pedidos a los cocineros, se utiliza la función *void asignarCocinero();* donde se realiza un recorrido preOrden en el árbol. Para poder asignar un pedido al cocinero se deben de cumplir tres condiciones, que en el nodo del árbol el status del cocinero sea igual a “A”, que su asignación o turno sea igual a “D” y que el valor de una variable sea igual a false.

void asignarCocinero(ARBOLB &arbol, float noPedido, int totalTacos, int totalHamburguesas, float total, bool &pedido, bool &asig){

if(arbol!= NULL){

if(strcmp(arbol->turno, "D")==0 && strcmp(arbol->status, "A")==0){

if(pedido==false){

strcpy(arbol->turno, "O");

arbol->pedidosA++;

arbol->promedio=(arbol->pedidosA/noPedido)\*100;

cout<<"Pedido asignado a "<<arbol->nombre<<endl;

cout<<"Atendiendo pedido."<<endl;

cout<<"Información del pedido: "<<endl;

cout<<"No. Pedido: "<<noPedido<<endl;

cout<<"Tacos: "<<totalTacos<<endl;

cout<<"SubTotal: $"<<totalTacos\*precioTacos<<endl;

cout<<"Hamburguesas: "<<totalHamburguesas<<endl;

cout<<"Subtotal: $"<<totalHamburguesas\*precioHamburguesa<<endl;

total=totalTacos\*precioTacos+totalHamburguesas\*precioHamburguesa;

arbol->ganancias=arbol->ganancias+total;

cout<<setw(30)<<"Total a pagar: $"<<total<<endl<<endl<<endl;

pedido=true;

asig=true;

}

}

asignarCocinero(arbol->izq,noPedido, totalTacos, totalHamburguesas, total, pedido, asig);

asignarCocinero(arbol->der,noPedido, totalTacos, totalHamburguesas, total, pedido, asig);

}

}

Cuando la función es llamada, trae como argumentos el árbol, los datos del pedido hecho y dos variables de tipo boleano que a continuación se explicará su funcionamiento.

Dado que la función es recursiva porque se realiza un recorrido por el árbol, es importante saber en qué momento se ha asignado el pedido y a quien. Cuando se ha asignado el pedido a un cocinero, se cambian ciertos valores del nodo, el primero de ellos es el del turno o asignación, dado que “D” significa que el cocinero está desocupado, cambiar esta letra por “O”, ese nodo no permitirá que se le hagan cambios. Luego se aumenta la variable del nodo *arbol->pedidosA* para indicar el número de pedidos que ha atendido el cocinero, posteriormente se cambia el promedio calculando el total de pedidos que ocurren en el día entre *arbol->pedidosA.* Finalmente a la variable pedido y asig se les asigna el valor de true. Pedido = true nos indica que el pedido ya ha sido atendido por un cocinero, de modo que, aunque encuentre otro nodo que cumpla con la condición *arbol->turno= “D”* y *arbol->status=”A”* si la segunda condición no se cumple, no se asigna el pedido. Esto evita que el pedido sea atendido por dos cocineros a la vez.

Asig = true es el valor que se compara en donde fue llamada la función, su valor true indica que el pedido fue atendido, por lo tanto si se cumple esta condición se guardan los datos del pedido en el archivo llamado “pedidos.xls”.

*void desocuparCocinero(ARBOLB, int);* realiza nuevamente un recorrido preOrden por el árbol comparando que *arbol->turno = “O”* y que *id = arbol->id\_cocinero*, *id* fue generado con un aleatorio previo a llamar a la función *void desocuparCocinero(ARBOLB, int);* donde se busca una coincidencia de estos valores con los de algún nodo del árbol, cuando las encuentre, a *arbol->turno* le da el valor de *“D”* indicando que el cocinero de ese nodo ha sido desocupado, por lo tanto podrá atender un pedido nuevo.

void desocuparCocinero(ARBOLB &arbol, int id){

if(arbol!=NULL){

if(id==arbol->id\_cocinero && strcmp(arbol->turno, "O")==0){

strcpy(arbol->turno, "D");

cout<<"Se desocupó :"<<arbol->nombre<<endl;

system("pause");

}

desocuparCocinero(arbol->izq, id);

desocuparCocinero(arbol->der, id);

}

}

### Administrador.h

En esta librería se encuentran todas las funciones de las acciones que puede realizar el administrador del sistema. Entre ellas se encuentran *bool agregarCocinero();* que genera un ID aleatorio y es mandado como parámetro junto con el árbol de los cocineros en la función *void insertar(ARBOLB, int);* cuando se agrega un nuevo cocinero, imprime un mensaje de confirmación.

