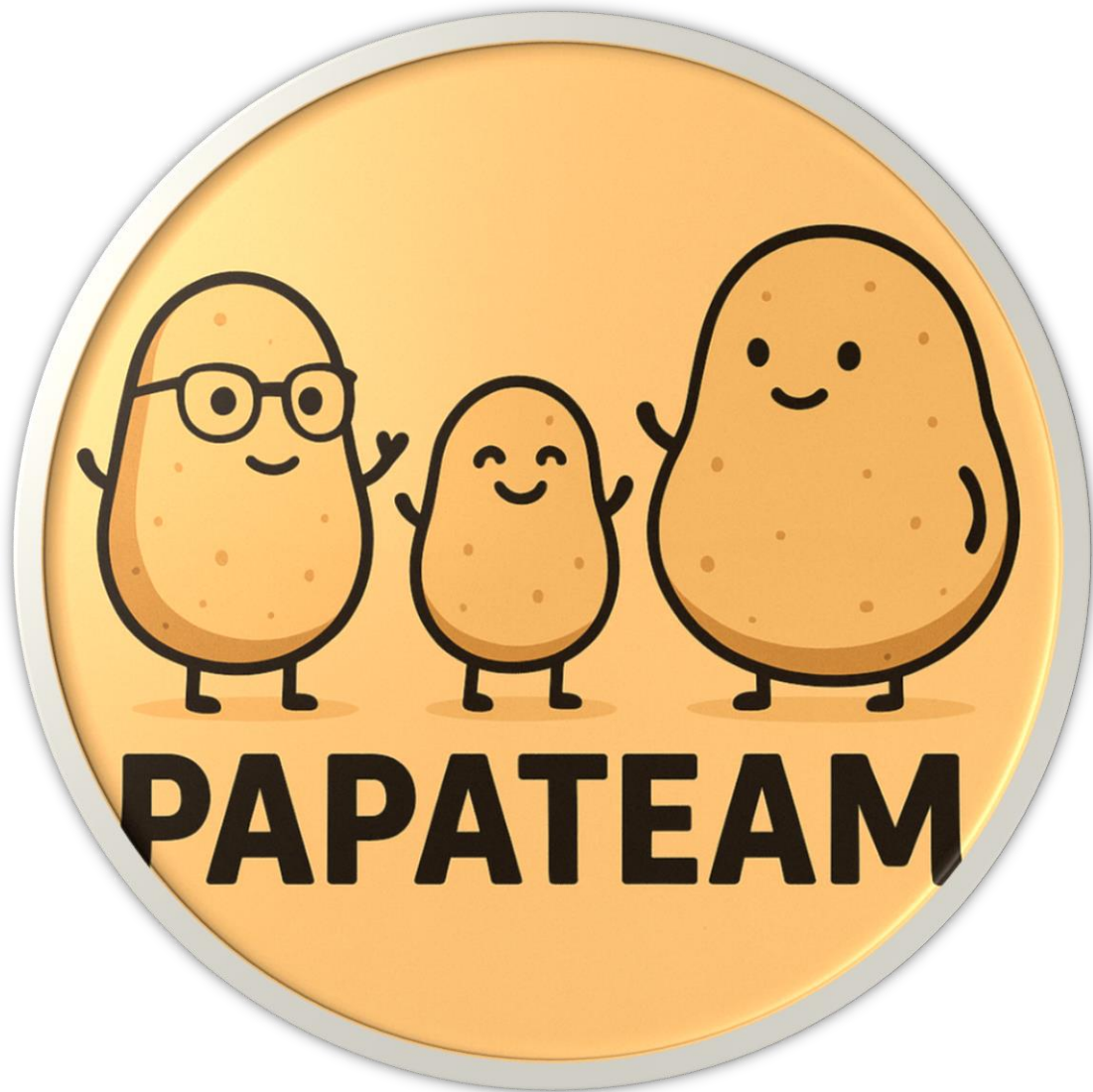


TRABAJO PRACTICO N°2

ALGORITMOS Y ESTRUCTURAS DE DATOS II



Docente:

