Food Notifier**®**

Informe de Software

**Participantes**

* Pedelhez, Matías Gabriel;
* Soto, Micaela Jazmín Elizabeth;
* X, Gabriela Agustina;
* Zini, Franco Joaquín.

**Descripción del proyecto**

**Food Notifier®** es un producto que se encarga de la administración inteligente de clientes dentro de los comercios. Entre sus funciones permite asignar turnos de forma dinámica a distintos clientes e informar, mediante un dispositivo inalámbrico, el momento en el cual el turno se encuentra disponible.

**Justificación del proyecto**

**Food Notifier®** encuentra su lugar en el mercado permitiendo a las empresas comunicarse con sus clientes de manera efectiva, rápida y de forma enlazada, sin preocuparse por problemas individuales que pudiesen surgir en cada usuario, como ser la imposibilidad de recibir el estado de su pedido (p. ej. si las notificaciones se hiciesen a través de una aplicación móvil y no desde un producto en sí), convirtiéndose en una opción sumamente confiable.

**Planificación del proyecto**

Se realizó una primera reunión mediante la plataforma Discord, con el motivo de debatir ideas sobre la funcionalidad del programa. Una vez decidida fueron planeadas otras reuniones (a través de Meet y Discord), además de la escritura del código y su publicación en la página GitHub para así intervenir en los avances logrados.

**Descripción de etapas**

**Etapa 1: Semana de repaso**

1. Se crea un tablero en Trello, donde se colocan las actividades pendientes de realizar y aquellas concretadas.
2. Se crea un archivo en lenguaje “C” que permite el ingreso de datos y almacenamiento de los mismos en forma binaria.

**-Análisis** - 10 de junio

Estudio del problema a fin de determinar los requerimientos del programa.

**-Diseño** - 12 de julio

Determinación del algoritmo de solución:

1. **Mostrar pantalla de inicio**
   1. esperarIntro()
   2. firstScreen()
   3. mostrarFechaYHora()
   4. mensajeBienvenida()
   5. mensaje()
2. **Mostrar menú con las siguientes opciones:**
   1. **Crear archivo .dat:** apertura del archivo .dat en modo escritura a través de la función generarBinario ().
   2. **Ingresar pedidos y guardar -grabar- comprobantes:** hay una función llamada “ingresarDatosClientes()” que contiene a su vez tres funciones secundarias: ingresarPedidoBebida(), ingresarPedidoComida() y ingresarIdCliente(). En las dos primeras, se muestran al usuario las opciones de bebida y comida mediante dos vectores opcionComidas y opcionBebidas, a los cuales se les asigna de antemano como valor los nombres de las diferentes opciones de comida, y bebida, respectivamente. Asimismo, se inicializan otros dos vectores que tienen por función contener los precios: preciosComidas y preciosBebidas. A sabiendas de que un cliente puede solicitar más de un tipo de comida y bebida en un mismo pedido, el usuario tiene la posibilidad de ingresar tantas opciones como sean necesarias. Una vez terminada la carga del identificador del cliente y los pedidos de comida y bebida, se muestran al usuario los datos cargados en el registro a través de la función mostrarPedidoCliente (). Dicha función, a su vez calcula el total acumulado del pedido mediante la función calcularCuentaTotal(), la cual se vale de los arreglos mencionados previamente.

Posteriormente, el usuario tiene la posibilidad de grabar el registro mediante la función grabarPedido().

Cabe mencionar, que se trabaja con una estructura de tipo tPedidoCliente, que incluye el número identificador del cliente, los pedidos de comida y bebida y el total gastado.

* 1. **Finalizar la carga:** cierre del archivo mediante la función cerrarArchivo ().
  2. **Mostrar los pedidos grabados en el archivo .dat:** haciendo uso de la función abrirArchivoLectura (), se abre el archivo binario en modo lectura y se procede a leer cada registro -leerRegistro ()-, a mostrar los datos del mismo -mostrarPedidoClienteDeArchivo ()- y verificar que si se encuentra ante el último -feof ()-, como condición de salida del while. Dichas funciones se encuentran contenidas en la función mostrarPedidosGrabados().
  3. **Salir**

**-Codificación en lenguaje .c, compilación, ejecución y verificación** - 12 al 14 de julio-

Se hizo uso de la metodología de pair programming, donde uno de los compañeros programaba y el resto comentaba. Ello se hizo de manera sincrónica -a través de Discord y Meet-.

**Etapa 2: Estructuras compuestas: listas (implementación estática mediante arrays)**

**-Análisis y Diseño** -29 de agosto al 4 de septiembre-

Determinación de las modificaciones a realizar en el algoritmo:

1. En cuanto a las estructuras, se incorpora “tCola”, la cual guarda el registro del pedido e indica el frente y el final de la misma.
2. Se modifica la función de ingreso de datos, para almacenar los registros en la cola “totalPedidos”.
3. Se agregan funciones para mostrar todos los datos de la cola -visualizarElementos (tCola)- y eliminar algún pedido en función del número identificador del cliente ingresado por el usuario, haciendo uso de la modalidad de frente fijo.
4. También se incorporan las funciones “inicializarCola ()”, “colaVacía ()” y “colaLlena ()”, como auxiliares para el tratamiento de los datos mediante listas estáticas de tipo cola.
5. En razón de que deben grabarse los registros a medida que se recorre la lista de pedidos, se implementa una función que recibe como parámetro la cola -grabarArchivoConsumoClientes(tCola)- y a su vez, dentro de una estructura de control “for”, llama a otra función -grabarRegistros(int)- que se encarga de grabar, propiamente, tantas veces como sea necesario hasta llegar al final de dicha cola.

-**Codificación en lenguaje .c, compilación, ejecución y verificación** -5 al 12 de septiembre-

Al igual que en la etapa anterior, se recurrió al método de programación de a pares. En contraste, se incorporó la modalidad de trabajo asincrónica. Ello fue posible gracias al portal GitHub, donde se subió el código en .c del programa, a fin de administrar el proyecto de manera más organizada. Ello permitió que los programadores revisores pudieran ver asincrónicamente el código y efectuar solicitudes de modificaciones. Además, facilitó el trabajo, en tanto tiene la función de comparar versiones, fusionarlas, e incluso restaurarlas.