Universidad Mariano Gálvez de Guatemala
Ingeniería en Sistemas de Información y ciencias de la Computación
Algoritmos
Ingeniero Hernán Noé Velásquez González

Proyecto Final
Sistema de Reservas de Hotel

Integrantes:

No.	Carné	Nombre	% de participación
1	0900-25-28628	Pedro Luis Cruz Marroquin	100%
2	0900-24-3139	Cristian José Ixén Hernández	100%
3	0900-20-16748	Gabriel Alejandro Velasquez Acú	100%
4	0900-04-3673	Ana Lucrecia Torres	100%

25/10/2025

Indice

Introducción

En el desarrollo de programación estructurada, se crea un sistema básico orientado a la gestión de reservaciones en un hotel. Este proyecto tiene como objeto aplicar de manera integral el ciclo de resolución de problemas, desde el análisis inicial hasta la implementación funcional, utilizando las técnicas y estructuras aprendidas durante el curso.

El sistema de reservaciones permitirá automatizar procesos como el registro de habitaciones y clientes, la asignación y cancelación de reservaciones, la consulta de disponibilidad y la generación de facturas. Para lograrlo, se utilizará el pseudocódigo, diagramas de flujo, estructuras selectivas y repetitivas, funciones, procedimientos, arreglos, cadenas, registros y manejo de archivos.

Además de aplicar las funciones aprendidas, el sistema incorporará buenas prácticas de modularización, validación de datos, garantizando claridad y eficiencia.

Este proyecto no solo representa una aplicación práctica de los conocimientos adquiridos, sino también una oportunidad para fortalecer el pensamiento lógico.

Planteamiento del problema y análisis de requerimientos

Actualmente hay hoteles de tamaño pequeño que gestionan sus procesos de reservación de manera manual, lo que lleva a una serie de dificultades como pérdida de información, errores en la asignación de habitaciones, duplicación de reservas, y falta de control sobre la disponibilidad y facturación. Esta situación no solo afecta la eficiencia operativa del hotel.

Ante esta falta de optimización de información, nace la necesidad de desarrollar un sistema informático básico de reservaciones de hotel que permita automatizar y optimizar las tareas administrativas. El sistema debe ser capaz de registrar habitaciones y clientes, gestionar reservaciones, modificar datos, consultar disponibilidad y generar facturas de manera precisa y ordenada. Además, debe garantizar la persistencia de los datos mediante el uso de archivos, permitiendo así la continuidad de la información entre sesiones.

Este proyecto se enmarca dentro del proceso de aprendizaje de técnicas fundamentales de programación, por lo que su desarrollo debe aplicar el ciclo completo de resolución de problemas: análisis, diseño, codificación, prueba y documentación. Se utilizarán pseudocódigo, diagramas de flujo, estructuras selectivas y repetitivas, funciones, procedimientos, arreglos, cadenas, registros y archivos.

Consiste en diseñar una solución modular, clara y funcional que no solo resuelva las necesidades del hotel, sino que también demuestre la correcta aplicación de los conocimientos adquiridos en el curso.

Estaremos utilizando:

Estructuras selectivas: if / switch para navegación por menú y validaciones.

Estructuras repetitivas: while / for para mantener el sistema activo y recorrer datos.

Funciones y procedimientos: modularización del código por operación.

Arreglos, registros y cadenas: organización eficiente de la información.

Algoritmo de ordenamiento: ordenar habitaciones por tipo o clientes por nombre.

Recursividad: retorno al menú o reintento de búsqueda en caso de error.

Pseudocódigo

Pseudocódigo Program

```
Inicio

Clase Estática Program:
    Método Público Estático Main(string[] args):
        Llamar a DataStore.Init()


        // Cargar datos existentes desde los archivos al iniciar
        Llamar a DataStore.LoadHabitacionesIntoConfig()
        Llamar a DataStore.LoadClientesIntoModule()


        Mientras Verdadero Hacer:

            Limpiar la consola
            Mostrar "=== SISTEMA DE RESERVAS GUATEMALA ==="
            Mostrar "1) Habitaciones"
            Mostrar "2) Clientes"
            Mostrar "3) Administración"
            Mostrar "0) Salir"
            Mostrar "Selecciona una opción: "
            Leer la opción del usuario → op (texto recortado)


            Según op Hacer:

                Caso "1":
                    Llamar a ConfigHabitaciones.MostrarMenu()
                    Romper del switch
                Caso "2":
                    Llamar a Clientes.MostrarMenu()
                    Romper del switch
                Caso "3":
                    Llamar a Administracion.LoginYMenu()
                    Romper del switch
                Caso "0":
                    Mostrar "¡Hasta luego!"
                    Salir del método Main (return)

                Caso predeterminado:

                    Llamar a Pausa("Opción inválida.")
```

```
                Romper del switch
            Fin Según
        Fin Mientras
    Fin Método Main


    Método Público Estático Pausa(string? mensaje = null):
        Si mensaje no es nulo ni está vacío Entonces:
            Mostrar mensaje

        Fin Si

        Mostrar "Presiona una tecla para continuar..."
        Esperar que el usuario presione una tecla sin mostrarla
    Fin Método Pausa
Fin Clase
```

Pseudocódigo DataStore

Inicio

Clase Estática DataStore:
  Propiedades:
    BaseDir: string (ruta base en el escritorio)
    HabitacionesDir: string (subcarpeta para habitaciones)
    ClientesDir: string (subcarpeta para clientes)
    FacturasDir: string (subcarpeta para facturas)

    HabitacionesFile: ruta completa al archivo de habitaciones
    ClientesFile: ruta completa al archivo de clientes
    FacturasFile: ruta completa al archivo de facturas

    HabitacionesErrFile: ruta al archivo de errores de habitaciones
    ClientesErrFile: ruta al archivo de errores de clientes

  Método Init():
    Obtener la ruta del escritorio del usuario

    Definir la carpeta base llamada "Reservas Guatemala"
    Crear subcarpetas:
      - Habitaciones
      - Clientes
      - Facturas

    Crear las carpetas si no existen

    Verificar si existen los archivos principales:
      - Habitaciones.txt
      - Clientes.txt
      - Facturas.txt

    Si no existen, crearlos vacíos

Método Estático guardar habitaciones(Habitacion[] rooms):
Si el arreglo 'rooms' es nulo, terminar el método

Abrir un archivo de texto para sobrescribir en la ruta de HabitacionesFile

Para cada habitación 'h' en el arreglo:
Escribir una línea en el archivo con los siguientes datos separados por '|':
- Número
- Tipo (convertido a int)
- Precio (formato invariable)
- Estado (convertido a int)

Método Estático LoadHabitaciones() retorna Habitacion[]:
Si el archivo HabitacionesFile no existe, retornar un arreglo vacío

Leer todas las líneas del archivo

Inicializar:
- Una lista vacía de habitaciones
- Una lista vacía para mensajes de error

Para cada línea en el archivo:
- Si la línea está vacía o en blanco, continuar a la siguiente

Intentar:
- Dividir la línea usando el separador '|'
- Validar que tenga al menos 4 partes
- Parsear cada campo a su tipo correspondiente:
  - Número de habitación (int)
  - Tipo de habitación (enum desde int)
  - Precio (decimal con formato invariable)
  - Estado (enum desde int)

- Crear una instancia de Habitacion con esos valores
- Agregar la habitación a la lista

Si ocurre una excepción:
- Agregar a la lista de errores el número de línea, contenido, y mensaje de error

Si hay errores:
- Agregar los errores al archivo de errores HabitacionesErrFile, con la fecha actual

Retornar el arreglo de habitaciones ordenado por número

Método Estático cargar HabitacionesIntoConfig():

Llamar a LoadHabitaciones() para obtener un arreglo de habitaciones → rooms
Si rooms.Length > 0 Entonces
Llamar a Config habitaciones cargar desde persistencia(rooms)
Fin si

Método Estático SaveClientes(IEnumerable<Cliente> clientes):
Abrir un StreamWriter apuntando a ClientesFile, en modo sobrescribir (false)

Para cada cliente c en clientes:
dias ← c.DiasEstancia
hab ← Si c.HabitacionNumero tiene valor Entonces c.HabitacionNumero.ToString() Sino string vacío

price ← 0m (decimal)
Si c.Estancias no es nulo Y c.Estancias tiene al menos un elemento Entonces
last ← el último elemento en c.Estancias
price ← last.PrecioNoche
Fin si

Escribir línea al archivo con los siguientes campos separados por '|':
Esc(c.NombreCompleto) |
Esc(c.DPI) |
Esc(c.Nit) |
Esc(c.Telefono) |
dias |
hab |
price (convertido a string con CultureInfo.InvariantCulture)
Fin para

Método Estático LoadClientes() retorna List<Cliente>:

Crear lista vacía de clientes → res
Si el archivo ClientesFile no existe:
retornar res

Crear lista vacía de errores → errores
Leer todas las líneas del archivo ClientesFile → lines

Para i desde 0 hasta lines.Length-1:
ln ← lines[i]
Si ln está vacía o solo espacios:
continuar al siguiente i

Intentar:
p ← SplitSafe(ln) // dividir la línea en partes seguidas
Si p.Length < 7 entonces:
lanzar nueva FormatException("se esperaban 7 columnas")

Crear nueva instancia de Cliente → c
c.NombreCompleto ← UnEsc(p[0])
c.DPI ← UnEsc(p[1])
c.Nit ← UnEsc(p[2])
c.Telefono ← UnEsc(p[3])
c.DiasEstancia ← si int.TryParse(p[4], out d) entonces d sino 0

Si int.TryParse(p[5], out hab) entonces
c.HabitacionNumero ← hab
Sino
c.HabitacionNumero ← null

precioNoche ← 0m
_ ← decimal.TryParse(p[6], NumberStyles.Number,
CultureInfo.InvariantCulture, out precioNoche)

```
c.Estancias ← nueva lista de Movimiento Estancia
            Si c.Dias estancia > 0 Y c.Habitacion número no es null entonces:
               Agregar a c.Estancias un nuevo Movimiento estancia con:
                  Concepto = "Reserva importada"
                  Dias = c.DiasEstancia
                  PrecioNoche = precioNoche

            Agregar c a res

         Capturar excepción ex:
            errores.Agregar($"Línea {i+1}: \"{ln}\" -> {ex.Message}")

   Si errores tiene elementos:
      Escribir (apilar) al archivo ClientesErrFile:
         Fecha actual + mensaje de “Errores al cargar Clientes.txt”
         Luego cada línea de errores
         Luego una línea de separación (“-----…”)

   Retornar res


Método Estático LoadClientesIntoModule():

   list ← Llamar al método LoadClientes()
   Si list.Count > 0 Entonces
      Llamar a Clientes.CargarDesdePersistencia(list)
   FinSi


Método Estático AppendFactura(string facturaTexto, string clienteNombre,
DateTime fecha, string tipo) retorna string:

   Asegurarse de que exista el directorio FacturasDir (Crear si no existe)

   Abrir un StreamWriter en el archivo FacturasFile en modo "añadir" (append =
true)
      Escribir línea: facturaTexto
      Escribir línea vacía
      Escribir línea: nueva cadena de ‘=’ repetida 70 veces
      Escribir línea vacía
```

```
    Cerrar StreamWriter

    safeClient ← SanitizeFileName(clienteNombre)
    stamp ← fecha formateada como "yyyyMMdd-HHmmss"
    filename ← safeClient + "_" + tipo.ToUpperInvariant() + "_" + stamp + ".txt"
    fullPath ← Path.Combine(FacturasDir, filename)
    Escribir todo el contenido facturaTexto en el archivo fullPath

    Retornar fullPath

Método Privado Estático Esc(string s) retorna string:

    Si s es nulo entonces s ← ""
    Reemplazar en s: cada "\" por "\\\\"
    Reemplazar en s: cada "|" por "\\p"
    Retornar el resultado

Método Privado Estático UnEsc(string s) retorna string:
    Si s es nulo entonces s ← ""
    Reemplazar en s: cada "\\p" por "|"
    Reemplazar en s: cada "\\\\" por "\"
    Retornar el resultado

Método Privado Estático SplitSafe(string line) retorna string[]:

    Crear lista vacía de strings → parts
    cur ← ""  // acumulador de caracteres
    slash ← false  // indica si el caracter anterior fue '\'

    Para cada caracter ch en line:
        Si slash = true Entonces
            Si ch == 'p' Entonces
                cur += "|"
            Sino
                cur += "\" + ch
            FinSi
            slash ← false
        Sino
            Si ch == '\' Entonces
                slash ← true
```

```
        Sino si ch == '|' Entonces
            parts.Agregar(cur)
            cur ← ""
        Sino
            cur += ch
        Fin si
    Fin si
Fin para

parts.Agregar(cur)  // agregar la última parte
Retornar parts como arreglo


Método Privado Estático SanitizeFileName(string s) retorna string:

    invalid ← Obtener arreglo de caracteres inválidos para nombres de archivo
    cleaned ← nuevo string formado por:
        Para cada ch en (s o "" si s es nulo):
            Si invalid contiene ch Entonces
                '_'
            Sino
                ch
    Fin para

    Si cleaned es nulo o sólo espacios Entonces
        cleaned ← "Cliente"
    FinSi

    Si cleaned.Length > 60 Entonces
        cleaned ← cleaned.Substring(0, 60)
    Fin si

    Retornar cleaned
```

Pseudocodigo Habitaciones

INICIO

Enumeración TipoHabitacion

1 - Sencilla
2- Doble
3- Suite

Enumeración Estado

1- Libre
2- Ocupada

ESTRUCTURA Habitación

- Numero: Entero
- Tipo: Tipo de habitación
- Precio: Decimal
- Estado: Estado

Configuración de Habitaciones

VARIABLE PRIVADA Habitaciones: Arreglo de Habitación (inicialmente vacío)
FUNCION PRIVADA Esta Inicializado ()
RETORNAR longitud de Habitaciones > 0

FUNCION PUBLICA Hay Habitaciones ()
RETORNAR longitud de Habitaciones > 0

FUNCION PUBLICA Obtener Habitaciones() : Arreglo de Habitación
RETORNAR Habitaciones

Función Cargar Desde Persistencia(rooms : Arreglo de Habitación)

