**Sistema de Gestión de Producción y Recursos**

Fecha: 18 de mayo de 2025

Joel Alvarado Tatiana Torres

Contenido

[Informe: Sistema de Gestión de Producción y Recursos 2](#_Toc198492912)

[1. Introducción 4](#_Toc198492913)

[2. Objetivos 5](#_Toc198492914)

[3. Formulación del Problema 5](#_Toc198492915)

[4. Análisis de Alternativas 5](#_Toc198492916)

[Alternativa 1: 5](#_Toc198492917)

[Alternativa 2: 6](#_Toc198492918)

[5. Selección de la Mejor Alternativa 7](#_Toc198492919)

[a) ¿Cuál de estas alternativas sería la más fácil de implementar y por qué? 7](#_Toc198492920)

[b)¿Cómo garantizaría que la solución seleccionada es la más eficiente en términos de tiempo de ejecución y claridad en el código? 8](#_Toc198492921)

[c) ¿Qué aspectos son importantes al elegir una solución eficiente? 8](#_Toc198492922)

[6. Desarrollo 9](#_Toc198492923)

[6.1 Diagrama de flujo del programa 9](#_Toc198492924)

[6.2 Enlace de Github del código fuente del programa: 9](#_Toc198492925)

[6.3 Explicación de las partes más importantes del código 9](#_Toc198492926)

[6.3 Imágenes y descripción de cada sección 11](#_Toc198492927)

[6.4 Imágenes y la ejecución de cada sección 22](#_Toc198492928)

[7. Conclusiones 25](#_Toc198492929)