*bool consultaCocinero()* que manda llamar a la función *void inor(ARBOLB);*  e imprime el ABB con los datos de los cocineros mediante un recorrido inOrden.

*bool modificarCocinero()* esta función recupera los datos del árbol mediante la función *void recuperarDatos(ARBOLB);* y posteriormente los imprime utilizando la función *void inor(ARBOLB);* se pide al usuario el ID del cocinero que se desea modificar y este ID es enviado como parámetro junto con el ABB en la función *void inorModificar(ARBOLB, int);* finalmente se llama a la función *void actualizarDatos();* para actualizar el archivo con las modificaciones hechas.

*bool reporteVentass();* utiliza una variable de tipo FILE para abrir el archivo “pedidos.xls” y leer los datos que contiene este archivo. La función imprime un reporte de ventas que incluye los datos del establecimiento, el número de pedidos que fueron atendidos y el total de las ganancias hechas hasta el momento.

*bool reporteCocinero();* al igual que *bool modificarCocinero()* , esta función recupera los datos del ABB, los imprime utilizando la función *void inor(ARBOLB);* y nuevamente se pide que ingrese el ID del usuario que se desea generar el reporte de ventas. Posteriormente se llama a la función *void inorReportar(ARBOLB, int);* con ABB y el ID ingresado por el usuario como argumentos.

*bool administrador();* esta es la función principal que maneja el administrador, en ella son llamadas cada una de las funciones anteriores. Mostrando un menú de lo que se puede hacer, el usuario elige una opción y utilizando un *switch* se manda llamar a la función de acuerdo a la elección hecha.

Todas las funciones retornan un valor false, para poder regresar al menú principal de la aplicación.

### Personal.h

if(totalHamburguesas==0 && totalTacos==0){

cout<<"Agregue tacos y/o hamburguesas."<<endl;

system("pause");

}else{

noPedido++;

asignarCocinero(arbol, noPedido, totalTacos, totalHamburguesas, total, pedido, asig);

if(asig==true){

total=totalTacos\*precioTacos+totalHamburguesas\*precioHamburguesa;

archivo=fopen("pedidos.xls", "a");

fprintf(archivo, "%f\t", noPedido);

fprintf(archivo, "%d\t", totalTacos);

fprintf(archivo, "%d\t", totalHamburguesas);

fprintf(archivo, "%.2f\n", total);

fclose(archivo);

}

else cout<<"No hay cocineros disponibles"<<endl;

}

cout<<endl<<"¡Gracias!"<<endl;

system("pause");

id=160726+rand()%11;

desocuparCocinero(arbol, id);

actualizarDatos();

rep=true;

Esta librería contiene la función *bool personal();* aquí se manejan los pedidos que se van a realizar. Se manda llamar a la función *void recuperarDatos()* para tener los datos actualizados de los cocineros. Posteriormente se utiliza una variable de tipo FILE para abrir el archivo “pedidos.xls” para recuperar el último número de pedido que se realizó. Posteriormente se muestra un menú con las opciones de agregar tacos y/o hamburguesas a la orden. Con un *switch,* a los contadores para el total de tacos y hamburguesas se le agrega la cantidad que ingresó el usuario para posteriormente calcular el total de la venta. Cuando se selecciona la opción “*finalizar pedido*”, se comprueba que el total de los tacos y de las hamburguesas no sean iguales a 0. Posteriormente se manda llamar a la función *void asignarCocinero();* con los datos de la venta. Con el valor retornado de esa función se hace una comparación donde, si el valor asig = true, se imprimen en el archivo los datos de la compra, de otro modo, asig= false significa que no se asignó el pedido, por lo tanto se imprime el mensaje *“No hay cocineros disponibles”*.

Una vez realizado esto, se genera un número aleatorio con los ID’s de los cocineros y es mandado como argumento junto con el ABB a la función *void desocuparCocinero();*

Finalmente se actualizan los datos del ABB.

### Creditos.h

Librería que contiene *void creditos()* e imprime los datos de los autores de la aplicación, junto con algunos de sus datos y datos de la institución a la que pertenecen.