SI rooms ES NULO ENTONCES
Habitaciones ← nuevo arreglo vacío
SINO
Habitaciones ← rooms
FIN SI

```
Función  Existe Habitación (numero: Entero) : Booleano
        Retornar  número >= 1 Y número <= longitud de Habitaciones

  Función  Esta Libre(numero : Entero)
        SI NO Existe Habitación (numero) ENTONCES
        RETORNAR FALSO
        FIN SI
Retornar  Habitaciones [numero - 1].Estado = Libre


Función  Ocupar (numero: Entero)

        SI NO Existe Habitación (numero) ENTONCES
        RETORNAR
        FALSO
        FIN SI
        i ← numero - 1


        SI Habitaciones[i].Estado = Ocupada ENTONCES
        RETORNAR FALSO
        FIN SI
         h ← Habitaciones[i]
         h.Estado ← Ocupada
         Habitaciones[i] ← h
         LLAMAR DataStore.GuardarHabitaciones(Habitaciones)

         RETORNAR VERDADERO


Función  Liberar (numero: Entero)
         SI NO Existe Habitación (numero) ENTONCES
            RETORNAR FALSO
         FIN SI
         i ← numero - 1
         h ← Habitaciones[i]
         h.Estado ← Libre
         Habitaciones[i] ← h
         LLAMAR Data Store Guardar Habitaciones (Habitaciones)

         RETORNAR VERDADERO

    Fin función

FIN
```

```
Función  Precio Por Noche(numero : Entero) : Decimal
        SI NO Existe Habitación (numero) ENTONCES
        RETORNAR 0
        FIN SI
Retornar  Habitaciones [numero - 1].Precio
Fin función

Función  Hay Libres () : Booleano
        PARA CADA h EN Habitaciones HACER
        SI h.Estado = Libre ENTONCES
        RETORNAR VERDADERO
        FIN SI
        FIN PARA
Retornar
Fin función

Procedimiento  Mostrar Menú()
        Mientras es verdadero
        Limpiar consola
        Imprimir  " === Configuración de Habitaciones === "
        Imprimir "1) Registrar habitaciones"
        Imprimir  "2) Listado de habitaciones (solo consulta)"
        Imprimir "3) Editar Información de habitación (tipo / precio)"
        Imprimir "0) Volver"
        Imprimir "Selecciona una opción: "

      Leer opción desde consola
      opción ← TRIM(opcion)

      Según  opción hacer
        Caso  "1":
        Llamar  Crear habitaciones ()
        Caso  "2":
        Llamar  listar habitaciones (pausar = VERDADERO)
        CASO "3":
        Llamar  editar habitación ()
        Caso  "0":
        Salir del procedimiento
        OTRO CASO:
        LLAMAR Pausa("Opción inválida.")
        Fin según
        Fin mientras
Fin procedimiento
```

```
Procedimiento  crear habitaciones ()
   Limpiar consola
   Imprimir  " === Opciones para habitación === "

SI Habitaciones ya han sido cargadas (es decir: _inicializado = verdadero) entonces
         Imprimir  "Ya existen habitaciones."
         Imprimir "1) Conservar y AGREGAR más"
         Imprimir "2) REEMPLAZAR todo el listado (se conservarán las ocupadas)"
         Imprimir "0) Cancelar y volver"
         Imprimir "Elige una opción: "

       Leer  respuesta desde consola
       Respuesta ← TRIM(respuesta)

       SI respuesta = "0" entonces
          Llamar  pausa("Operación cancelada.")
          SALIR DEL PROCEDIMIENTO
       SINO SI respuesta = "1" ENTONCES
          agregar ← VERDADERO
       SINO SI respuesta = "2" ENTONCES
          agregar ← FALSO
       SINO
          LLAMAR Pausa("Opción inválida. Operación cancelada.")
          Salir del procedimiento
       Fin si
   Fin si

Cantidad ← leer entero positivo (
   SI agregar entonces
      "¿Cuántas habitaciones adicionales  deseas registrar?"
   SINO
      "¿Cuántas habitaciones deseas registrar?"


SI agregar ENTONCES
         old ← longitud de Habitaciones
         nuevas ← nuevo arreglo de Habitación con tamaño old + cantidad

COPIAR elementos de Habitaciones en nuevas

         Imprimir  línea en blanco
         Imprimir "Tipos: 1) Sencilla  2) Doble  3) Suite"
         Imprimir  línea en blanco
```

```
Para  i desde old HASTA nuevas Length - 1 HACER
Imprimir  "--- Habitación #" + (i + 1) + " ---"

        Tipo ← Leer tipo habitación ("Tipo (1-3): ")
        Precio ← Leer decimal positivo ("Precio (ej. 1200.50): ")

Nuevas[i] ← nueva Habitación con:
          Número ← i + 1
          Tipo ← tipo
          Precio ← precio
          Estado ← Libre
  Imprimir  línea en blanco
   Fin para

Habitaciones ← nuevas
   Llamar  guardar habitaciones (Habitaciones)
   Llamar  Pausa ("Habitaciones agregadas correctamente.")
   Salir del procedimiento
Fin si

Ocupadas ← nueva lista vacía de Habitación
mapOldToNew ← nuevo diccionario (clave: entero, valor: entero)

SI Habitaciones ya han sido cargadas entonces
Para cada  h en Habitaciones hacer
SI h.Estado = Ocupada entonces
Anadir  h A ocupadas
Fin si
Fin para

Ordenar  ocupadas por  Número de habitación (ascendente)
Fin si

Total ← cantidad + cantidad de habitaciones ocupadas
Resultado ← nuevo arreglo de habitación con tamaño total

Para  i desde  0 hasta  cantidad de ocupadas – 1 hacer
   h ← ocupadas[i]
   Nuevo número ← i + 1

mapOldToNew[h.Numero] ← nuevoNumero
   h.Numero ← nuevoNumero
   resultado[i] ← h
Fin para
```

```
Imprimir  línea en blanco
Imprimir  "Tipos: 1) Sencilla  2) Doble  3) Suite"
Imprimir línea en blanco

Para  i desde  cantidad de ocupadas hasta  total - 1 hacer
   Imprimir  "--- Habitación #" + (i + 1) + " ---"


Tipo ← Leer tipo habitación ("Tipo (1-3): ")
   Precio ← leer decimal positivo ("Precio (ej. 1200.50): ")

  Resultado[i] ← nueva habitación con:
       Número ← i + 1
       Tipo ← tipo
       Precio ← precio
       Estado ← Libre

   Imprimir  línea en blanco
Fin para

Habitaciones ← resultado

SI mapOldToNew contiene elementos ENTONCES
   LLAMAR clientes Aplicar remapeo habitaciones (mapOldToNew)
Fin si

Llamar  Guardar Habitaciones (Habitaciones)
Llamar  Guardar Clientes (Clientes.ObtenerClientesParaGuardar())

Llamar Pausa ("Proceso completado (ocupadas conservadas y nuevas libres).")

Procedimiento  EditarHabitacion()

         Si no  hay habitaciones cargadas entonces
         Llamar  Pausa("Aún no hay habitaciones para editar.")
         Salir del procedimiento
         Fin si

   Limpiar pantalla
   Imprimir  "=== editar habitación  ==="
   Llamar  listar habitaciones (pausar = falso )
   Imprimir  "Ingresa el número de la habitación a editar (0 = cancelar):"
   Leer  num como entero
```

```
SI entrada no es válida O num < 0 O num > cantidad de habitaciones entonces
      Llamar Pausa("Entrada inválida.")
      Salir del procedimiento
   Fin si

   SI num = 0 entonces
      Salir del procedimiento
   FIN SI

   idx ← num - 1
   h ← Habitaciones[idx]

   SI h.Estado = Ocupada ENTONCES
      LLAMAR Pausa("La habitación está OCUPADA y no puede ser modificada.")
      Salir del procedimiento
   Fin si
        Imprimir  "Editando Habitación #" + h.Numero
        Imprimir  "(Tipo actual: " + h.Tipo + ", Precio actual: " + h.Precio + ")"
        Imprimir  "Deja vacío para conservar el valor actual."

Imprimir  "Nuevo tipo (1=Sencilla, 2=Doble, 3=Suite):"
   Leer  tipoTxt como texto

   SI tipoTxt NO está vacío Y ES "1", "2" o "3" entonces
      h.Tipo ← convertir tipoTxt a TipoHabitacion
   Fin si

Imprimir  "Nuevo precio por noche:"
Leer precioTxt COMO TEXTO

SI precioTxt NO está vacío ENTONCES
      SI precioTxt puede convertirse a decimal válido (cultura local o invariante) Y ≥
0 ENTONCES
         h.Precio ← valor convertido
      SINO
         LLAMAR Pausa ("Precio inválido. No se cambió el precio.")
      FIN SI
   FIN SI

Habitaciones [idx] ← h
   LLAMAR Guardar habitaciones (Habitaciones)
   LLAMAR Pausa ("Habitación actualizada.")

Fin procedimiento
```

```
Procedimiento Listar habitaciones  (pausar : Booleano)
        Imprimir  "No.      | Tipo      | Precio Noche          | Estado"
        Imprimir  "- - - - - - - - - - - - - - - - - - - - - - - - -"

   Para cada  h en  Habitaciones hacer
      Imprimir  número, tipo, precio y estado con formato
   Fin para

   Imprimir  línea en blanco

   SI pausar = verdadero entonces
      Llamar  Pausa()
   Fin si
Fin procedimiento

Función  Leer entero positivo(prompt : Texto) : Entero
   Mientras  verdadero hacer
      Imprimir  prompt
      Leer txt como  texto desde consola

      SI txt puede convertirse a entero v Y v > 0 entonces
         Retornar v
      Fin si

Imprimir  "Valor inválido. Ingresa un entero positivo."
Fin mientras
Fin función

Función  Leer decimal positivo(prompt : Texto) : Decimal
   Mientras  verdadero hacer
      Imprimir  prompt
      Leer txt como  texto desde consola

      SI txt puede convertirse a decimal v (usando cultura actual) Y v ≥ 0 entonces
         Retornar  v
      SI txt puede convertirse a decimal v (usando cultura invariante) Y v ≥ 0 entonces
         Retornar v

      Imprimir  "Valor inválido. Ingresa un número decimal (ej. 1200.50)."
   Fin mientras
Fin función
```

```
Función  leer tipo habitación (prompt : Texto) : TipoHabitacion
   Mientras  verdadero hacer
      Imprimir  prompt
      Leer  txt COMO texto desde consola

      Si  txt ES "1" o "2" o "3" entonces
         Retornar  tipo habitación convertido desde txt
      Fin si

      Imprimir  "Opción inválida. Usa 1, 2 o 3."
   Fin mientras
Fin función
```

Pseudocodigo Clientes

Inicio

Función movimiento estancia

Atributo: Concepto de tipo Texto (inicializado como cadena vacía)
Atributo: Dias de tipo Entero
Atributo: Precio noche de tipo Decimal
Atributo: Fecha de tipo Fecha/Hora (inicializado con la fecha y hora actual)
Método Constructor (opcional)
Inicializa los atributos si se desea establecer valores por defecto

Fin función

Clase Cliente

Atributo: NombreCompleto de tipo Texto (inicializado como cadena vacía)
Atributo: DPI de tipo Texto (inicializado como cadena vacía)
Atributo: Nit de tipo Texto (inicializado como cadena vacía)
Atributo: Telefono de tipo Texto (inicializado como cadena vacía)

// Información de la estancia
Atributo: Días estancia de tipo Entero
Atributo: Habitación número de tipo Entero (puede ser nulo)

// Información administrativa
Atributo: ultimo comprobante de tipo Texto (inicializado como cadena vacía)

// Lista de movimientos o cargos en la estancia del cliente
Atributo: Estancias de tipo Lista de MovimientoEstancia (inicializada vacía)

Fin función

Función Estática Clientes

Atributo Privado: Lista _clientes de tipo Lista de Cliente (inicializada vacía)
Método Público: ObtenerClientesParaGuardar
Retorna la lista _clientes

Método Público: CargarDesdePersistencia (parámetro: lista de clientes)
Limpiar la lista _clientes
Agregar todos los elementos de la lista recibida a _clientes

Método MostrarMenu
Repetir indefinidamente:
Limpiar pantalla
Mostrar el menú de opciones:
1) Registrar cliente
2) Listado de clientes
0) Volver
Solicitar opción al usuario

Según la opción ingresada:
Si opción es "1":
Llamar al método CrearCliente
Si opción es "2":
Llamar al método ListadoClientesMenu
Si opción es "0":
Salir del método (terminar menú)
Cualquier otro valor:
Mostrar mensaje: "Opción inválida"
Llamar a método Pausa del programa principal

Fin función

Función Estático Crear Cliente

Limpiar pantalla
Mostrar "=== Registrar Cliente ==="

Crear un nuevo objeto Cliente llamado c
Asignar a c los siguientes valores obtenidos desde consola:
- Nombre completo: leer texto no vacío con el mensaje "Nombre completo:"
- DPI: leer texto no vacío con el mensaje "DPI:"
- Nit: leer texto opcional con el mensaje "NIT (Enter si no tiene):"
- Teléfono: leer texto no vacío con el mensaje "Teléfono:"
- DiasEstancia: leer número entero no negativo con el mensaje "Días de estancia (0 si solo registro):"

Si el campo Nit está vacío o solo contiene espacios
Asignar "CF" como valor predeterminado a Nit

Si DiasEstancia es mayor que 0
Si no hay habitaciones o no hay habitaciones libres disponibles
Mostrar mensaje de advertencia: "No hay habitaciones libres. Se registrará al cliente sin hospedaje."
Establecer Días estancia a 0
Establecer Habitación número como nulo
De lo contrario
Mostrar línea en blanco

Si se puede elegir una habitación libre y se obtiene el número de habitación (numHab)
Asignar numHab a Habitación número del cliente
Marcar esa habitación como ocupada

Obtener el precio por noche de esa habitación (px)

Agregar a la lista Estancias del cliente un nuevo MovimientoEstancia con:
- Concepto: "Reserva inicial"
- Dias: valor de Dias estancia del cliente
- PrecioNoche: valor px obtenido
Si la elección de habitación fue cancelada
Establecer Dias estancia a 0
Establecer Habitacion numero como nulo
Mostrar mensaje: "Asignación cancelada. El cliente quedó sin hospedaje."

Agregar el cliente a la lista _clientes
Guardar la lista de clientes en almacenamiento (DataStore.SaveClientes)
Mostrar mensaje: "Cliente registrado correctamente."

Fin fusión

Función Estático ListadoClientesMenu

Repetir indefinidamente:

Limpiar la pantalla
Mostrar "=== LISTADO DE CLIENTES ==="

Si la lista de clientes (_clientes) está vacía

Mostrar "No hay clientes registrados."
Sino
Llamar al método Imprimir tabla clientes para mostrar la lista de clientes en formato tabla

Mostrar opciones disponibles al usuario:
1) Asignar hospedaje
2) Cancelar estancia
3) Finalizar estancia
4) Editar información del cliente
5) Eliminar cliente
0) Volver

Solicitar al usuario que ingrese una opción
Leer y limpiar el valor ingresado

Según la opción seleccionada:
Si es "1"
Llamar al método Crear hospedaje
Si es "2"
Llamar al método Cancelar estancia
Si es "3"
Llamar al método Finalizar estancia
Si es "4"
Llamar al método Editar cliente
Si es "5"
Llamar al método Eliminar cliente
Si es "0"
Salir del método (terminar el menú)
En cualquier otro caso
Mostrar mensaje de error: "Opción inválida."
Llamar al método Pausa del programa para esperar al usuario

Fin función

Función ImprimirTablaClientes

Escribir en consola: "Id | Nombre | DPI | Teléfono | Días | Habitación"
Escribir en consola: "----------"

