Universidad de las Américas
Programación 1

Integrantes: Brandon Arellano – Josué Riera

Proyecto Final – Progreso 3

Introducción

El proyecto se enfoca en la búsqueda de una solución que nos permita gestionar los datos de un centro de maquinaria de una empresa de fabricación, teniendo en cuenta los 3 turnos diarios y calculando el desperdicio de materia prima. De esta manera, podremos determinar la eficiencia de cada estación de máquinas. Para resolver este problema, utilizaremos herramientas y lenguajes de programación específicos y manejo de archivos planos, además de implementar funcionalidades adicionales y la lógica de programación necesaria.

Objetivo:

Implementar una solución eficiente por medio de la lógica de programación utilizando funciones y archivos, para el saciar las necesidades de la empresa.

Objetivo específico

Utilizar archivos planos para tener un mayor control en la fábrica y calcular la eficiencia que genera cada turno de trabajo.

Tabla de identificación:

Tabla de identificación	
Tipos de usuarios:	Los usuarios del programa abarcarán a cualquier persona con un conocimiento básico de informática y habilidades en el manejo de computadoras.
Contexto del problema:	El problema se presenta en una empresa de maquinaria pesada con un número reducido de empleados y la posibilidad de ser escalable en el futuro. Dado un presupuesto limitado para tecnología, se busca una solución que ofrezca un equilibrio entre costo y beneficio. La funcionalidad del software consiste en recopilar datos y generar informes sobre la eficiencia de los turnos de las estaciones de maquinaria, lo que permitirá llevar un control adecuado de los procesos. Los usuarios del programa son personas que trabajan en la empresa.
Restricciones:	Se identificaron varias restricciones durante el proceso, entre las cuales se destacan la falta de información disponible, la incertidumbre sobre la veracidad de los datos existentes y la ausencia de información en algunos casos específicos. Estas limitaciones representan desafíos significativos para garantizar la calidad y confiabilidad de los datos utilizados en el proyecto. Será necesario abordar estas restricciones.

Limitaciones:	Se identificaron limitaciones en cuanto a la interfaz gráfica del software. El manejo de la consola puede resultar menos práctico en comparación con otras opciones más intuitivas presentes en otros lenguajes de programación. Esto puede afectar la usabilidad y la experiencia del usuario al interactuar con el programa. Además, se debe tener en cuenta que el software se ejecutará en máquinas expuestas en proximidad a la maquinaria pesada. Esta ubicación plantea desafíos adicionales, como la necesidad de garantizar la resistencia y la protección de las máquinas contra condiciones ambientales adversas, vibraciones y posibles interferencias electromagnéticas.

Planteamientos de propuestas de solución:

Propuesta 1: La utilización de arrays de varias dimensiones para almacenar los datos de las estaciones de maquinaria por turnos, el uso de punteros para que el programa sea más eficiente en cuestión de recursos computacionales y el manejo de archivos planos para generar un informe. Esto permitirá registrar y rastrear la información relevante, como la cantidad de materia prima utilizada y la cantidad que no se pudo utilizar en cada turno. Con base en estos datos, se podrán realizar cálculos para determinar la eficiencia de los turnos y generar informes correspondientes. La utilización de arrays de varias dimensiones proporcionará una estructura organizada y eficiente para almacenar y acceder a los datos necesarios, el uso de punteros permitirá tener un mejor manejo de memoria y el manejo de archivos permitirá tener un registro de la información. Además, permitirá llevar un seguimiento detallado de las operaciones realizadas en cada turno de trabajo, facilitando así el control y la evaluación de la eficiencia.

Esta propuesta busca optimizar el manejo de los datos y brindar una solución efectiva para el control en la empresa.

Propuesta 2: La utilización de archivos planos para el almacenamiento de los datos de cada turno de las estaciones de máquinas. Cada archivo contendrá la información correspondiente a un turno específico, permitiendo un registro ordenado de los datos. Al ingresar los datos en cada archivo, se realizarán los cálculos necesarios para determinar la cantidad de materia prima utilizada y la cantidad de materia prima desechada en cada turno. Estos cálculos proporcionarán información precisa sobre el desperdicio de materia prima y permitirán evaluar la eficiencia de las estaciones de máquinas. El uso de archivos planos como método de almacenamiento ofrece una solución práctica y accesible, especialmente considerando el presupuesto reducido para tecnología. Además, brinda la flexibilidad necesaria para manejar y analizar los datos de manera eficiente.