# Informe: Sistema de Gestión de Producción y Recursos

|  |  |
| --- | --- |
| **Nombre del sistema:** | Gestión de producción |
| **Usuarios:** | Analista de producción (persona encargada de procesar pedidos) |
| **Objetivo del sistema:** | * Validar la disponibilidad de recursos (componentes y horas hombre) en relación con los requerimientos de un pedido recibido. * Alertar al usuario sobre la insuficiencia de recursos en componentes, ofreciendo la posibilidad de gestionar una compra oportuna para aumentar el stock. * Alertar al usuario sobre la insuficiencia de recursos en tiempo, ofreciendo la posibilidad de contratar personal adicional para cubrir la demanda. * Minimizar la pérdida de ventas y mejorar la productividad general mediante la gestión proactiva de la falta de recursos. |
| **Variables de entrada:** | **Datos iniciales:**   * Tiempo total disponible para la producción en minutos. * Cantidad disponible de cada recurso (Pantallas, Altavoces, Procesadores, Placas).   **Datos de los productos (por cada producto):**   * Nombre del producto (cadena de texto). * Cantidad demandada (entero positivo). * Tiempo de producción por unidad (entero positivo, en minutos). * Unidades requeridas de cada recurso por unidad de producto (enteros positivos).   **Opciones del menú:**   * Selección numérica para agregar, editar, eliminar, mostrar productos, calcular producción, modificar recursos disponibles, modificar tiempo disponible y salir.   **Nuevos valores para edición/modificación:**   * Nuevos nombres de productos (cadenas de texto). * Nuevas cantidades demandadas (enteros positivos). * Nuevos tiempos por unidad (enteros positivos). * Nuevas cantidades de recursos por unidad (enteros positivos). * Nuevo tiempo total disponible (entero positivo). * Nuevas cantidades de recursos disponibles (enteros positivos). |
| **Variables de salida:** | **Información mostrada al usuario:**   * Menú de opciones. * Mensajes de solicitud de entrada de datos. * Mensajes de error (ej. nombre inválido, producto no encontrado, número no positivo, máximo de productos alcanzado). * Lista de productos con sus detalles (nombre, cantidad, tiempo por unidad, recursos por unidad). * Resultado de la evaluación de producción para cada producto (si se puede o no producir, tiempo faltante, recursos faltantes). * Resumen final de la evaluación de producción (tiempo restante, recursos restantes, indicación si se pudo producir al menos un producto). * Mensajes de confirmación de edición, eliminación y modificación de recursos/tiempo. * Mensaje de salida del programa. |
| **Procesos asociados:** | **Inicialización:** Lectura del tiempo total disponible y las cantidades iniciales de los recursos.  **Gestión de productos:**   * **Agregar:** Solicitar y almacenar la información de un nuevo producto (nombre, cantidad, tiempo, recursos). * **Editar:** Buscar un producto por nombre y permitir la modificación de sus datos. * **Eliminar:** Buscar un producto por nombre y eliminarlo de la lista. * **Mostrar:** Presentar la lista de productos con sus detalles.   **Cálculo de producción:**   * Iterar a través de la lista de productos. * Para cada producto, calcular el tiempo y los recursos totales necesarios. * Verificar si el tiempo necesario es menor o igual al tiempo restante y si los recursos necesarios son menores o iguales a los recursos restantes. * Actualizar el tiempo y los recursos restantes si el producto puede ser "producido". * Mostrar el resultado de la evaluación para cada producto. * Mostrar un resumen final de los recursos y el tiempo restantes.   **Modificación de parámetros:**   * Modificar recursos disponibles: Permitir la actualización de las cantidades disponibles de cada recurso. * Modificar tiempo disponible: Permitir la actualización del tiempo total disponible para la producción.   **Validación de entrada:** Verificar que los nombres de los productos sean válidos (sin números ni vacíos) y que las cantidades y tiempos sean enteros positivos.  **Búsqueda de productos:** Localizar un producto en la lista por su nombre. |
| **Restricciones** (Consideraciones externas que se deben tener para diseñar y desarrollar el proyecto, ejemplo: porcentaje de impuesto, registro sanitario u otros) | Por el momento, no se definen restricciones externas específicas en el código proporcionado (como impuestos, registros sanitarios, etc.) |
| **Limitaciones** (Consideraciones internas que se deben tener para diseñar y desarrollar el proyecto, ejemplo: lenguaje de programación, tipos de almacenamiento u otros) | * Lenguaje de programación: C. * Tipo de almacenamiento: Arreglos de tamaño fijo (nombres[5], cantidades[5], etc.). Esto limita la cantidad máxima de productos que se pueden gestionar a 5. * Estructura de datos simple: Se utilizan arreglos para almacenar la información de los productos. Para una mayor cantidad de datos o funcionalidades más complejas, se podrían considerar estructuras de datos más avanzadas. * Interfaz de usuario: Basada en texto plano en la consola. Esto limita la interactividad y la presentación visual de la información. * Cálculo de producción secuencial: La evaluación de la producción se realiza en el orden en que se ingresan los productos. No se considera la optimización de la producción basada en la disponibilidad de recursos o la prioridad de los productos. * Manejo básico de errores: El manejo de errores se limita a mensajes simples en la consola |

## 1. Introducción

El presente informe describe el desarrollo de un sistema de gestión para controlar la producción de productos en función de recursos disponibles y tiempo limitado. Este sistema permite agregar, editar, eliminar y mostrar productos, además de calcular si la producción es viable dadas las restricciones de tiempo y recursos.

## 2. Objetivos

* **General:** Desarrollar un sistema computacional que facilite la gestión y planificación de la producción de productos en un contexto de recursos y tiempo limitados.
* **Específicos:**
* Validar la disponibilidad de recursos (componentes y horas hombre) en relación con los requerimientos de un pedido recibido.
* Alertar al usuario sobre la insuficiencia de recursos en componentes, ofreciendo la posibilidad de gestionar una compra oportuna para aumentar el stock.
* Alertar al usuario sobre la insuficiencia de recursos en tiempo, ofreciendo la posibilidad de contratar personal adicional para cubrir la demanda.
* Minimizar la pérdida de ventas y mejorar la productividad general mediante la gestión proactiva de la falta de recursos.