Para i desde 0 hasta (_clientes.Count - 1) hacer:
c = _clientes[i] // Obtener cliente actual

```
        Si c.HabitacionNumero tiene valor
            hab = convertir a texto c.Habitacion numero
        Sino
            hab = "-"

        Escribir en consola:
            Número (i + 1),
            NombreCompleto
            DPI
            Teléfono
            Días de estancia
            Número de habitación

Fin función



Función  EditarCliente

    Si la lista _clientes está vacía entonces
        Mostrar mensaje: "No hay clientes para editar."
        Esperar pausa del usuario
        Terminar método
    Fin Si

    Mostrar mensaje: "Ingresa el Id del cliente a editar (0 = cancelar):"
    Leer entrada del usuario y tratar de convertirla a entero idx1

    Si la conversión falla o idx1 es menor que 0 o mayor que la cantidad de clientes
entonces
        Mostrar mensaje: "Id inválido."
        Esperar pausa del usuario
        Terminar método
    Fin Si

    Si idx1 es 0 entonces
        Terminar método (cancelar edición)
    Fin Si

    Calcular índice real: idx = idx1 - 1
    Obtener cliente c de la lista _clientes en posición idx
```

```
    Mostrar información actual del cliente (Nombre, DPI, Teléfono, NIT)
    Mostrar mensaje: "Deja vacío para conservar el valor actual."

    Solicitar al usuario ingresar nuevo nombre
    Leer texto nombre
    Si nombre no está vacío ni es solo espacios en blanco entonces
        Asignar nombre limpio (sin espacios extras) a c.Nombre completo
    Fin Si

    Solicitar nuevo DPI
    Leer texto dpi
    Si dpi no está vacío ni es solo espacios en blanco entonces
        Asignar dpi limpio a c.DPI
    Fin Si

    Solicitar nuevo teléfono
    Leer texto tel
    Si tel no está vacío ni es solo espacios en blanco entonces
        Asignar tel limpio a c.Telefono
    Fin Si

    Solicitar nuevo NIT (con instrucción que Enter asigna "CF")
    Leer texto nit
    Si nit está vacío o solo espacios entonces
        Asignar "CF" a c.Nit
    Sino
        Asignar nit limpio a c.Nit
    Fin Si

    Guardar la lista de clientes actualizada en almacenamiento
(DataStore.SaveClientes)

    Mostrar mensaje: "Cliente actualizado."
    Esperar pausa del usuario

Fin función



Función EliminarCliente

    Si la lista _clientes está vacía entonces
        Mostrar mensaje: "No hay clientes para eliminar."
        Esperar pausa del usuario
```

```
    Terminar método
  Fin si

  Mostrar mensaje: "Ingresa el Id del cliente a eliminar (0 = cancelar):"
  Leer entrada y tratar de convertirla a entero idx1

  Si la conversión falla o idx1 es menor que 0 o mayor que la cantidad de clientes
entonces
    Mostrar mensaje: "Id inválido."
    Esperar pausa del usuario
    Terminar método
  Fin Si

  Si idx1 es 0 entonces
    Terminar método (cancelar operación)
  Fin Si

  Calcular índice real: idx = idx1 - 1
  Obtener cliente c de la lista _clientes en la posición idx

  Si cliente tiene estancia activa (Dias estancia > 0) y habitación asignada
(Habitación número != null) entonces
    Mostrar mensaje:
      "El cliente tiene una reserva ACTIVA. Debes elegir una opción:"
      "1) Finalizar estancia"
      "2) Cancelar estancia (reembolso 75% de lo no usado)"
      "0) Volver (no eliminar)"
    Solicitar opción al usuario
    Leer opción op


    Si op es "1" entonces
      Llamar a método Finalizar estancia de cliente(idx)

    Sino si op es "2" entonces
      Llamar a método Cancelar estancia de cliente(idx)

    Sino
      Mostrar mensaje: "Operación cancelada. El cliente no fue eliminado."
      Esperar pausa del usuario
      Terminar método
    Fin Si
  Fin Si
```

```
    Eliminar cliente en índice idx de la lista _clientes
    Guardar lista actualizada en almacenamiento (DataStore.SaveClientes)
    Mostrar mensaje: "Cliente eliminado correctamente."
    Esperar pausa del usuario

Fin función


Método CrearHospedaje

    Mostrar mensaje: "Ingresa el Id del cliente (0 = cancelar):"
    Leer entrada y tratar de convertirla a entero idx1

    Si la conversión falla o idx1 es menor que 0 o mayor que la cantidad de clientes
entonces
        Mostrar mensaje: "Id inválido."
        Esperar pausa del usuario
        Terminar método
    Fin Si

    Si idx1 es 0 entonces
        Terminar método (cancelar operación)
    Fin Si

    Calcular índice real: idx = idx1 - 1
    Obtener cliente c de la lista _clientes en la posición idx

    Si cliente tiene días de estancia activos (c.Dias estancia > 0) y tiene habitación
asignada (c.Habitacion número es habActual) entonces
        Mostrar mensaje:
            "Este cliente ya tiene {c.Dias estancia} día(s) en la habitación
#{habActual}."

        Si el usuario confirma (pregunta: "¿Desea aumentar los días de estancia?
(S/N):") entonces
            Pedir al usuario cuántos días adicionales desea agregar
            Leer entero positivo extra

            Si el usuario confirma mantener la misma habitación (pregunta:
"¿Mantener la misma habitación? (S/N):") entonces
                Sumar extra a c.Dias estancia
                Agregar a la lista c.Estancias un nuevo Movimiento estancia con:
                    Concepto = "Extensión"
                    Dias = extra
```

Precio noche = obtener precio por noche para la habitación habActual

Guardar lista actualizada de clientes (DataStore.SaveClientes)
Mostrar mensaje: "Días de estancia actualizados en la misma habitación."
Esperar pausa del usuario
Terminar método
Fin Si
Fin Si
Fin Si

Fin función

Si el usuario NO desea mantener la misma habitación entonces

Solicitar al usuario el código de comprobante único para la primera reserva y guardarlo en comp

Si no hay habitaciones libres entonces
Mostrar mensaje: "No hay habitaciones libres para el cambio. Operación cancelada."
Esperar pausa del usuario
Terminar método
Fin Si

Calcular total pagado hasta ahora como la suma de (Dias * PrecioNoche) de todos los movimientos en c.Estancias

Construir factura de texto llamada factura cambio con:
Cliente c
Comprobante comp
esCancelacion = false
diasUsados = c.DiasEstancia
diasNoUsados = 0
montoUsados = totalPagadoHastaAhora
reembolso = 0

Obtener fecha/hora actual tsCambio
Guardar factura en archivo con DataStore.AppendFactura(facturaCambio, nombre cliente, tsCambio, "CambioHabitacion") y guardar ruta en pathCambio

```
   Mostrar mensaje con ruta del archivo generado
   Mostrar línea vacía para separación
   Solicitar al usuario elegir una habitación libre con validación

   Si el usuario cancela (no elige habitación) entonces
      Mostrar mensaje: "Cambio cancelado por el usuario."
      Esperar pausa
      Terminar método
   Fin Si

   Liberar habitación actual habActual
   Ocupar la nueva habitación elegida nuevaHab

   Actualizar cliente c con:
      HabitacionNumero = nuevaHab
      DiasEstancia = extra (días que se agregaron)
      UltimoComprobante = comp
      Reemplazar lista c.Estancias por una nueva lista con un MovimientoEstancia
que contiene:
         Concepto = "Reserva (cambio de habitación)"
         Dias = extra
         PrecioNoche = precio por noche de nuevaHab

   Guardar configuración de habitaciones actualizada
   Guardar lista de clientes actualizada

   Mostrar mensaje: "Cambio realizado. Nueva habitación #{nuevaHab}. Días
asignados: {c.DiasEstancia}."
   Esperar pausa
   Terminar método

Sino (si el usuario NO desea aumentar días o no hay hospedaje previo)
   Mostrar mensaje: "No se realizaron cambios."
   Esperar pausa
   Terminar método
Fin Si

Si no hay habitaciones configuradas o no hay habitaciones libres entonces
   Mostrar mensaje: "No hay habitaciones libres para asignar."
   Esperar pausa
   Terminar método
Fin Si
```

```
Pedir al usuario ingresar cantidad de días de estancia positivos
Guardar en c.DiasEstancia
Solicitar al usuario elegir una habitación libre con validación

Si el usuario elige una habitación numHab entonces
   Asignar numHab a c.HabitacionNumero
   Marcar habitación numHab como ocupada

   Reemplazar c.Estancias por una nueva lista con un MovimientoEstancia:
      Concepto = "Reserva inicial"
      Dias = c.DiasEstancia
      PrecioNoche = precio por noche de numHab

   Guardar configuración de habitaciones actualizada
   Guardar lista de clientes actualizada

   Mostrar mensaje: "Hospedaje creado y habitación asignada."
   Esperar pausa
Sino

   Establecer c.DiasEstancia = 0
   Establecer c.HabitacionNumero = null
   Limpiar lista c.Estancias
   Guardar lista de clientes actualizada
   Mostrar mensaje: "Asignación cancelada. El cliente continúa sin hospedaje."
   Esperar pausa
Fin Si


Función  CancelarEstancia

   Si todos los clientes en _clientes tienen DiasEstancia igual a 0 entonces
      Mostrar mensaje: "Ningún cliente tiene estancia para cancelar."
      Esperar pausa del usuario
      Terminar método
   Fin Si

   Mostrar mensaje: "Ingresa el Id del cliente (0 = cancelar):"
   Leer entrada y tratar de convertirla a entero idx1
```

Si la conversión falla o idx1 es menor que 0 o mayor que la cantidad de clientes entonces
Mostrar mensaje: "Id inválido."
Esperar pausa del usuario
Terminar método
Fin Si

Si idx1 es 0 entonces
Terminar método (cancelar operación)
Fin Si

Calcular índice real: idx = idx1 - 1
Llamar al método CancelarEstanciaDeCliente pasando idx

Fin función

Función CancelarEstanciaDeCliente con parámetro idx (entero)

Obtener cliente c de la lista _clientes en la posición idx

Si c.DiasEstancia es 0 o c.HabitacionNumero es null entonces
Mostrar mensaje: "Ese cliente no tiene una estancia activa."
Esperar pausa del usuario
Terminar método
Fin Si

Pedir al usuario ingresar días usados (entero entre 0 y c.DiasEstancia) y guardar en usados

Calcular noUsados como c.DiasEstancia - usados

Solicitar al usuario un comprobante único de pago de los días usados y guardar en comp

Llamar a CalcularMontosUsadosNoUsados pasando c.Estancias y usados, y obtener montoUsados y montoNoUsados

Calcular reembolso como 75% de montoNoUsados (reembolso = montoNoUsados * 0.75)

Construir texto de factura llamando a ConstruirFacturaTexto con parámetros:
cliente c
Comprobante comp

esCancelacion = true
diasUsados = usados
diasNoUsados = noUsados
montoUsados = montoUsados
reembolso = reembolso

Mostrar la factura por consola

Obtener fecha/hora actual y guardar en ts

Guardar la factura en un archivo con DataStore.AppendFactura(factura, nombre cliente, ts, "Cancelacion")
Mostrar mensaje con la ruta del archivo guardado

Liberar la habitación ocupada por el cliente llamando Config habitaciones Liberar con c.habitacion numero

Actualizar cliente c:
DiasEstancia = 0
HabitacionNumero = null
Limpiar lista c.Estancias
UltimoComprobante = comp

Guardar configuración de habitaciones actualizada
Guardar lista de clientes actualizada

Mostrar mensaje: "Cancelación procesada."
Esperar pausa del usuario

Fin función

Función FinalizarEstancia

Crear lista con reserva:
De la lista _clientes, seleccionar pares (cliente c, índice i)
Filtrar para obtener solo clientes donde:
c.Dias estancia > 0
y c.Habitacion numero no es null

```
  Si con reserva está vacía entonces
     Mostrar mensaje: "No hay clientes con reserva activa para finalizar."
     Esperar pausa del usuario
     Terminar método
  Fin Si

  Mostrar encabezado de tabla:
     "=== Cliente con estancia activa ==="
     "Id | Nombre  | Hab | Días | Teléfono"

  Para cada elemento x en conReserva hacer
     Mostrar línea con:
        Id = x.i + 1
        Nombre truncado a 29 caracteres
        Número de habitación
        Días de estancia
        Teléfono truncado a 10 caracteres
  Fin Para

  Mostrar línea en blanco

  Mostrar mensaje: "Elige el Id del cliente (0=cancelar):"
  Leer entrada del usuario y convertir a entero idxShown

  Si la conversión falla o idxShown < 0 entonces
     Mostrar mensaje: "Id inválido."
     Esperar pausa del usuario
     Terminar método
  Fin Si

  Si idxShown es 0 entonces
     Terminar método (cancelar operación)
  Fin Si

  Calcular índice global: idxGlobal = idxShown - 1
  Llamar método Finalizar estancia de cliente pasando idxGlobal

Fin función
```

```
Función FinalizarEstanciaDeCliente con parámetro idx (entero)

    Obtener cliente c de la lista _clientes en la posición idx

    Si c.DiasEstancia es 0 o c.HabitacionNumero es null entonces
        Mostrar mensaje: "Ese cliente no tiene una estancia activa."
        Esperar pausa del usuario
        Terminar método
    Fin Si

    Pedir al usuario que ingrese un comprobante único:
        comp = LeerComprobanteUnico("Comprobante de pago de la habitación: ")

    Calcular total a pagar:
        total = suma de (para cada movimiento en c.Estancias) (movimiento.Dias *
movimiento.PrecioNoche)

    Construir texto de factura usando:
        cliente c,
        comprobante comp,
        esCancelacion = falso,
        días usados = c.DiasEstancia,
        días no usados = 0,
        monto usados = total,
        reembolso = 0

    Mostrar factura en consola

    Guardar factura en archivo:
        ts = fecha y hora actual
        path = DataStore.AppendFactura(factura, nombre del cliente, ts,
"Finalizacion")

    Mostrar ruta del archivo guardado

    Liberar la habitación ocupada:
        ConfigHabitaciones.Liberar(número de habitación de c)

    Reiniciar datos del cliente:
        c.DiasEstancia = 0
        c.HabitacionNumero = null
        Limpiar lista c.Estancias
        c.UltimoComprobante = comp
```

```
    Guardar datos actualizados de habitaciones y clientes:
        DataStore.SaveHabitaciones(lista de habitaciones)
        DataStore.SaveClientes(lista de clientes)

    Mostrar mensaje "Estancia finalizada y facturada."
    Esperar pausa del usuario

Fin función


Método leer comprobante único con parámetro prompt (texto) devuelve string

    Mientras verdadero
        comp = LeerNoVacio(prompt)  // Pedir al usuario que ingrese texto no vacío

        Si ComprobanteExistente(comp) es verdadero entonces
            Mostrar mensaje: ">> Comprobante ya registrado. Asigna otro
comprobante válido."
            Continuar ciclo (pedir otro comprobante)
        Fin Si

        Retornar comp
    Fin Mientras

Fin función


Método comprobante existente con parámetro comprobante (texto)

        Obtener ruta escritorio (desk)
        Construir ruta carpeta facturas dentro de "Reservas Guatemala" en escritorio
(facturasDir)

        Si la carpeta facturasDir no existe entonces
            Retornar falso
        Fin Si

        Para cada archivo con extensión ".txt" en facturasDir
            Leer contenido del archivo (text)
            Si text contiene la palabra "Comprobante" y contiene el texto comprobante
entonces
                Retornar verdadero
            Fin Si
        Fin Para
```

```
      Retornar falso
   Capturar cualquier error
      Retornar falso

Fin función



Función  construir factura texto con parámetros:
   Cliente c,
   string  comprobante,
   bool esCancelacion,
   int diasUsados,
   int diasNoUsados,
   decimal montoUsados,
   decimal reembolso
Devuelve string

   Crear un objeto StringBuilder sb

   Agregar línea vacía a sb
   Agregar líneas de encabezado con título "FACTURA / RECIBO DE ESTANCIA"
   Agregar línea con "Hotel / Razón social  : Reservas Guatemala SA"
   Agregar línea con "Folio" y valor generado por GenerarFolio()
   Agregar línea con fecha y hora actual
   Agregar línea con "Comprobante" y el parámetro comprobante
   Agregar línea separadora

   Agregar línea con "Cliente" y c.NombreCompleto
   Agregar línea con "NIT" mostrando c.Nit si no vacío, si no c.DPI
   Agregar línea con "Teléfono" y c.Telefono
   Agregar línea con "Habitación" mostrando número o "-" si es nulo
   Agregar línea separadora

   Si c.Estancias está vacío entonces
      Agregar línea "No hay movimientos registrados."
   Sino
      Agregar línea "DETALLE:"
      Inicializar contador i = 1

      Para cada movimiento m en c.Estancias
         Calcular subtotal sub = m.Dias * m.PrecioNoche
         Agregar línea con formato:
            "{i}. {m.Concepto}  {m.Dias} día(s) × {m.PrecioNoche} = {sub}"
```

```
            Incrementar i
        Fin Para
    Fin Si

    Calcular total = suma de (Dias * PrecioNoche) en c.Estancias
    Agregar línea separadora
    Agregar línea con "TOTAL ESTANCIA" y el total formateado como moneda

    Si es cancelación es verdadero entonces
        Agregar línea vacía
        Agregar línea ">>> CANCELACIÓN"
        Agregar línea con "Días utilizados" y diasUsados
        Agregar línea con "Días sin usar" y diasNoUsados
        Agregar línea con "Monto días usados" y montoUsados formateado como
moneda
        Agregar línea con "Reembolso (75% sobre reservacion no utilizada)" y
reembolso formateado como moneda
        Agregar línea separadora
        Agregar línea con "TOTAL A DEVOLVER" y reembolso formateado como
moneda
    Fin Si

    Agregar línea final separadora
    Agregar línea vacía

    Retornar sb convertido a texto

Fin función



Función Generar folio que devuelve string

    Crear un objeto Random rnd

    Obtener la fecha y hora actual en formato "yyyyMMdd-HHmmss"

    Generar un número aleatorio entre 100 y 999

    Construir y retornar un string con formato:
        "{fechaHora}-{númeroAleatorio}"

Fin Método
```

```
Función Calcular montos usadosNoUsados con parámetros:
   List<MovimientoEstancia> movs,
   int diasUsados,
   out decimal montoUsados,
   out decimal montoNoUsados

   Inicializar restantesUsados = diasUsados
   Inicializar montoUsados = 0
   Inicializar total = 0

   Para cada movimiento m en movs
      Sumar al total: m.Dias * m.PrecioNoche

      Si restantesUsados <= 0 entonces
         Continuar al siguiente movimiento
      Fin Si

      calcular consumir = mínimo entre restantesUsados y m.Dias
      Sumar a montoUsados: consumir * m.PrecioNoche
      Restar consumir de restantesUsados
   Fin Para

   Calcular montoNoUsados = total - montoUsados

Fin función



Función  Leer No Vacío

   Mientras verdadero
      Mostrar en consola el mensaje prompt
      Leer la entrada del usuario y eliminar espacios en blanco al inicio y al final,
asignar a s
      Si s no está vacío ni es solo espacios en blanco
         Retornar s
      Fin Si
      Mostrar en consola "No puede quedar vacío."
   Fin Mientras

Fin función
```