Con esta propuesta, se busca obtener una solución efectiva que permita llevar un control adecuado de los recursos utilizados y evaluar la eficiencia de los turnos de trabajo.

Análisis de propuestas

Dado que la primera propuesta implica una alta complejidad de programación y puede resultar robusta, es importante considerar la viabilidad y los recursos disponibles para desarrollar y mantener dicho programa. Si la capacidad y los recursos están disponibles, puede ser una opción sólida para lograr una solución más completa y funcional.

Sin embargo, es fundamental tener en cuenta la capacidad de los usuarios finales para utilizar y comprender el programa. Si los trabajadores tienen poco conocimiento de informática, puede resultar difícil para ellos aprovechar al máximo las funcionalidades más complejas. En ese caso, sería beneficioso considerar una interfaz de usuario más intuitiva y fácil de usar.

Dado que el manejo de archivos planos por turnos no es viable debido a la falta de registro de días anteriores y la limitada capacidad de los trabajadores en términos de habilidades informáticas, es importante descartar esta opción.

Considerando las restricciones mencionadas, es necesario buscar una alternativa que sea más adecuada y fácil de usar para los usuarios finales con poco conocimiento en informática.

Al tomar en cuenta las limitaciones y los recursos disponibles, es esencial buscar una solución que sea práctica, accesible y brinde los resultados deseados.

Selección

Aunque puede requerir un poco más de esfuerzo y complejidad en la programación, optamos por la primera propuesta, es una opción eficiente y proporciona un mayor potencial de crecimiento a largo plazo. Al desarrollar una solución robusta, se podrá aprovechar al máximo la funcionalidad del software y obtener una visión clara de la eficiencia de los turnos de trabajo. A medida que la empresa crezca, esta solución proporcionará una base sólida para gestionar y analizar los datos, permitiendo tomar decisiones informadas y mejorar los procesos en función de los resultados obtenidos. Es importante considerar tanto la viabilidad actual como el potencial de escalabilidad futura al seleccionar esta solución, es por eso que la solución número uno cumple con ambos aspectos y, aunque pueda implicar un poco más de esfuerzo en la programación, valdrá la pena el resultado final.

Diagrama de flujo del programa.

Diagrama de la función EscribirMuebles

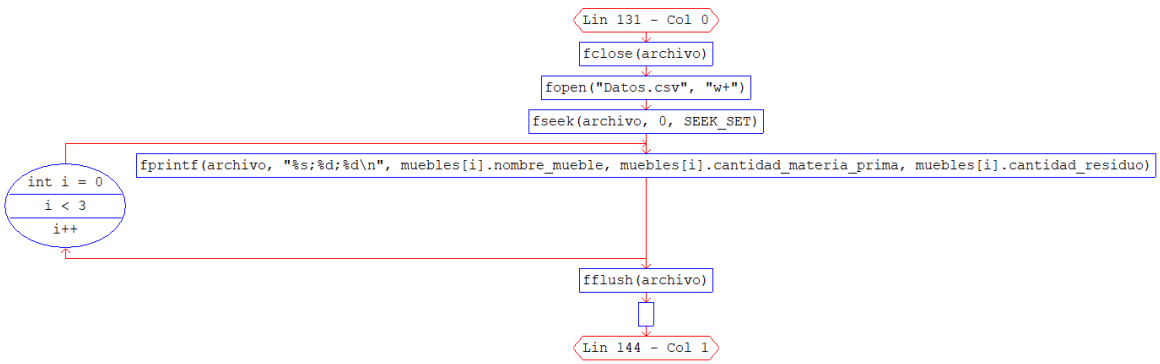


Diagrama de la función principal(Main)