## 3. Formulación del Problema

| Las empresas necesitan un control eficiente sobre la producción, tomando en cuenta limitaciones en recursos y tiempos. La ausencia de una herramienta que permita evaluar y planificar adecuadamente la producción puede ocasionar desperdicio o incumplimiento en los pedidos. Por tanto, se requiere un sistema que gestione estos factores de forma sencilla y confiable. |  |
| --- | --- |

## 4. Análisis de Alternativas

### Alternativa 1:

if-else: Para validar datos y tomar decisiones simples, como verificar si un nombre es válido o si un número es positivo.

fory while: Para recorrer arreglos, leer múltiples datos y validar entradas.

Se usa programación modular al definir funciones separadas para tareas específicas: validar si un carácter es dígito ( esDigito), validar nombres ( esNombreValido), leer enteros positivos ( leerEnteroPositivo) y buscar productos ( buscarProducto).

Los productos, sus cantidades, tiempos y recursos asociados se guardan en arreglos paralelos (varios arreglos que guardan información relacionada por índice).

Por ejemplo, el producto en nombres[i]tiene cantidad en cantidades[i], tiempo en tiempos[i]y recursos en recursos[i][j].

Se usan punteros para recorrer cadenas (por ejemplo, en la función esNombreValidocon \*(cadena + i)), lo que es más eficiente y directo en C.

Se valida que los números sean positivos y que los nombres no contengan dígitos.

El programa solicita repetidamente la entrada hasta que se reciba un dato válido.

Está bien organizado y modularizado sin complejidades innecesarias.

Ofrece un modelo realista y funcional para la gestión de productos y recursos.

|  |  |
| --- | --- |
| Ventajas | Desventajas |
| Simplicidad y claridad: El uso de if-else es directo y fácil de entender, ideal para principiantes. | Puede ser menos escalable: Cuando el menú crece mucho, varios if-else pueden hacer el código menos legible |
| Modularización básica: Uso de funciones para validar y buscar mantiene el uso de funciones para validar y buscar mantiene el código organizado sin complejidades. | No aproveche el uso de switch-case que puede ser más eficiente para menús numéricos |
| Facilidad para validar entradas: La lógica de validación está claramente expresada |  |
| Flexibilidad: Se puede modificar fácilmente para agregar nuevas opciones sin aprender sintaxis nueva |  |

### Alternativa 2:

Se utiliza una estructura switch para manejar el menú principal, mejorando la claridad del código y la navegación entre las diferentes opciones del programa

Mantiene las funciones que validan datos y gestionan operaciones específicas, como lectura de enteros, validación de nombres y búsqueda, manteniendo el código organizado

Igual que en la versión seleccionada, los datos de productos y recursos se guardan en arreglos paralelos.

Se asegura que el usuario ingresa datos válidos, con mensajes claros y repetición si hay errores.

Se controlan límites máximos (ejemplo: máximo 5 productos)

|  |  |
| --- | --- |
| Ventajas | Desventajas |
| Organización clara para menús: switch-case ofrece una estructura limpia para manejar múltiples opciones, mejora la legibilidad en menús grandes. | Puede ser un poco más complejo para principiantes: La sintaxis de switchy el manejo de breakre quiere cuidado para evitar errores |
| Eficiencia en la evaluación: Algunos compiladores optimizan switchpara ejecución más rápida que if-elseMúltiple. | No hay una gran diferencia funcional para este tamaño de programa: La mejora es más estética o estructural, pero funcionalmente similar. |
| Mejor para escalabilidad: Facilita agregar más casos sin anidar más condicionales. |  |
| Flexibilidad: Se puede modificar fácilmente para agregar nuevas opciones sin aprender sintaxis nueva |  |

## 5. Selección de la Mejor Alternativa

### a) ¿Cuál de estas alternativas sería la más fácil de implementar y por qué?

La opción más fácil de implementar es la opción 1.

Es más directo para validar cada condición de forma secuencial y manejar entradas erróneas sin complicaciones.