- Mónica Nano

Alumnos:

- Agostini Santiago
- Pineda Agustin Nahuel
- Poviña Alejo

OBJETIVOS DE LA ACTIVIDAD

- Implementar un **Montículo Binario** con inserción, consulta de raíz, extracción y heapify.
- Separar la **lógica algorítmica** de la **visualización** usando C++ y SFML.
- Elegir y justificar un **caso real** en el que se aplique el montículo.
- Utilizar la **Inteligencia Artificial Generativa (IAG)** como **compañera de codificación** para mejorar el diseño, la refactorización y las pruebas.
- Presentar un informe claro y ordenado que documente decisiones, pruebas y resultados.

CONSIGNAS:

1. Implementación algorítmica

- a. Crear una clase Heap basada en `std::vector<int>`.
- b. Implementar: insertar, raíz, extraerRaíz, heapify y, opcionalmente, heapsort.
- c. Probar con el dataset de la clase: 10, 20, 30, 15, 50, 12, 40, 7, 18, 45, 1.

2. Visualización con SFML

- a. Representar el árbol por niveles y el vector subyacente.
- b. Resaltar comparaciones e intercambios con colores o animaciones.
- c. Incluir controles básicos: insertar, extraer raíz, heapify y alternar Min/Max.

3. Caso real

- a. Elegir un escenario y justificar qué significa la prioridad.
- b. Simular entradas (insert) y salidas (extract), mostrando resultados y métricas (por ejemplo, tiempo de espera promedio).

4. Uso de IAG (obligatorio)

- a. Documentar **tres prompts** usados con el Asistente de la materia o Copilot.
- b. Explicar qué tomaste y qué descartaste de la respuesta.

- c. Incluir al menos un test de borde sugerido por la IAG (heap vacío, duplicados, gran cantidad de elementos).

EXPECTATIVAS DEL TRABAJO PRACTICO

Tenemos ciertas ideas a aplicar en este trabajo, los puntos ideales que queremos tratar son:

(Este punto se va a ir modificando respecto al avance del código, si hay algún “desafío” que se nos pueda otorgar sería lo idóneo. Además, iremos incrementando nosotros mismos las expectativas del trabajo)

- Tenga una interfaz en el menú
- Visualizar la inserción del nodo (contenido)
- Un menú que administre la cola de las estaciones de trabajo (plancha, frituras y horno)
- Establecer un valor fijo a cada opción para facilitar la carga de datos
- Simular entradas (insert) y salidas (extract), mostrando resultados y métricas (por ejemplo, tiempo de espera promedio).
- Identificar el tiempo de demora respecto a la cantidad de elementos cargados, es decir, visualizar una posible demora en la comida
- Generar un “código vivo”
- Generar una vinculación con la hora del dispositivo (mostrarlo también en el código)

ENTORNO DEL TRABAJO PRACTICO

Solicitud de desarrollo de software

La tienda de comida “Papateam” se encuentra en un proceso de modernización a la hora de realizar sus procesos internos y de atención al cliente. Con el fin de mejorar la organización del trabajo y ofrecer un servicio más eficiente, se nos solicitó el desarrollo de un software que les permita gestionar las comandas de manera sencilla y optima.

Actualmente, el registro y seguimiento de los pedidos se realiza de forma manual, lo que genera demoras, errores y dificultades para sacar los platos a tiempo.

Este nuevo sistema tiene como objetivo principal **optimizar el flujo de trabajo**, facilitando la comunicación interna, mayor precisión de tiempos y una interfaz amigable, logrando así, un mayor control sobre la atención al cliente.

INFORME TÉCNICO

INTRODUCCIÓN

En este trabajo practico tiene como enfoque el diseño y desarrollo de aplicaciones interactivas utilizando SFML para generar la interfaz gráfica. Tras haber mencionado previamente el entorno de desarrollo y los objetivos pasaremos a explicar el proceso y código del mismo.