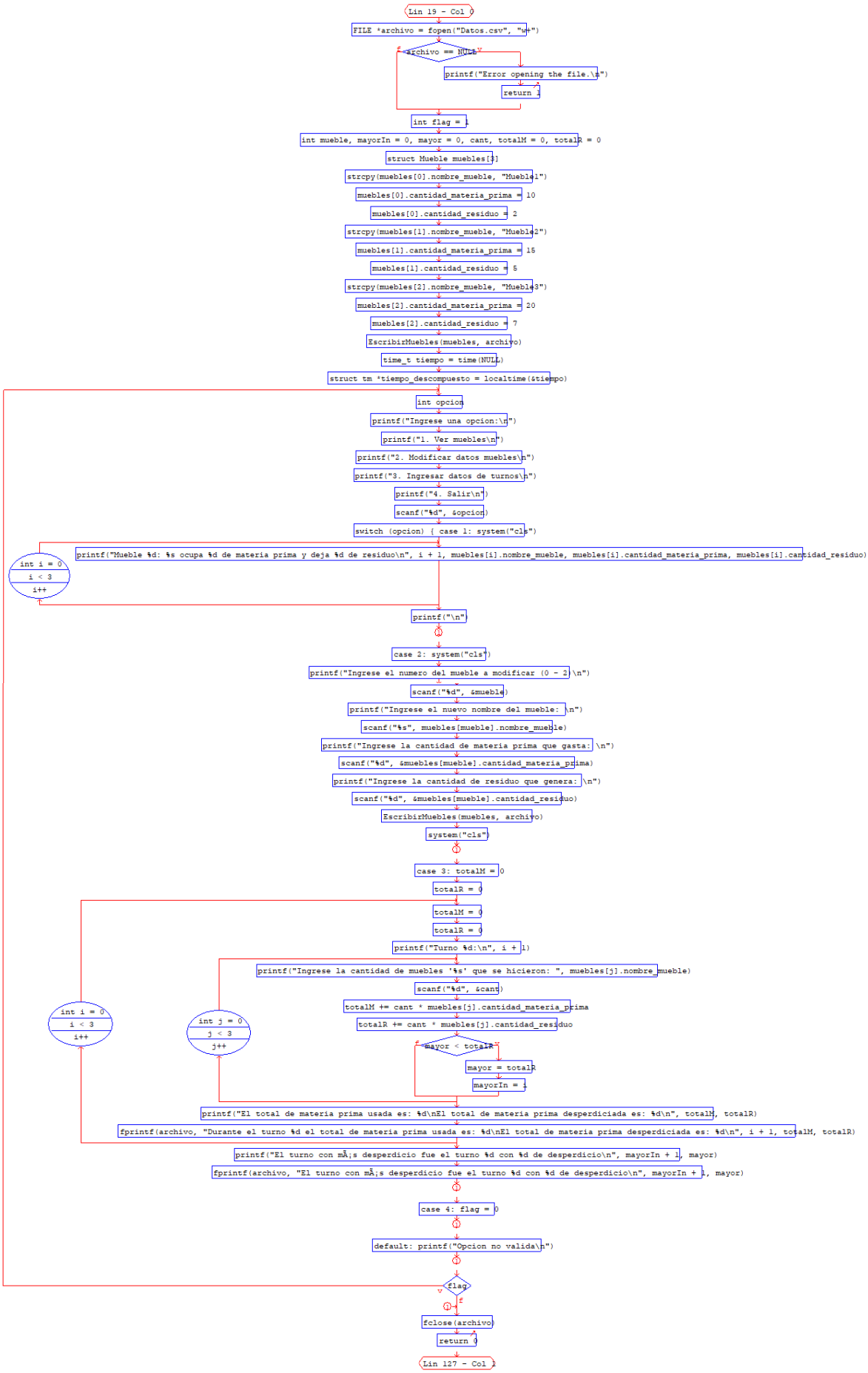
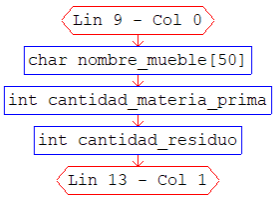