**Justificación técnica:**

* Simplicidad y facilidad de implementación: El uso de if-elsees más simple lo que facilita escribir, entender y mantener el código sin introducir errores relacionados con estructuras más complejas.
* Claridad en la lógica: La secuencia condicional lineal de if-else, hace que cada condición se evalúa explícitamente y sea fácil de seguir, lo cual es crucial para validar entradas y controlar flujos en un programa que gestiona datos como nombres y cantidades.
* Suficiente para el tamaño y alcance del programa: En programas pequeños o medianos, la diferencia de rendimiento entre if-elsey switch-casees insignificante. Lo importante es mantener un código claro y funcional.
* Facilita la depuración: La estructura sencilla hace más fácil detectar errores lógicos o de validación.

### b)¿Cómo garantizaría que la solución seleccionada es la más eficiente en términos de tiempo de ejecución y claridad en el código?

* Medición práctica del rendimiento: Para programas pequeños, el tiempo de ejecución puede medirse con pruebas reales (ejecutar el programa varias veces y comparar tiempos), aunque para este caso las diferencias serán mínimas.
* Claridad mediante revisión y comentarios: Garantizar la legibilidad con una buena estructura, comentarios claros y nombres descriptivos para funciones y variables.
* Pruebas funcionales exhaustivas: Asegurar que el código maneje correctamente todos los casos posibles (entradas válidas e inválidas) sin errores o comportamientos inesperados.
* Cumplimiento de las restricciones del proyecto: Mantener el código dentro de las limitaciones ayuda a evitar complejidades innecesarias que puedan entorpecer la comprensión.
* Facilidad de mantenimiento: Un código claro y simple facilita futuras modificaciones, lo que es clave para un desarrollo eficiente a largo plazo.

### c) ¿Qué aspectos son importantes al elegir una solución eficiente?

* Simplicidad y claridad del código: Que el código sea fácil de entender y mantener.
* Facilidad de validación y manejo de errores: Que sea fácil controlar que las entradas sean correctas.
* Escalabilidad: Que el código pueda crecer sin volverse confuso o demasiado complejo.
* Eficiencia: Que las operaciones internas (búsqueda, cálculos) no sean innecesariamente largos.
* Aprendizaje y dominio personal: Que se adapta al nivel de conocimiento para evitar frustraciones y facilitar la comprensión.

## 6. Desarrollo

### 6.1 Diagrama de flujo del programa

*Aquí insertar el diagrama de flujo realizado en Draw.io*

### 6.2 Enlace de Github del código fuente del programa:

[**https://github.com/TatianaTorres-code/PROYECTOS/tree/caded81f01e677d364ee80b2c129f3f800ec42f0/FABRICA**](https://github.com/TatianaTorres-code/PROYECTOS/tree/caded81f01e677d364ee80b2c129f3f800ec42f0/FABRICA)

### 6.3 Explicación de las partes más importantes del código

* **Validación de nombre y números:** Funciones esNombreValido y esDigito garantizan que los datos ingresados sean válidos.