Función Leer Opcional

    Mostrar en consola el mensaje prompt
    Leer la entrada del usuario (puede ser vacía)
    Eliminar espacios en blanco al inicio y al final
    Retornar la cadena resultante

Fin función

Función Confirmar con parámetro string

    Mientras verdadero
        Mostrar en consola el mensaje prompt
        Leer la entrada del usuario, eliminar espacios y convertir a mayúsculas, asignar a s
        Si s es "S" o "SI" o "SÍ"
            Retornar true
        Si s es "N" o "NO"
            Retornar false
        Mostrar en consola "Responde S/N."
    Fin Mientras

Fin función

Función Leer Entero Positivo con parámetro string

    Mientras verdadero
        Mostrar en consola el mensaje prompt
        Leer entrada del usuario
        Intentar convertir la entrada a entero y asignar a v
        Si la conversión fue exitosa y v > 0
            Retornar v
        Fin Si
        Mostrar en consola "Ingresa un entero positivo."
    Fin Mientras

Fin función

```
Función  Leer Entero No Negativo

   Mientras verdadero
      Mostrar en consola el mensaje prompt
      Leer entrada del usuario
      Intentar convertir la entrada a entero y asignar a v

      Si la conversión fue exitosa y v >= 0
         Retornar v
      Fin Si

      Mostrar en consola "Ingresa un entero ≥ 0."
   Fin Mientras

Fin Método

Función  Leer Entero En Rango
   Mientras verdadero
      Mostrar en consola el mensaje prompt
      Leer entrada del usuario
      Intentar convertir la entrada a entero y asignar a v
      Si la conversión fue exitosa y v >= min y v <= max
         Retornar v
      Fin Si
      Mostrar en consola "Ingresa un entero entre min y max."
   Fin Mientras

Fin función




Función Elegir Habitacion Libre Con Validacion(salida numeroSeleccionado: entero)
   numeroSeleccionado = -1

   Si NO hay habitaciones libres Entonces
      Mostrar "No hay habitaciones libres."
      Retornar falso
   FinSi

   Mientras Verdadero Hacer
      MostrarHabitaciones()
      Mostrar "Elige el número de habitación (0 = cancelar): "
      Leer entrada Usuario
```

```
        Si entradaUsuario no es entero Entonces
            Mostrar "Entrada inválida."
            Continuar ciclo
        FinSi

        num = convertir entradaUsuario a entero

        Si num == 0 Entonces
            Retornar falso
        FinSi

        Si NO existe la habitación num Entonces
            Mostrar "No existe esa habitación."
            Continuar ciclo
        FinSi

        Si la habitación num NO está libre Entonces
            Mostrar "Habitación ocupada. Elige otra."
            Continuar ciclo
        FinSi

        numeroSeleccionado = num
        Retornar verdadero
    FinMientras

Fin Función


Función MostrarHabitaciones()
    Si NO hay habitaciones configuradas Entonces
        Mostrar "No hay habitaciones configuradas."
        Retornar
    FinSi

    Mostrar línea vacía
    Mostrar encabezado: "No.  | Tipo      | Precio       | Estado"
    Mostrar línea separadora: "----------------------------"

    Para cada habitación h en ObtenerHabitaciones() Hacer
        Mostrar número de habitación h.Numero alineado a la derecha con 3
espacios
        Mostrar tipo h.Tipo alineado a la izquierda con 9 espacios
```

```
        Mostrar precio h.Precio con formato moneda, alineado a la derecha con 10
espacios
        Mostrar estado h.Estado
    FinPara

    Mostrar línea vacía

Fin función


Función Trunc(s: cadena, n: entero) : cadena
    Si longitud de s <= n Entonces
        Retornar s
    Sino
        Retornar subcadena de s desde índice 0 hasta n-1 concatenada con "…"
    FinSi

Fin función


Función Aplicar Remapeo Habitaciones (mapa: Diccionario<entero, entero>)
    Si mapa es nulo o mapa está vacío Entonces
        Retornar
    FinSi

    Para i desde 0 hasta cantidad_de_clientes - 1 Hacer
        cliente = cliente en índice i
        Si cliente tiene habitación asignada Y mapa contiene la habitación actual
Entonces
            Asignar al cliente la nueva habitación del mapa correspondiente
        FinSi
    FinPara

    Guardar clientes actualizados en el almacenamiento

Fin función
```

Pseudocodigo Administración

PROCEDIMIENTO Iniciar sesión y Menú

Pantalla vacía (limpia)
Escribir "=== ADMINISTRACIÓN ==="
Ingresamos "Usuario: " sin espacios”
Leer usuario
Ingresar Contraseña: " sin espacios”
Leer contraseña

Comparación

SI user <> "Admin" O pass <> "Admin123" ENTONCES
Pausa("Credenciales inválidas.")
RETORNAR
FIN_SI

Repetir

Pantalla vacía (limpia)
Escribir "=== ADMINISTRACIÓN ==="
Escribir número "1) Resumen general (clientes, reservas y habitaciones)"
Escribir número "2) Comparativa de ingresos/egresos por mes"
Escribir número "3) Facturas"
Escribir número "0) Volver"
Escribir "Elige una opción: "sin espacio”
Leer opción

Opción hacer

Opción "1":
Mostrar Resumen()

Opción "2":
Mostrar Ingresos, egresos mensuales ( )

Opción "3":
Menú Facturas()

Opción "0":
Salir_del_bucle

De otro modo:
Pausa("Opción inválida.")

fin_segun
hasta_falso
fin_procedimiento

Procedimiento Mostrar Resumen

Pantalla vacía (limpia)
Escribir "=== RESUMEN GENERAL ==="

Clientes <- Obtener clientes para guardar
totalClientes <- contar(clientes)
conReserva <- contar(clientes DONDE Dias estancia > 0 Y HabitacionNumero != NULO)
sinReserva <- totalClientes - conReserva

Escribir "[CLIENTES]"
Escribir " - Totales:", totalClientes
Escribir " - Con reserva:", conReserva
Escribir " - Sin reserva:", sinReserva
salto_linea

Escribir "[RESERVAS ACTIVAS]"

SI conReserva = 0 ENTONCES
ESCRIBIR " No hay reservas activas."
Sino

Escribir "Id | Cliente | Hab | Días | Precio/Noche | Subtotal"
indice <- 1
PARA cada cliente EN clientes DONDE DiasEstancia > 0 Y HabitacionNumero != NULO HACER
hab <- cliente.HabitacionNumero
precio <- Precio por noche (habitación)
subtotal <- precio * cliente.DiasEstancia
ESCRIBIR indice, cliente.NombreCompleto, hab, cliente.Dias-Estancia, precio, subtotal

indice <- indice + 1

FIN_PARA

FIN_SI

SALTO_LINEA

```
    Habitaciones <- Obtener Habitaciones()
    ESCRIBIR "[HABITACIONES]"
    SI habitaciones VACIO ENTONCES
    ESCRIBIR "No hay habitaciones configuradas."

    Sino
    totalHab <- CONTAR(habitaciones)
    libres <- CONTAR(habitaciones DONDE Estado = "Libre")
    ocupadas <- CONTAR(habitaciones DONDE Estado = "Ocupada")
    Escribir "Totales:", totalHab
     Escribir "Libres:", libres
     Escribir "Ocupadas:", ocupadas
     Escribir "No | Tipo | Precio | Estado"

     PARA cada h EN habitaciones HACER
        Escribir h.Numero, h.Tipo, h.Precio, h.Estado
     FIN_PARA
  FIN_SI



Escribir  "[CLIENTES - LISTA GENERAL]"

    SI totalClientes = 0 ENTONCES
    Escribir "No hay clientes registrados."
    SINO
    Escribir "Id | Nombre | DPI | Teléfono | Días | Habitación"

    PARA i DESDE 1 HASTA totalClientes HACER
    c <- clientes[i]
    hab <- SI c.Habitacion Numero = NULO ENTONCES "-" SINO
    c.HabitacionNumero
    ESCRIBIR i, c.Nombre Completo, c.DPI, c.Telefono, c.DiasEstancia, hab
    FIN_PARA
    FIN_SI
  Pausa()

FIN_PROCEDIMIENTO
```

```
PROCEDIMIENTO MostrarIngresosEgresosMensuales()

    Pantalla vacía (limpia)
    Escribir "=== INGRESOS Y EGRESOS POR MES ==="
    porMes <- CalcularIngresosPorMes()


    SI porMes.ESTA_VACIO ENTONCES
    Pausa("No se encontraron facturas.")
    RETORNAR
    FIN_SI


   Ingresar    "Mes | Ingresos | Egresos | Utilidad"

PARA cada mes, ingreso EN porMes ORDENADO_POR_MES HACER
egresos <- ingreso * 0.35
utilidad <- ingreso - egresos
ESCRIBIR mes, ingreso, egresos, utilidad
FIN_PARA
FIN_PROCEDIMIENTO



  FUNCIÓN CalcularIngresosPorMes()

    dict <- Diccionario_Vacío
    PARA cada archivoFactura EN EnumerarFacturasIndividuales() HACER
    Texto <- LeerArchivo(archivoFactura)
    Fecha <- ExtraerFechaHora(texto)

    SI fecha = NULO ENTONCES CONTINUAR
    Clave <- FORMATO(fecha, "YYYY-MM")
    Ingreso <- ExtraerIngresoSegunTipo(texto)

    SI ingreso < 0 ENTONCES ingreso <- 0
    dict[clave] <- dict[clave] + ingreso
    FIN_PARA
    RETORNAR dict
    FIN_FUNCIÓN
```

```
FUNCIÓN ExtraerIngresoSegunTipo (facturaTexto) DEVUELVE Decimal

    SI facturaTexto.CONTIENE("CANCELACIÓN") ENTONCES
    SI BuscarMonto(facturaTexto, "Monto días usados", usados) ENTONCES
    RETORNAR usados
    SINO
    RETORNAR 0
    SINO

    SI BuscarMonto(facturaTexto, "TOTAL ESTANCIA", total) ENTONCES
    RETORNAR total
    SINO
    RETORNAR 0
    FIN_SI
    FIN_FUNCIÓN
```

PROCEDIMIENTO MenuFacturas()

```
  REPETIR

    Escribir "=== FACTURAS ==="
    Escribir ("1) Ver facturas (consolidado)"
    Escribir ("2) Buscar facturas individuales"
    Escribir ("0) Volver"
    Escribir  "Elige una opción:
    LEER opción

Según opción HACER

    CASO "1":
    Ver Consolidado ()

    CASO "2":
    BuscarFacturasIndividuales()

    CASO "0":
    SALIR_DEL_BUCLE
    DE_OTRO_MODO:
    Pausa("Opción inválida.")

    FIN_SEGUN
    FIN_PROCEDIMIENTO
```

PROCEDIMIENTO VerConsolidado()

```
        Ruta <- ObtenerRutaFacturas() + "/facturas.txt"
        SI NO EXISTE_ARCHIVO(ruta) ENTONCES
        Pausa("No existe el consolidado facturas.txt")
        RETORNAR
        FIN_SI


lineas <- leer_lineas(ruta)
        Para i dese 1 hasta lineas.LONGITUD PASO 40 HACER
        Pantalla vacía (limpia)
        Escribir  "Mostrando líneas", i, "a", MIN(i+40, lineas.LONGITUD)
        Mostrar  40_lineas (lineas, i)

        Si i + 40 < lineas.LONGITUD ENTONCES Pausa("Continuar...")
        fin_para
        Pausa("Fin de archivo.")
        FIN_PROCEDIMIENTO

   PROCEDIMIENTO BuscarFacturasIndividuales()
      Pantalla vacía (limpia)
      Escribir ("=== BÚSQUEDA DE FACTURAS ==="
      Escribir ("1) Por cliente"
      Escribir  ("2) Por mes (YYYY-MM)"
      Escribir  ("3) Por habitación"
      Escribir ("0) Volver"
      Leer opción

Archivos <- EnumerarFacturasIndividuales()
        SI vacio (archivos) ENTONCES
        Pausa ("No se encontraron facturas.")
        Retornar
        FIN_SI


       SEGUN opción HACER

        CASO "1":
        Nombres <- ExtraerNombresClientes(archivos)
        MOSTRAR_LISTA(nombres)
        LEER indiceCliente
        SI indiceCliente = 0 ENTONCES RETORNAR
        Elegido <- nombres[indiceCliente]
        encontrados <- FILTRAR(archivos, contiene nombre)
```

CASO "2":

meses <- ExtraerMesesDisponibles(archivos)
MOSTRAR_LISTA(meses)
LEER indiceMes
SI indiceMes = 0 ENTONCES RETORNAR
mesElegido <- meses[indiceMes]
encontrados <- FILTRAR(archivos, pertenece a mesElegido)

CASO "3":

firmas <- ExtraerFirmasHabitacion(archivos)
MOSTRAR_LISTA(firmas)
LEER índice.Firma
SI índice.Firma = 0 ENTONCES RETORNAR
Firma.Elegida <- firmas[indiceFirma]
Encontrados <- FILTRAR(archivos, coincideConFirma(firmaElegida))

CASO "0":

Retornar
DE_OTRO_MODO:
Pausa("Opción inválida.")
Retornar
FIN_SEGUN

SI VACIO(encontrados) ENTONCES
Pausa("No se encontraron coincidencias.")
RETORNAR
FIN_SI
REPETIR

Pantalla vacía (limpia)
MOSTRAR_LISTA(encontrados)
ESCRIBIR "Elige un índice (0=volver):"
LEER idx
SI idx = 0 ENTONCES RETORNAR
archivo <- encontrados[idx]

Lista<string> lista = convertir encontrados a lista

Si lista está vacía:
Mostrar mensaje: "No se encontraron coincidencias."
Terminar función

Mientras verdadero:
Limpiar consola
Mostrar título: "=== RESULTADOS ==="

Para cada archivo en lista con índice i:
Mostrar: "i+1) NombreDelArchivo"

Mostrar: "Elige un índice para ver (0 = volver): "
Leer entrada del usuario como entero "idx"

Si idx no es válido o está fuera de rango:
Mostrar: "Índice inválido"
Continuar al siguiente ciclo

Si idx es 0:
Terminar función

archivo = lista[idx - 1]
Limpiar consola
Mostrar título con nombre del archivo
Leer y mostrar contenido del archivo
Esperar que el usuario presione una tecla

GetFacturasDir()

Obtener ruta del escritorio
Concatenar con "Reservas Guatemala\facturas"
Retornar ruta completa

EnumerarArchivosFacturaIndividual()