DESARROLLO

- Visualización de Heap
- Insertar elementos del dataset provisto en el enunciado.
- Obtener y extraer la raíz.
- Heapify, para reordenar el arreglo en forma de montículo.
- Toggle entre heap mínimo y máximo.
- Heapsort paso a paso o automático, con registro de comparaciones, intercambios y nodos ordenados.

La interfaz muestra el heap en forma de árbol binario y como arreglo lineal. El sistema utiliza colores para destacar comparaciones, intercambios y elementos ya ordenados. El control se realiza mediante botones y utilizamiento del teclado, lo que permite una facilidad de uso.

GESTION DE PEDIDOS

Cada pedido contiene ítems con tiempos de preparación asignados, los cuales 1 minuto representa un 1 segundo.

El código está pensado para que los elementos repetidos no generen suma en el tiempo de elaboración (De un mismo pedido), dando lugar a una demora máxima de 45 segundos.

Se establecen horarios de demora en función de la hora local y se detectan automáticamente las horas pico (13:00 y 21:00), es decir, al ser un código vivo la relación de ejecución respecto a la hora del dispositivo dictará automáticamente la demora del mismo. *El código se ve representado en la siguiente imagen.*

```

// Calcula duracion total por tipos unicos
int calcularTotalSeg(const vector<OrderItem>& items) {
    set<string> unicos;
    for (auto& it : items) unicos.insert(it.tipo);
    int total = 0;
    for (auto& t : unicos) {
        auto it = DURACIONES.find(t);
        if (it != DURACIONES.end()) total += it->second;
    }
    return total;
}

// Genera entre 1 y 3 items, pudiendo repetir tipos (los repetidos no suman para el tiempo)
vector<OrderItem> generarItemsAleatorios(const string& tipoPreferente = "") {
    // Pool de tipos posibles
    static vector<string> pool = { "lomito", "hamburguesa", "papas", "postre", "pizza" };
    int n = 1 + (rand() % 3); // 1..3
    vector<OrderItem> out;
    auto pushOrInc = [&](const string& t) {
        auto it = find_if(out.begin(), out.end(), [&](const OrderItem& o) { return o.tipo == t; });
        if (it == out.end()) out.push_back({ t, 1 });
        else it->qty++;
    };
}

```

Los pedidos se eliminan automáticamente al cumplirse el tiempo de preparación, registrándose en un archivo .txt. *El código se ve representado en la siguiente imagen.*

```

void actualizarYEliminarVencidos() {
    time_t now = time(nullptr);
    pedidos.erase(remove_if(pedidos.begin(), pedidos.end(), [&](const Order& o) {
        double elapsed = difftime(now, o.timestamp);
        if (elapsed >= o.totalSeg) {
            cout << "Pedido #" << o.id << " eliminado automaticamente ("
                << o.totalSeg << "s)." << endl;

            ofstream archivo("../pedidos_finalizados.txt", ios::app);
            if (archivo.is_open()) {
                tm lt{};
                localtime_s(&lt, &now);
                char buffer[30];
                strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", &lt);

                archivo << "[Finalizado: " << buffer << "] Pedido #" << o.id << " → ";
                for (auto& it : o.items) {
                    archivo << it.tipo << "x" << it.qty << " ";
                }
                archivo << endl;
            }
        }
    }));
}

```

La interfaz gráfica permite agregar pedidos mediante botones diferenciados por color, visualizar los pedidos en curso y mostrar mensajes de advertencia en caso de demoras. *El código se ve representado en la siguiente imagen.*



Pedidos en curso:
Pedido #1 [papas x1 hamburguesa x1] Restante: 11s
Pedido #2 [papas x1] Restante: 2s
Pedido #3 [papas x2 postre x1] Restante: 9s

El código está pensado para que los elementos repetidos no generen suma en el tiempo de elaboración (De un mismo pedido), dando lugar a una demora máxima de 45 segundos.