Texto

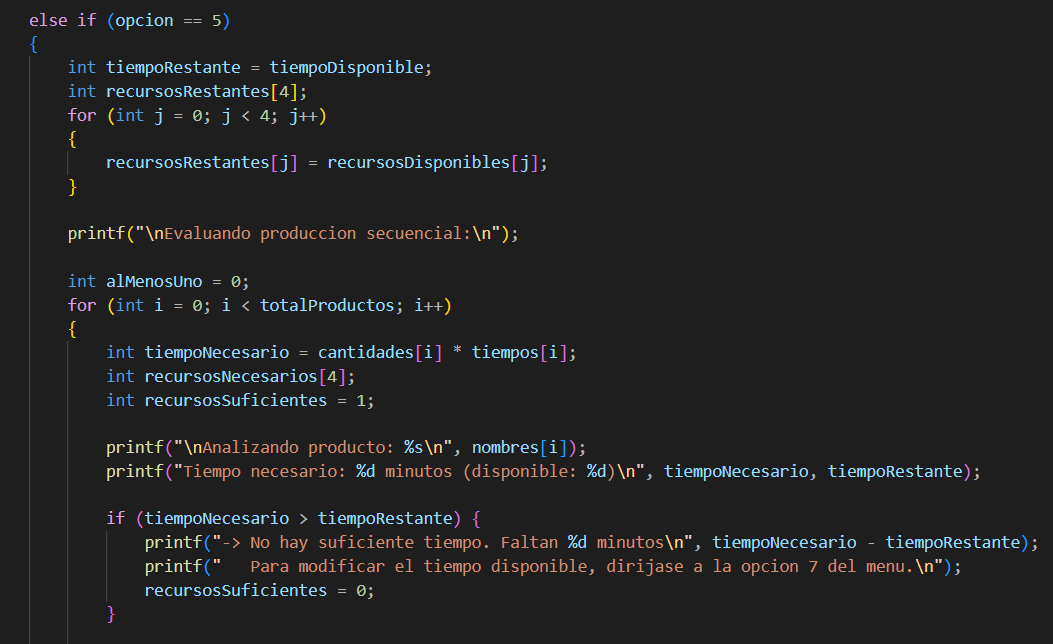
Descripción generada automáticamente

* **Gestión de productos:** Uso de arreglos para guardar datos y funciones para agregar, editar y eliminar.

Texto

Descripción generada automáticamente

* **Cálculo de producción:** Evalúa si hay recursos y tiempo suficiente para cumplir con la producción requerida.



Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

### 6.3 Imágenes y descripción de cada sección

**1.- Librerías:**

* stdio.h: Esta librería estándar de entrada/salida proporciona funciones como printf (para imprimir en la consola) y scanf (para leer datos desde la consola).
* string.h: Esta librería proporciona funciones para manipular cadenas de caracteres, como strlen (para obtener la longitud de una cadena) y strcmp (para comparar dos cadenas).

****

**2. Función esDigito:**

* Esta función toma un carácter c como entrada.
* Devuelve 1 (verdadero) si el carácter c es un dígito (entre '0' y '9').
* Devuelve 0 (falso) en caso contrario.
* **Propósito:** Validar si un carácter es un número.

Texto

Descripción generada automáticamente

**3. Función esNombreValido:**

* Esta función toma una cadena de caracteres nombre como entrada.
* Primero, verifica si la longitud de la cadena es 0. Si lo es, devuelve 0 (falso), indicando que el nombre no es válido porque está vacío.
* Luego, itera a través de cada carácter de la cadena.
* Dentro del bucle, llama a la función esDigito para verificar si el carácter actual es un dígito. Si encuentra un dígito, devuelve 0 (falso), ya que el nombre no debe contener números.
* Si el bucle termina sin encontrar ningún dígito, devuelve 1 (verdadero), indicando que el nombre es válido.
* Propósito: Validar si un nombre de producto no está vacío y no contiene números.

Texto

Descripción generada automáticamente

**4. Función leerEnteroPositivo:**

* Esta función toma un mensaje como entrada, que se mostrará al usuario.
* Utiliza un bucle do-while para solicitar al usuario que ingrese un número entero.
* Después de leer el número con scanf, verifica si el valor es menor que 0.
* Si el valor es negativo, muestra un mensaje de error y vuelve a pedir al usuario que ingrese un número.
* El bucle continúa hasta que el usuario ingresa un número mayor o igual a 0.
* Finalmente, devuelve el valor entero positivo ingresado por el usuario.
* Propósito: Asegurar que el usuario ingrese un número entero no negativo.