Lin 19 - Col 0
FILE *archivo = fopen("Datos.csv", "w+")
archivo == NULL
printf("Error opening the file.\n")
return 1
int flag = 1
int mueble, mayorIn = 0, mayor = 0, cant, totalM = 0, totalR = 0
struct Mueble muebles[3]
strcpy(muebles[0].nombre_mueble, "Mueble1")
muebles[0].cantidad_materia_prima = 10
muebles[0].cantidad_residuo = 2
strcpy(muebles[1].nombre_mueble, "Mueble2")
muebles[1].cantidad_materia_prima = 15
muebles[1].cantidad_residuo = 5
strcpy(muebles[2].nombre_mueble, "Mueble3")
muebles[2].cantidad_materia_prima = 20
muebles[2].cantidad_residuo = 7
EscribirMuebles(muebles, archivo)
time_t tiempo = time(NULL)
struct tm *tiempo_descompuesto = localtime(&tiempo)
int opcion
printf("Ingrese una opcion:\n")
printf("1. Ver muebles\n")
printf("2. Modificar datos muebles\n")
printf("3. Ingresar datos de turnos\n")
printf("4. Salir\n")
scanf("%d", &opcion)
switch (opcion) { case 1: system("cls")
int i = 0
i < 3
i++
printf("Mueble %d: %s ocupa %d de materia prima y deja %d de residuo\n", i + 1, muebles[i].nombre_mueble, muebles[i].cantidad_materia_prima, muebles[i].cantidad_residuo)
printf("\n")
case 2: system("cls")
printf("Ingrese el numero del mueble a modificar (0 - 2)\n")
scanf("%d", &mueble)
printf("Ingrese el nuevo nombre del mueble: \n")
scanf("%s", muebles[mueble].nombre_mueble)
printf("Ingrese la cantidad de materia prima que gasta: \n")
scanf("%d", &muebles[mueble].cantidad_materia_prima)
printf("Ingrese la cantidad de residuo que genera: \n")
scanf("%d", &muebles[mueble].cantidad_residuo)
EscribirMuebles(muebles, archivo)
system("cls")
case 3: totalM = 0
totalR = 0
totalM = 0
totalR = 0
printf("Turno %d:\n", i + 1)
int i = 0
i < 3
i++
printf("Ingrese la cantidad de muebles '%s' que se hicieron: ", muebles[j].nombre_mueble)
scanf("%d", &cant)
int j = 0
j < 3
j++
totalM += cant * muebles[j].cantidad_materia_prima
totalR += cant * muebles[j].cantidad_residuo
mayor < totalR
mayor = totalR
mayorIn = i
printf("El total de materia prima usada es: %d\nEl total de materia prima desperdiciada es: %d\n", totalM, totalR)
fprintf(archivo, "Durante el turno %d el total de materia prima usada es: %d\nEl total de materia prima desperdiciada es: %d\n", i + 1, totalM, totalR)
printf("El turno con mÃ¡s desperdicio fue el turno %d con %d de desperdicio\n", mayorIn + 1, mayor)
fprintf(archivo, "El turno con mÃ¡s desperdicio fue el turno %d con %d de desperdicio\n", mayorIn + 1, mayor)
case 4: flag = 0
default: printf("Opcion no valida\n")
flag
fclose(archivo)
return 0
Lin 127 - Col 1

Diagrama del Struct



Identificación de variables de entrada y salida.

Variables de Entrada:

opcion (entero): Almacena la opción seleccionada por el usuario en el menú.

mueble (entero): Almacena el número del mueble a modificar.

cant (entero): Almacena la cantidad de muebles producidos en cada turno.

Variables de Salida:

muebles (arreglo de estructuras Mueble): Almacena los datos de los muebles, incluyendo el nombre, cantidad de materia prima y cantidad de residuo.

archivo (puntero a FILE): Apunta al archivo "Datos.csv" para leer y escribir los datos de los muebles.

totalM (entero): Almacena el total de materia prima utilizada en cada turno.

totalR (entero): Almacena el total de residuo generado en cada turno.

mayor (entero): Almacena el valor del mayor residuo generado en los turnos.

mayorIn (entero): Almacena el número del turno con el mayor residuo.

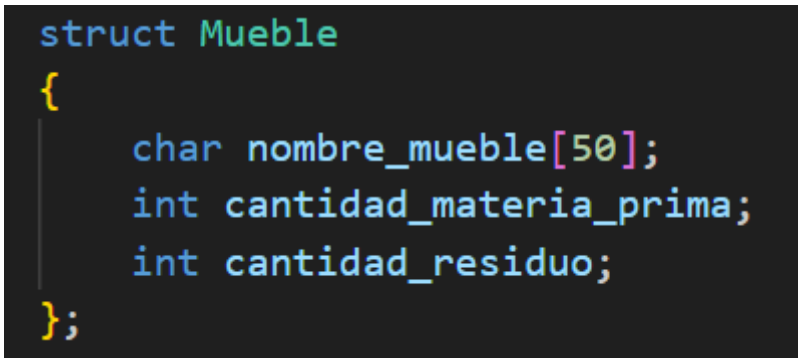
Enlace de GitHub al código fuente del programa.
Enlace de GitHub: https://github.com/BrandonArellanoU/ProyectoP3/tree/main

Imágenes de las partes más importantes del código explicadas.