Obtener ruta de facturas
Si no existe el directorio, terminar

Para cada archivo .txt en ese directorio:
Si el nombre es "facturas.txt", ignorar
Retornar archivo

Procedimiento ExtraerFechaHora(facturaTexto)

Leer línea por línea
Si línea empieza con "Fecha/Hora":
Extraer texto después de ":"
Intentar convertir a DateTime
Si es válido, retornar fecha
Si no se encuentra,
Retornar null

Procedimiento TryFindCurrencyAfterLabel(texto, etiqueta, out valor)

Leer línea por línea
Si línea contiene la etiqueta:
Extraer texto después de ":"
Quitar "Q", "GTQ", espacios
Intentar convertir a decimal (usando culturas diferentes)
Si es válido, retornar true con valor

Si no se encuentra,
Retornar falso

Procedimiento ObtenerMesArchivo(path)

Leer contenido del archivo
Extraer fecha/hora
Si se obtuvo, retornar formato "yyyy-MM"
Si falla,
Retornar null

Procedimiento ContieneEnArchivo(path, etiqueta, valor, ignoreCase)

Leer archivo línea por línea
Si línea contiene etiqueta y valor (con o sin sensibilidad a mayúsculas):
Retornar true
Si no se encuentra
Retornar falso

Procedimiento ExtraerNombresClientesDeFacturas(archivos)

    Inicializar lista de nombres

    Para cada archivo:
    Leer línea por línea
    Si línea empieza con "Cliente":
    Extraer nombre
    Agregar a lista

    Eliminar duplicados
    Retornar lista

Procedimiento ConstruirFirmasHabitacionDesdeFacturas(archivos)

    Inicializar conjunto de firmas únicas
    Inicializar lista de firmas

    Para cada archivo:
    Leer contenido
    Extraer número y tipo de habitación
    Extraer precios unitarios

    Si no hay precios:
    Crear firma con precio 0
    Para cada precio:
    Crear firma con ese precio

    Ordenar lista por número, tipo y precio
    Retornar lista

Procedimiento ExtraerNumeroHabitacion(texto)

    Buscar línea que empiece con "Habitación"
    Extraer número después de ":"
    Retornar como entero
    Si no es válido, retornar -1

Procedimiento ExtraerTipoHabitacion(texto)

    Buscar línea que empiece con "Tipo habitación" o "Tipo"
    Extraer texto después de ":"
    Retornar texto limpio

Si no se encuentra
retornar null

Procedimiento ExtraerPreciosUnitarios(texto)

Buscar líneas que contengan "día(s)" y "×"
Después del "×", buscar texto antes del "="
Quitar símbolos "Q", "GTQ"
Extraer número
Intentar convertir a decimal
Agregar a lista si es válido

Eliminar duplicados
Retornar lista

Procedimiento CoincideFacturaConFirma(path, firma)

Leer archivo
Extraer número, tipo y precios
Comparar con los valores de la firma:
- Si número o tipo no coinciden, retornar false
- Si no hay precios, firma debe tener precio 0
- Si hay precios, alguno debe coincidir

Retornar si coincide

Procedimiento Trunc(string s, int n)

Si longitud de s ≤ n:
  Retornar s
Sino:
  Retornar primeros n-1 caracteres + "…"

Procedimiento LeerPassword()

Inicializar cadena vacía
Mientras no se presione Enter:
  Leer tecla sin mostrarla
  Si es Backspace, borrar último carácter
  Si es carácter visible, agregar a cadena

Retornar contraseña ingresada

Diagrama de Flujo

Sistema de Reservas de Hotel

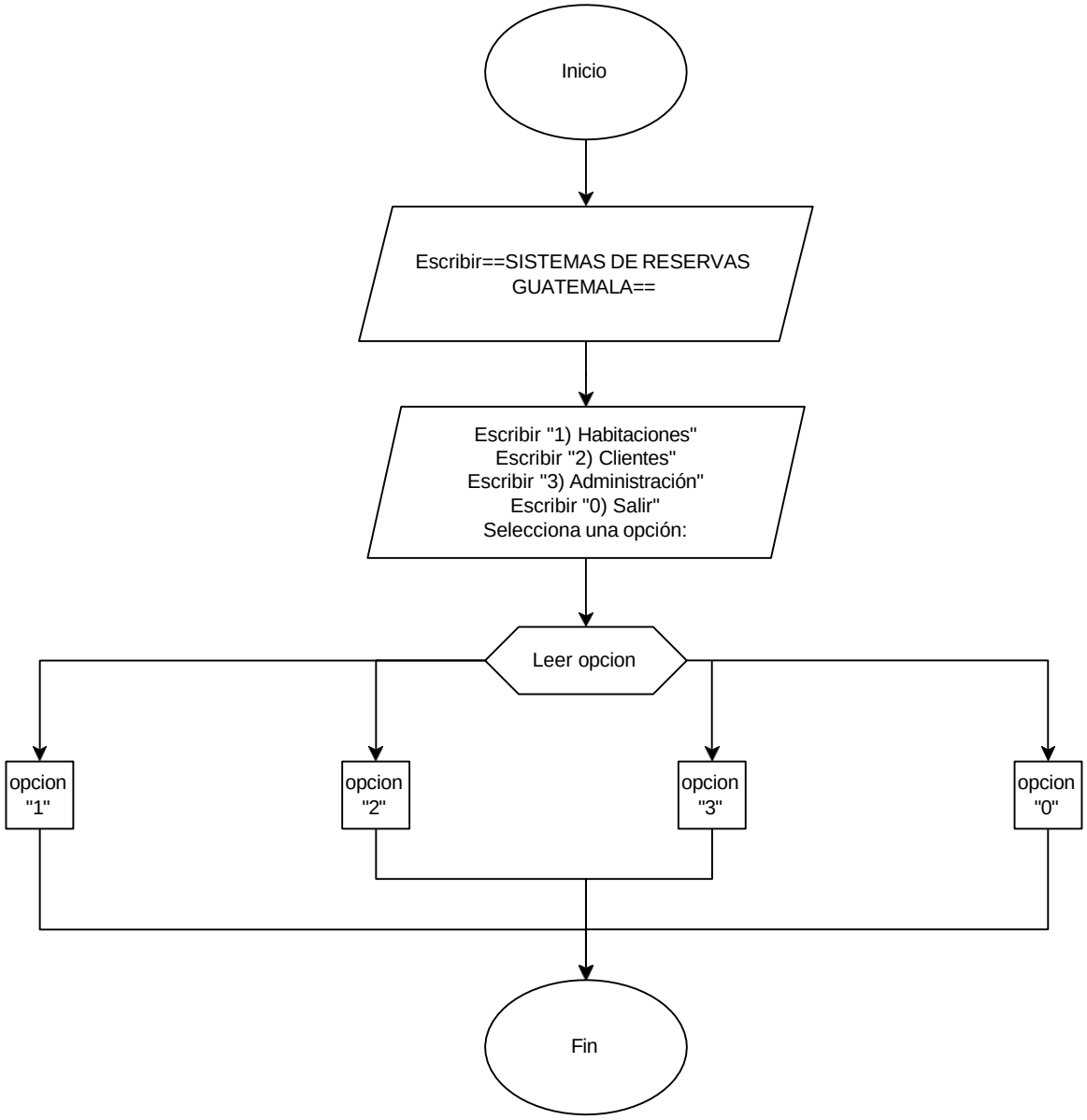


Diagrama de Flujo
Sistema de Reservas de Hotel

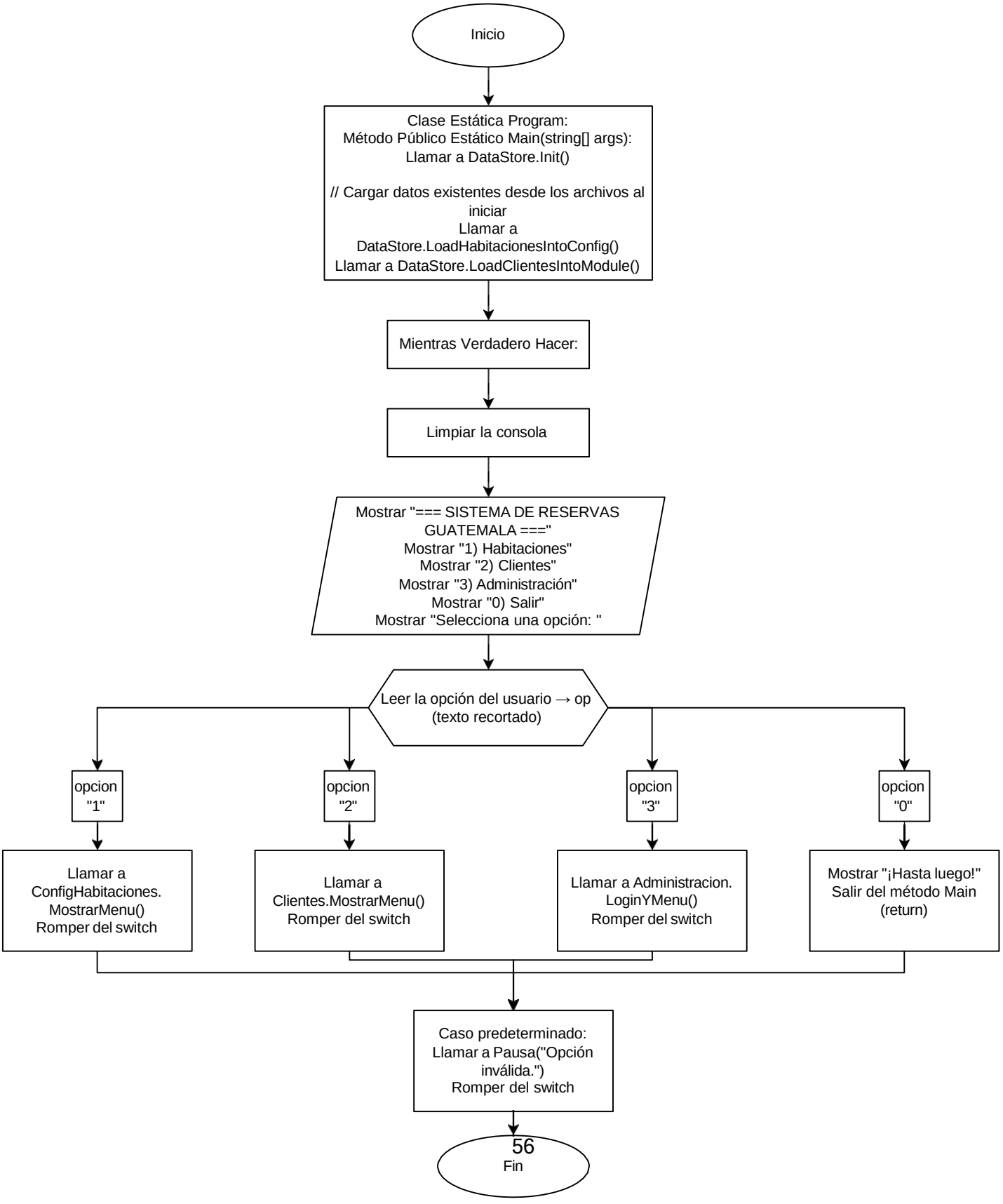
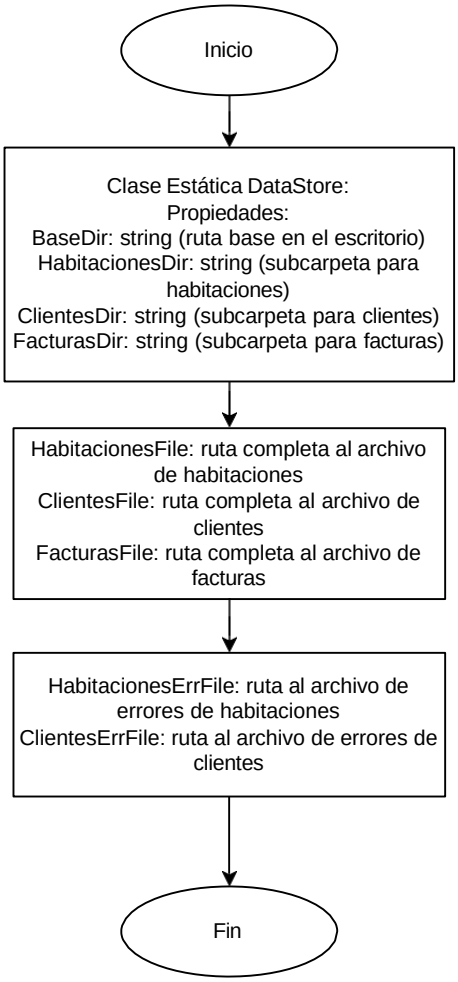


Diagrama de Flujo
Sistema de Reservas de Hotel



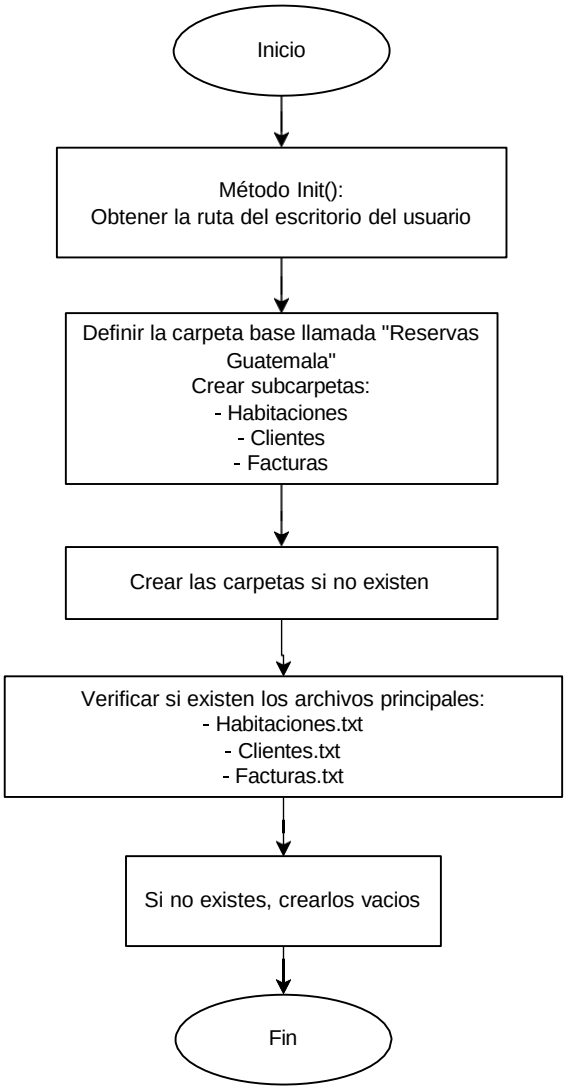
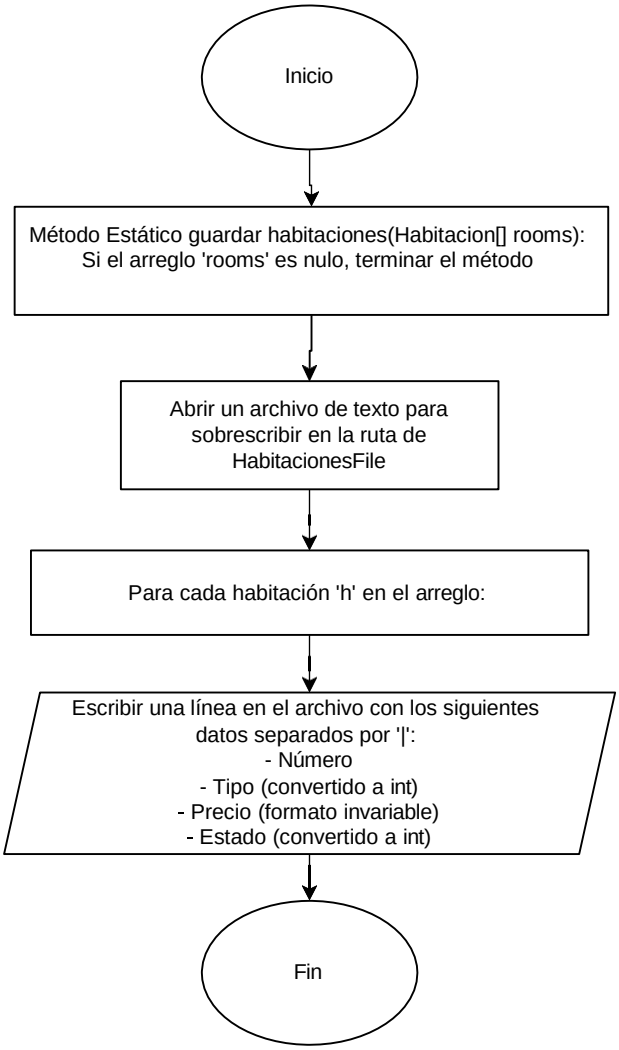