Texto

Descripción generada automáticamente

**5. Función buscarProducto:**

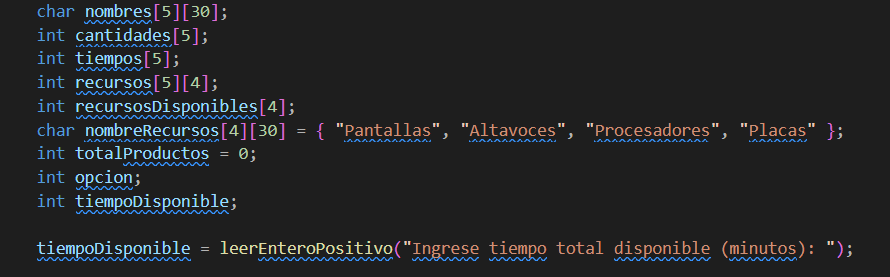
* Esta función toma un array bidimensional de nombres de productos (nombres), el número total de productos (total), y el nombre del producto que se desea buscar (nombreBuscado).
* Itera a través del array de nombres.
* En cada iteración, utiliza la función strcmp para comparar el nombre actual en el array con el nombreBuscado. strcmp devuelve 0 si las dos cadenas son iguales.
* Si encuentra una coincidencia, devuelve el índice i donde se encontró el producto.
* Si el bucle termina sin encontrar el producto, devuelve -1.
* Propósito: Buscar un producto por su nombre en la lista de productos y devolver su índice.

**6. Función main:**

Esta es la función principal donde comienza la ejecución del programa.

**6.1. Declaración de variables:**

* nombres[5][30]: Para almacenar los nombres de hasta 5 productos (cada nombre con una longitud máxima de 29 caracteres + el carácter nulo).
* cantidades[5]: Array para almacenar la cantidad demandada de cada producto.
* tiempos[5]: Para almacenar el tiempo de producción por unidad de cada producto (en minutos).
* recursos[5][4]: Para almacenar la cantidad de cada uno de los 4 recursos necesarios para producir una unidad de cada producto.
* recursosDisponibles[4]: Para almacenar la cantidad disponible de cada uno de los 4 recursos.
* nombreRecursos[4][30]: Para almacenar los nombres de los 4 recursos ("Pantallas", "Altavoces", "Procesadores", "Placas").
* totalProductos: Variable para llevar la cuenta del número actual de productos ingresados. Inicialmente es 0.
* opcion: Variable para almacenar la opción seleccionada por el usuario del menú.
* tiempoDisponible: Variable para almacenar el tiempo total disponible para la producción (en minutos).



**6.2. Inicialización del tiempo y los recursos disponibles:**

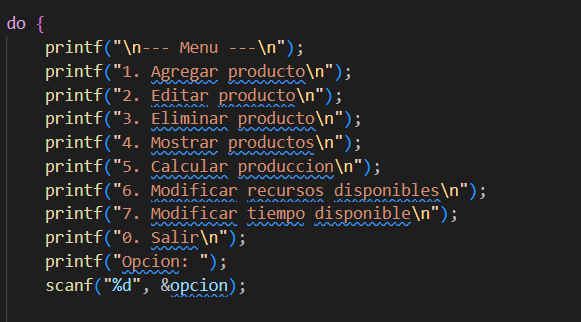
* Se llama a la función leerEnteroPositivo para que el usuario ingrese el tiempo total disponible.
* Se utiliza un bucle for para iterar a través de los nombres de los recursos y solicitar al usuario que ingrese la cantidad disponible para cada recurso, utilizando también leerEnteroPositivo para asegurar que se ingresen valores no negativos.

Texto

Descripción generada automáticamente

**6.3. Menú principal (bucle do-while):**

* Este bucle do-while muestra el menú de opciones al usuario y se repite hasta que el usuario selecciona la opción 0 (Salir).
* Dentro del bucle, se imprime el menú con las diferentes opciones.
* Se lee la opción ingresada por el usuario utilizando scanf.
* A continuación, se utiliza una serie de sentencias if-else if-else para ejecutar la acción correspondiente a la opción seleccionada.



**6.4. Manejo de las opciones del menú:**

* Opción 1: Agregar producto:
* Verifica si se ha alcanzado el límite máximo de productos (5).
* Solicita al usuario el nombre del producto y utiliza un bucle do-while junto con esNombreValido para asegurarse de que el nombre ingresado sea válido.
* Copia el nombre válido al array nombres.
* Solicita la cantidad demandada y el tiempo por unidad utilizando leerEnteroPositivo.
* Solicita la cantidad de cada recurso necesario por unidad del producto, utilizando un bucle for y leerEnteroPositivo.
* Incrementa el contador totalProductos.