PROBLEMAS DETECTADOS Y SOLUCIONES

Durante el desarrollo se presentaron algunos inconvenientes (En la bitácora diaria están nombrados):

- Carga de librerías: errores iniciales al vincular Ctime y SFML se resolvieron creando un proyecto de Visual Studio configurado correctamente para las dependencias gráficas.
- Gestión de archivos: el archivo .txt se generaba en rutas incorrectas. El problema estaba en el uso de mayúsculas en el nombre, y se solucionó normalizando el nombre en minúsculas.
- Optimización de interfaz: se ajustaron colores, tamaños y organización de elementos gráficos para mejorar la legibilidad y claridad de la simulación.

RESULTADOS

- El visualizador de heaps permite mostrar gráficamente las operaciones internas de la estructura.
- El sistema de pedidos integra lógica de tiempos, guardado de datos y simulación en tiempo real, representando la gestión de recursos.

UTILIZACIÓN DE IA

Prompts utilizados:

(El resultado del mismo está cargado en un .txt)

1-Puedes explicarme las siguientes advertencias:

En este caso las advertencias fueron explicadas porque sucedían y fueron solucionadas

2- puedes decirme como hacer una funcion para que se cree un .txt con los datos al final del pedido de ejecucion, de manera que se modifique con cada vez que se ejecuta el código?

Como respuesta fue explicado como se crea un .txt con los datos del pedido y fue introducido al código correctamente

3- podrias explicarme de donde surge el siguiente error? Gravedad Código
Descripción Proyecto Archivo Línea Estado suprimido Detalles
Error (activo) E0864 function no es una plantilla Ejemplo_elegido
C:\Users\santi\source\repos\Ejemplo_elegido\Ejemolo_elegido.cpp 114

Como respuesta de este error fue explicado correctamente y fue corregido

BITACORA DIARIA

La bitácora presentará un informe detallado de lo avanzado en cada día, avances, dificultades, decisiones, expectativas para el siguiente día y visiones a futuro.

Día 1 – 22/9/2025

Al comenzar este trabajo práctico determinamos la manera más adecuada de elaborar el informe. Tras conversar con la docente a cargo, Mónica Nano, concluimos que la realización de una bitácora nos permitirá reflejar con mayor precisión la evolución del trabajo.

También, decidimos implementar una vinculación en el código con la hora indicada por el dispositivo, lo que permite generar intervalos de tiempo para identificar la demanda en los horarios pico. De esta manera, el código se adapta automáticamente al horario del dispositivo, logrando un funcionamiento óptimo.

Respecto a la tarea que subió la docente nano (Tarea N°9) los avances previstos para el día de hoy no fueron muy amplios.

La expectativa para del día 2 son:

- Avanzar con el código logrando la vinculación del dispositivo con el código
- Establecer valores clave para optimizar la carga de datos de los mozos
- Encontrar un nuevo reto elevando la dificultad del desafío
- Preguntar a la docente hacia donde inclinar el informe (tener un cierto nivel de seriedad o que sea informativo, fácil de leer y con un poco de humor).
- Evaluar la posibilidad de que el código pueda generar sugerencias respecto a la demanda que hay.

Para concluir, una pequeña puntuación del día:

Avances: 6/10

Expectativas del trabajo: 10/10

Calidad del trabajo de hoy: 8/10

Resolución de problemas:10/10



Día 2 – 24/9/2025

Lo primero que logramos este día fue hacer funcionar en nuestro código el SFML

[Demostarcion de SFMLN](#)

Logramos plantear un nuevo desafío para el código: que los valores límite se actualicen en función del tiempo del dispositivo, permitiendo así generar la recomendación de una comida adicional

Posteriormente, iniciamos la vinculación del código con el dispositivo mediante la utilización de la librería “Ctime”. Durante el desarrollo fuimos encontrando diversas dificultades:

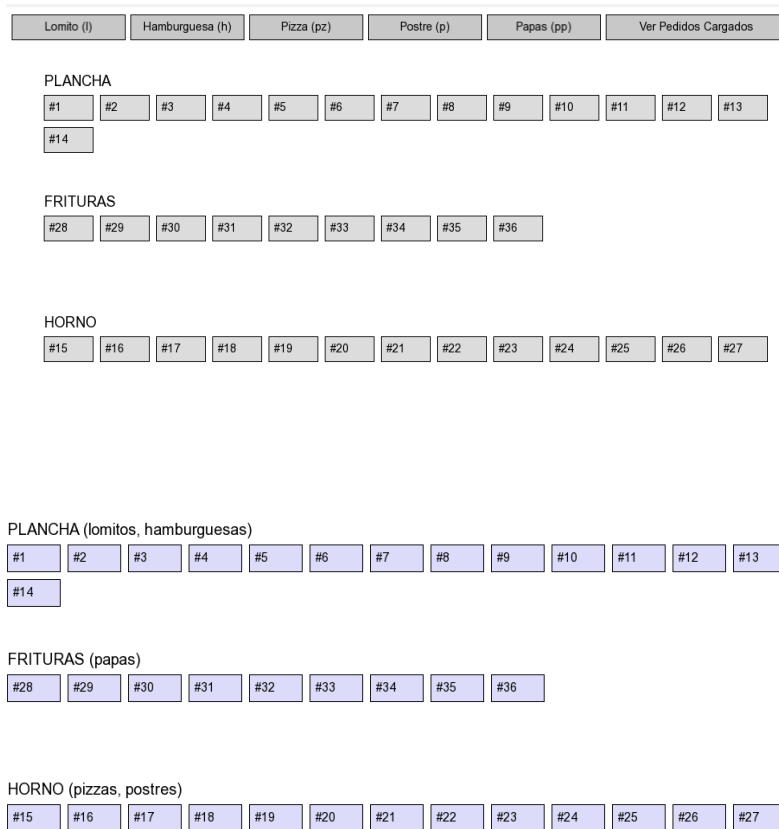
E0864	optional no es una plantilla	SFML para probar	ejemplo_elegido.cpp	92
E0020	el identificador "nullptr" no está definido	SFML para probar	ejemplo_elegido.cpp	92
C2065	'optional': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	92
C2275	'Order': se esperaba una expresión en lugar de un tipo	SFML para probar	ejemplo_elegido.cpp	92
C2974	'std::vector': argumento plantilla no válido para ",T)", tipo esperado	SFML para probar	ejemplo_elegido.cpp	92
C2976	'std::vector': no hay suficientes argumentos de plantilla	SFML para probar	ejemplo_elegido.cpp	92
C2059	error de sintaxis: "<"	SFML para probar	ejemplo_elegido.cpp	92
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	160
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	161
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	167
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	217
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	223
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	224
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	237
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	374
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	379
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	385
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	389
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	397
C2065	'processing': identificador no declarado	SFML para probar	ejemplo_elegido.cpp	481

El primer inconveniente surgió al instalar la librería “Ctime”, ya que generó varios errores. La solución fue crear un nuevo proyecto en Visual Studio que permitiera utilizar SFML sin la necesidad de incorporar una gran cantidad de archivos.h.

Una vez finalizada la incorporación de la librería *Ctime*, procedimos a generar los valores correspondientes a la comida y a definir su destino. Para evaluar el funcionamiento del código, decidimos trabajar con cinco elementos, los cuales fueron:

- Lomito
- Pizza
- Papas
- Hamburguesa
- Postre

```
if (tipo == "lomito" || tipo == "hamburguesa") {
    o.estacion = "PLANCHA";
    STATIONS[0].queue.push_back(o);
    pedidosPlancha.push_back(o);
}
else if (tipo == "papas") {
    o.estacion = "FRITURAS";
    STATIONS[1].queue.push_back(o);
    pedidosFreidora.push_back(o);
}
else if (tipo == "pizza" || tipo == "postre") {
    o.estacion = "HORNO";
    STATIONS[2].queue.push_back(o);
    pedidosHorno.push_back(o);
}
```



Al resolver esta instancia, se nos ocurrió asignar un tiempo específico a cada elemento y, una vez cumplido el tiempo de cocción establecido, el sistema indique que el pedido ha finalizado. Posteriormente, dicho pedido se eliminará automáticamente de la cola.