Diagrama de Flujo
Sistema de Reservas de Hotel



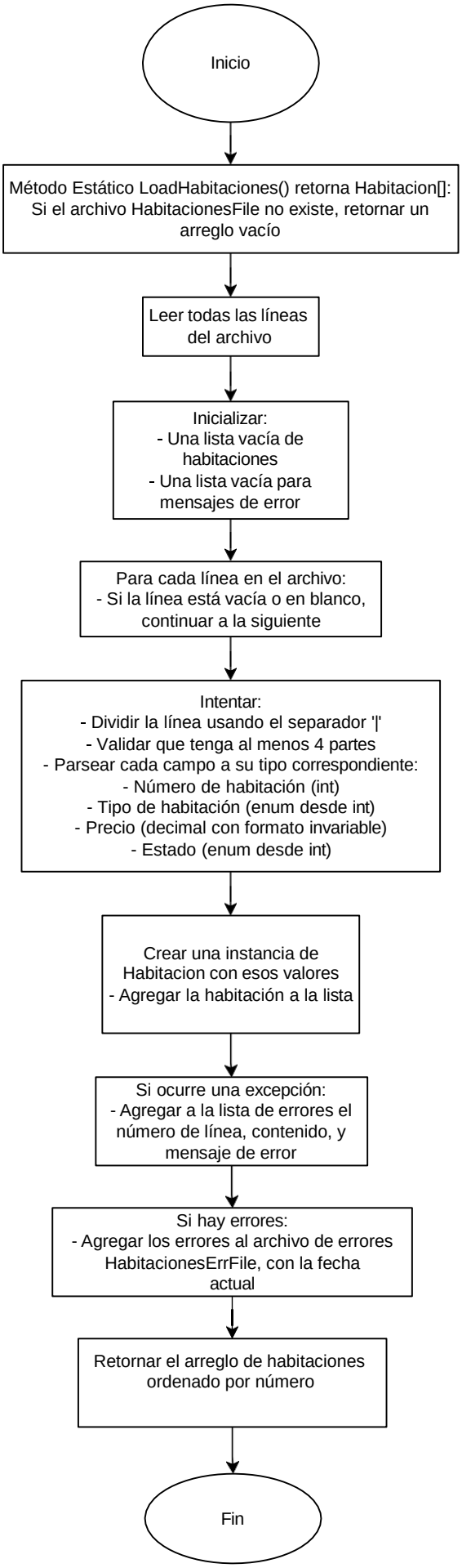


Diagrama de Flujo
Sistema de Reservas de Hotel

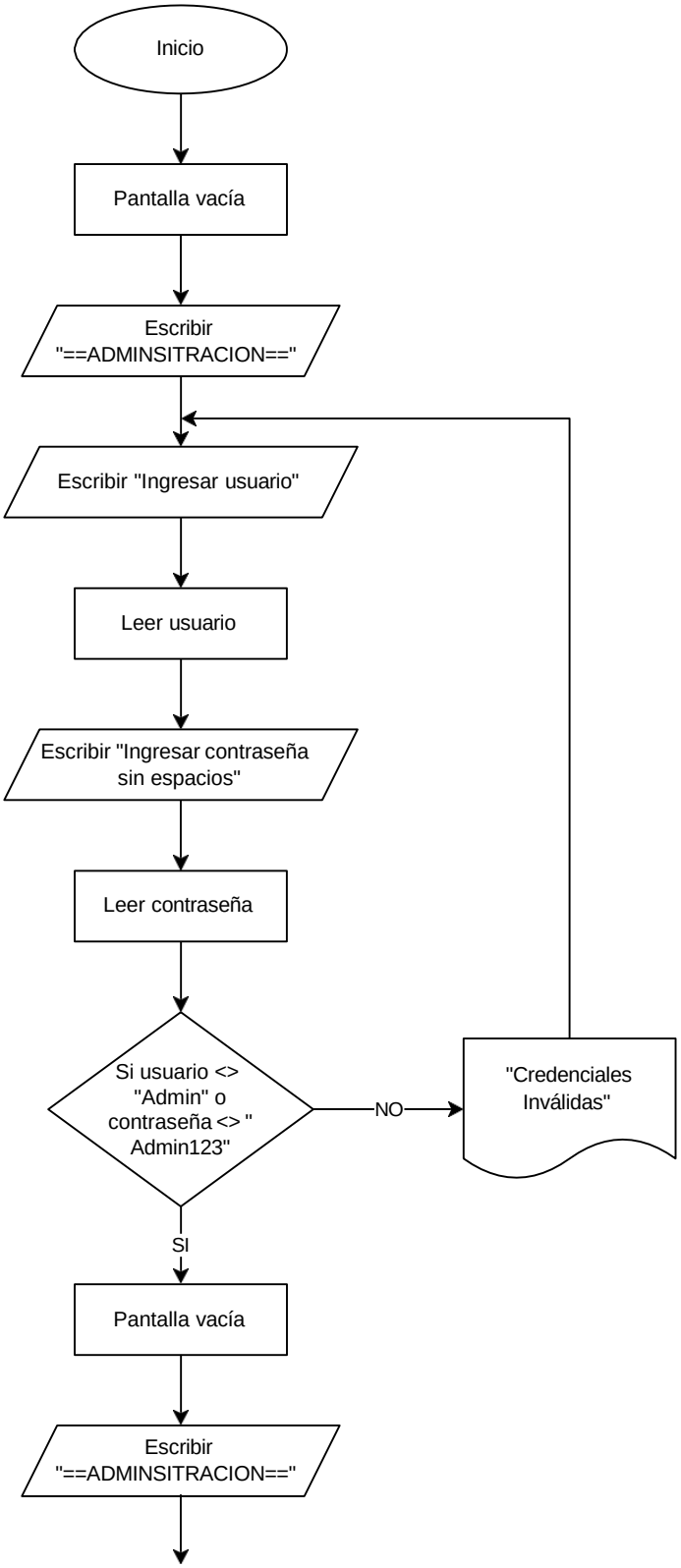


Diagrama de Flujo
Sistema de Reservas de Hotel

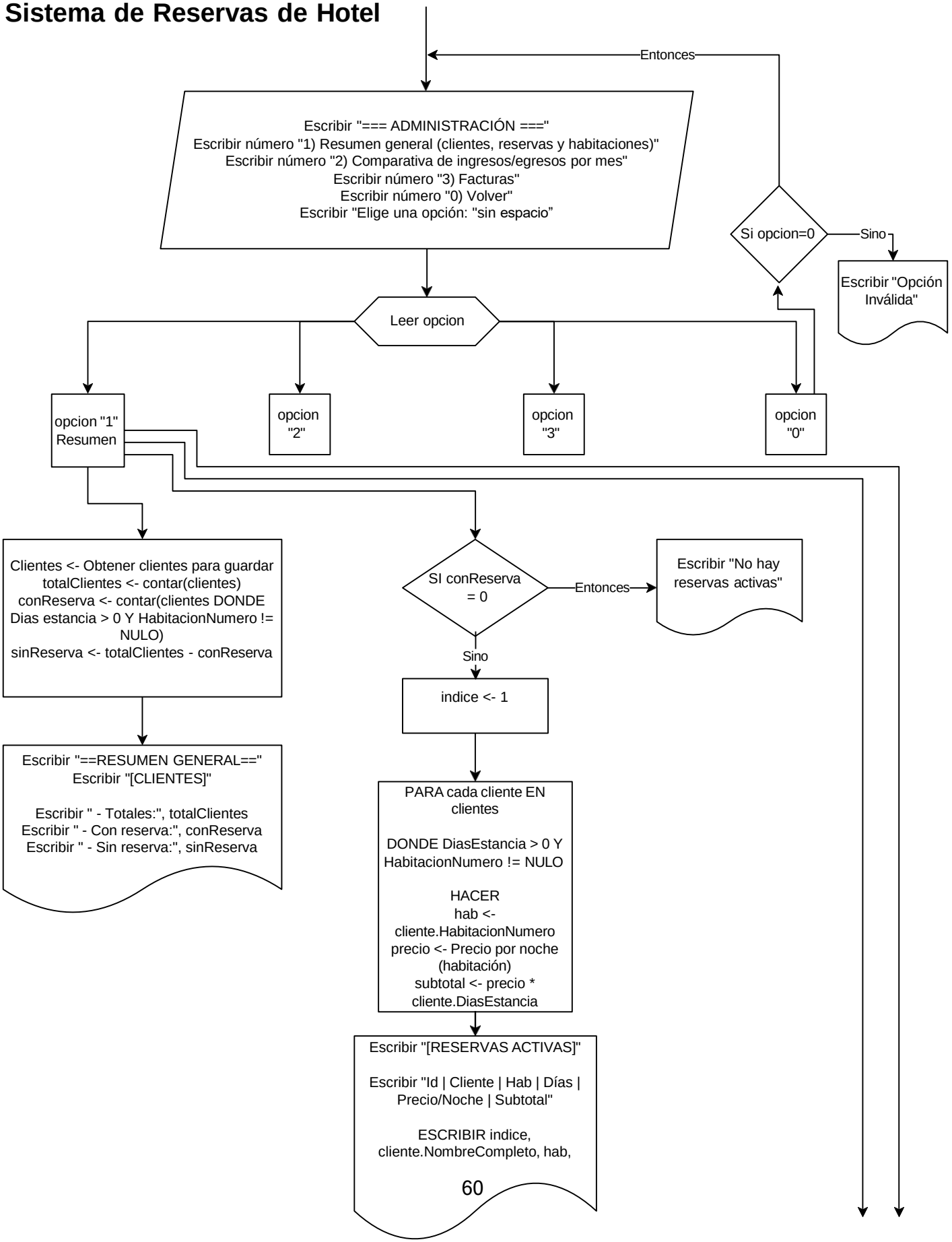


Diagrama de Flujo
Sistema de Reservas de Hotel

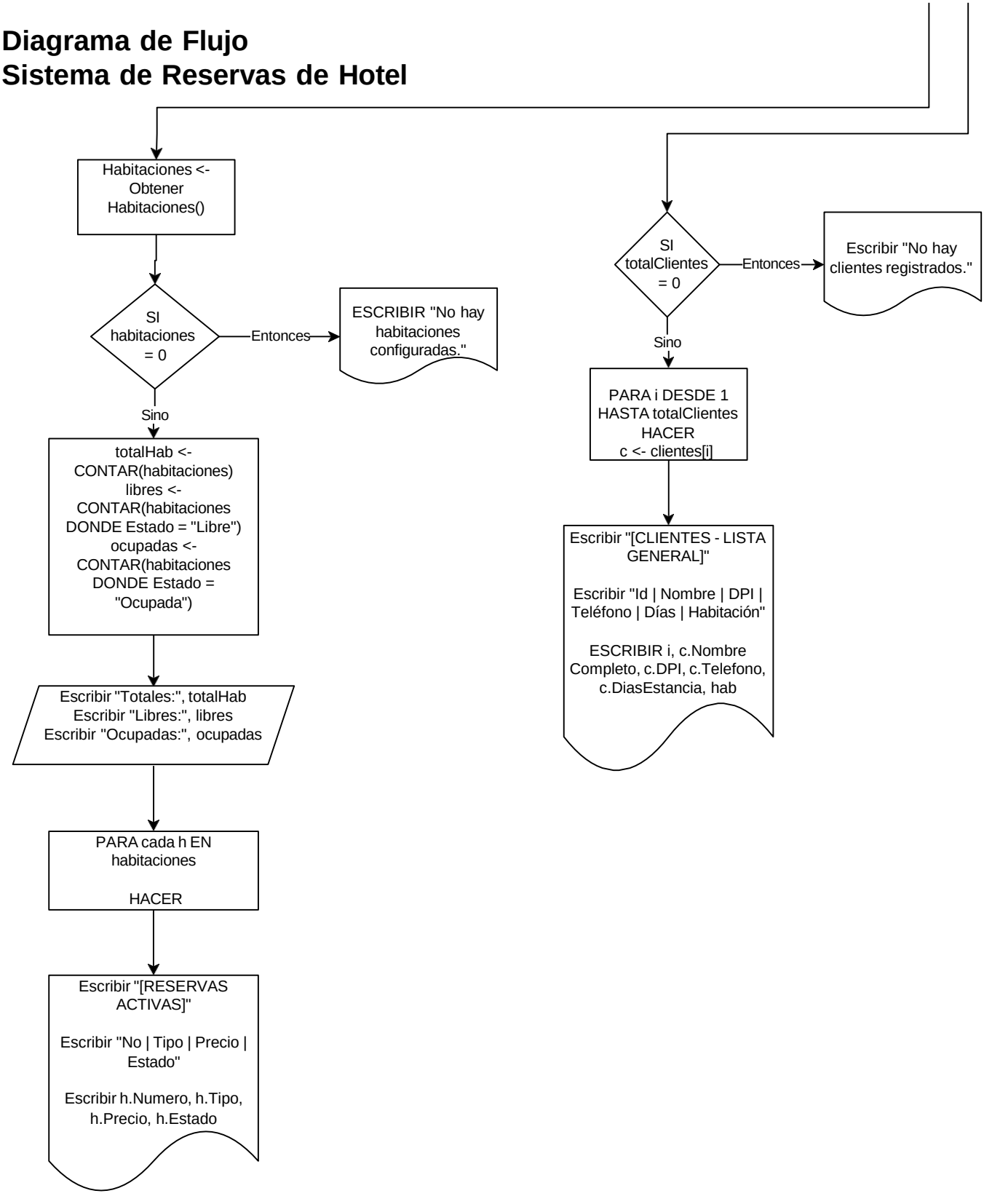


Diagrama de Flujo
Sistema de Reservas de Hotel

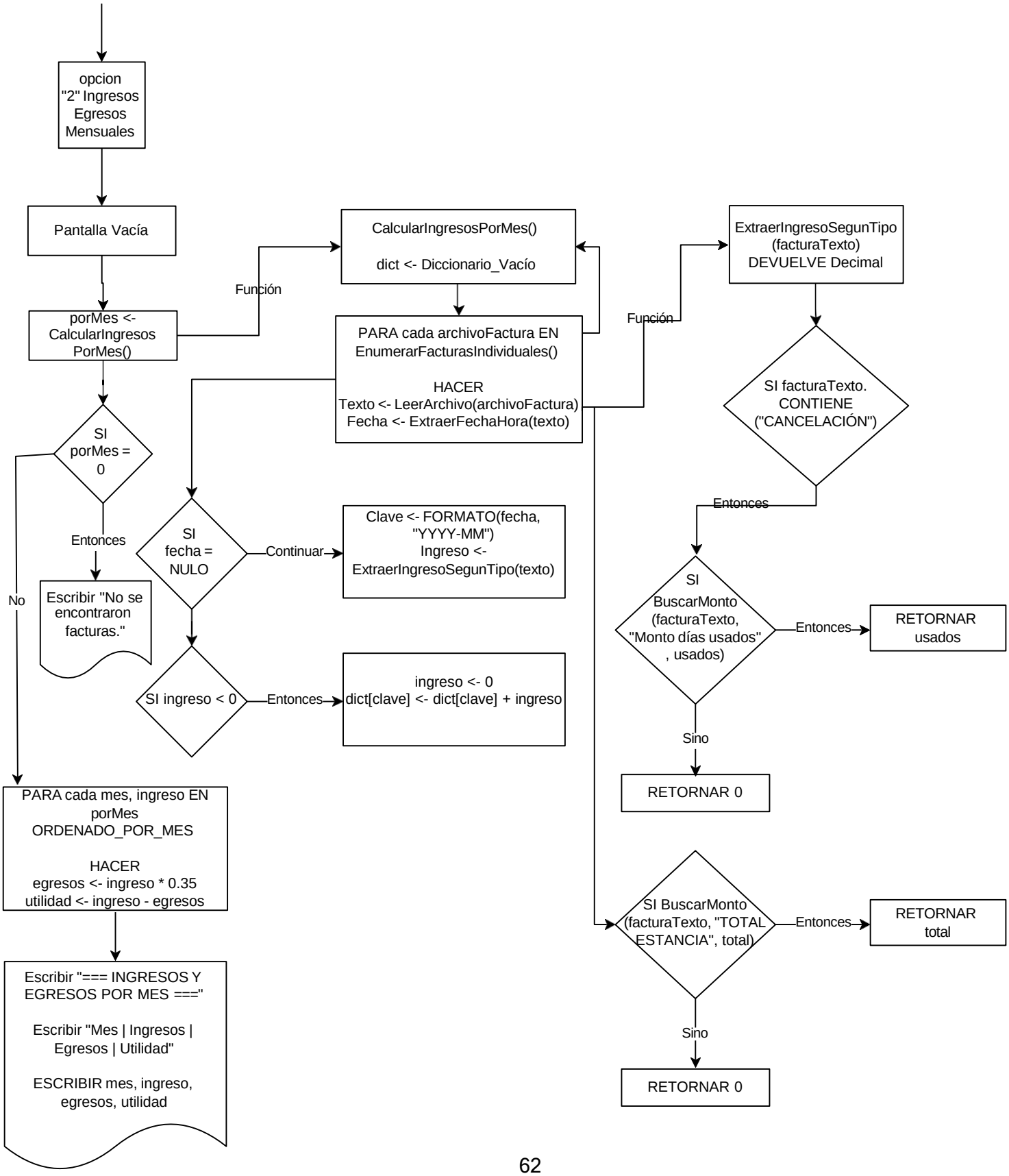


Diagrama de Flujo
Sistema de Reservas de Hotel

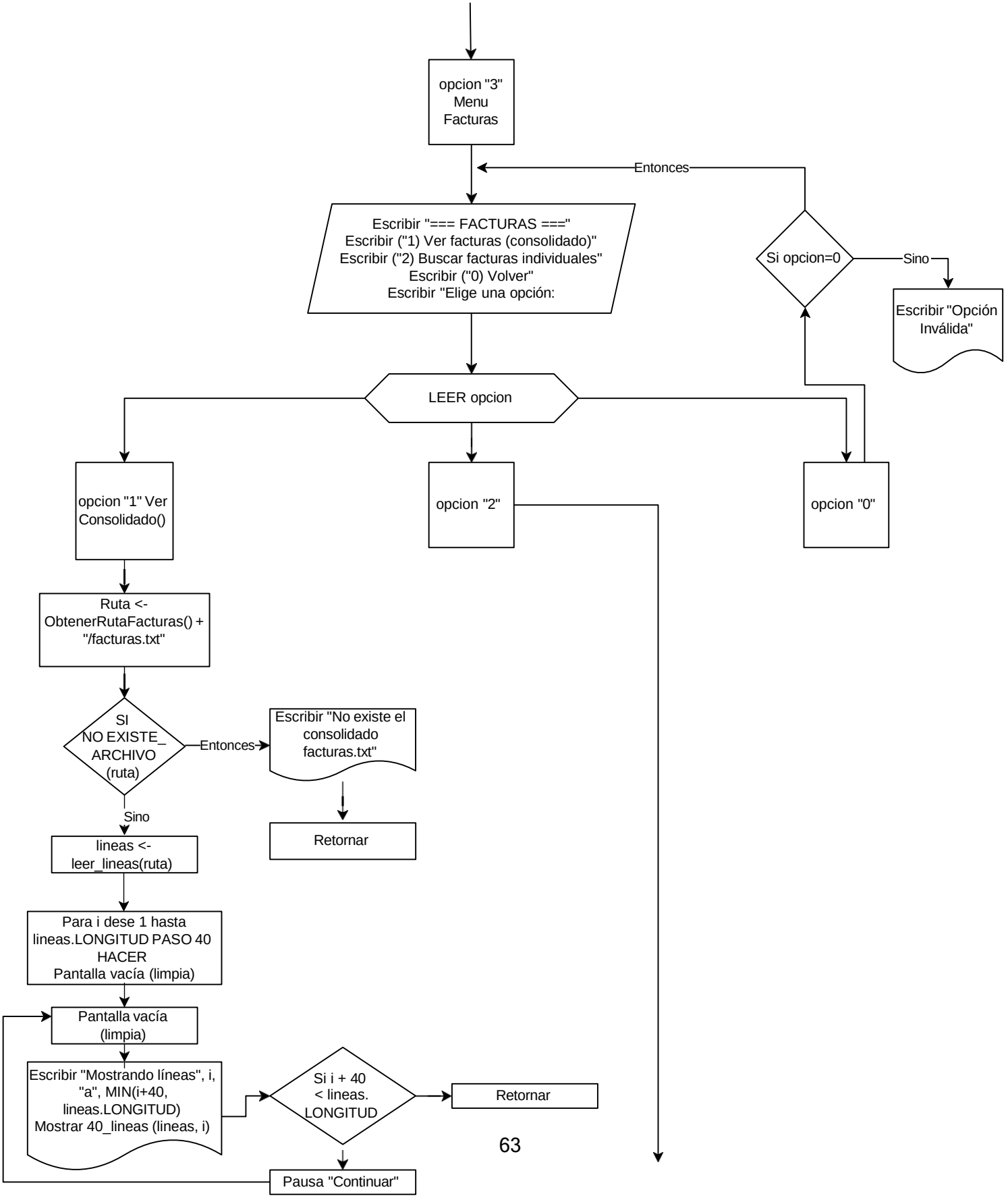
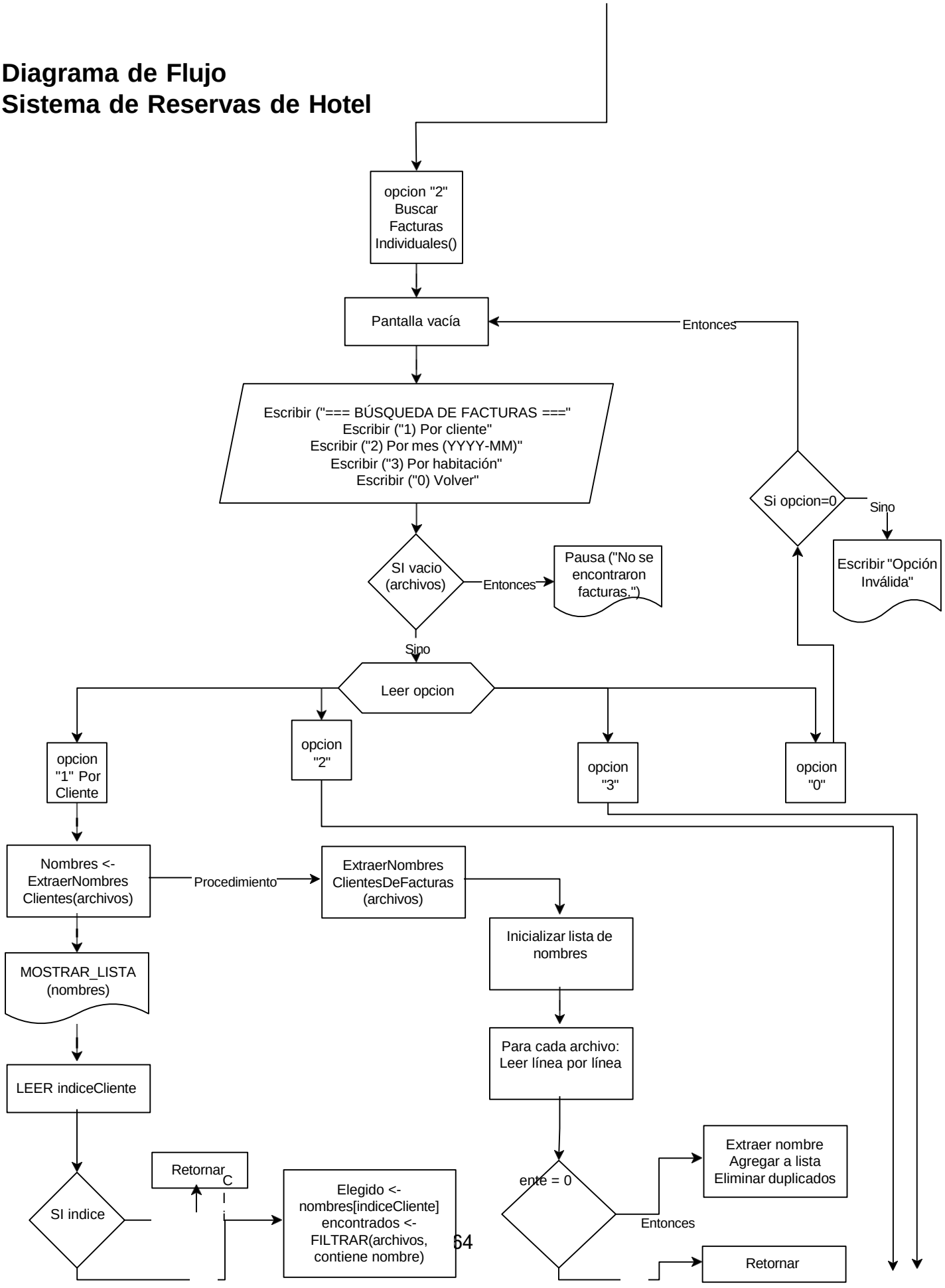


Diagrama de Flujo
Sistema de Reservas de Hotel



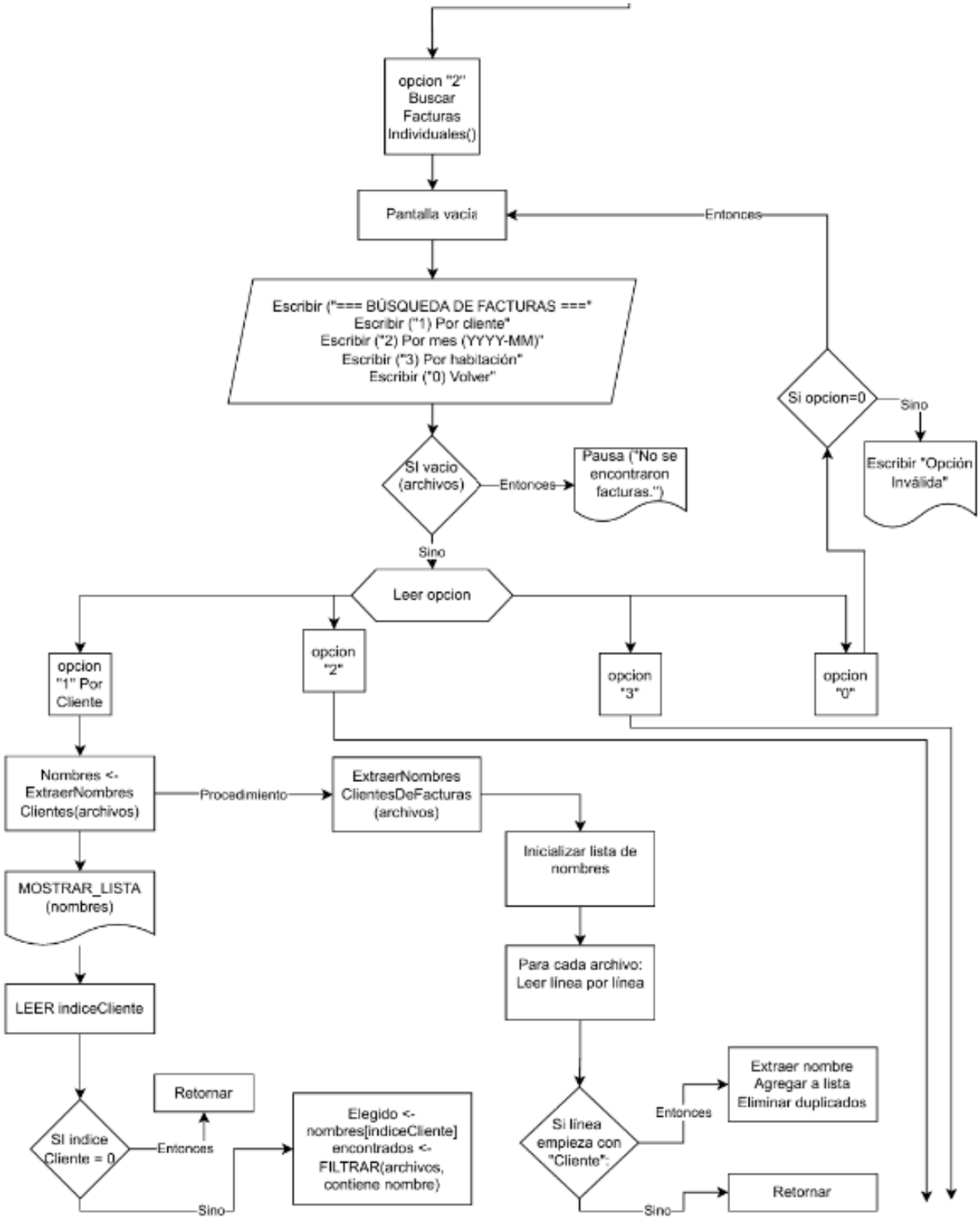

opcion "2" Buscar Facturas Individuales()
Pantalla vacia
Entonces
Escribir ("=== BÚSQUEDA DE FACTURAS ===" Escribir ("1) Por cliente" Escribir ("2) Por mes (YYYY-MM)" Escribir ("3) Por habitación" Escribir ("0) Volver"
Si opcion=0
Sino
Escribir "Opción Inválida"
SI vacio (archivos)
Entonces
Pausa ("No se encontraron facturas.")
Sino
Leer opcion
opcion "1" Por Cliente
opcion "2"
opcion "3"
opcion "0"
Nombres <- ExtraerNombres Clientes(archivos)
Procedimiento
ExtraerNombres ClientesDeFacturas (archivos)
Inicializar lista de nombres
MOSTRAR_LISTA (nombres)
Para cada archivo: Leer línea por línea
LEER indiceCliente
Retornar
SI indice Cliente = 0
Entonces
Elegido <- nombres[indiceCliente] encontrados <- FILTRAR(archivos, contiene nombre)
Sino
Si línea empieza con "Cliente":
Entonces
Extraer nombre Agregar a lista Eliminar duplicados
Sino
Retornar