* Opción 2: Editar producto:
* Solicita el nombre del producto que se desea editar.
* Llama a la función buscarProducto para encontrar el índice del producto en el array nombres.
* Si el producto no se encuentra, muestra un mensaje.
* Si se encuentra, solicita el nuevo nombre (validándolo con esNombreValido), la nueva cantidad demandada, el nuevo tiempo por unidad y las nuevas cantidades de cada recurso por unidad, actualizando los arrays correspondientes en el índice encontrado.

Texto

Descripción generada automáticamente

* Opción 3: Eliminar producto:
* Solicita el nombre del producto que se desea eliminar.
* Llama a la función buscarProducto para encontrar el índice del producto.
* Si el producto no se encuentra, muestra un mensaje.
* Si se encuentra, desplaza todos los elementos siguientes en los arrays nombres, cantidades, tiempos y recursos una posición hacia arriba para "eliminar" el producto.
* Decrementa el contador totalProductos.

Texto

Descripción generada automáticamente

* Opción 4: Mostrar productos:
* Imprime un encabezado "Productos:".
* Utiliza un bucle for para iterar a través de los productos ingresados (hasta totalProductos).
* Para cada producto, imprime su nombre, cantidad demandada y tiempo por unidad.
* Luego, utiliza otro bucle for anidado para imprimir la cantidad de cada recurso necesario por unidad de ese producto.

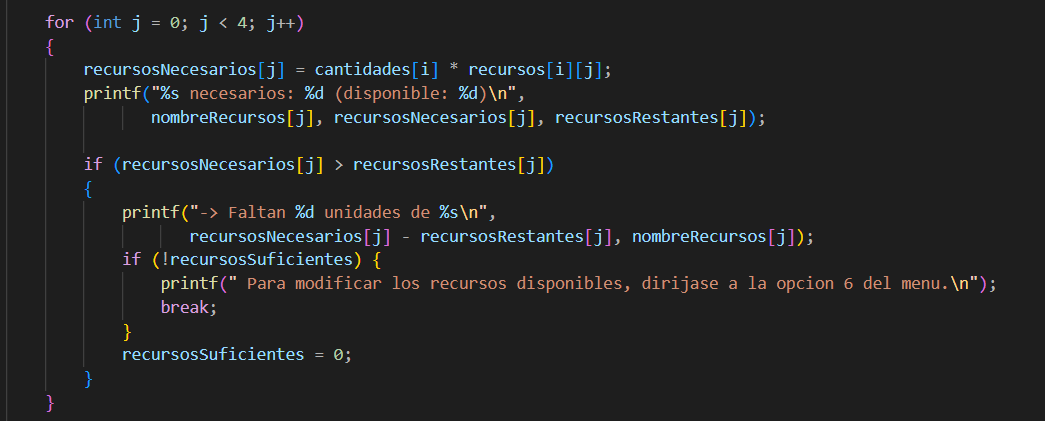
Texto

Descripción generada automáticamente

* Opción 5: Calcular producción:
* Inicializa tiempoRestante con el tiempoDisponible actual y crea un array recursosRestantes copiando los recursosDisponibles.
* Imprime un encabezado "Evaluando produccion secuencial:".
* Utiliza un bucle for para iterar a través de cada producto.
* Para cada producto:
  + Calcula el tiempoNecesario y las recursosNecesarios.
  + Verifica si hay suficiente tiempo disponible. Si no, muestra un mensaje y establece recursosSuficientes en 0.
  + Verifica si hay suficientes unidades de cada recurso disponible. Si no, muestra un mensaje y, si recursosSuficientes aún no es 0 (es decir, no se había quedado sin tiempo), lo establece en 0 y detiene la verificación de recursos para este producto (break).
  + Si tanto el tiempo como los recursos son suficientes (recursosSuficientes es 1), imprime un mensaje indicando que se puede producir el producto, actualiza el tiempoRestante y los recursosRestantes, y establece la bandera alMenosUno en 1 (para indicar que al menos un producto se puede producir completamente).
  + Si no hay suficientes recursos o tiempo, imprime un mensaje indicando que no se puede producir el producto.
* Después de analizar todos los productos, si alMenosUno es 0, imprime un mensaje indicando que no se puede producir ningún producto completamente.
* Finalmente, imprime un resumen del tiempoRestante y las cantidades restantes de cada recurso.