Designamos a cada elemento un tiempo tal que quedaría así:

- Hamburguesa = 10 Min
- Lomo = 8 Min
- Papas = 5 Min
- Pizzas = 15 Min
- Postre = 7 Min

Determinamos que el tiempo máximo de demora será de 45 minutos, dado que, en caso de repetirse los elementos, los tiempos no se sumarán por encontrarse en el mismo sector de la cocina. Para la representación en el código, se estableció que 1 minuto equivalga a 1 segundo, con el fin de agilizar las pruebas.

Otro de los errores que nos encontramos fue este:

Código	Descripción	Proyecto	Archivo	Línea	Estado	Suprimido	Detalles
abc	E0864	function no es una plantilla	Ejemplo_elegido	Ejemolo_elegido.cpp	114		

Debido a la alta eficiencia que estamos dictando en trabajo, llega un punto en el cual los futuros ingenieros experimentan una especie de 'cortocircuito', lo que a veces genera resultados tan creativos como el que se observa en la imagen. Incluso con un nivel de agilidad y eficiencia elevado, pueden surgir pequeños errores, como escribir “fuction” en lugar de “función”.

Aprovechamos para realizar un menú facilitando el uso de este, la interfaz es sencilla, practica y amigable.

Expectativas para el día 3

- Lograr dejar el trabajo en instancias finales
- Realizar el Readme.txt
- Generar la documentación del uso de IA

Nuevo (Lomito)	Nuevo (Hamburguesa)	Nuevo (Pizza)	Nuevo (Postre)	Nuevo (Papas)
----------------	---------------------	---------------	----------------	---------------

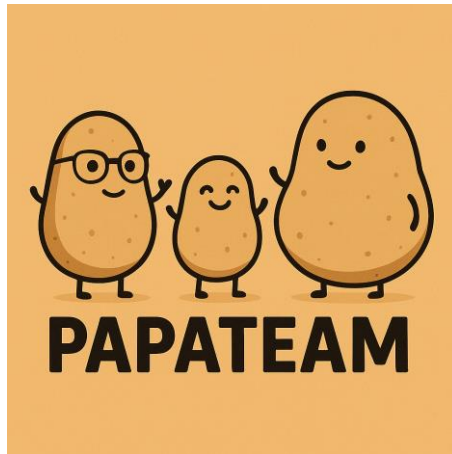
Pedidos (auto-eliminacion por tiempo):

Nuevo (Lomito)	Nuevo (Hamburguesa)	Nuevo (Pizza)	Nuevo (Postre)	Nuevo (Papas)
----------------	---------------------	---------------	----------------	---------------

Pedidos (auto-eliminacion por tiempo):

Pedido #16 [lomito x2 pizza x1] Restante: 11s (Total: 23s)
Pedido #17 [hamburguesa x1 postre x1] Restante: 5s (Total: 17s)
Pedido #18 [postre x1 hamburguesa x1] Restante: 7s (Total: 17s)
Pedido #19 [papas x1 hamburguesa x1 lomito x1] Restante: 14s (Total: 23s)

Nos encargamos de diseñar un logotipo que nos otorga una identidad visual y representa quiénes somos como equipo. Fue una tarea sencilla, ya que utilizamos la IA para generar nuestro logotipo y ahorrar un poco de tiempo. El resultado es el siguiente:



Para concluir, una pequeña puntuación del día (hoy ganaron los errores):

Avances: 8/10

Expectativas del trabajo: 10/10

Calidad del trabajo de hoy: 8/10

Resolución de problemas 6/10

Errores 12/10



Día 3 – 29/9/2025

Con unas altas expectativas sobre un gran avance en el día de hoy, damos por comenzado el día 4 (Esperamos con ansias que no ganen nuevamente los errores)

El primer pensamiento fue en el informe del proyecto, dado que la bitácora diaria que estamos elaborando abarca el registro del proceso, pero no constituye un informe del proyecto en sí mismo.