Como se muestra en la siguiente imagen, se puede observar las librarías que se usaron para el programa.

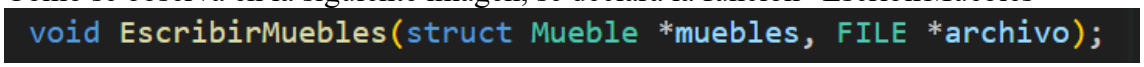
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
```

Como se observa en la siguiente imagen, se define una estructura llamada "Mueble" que tiene tres miembros: "nombre_mueble" (cadena de caracteres para almacenar el nombre del mueble), "cantidad_materia_prima" (entero para almacenar la cantidad de materia prima que requiere el mueble) y "cantidad_residuo" (entero para almacenar la cantidad de residuos que genera el mueble).



```c
struct Mueble
{
    char nombre_mueble[50];
    int cantidad_materia_prima;
    int cantidad_residuo;
};
```

Como se observa en la siguiente imagen, se declara la función “EscribirMuebles”



```c
void EscribirMuebles(struct Mueble *muebles, FILE *archivo);
```

En la siguiente imagen, observamos el main es decir nuestra función cabecera, en ella se crea el archivo llamado “Datos.csv” , siguiente de una condicional si no se encuentra el archivo nos demarca un mensaje de error, siguiente a eso se declaran las variables a usar y se encuentran inicializadas entre ellas el struct.

```c
int main()
{
    FILE *archivo = fopen("Datos.csv", "w+");// Apertura del archivo "Dat

    if (archivo == NULL)
    {
        printf("Error opening the file.\n"); // Muestra un mensaje de err
        return 1;
    }

    int flag = 1;
    int mueble, mayorIn = 0, mayor = 0, cant, totalM = 0, totalR = 0; //

    struct Mueble muebles[3]; // Arreglo de estructuras "Mueble" para alm
```

En consiguiente como se muestra en la imagen, inicializamos los datos de los muebles, seguimos con la llamada a la función “EscribirMuebles” y mandamos los parámetros de muebles y archivo, obtenemos el tiempo con la función, descomponemos el tiempo y lo

asignamos con el “localtime”, usamos un puntero.

```
// Inicialización de los datos de los muebles
strcpy(muebles[0].nombre_mueble, "Mueble1");
muebles[0].cantidad_materia_prima = 10;
muebles[0].cantidad_residuo = 2;
strcpy(muebles[1].nombre_mueble, "Mueble2");
muebles[1].cantidad_materia_prima = 15;
muebles[1].
strcpy(mueb
muebles[2].cantidad_materia_prima = 20;
muebles[2].cantidad_residuo = 7;

EscribirMuebles(muebles, archivo); // Llamada a la fun

time_t tiempo = time(NULL);
struct tm *tiempo_descompuesto = localtime(&tiempo);
```

int Mueble::cantidad_materia_prima

Como se puede observar en la siguiente imagen, utilizamos un Do While para realizar el menú y dentro utilizamos un Switch para las opcoines

```
do
{
    int opcion;
    printf("Ingrese una opcion:\n");
    printf("1. Ver muebles\n");
    printf("2. Modificar datos muebles\n");
    printf("3. Ingresar datos de turnos\n");
    printf("4. Salir\n");
    scanf("%d", &opcion); // Lee la opción ingresad

    switch (opcion)
} while (flag);
```

Detallamos las funcoinalidades de cada opcion del menu, como se puede ver en la siguietne imagen.

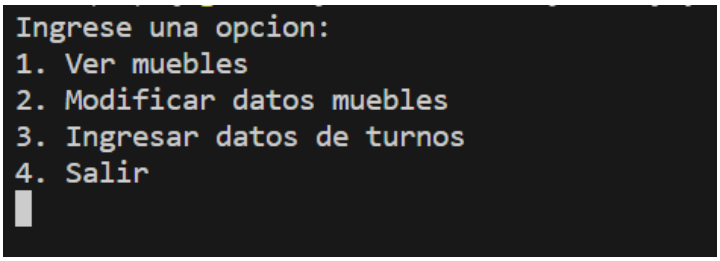
```c
switch (opcion)
{
case 1:
    system("cls");
    // Muestra en pantalla la información de los muebles almacenados en el arreglo "muebles"
    for (int i = 0; i < 3; i++)
    {
        printf("Mueble %d: %s ocupa %d de materia prima y deja %d de residuo\n", i + 1, muebles[i].nombre_mueble, muebles[i].cantidad_materia_prima, muebles[i].cantidad_residuo);
    }
    printf("\n");
    break;