Texto

Descripción generada automáticamente



Texto

Descripción generada automáticamente

* Opción 6: Modificar recursos disponibles:
* Imprime un encabezado "--- Modificar Recursos Disponibles ---".
* Muestra los recursos disponibles actuales.
* Utiliza un bucle for para solicitar al usuario que ingrese las nuevas cantidades para cada recurso, utilizando leerEnteroPositivo.
* Muestra un mensaje de éxito y las nuevas cantidades de los recursos.

Texto

Descripción generada automáticamente

* Opción 7: Modificar tiempo disponible:
* Imprime un encabezado "--- Modificar Tiempo Disponible ---".
* Muestra el tiempo disponible actual.
* Solicita al usuario que ingrese el nuevo tiempo disponible utilizando leerEnteroPositivo.
* Muestra un mensaje de éxito y el nuevo tiempo disponible.

Texto

Descripción generada automáticamente

* Opción 0: Salir:
* Imprime un mensaje "Saliendo del programa...".

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

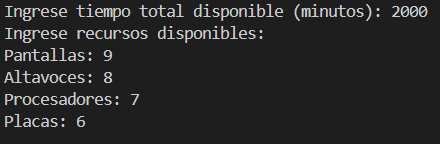
* Opción inválida:
* Si el usuario ingresa cualquier otra opción, se muestra un mensaje de "Opcion invalida."

Logotipo

Descripción generada automáticamente con confianza baja

### 6.4 Imágenes y la ejecución de cada sección

Ingresamos cantidad de recursos (stock actual)

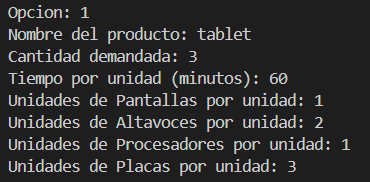


Menú de opciones:

Texto

Descripción generada automáticamente

1.- Agregar producto a fabricar



Opción 5: Analizamos si es posible la producción en función del stock de componentes y del tiempo de fabricación.

En este caso, el programa arroja un déficit de 3 placas, por lo que nos arroja el mensaje de que no es posible la fabricación y nos da opciones:

* Con opción 6: Modificar stock
* Con opción 2: Modificar el producto a fabricar

**Texto

Descripción generada automáticamente**

Opción 6: Modificamos el stock con 9 placas para esta producción:

**Texto

Descripción generada automáticamente**

Opción 5.- Validamos si ya es posible la fabricación

Texto

Descripción generada automáticamente

El programa también nos da el stock restante

**Texto

Descripción generada automáticamente**

Opción 4: Mostramos el detalle de producto fabricado y los insumos requeridos

**Texto

Descripción generada automáticamente**

Opción 3: Eliminamos el producto “Tablet”

**Texto

Descripción generada automáticamente con confianza baja**

Opción 4: Volvemos a pedir que el sistema muestre los productos fabricados y vemos que está en 0

**Forma

Descripción generada automáticamente con confianza media**

Opción 0: Salir



## 7. Conclusiones

* La opción seleccionada es la más fácil de usar en entender (o para nosotros).
* El sistema cumple con la gestión integral de productos y recursos, permitiendo planificar la producción según disponibilidad.
* La validación de datos mejora la confiabilidad y evita errores comunes.
* El uso de arreglos de tamaño fijo limita la capacidad del programa para escalar a un número mayor de productos sin modificar el código. Esperamos a futuro trabajar en mejoras.
* La interfaz por consola, aunque básica, es funcional y clara.