Al iniciar el informe técnico, se nos ocurrió que los datos cargados en el código pudieran almacenarse en un archivo `.txt`, con el fin de registrar la demanda correspondiente al día.

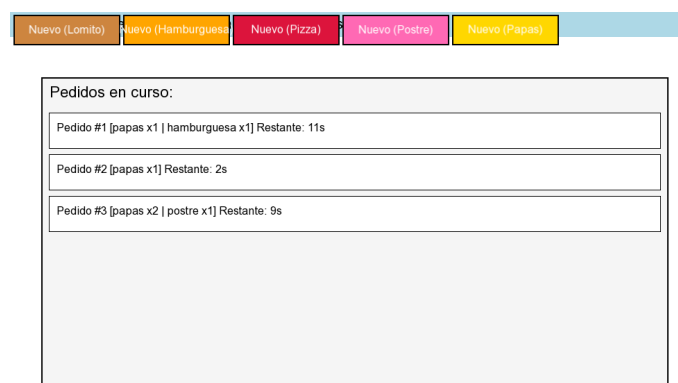
Al comenzar con la incorporación de esta nueva función al código nos encontramos con un problema: el archivo `.txt` se generaba en una ruta incorrecta. El error persistió hasta que observamos que el nombre del archivo debía escribirse en minúsculas y no iniciar con una mayúscula. Una vez corregido, el problema se resolvió y el programa volvió a funcionar correctamente.

[Video demostracion menu](#)

Resultado del código:

```
[Finalizado: 2025-09-29 09:10:34] papasx1
[Finalizado: 2025-09-29 09:10:35] postrex1
[Finalizado: 2025-09-29 09:10:35] papasx1
[Finalizado: 2025-09-29 09:10:35] papasx2
[Finalizado: 2025-09-29 09:10:37] papasx1
[Finalizado: 2025-09-29 09:10:37] papasx2
[Finalizado: 2025-09-29 09:10:37] papasx1
```

Una vez finalizada esta etapa de trabajo, continuamos con la mejora de la interfaz del código, incorporando colores, organizándolos adecuadamente y ajustando los tamaños correspondientes.



The screenshot displays a web application interface. At the top, there is a horizontal menu with five items: 'Nuevo (Lomito)', 'Nuevo (Hamburguesa)', 'Nuevo (Pizza)', 'Nuevo (Postre)', and 'Nuevo (Papas)'. Below the menu is a section titled 'Pedidos en curso:' (Orders in progress:). This section contains three rows, each representing an order with its details and remaining time: 'Pedido #1 [papas x1 | hamburguesa x1] Restante: 11s', 'Pedido #2 [papas x1] Restante: 2s', and 'Pedido #3 [papas x2 | postre x1] Restante: 9s'.

Una vez finalizado el código, decidimos iniciar con la elaboración del informe técnico sobre el uso de la IA, incorporando capturas de pantalla y desarrollando su defensa en otra sección de este documento.

Con todos estos avances damos por concluido este trabajo práctico. A lo largo de los días de desarrollo realizamos una autoevaluación crítica, registrando errores y soluciones,

así como también avances y demoras. Consideramos que el trabajo realizado, junto con los informes, refleja un nivel de desempeño que incluso superó nuestras propias expectativas. La experiencia resultó diversa: en ocasiones divertida, en otras exigente, compleja o tranquila, pero siempre desafiante. Sin duda, ha sido el trabajo en el que más expectativas hemos tenido como grupo. Al finalizarlo sentimos una mezcla de alegría y cierta tristeza: alegría por haber alcanzado el objetivo y tristeza porque debemos continuar con el Trabajo Práctico n° 9.



Para concluir, una pequeña puntuación del día:

Avances: 10/10

Expectativas del trabajo: 10/10

Calidad del trabajo de hoy: 1/10

Resolución de problemas 10/10

Errores 1/10