case 2:
    system("cls");
    printf("Ingrese el numero del mueble a modificar (1 - 3)\n");
    scanf("%d", &mueble);
    printf("Ingrese el nuevo nombre del mueble: \n");
    scanf("%s", muebles[mueble].nombre_mueble);
    printf("Ingrese la cantidad de materia prima que gasta: \n");
    scanf("%d", &muebles[mueble].cantidad_materia_prima);
    printf("Ingrese la cantidad de residuo que genera: \n");
    scanf("%d", &muebles[mueble].cantidad_residuo);
    EscribirMuebles(muebles, archivo); // Actualiza los datos de los muebles en el archivo
    system("cls");
    break;

case 3:
    totalM = 0;
    totalR = 0;
    for (int i = 0; i < 3; i++)
    {
        totalM = 0;
        totalR = 0;
        printf("Turno %d:\n", i + 1);
        for (int j = 0; j < 3; j++)
        {
            printf("Ingrese la cantidad de muebles '%s' que se hicieron: ", muebles[j].nombre_mueble);
            scanf("%d", &cant);
            totalM += cant * muebles[j].cantidad_materia_prima;
            totalR += cant * muebles[j].cantidad_residuo;
            if (mayor < totalR)
            {
                mayor = totalR;
                mayorIn = i;
            }
        }
        printf("El total de materia prima usada es: %d\nEl total de materia prima desperdiciada es: %d\n", totalM, totalR);
        fprintf(archivo, "Durante el turno %d el total de materia prima usada es: %d\nEl total de materia prima desperdiciada es: %d\n", i + 1, totalM, totalR);
    }

    printf("El turno con más desperdicio fue el turno %d con %d de desperdicio\n", mayorIn + 1, mayor);
    fprintf(archivo, "El turno con más desperdicio fue el turno %d con %d de desperdicio\n", mayorIn + 1, mayor);

    break;

case 4:
    flag = 0;
    break;

default:
    printf("Opcion no valida\n");
    break;
}
```

Y finalmente como se puede ver en la imagen, se desarrolla la funcion “EscribirMuebles ” cerrando el archivo, abriendolo y establece la posicion del puntero con la funcion fseek, utilizando un for para imprimir los datos y usando la funcion fflush para limpiar el bufer de salida.

```c
void EscribirMuebles(struct Mueble *muebles, FILE *archivo)
{
    fclose(archivo);          // Cierra el archivo
    fopen("Datos.csv", "w+"); // Vuelve a abrir el archivo en modo escritura y lectura

    fseek(archivo, 0, SEEK_SET); // Establece la posición del puntero de archivo al principio del archivo

    // Escribe los datos de los muebles en el archivo, separados por punto y coma
    for (int i = 0; i < 3; i++)
    {
        fprintf(archivo, "%s;%d;%d\n", muebles[i].nombre_mueble, muebles[i].cantidad_materia_prima, muebles[i].cantidad_residuo);
    }

    fflush(archivo); // Limpia el búfer de salida para asegurar que los datos se escriban en el archivo
}
```

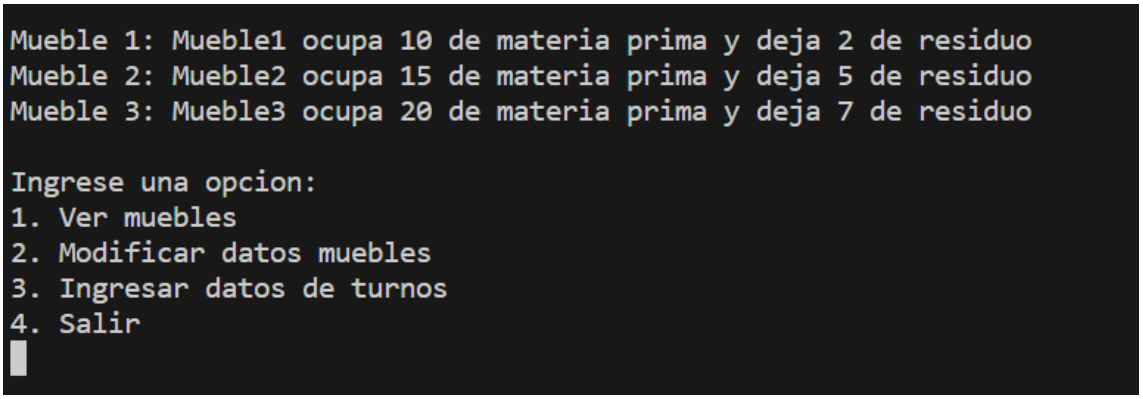
Imágenes de la ejecución de cada sección del programa explicadas.

Como se puede mirar en la siguiente imagen, el programa cuando se ejecuta se muestra el nuestro menú principal.



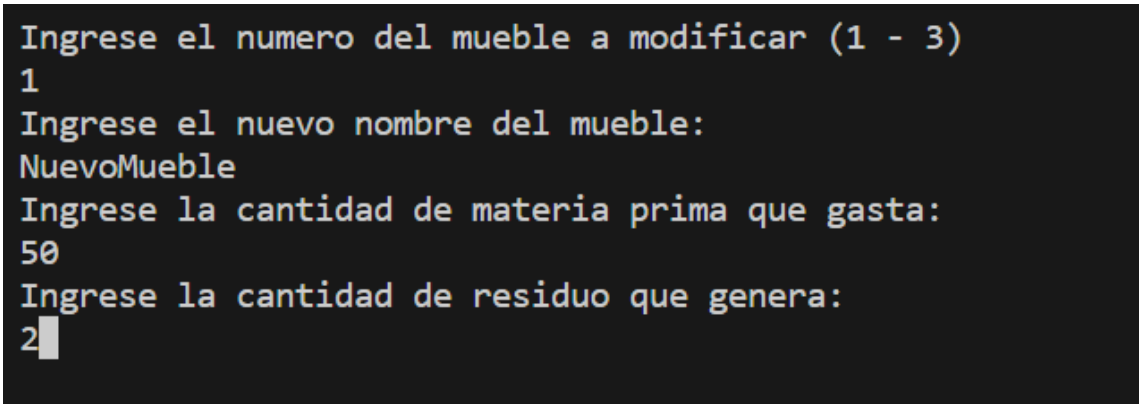
Opción 1:

Como se muestra en la siguiente imagen, al ingresar la opción 1, nos muestra los muebles que se realizaron que en nuestro caso son 3 tipos de muebles, presenta el nombre del mueble, el material de materia prima que usa y el residuo que nos deja. Posteriormente nos regresa al menú principal.



Opción 2:

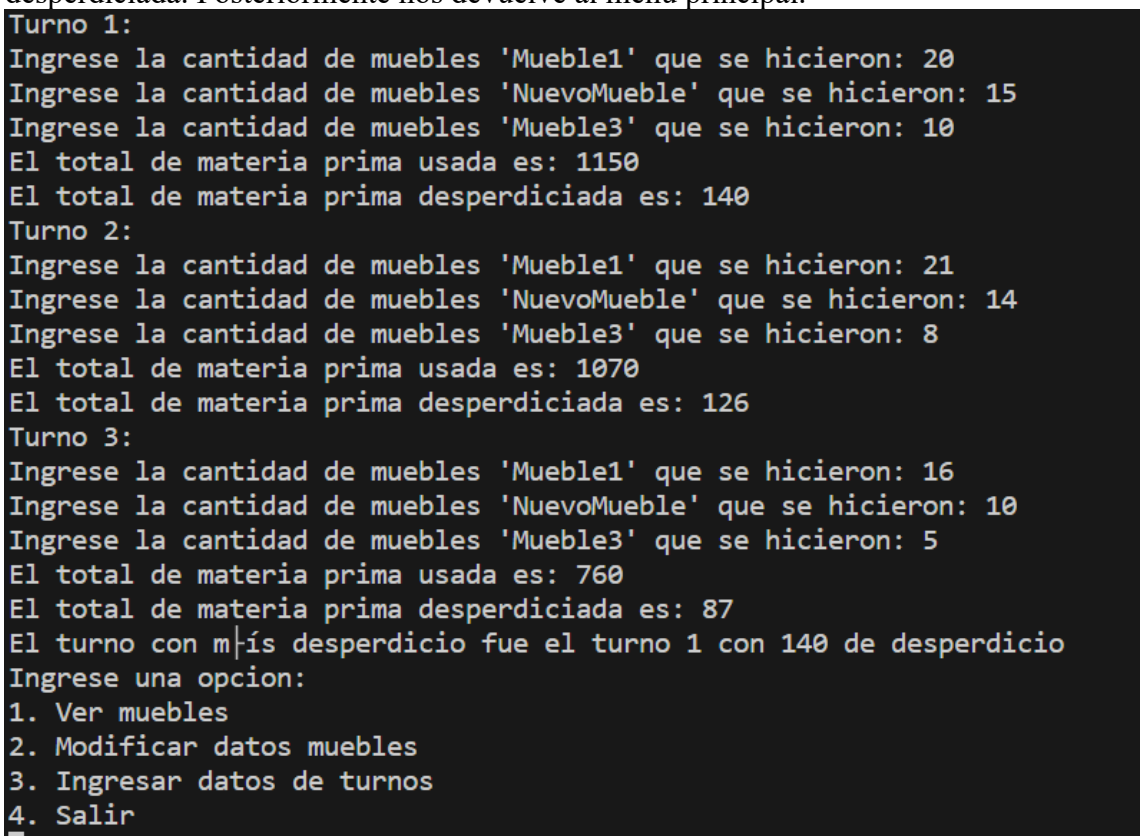
Como se muestra en la siguiente imagen, la opción numero dos nos permite modificar los datos del mueble escogido, primero nos pide el mueble a modificar es decir el 1,2 o 3, escogimos el 1, nos solicita el nombre nuevo del mueble, nosotros ingresamos "NuevoMueble", nos solicita la cantidad de materia prima que gasta, en nuestro caso 50 y finalmente el residuo que genera, para nosotros es 2, aplastamos 'Enter' y nos regresa al menú principal.



Opción 3:

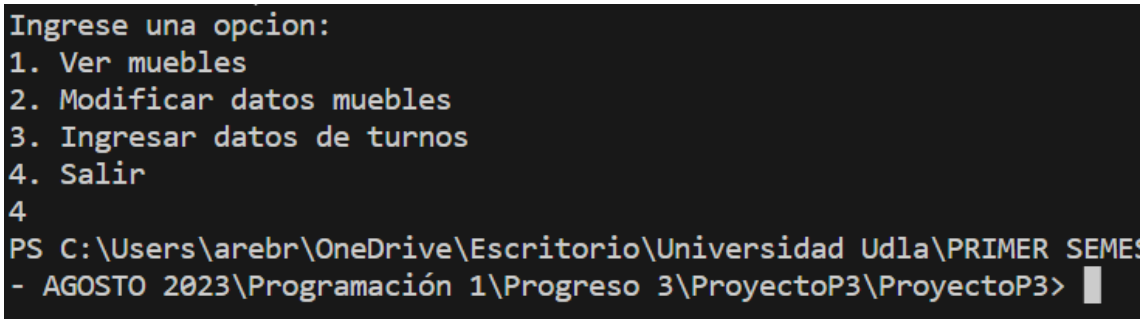
Como se muestra en la siguiente imagen, la opción 3 nos pide la cantidad de muebles que se realizaron por turnos, y nos devuelve la materia prima usada y la materia prima

desperdiciada. Posteriormente nos devuelve al menú principal.



Opción 4:

Como se muestra en la imagen, la opción número 4 hace que cierre el programa terminado con la ejecución.



Conclusiones y recomendaciones

En conclusión, A través de la implementación de la solución propuesta, hemos logrado gestionar de manera eficiente los datos del centro de maquinaria de la empresa de fabricación. La utilización de archivos planos nos ha permitido tener un mayor control y seguimiento de las actividades en la fábrica, facilitando el registro y almacenamiento de la información necesaria para el cálculo de la eficiencia. La lógica de programación y las funciones implementadas han sido clave para automatizar los procesos y realizar cálculos precisos.

Recomendaciones

Es recomendable mantener un seguimiento y actualización constante de los archivos planos utilizados, asegurándose de que reflejen correctamente los datos y cambios en la fábrica.

Es importante contar con un equipo de soporte y mantenimiento que pueda dar seguimiento a la solución implementada, brindando asistencia técnica y realizando mejoras o ajustes según las necesidades cambiantes de la empresa.