Diagrama de Flujo
Sistema de Reservas de Hotel

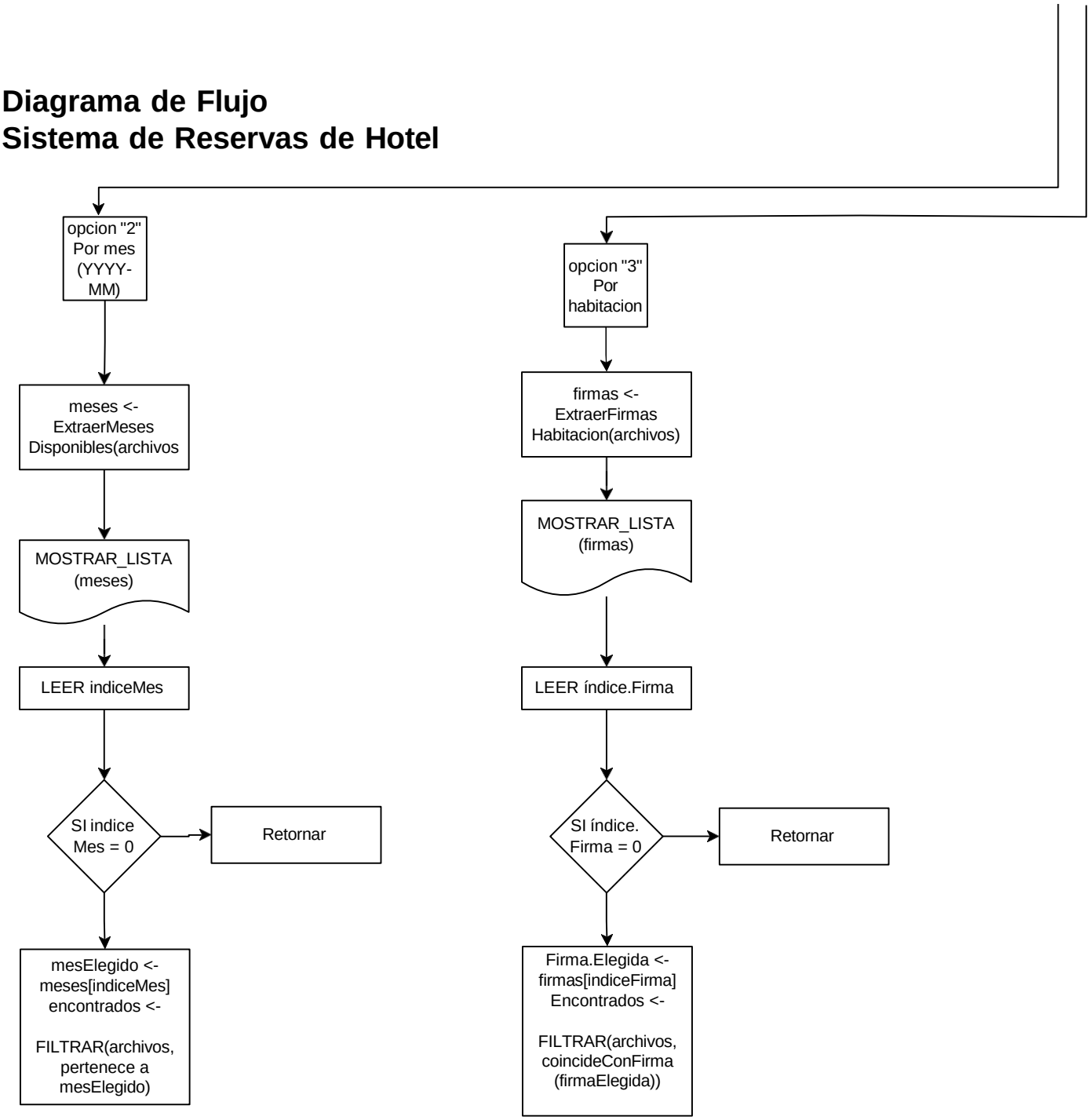


Diagrama de Flujo
Sistema de Reservas de Hotel

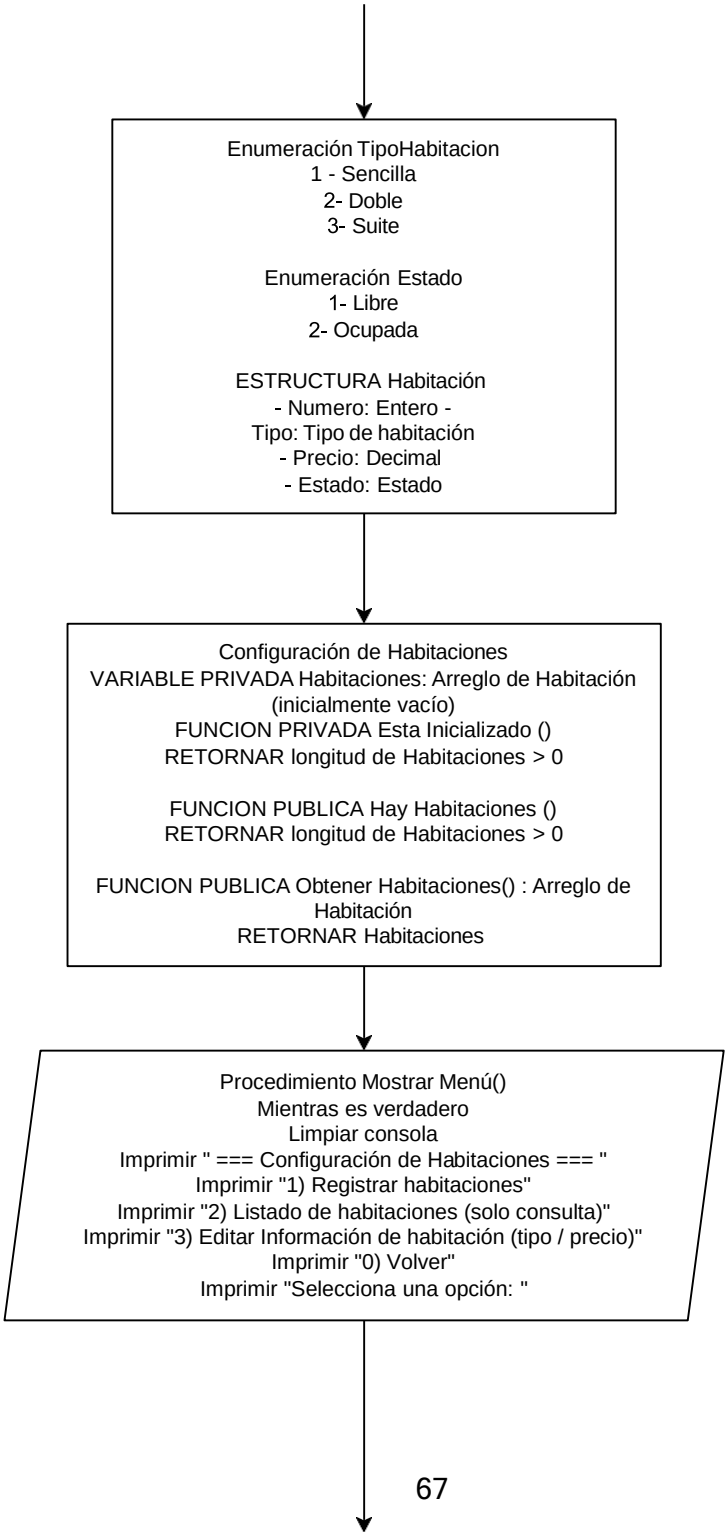
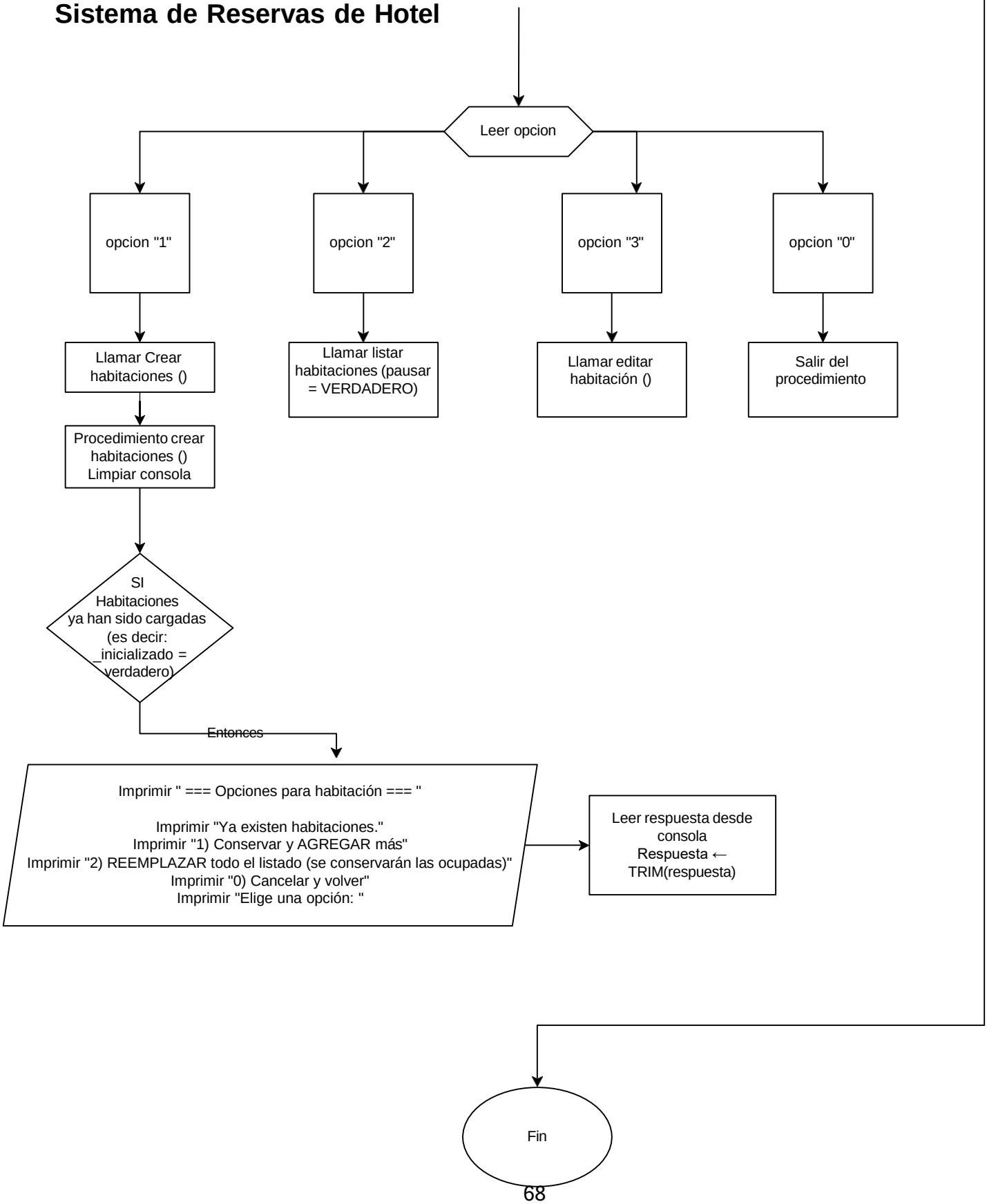


Diagrama de Flujo
Sistema de Reservas de Hotel



Pruebas

Inicio del programa

```
=== SISTEMA DE RESERVAS GUATEMALA ===
1) Habitaciones
2) Clientes
3) Administración
0) Salir
Selecciona una opción:
```

Opción 1

```
 === Configuracion de Habitaciones ===
1) Registrar habitaciones
2) Listado de habitaciones (solo consulta)
3) Editar Informacion de habitación (tipo / precio)
0) Volver
Selecciona una opción:
```

Opción 2

```
 === Opciones para clientes ===
1) Registrar cliente
2) Listado de clientes
0) Volver
Selecciona una opción:
```

Opción 3

```
=== ADMINISTRACIÓN ===
Usuario:
```

Habitaciones

```
 === Configuracion de Habitaciones ===
1) Registrar habitaciones
2) Listado de habitaciones (solo consulta)
3) Editar Informacion de habitación (tipo / precio)
0) Volver
Selecciona una opción:
```

Opción 1

```
 === Opciones para habitación ===
Ya existen habitaciones.
1) Conservar y AGREGAR más
2) REEMPLAZAR todo el listado (se conservarán las ocupadas)
0) Cancelar y volver
Elige una opción:
```

Opción 2

```
Selecciona una opción: 2
No.  | Tipo      | Precionoche       | Estado
-----------------------------------------------
  1  | Doble     |    Q600.00 | Libre

Presiona una tecla para continuar...
```

Opción 3

```
=== EDITAR HABITACIÓN ===
No.  | Tipo      | Precionoche       | Estado
-----------------------------------------------
  1  | Doble     |    Q600.00 | Libre

Ingresa el número de la habitación a editar (0 = cancelar):
```

Clientes

```
 === Opciones para clientes ===
1) Registrar cliente
2) Listado de clientes
0) Volver
Selecciona una opción:
```

Opción 1

```
=== Registrar Cliente ===
Nombre completo: JUAN PEREZ
DPI: 1234567890
NIT (Enter si no tiene): 987654
Teléfono: 10293847
Días de estancia (0 si solo registro): 2


No.  | Tipo      | Precio       | Estado
----------------------------------------
  1  | Doble     |    Q600.00 | Libre

Elige el número de habitación (0 = cancelar): 1
Cliente registrado correctamente.
Presiona una tecla para continuar...
```

Opción 2

```
=== LISTADO DE CLIENTES ===
Id | Nombre                        | DPI           | Teléfono      | Días | Habitación
-------------------------------------------------------------------------------------
  1 | JUAN PEREZ                      | 1234567890     | 10293847      |    2 |         1

Acciones:
1) Asignar hospedaje
2) Cancelar estancia
3) Finalizar estancia
4) Editar información del cliente
5) Eliminar cliente
0) Volver
Elige una acción:
```

Administración

```
=== ADMINISTRACIÓN ===
Usuario: Admin
Contraseña:
```

```
=== ADMINISTRACIÓN ===
1) Resumen general (clientes, reservas y habitaciones)
2) Comparativa de ingresos/egresos por mes
3) Facturas
0) Volver
Elige una opción:
```

Opción 1

```
=== RESUMEN GENERAL ===
[CLIENTES]
  - Totales        : 1
  - Con reserva    : 1
  - Sin reserva    : 0

[RESERVAS ACTIVAS]
Id  | Cliente                        | Hab | Días | Precio/Noche | Subtotal
----+--------------------------------+-----+------+--------------+------------
  1 | JUAN PEREZ                     |   1 |    2 |      Q600.00 |   Q1,200.00

[HABITACIONES]
  - Totales    : 1
  - Libres     : 0
  - Ocupadas   : 1

No.  | Tipo      | Precio       | Estado
-----+-----------+--------------+---------
  1  | Doble     |    Q600.00 | Ocupada

[CLIENTES - LISTA GENERAL]
Id  | Nombre                         | DPI            | Teléfono       | Días | Habitación
----+--------------------------------+----------------+----------------+------+-----------
  1 | JUAN PEREZ                     | 1234567890     | 10293847       |    2 |          1
Presiona una tecla para continuar...
```

Opción 2

```
=== INGRESOS Y EGRESOS POR MES ===
Mes         | Ingresos      | Egresos       | Utilidad
------------------------------------------------------------
2025-10     |        Q0.00 |        Q0.00 |        Q0.00
Presiona una tecla para continuar...
```

Opción 3

```
=== FACTURAS ===
1) Ver facturas (consolidado)
2) Buscar facturas individuales
0) Volver
Elige una opción:
```

Conclusiones

El desarrollo del sistema permitió aplicar todas las etapas del ciclo de resolución de problemas: análisis, diseño, codificación, prueba y documentación. Esto consolidó la capacidad de abordar desafíos técnicos de forma estructurada y eficiente.

Se logró implementar estructuras selectivas y repetitivas, funciones, procedimientos, arreglos y registros.

La implementación de arreglos de registros para habitaciones y clientes permitió una estructura clara y accesible para almacenar y manipular la información.

El manejo de archivos para cargar y guardar datos aseguró la constancia entre sesiones, lo que permite una operación continua en sistemas administrativos.

El programa desarrollado reproduce de forma básica pero funcional las operaciones de un hotel, como reservaciones, cancelaciones, modificaciones y facturación.

Recomendaciones

Implementar un sistema de almacenamiento con amplia capacidad, que garantice la conservación segura y organizada de los datos.

Establecer mecanismos de respaldo automático para proteger la información ante posibles pérdidas o fallos, según sea necesario.

Brindar capacitación continua al personal sobre el uso, manejo y buenas prácticas del programa, asegurando una operación eficiente y segura.

Anexos

BITACORA

1. Introducción

Esta bitácora documenta el proceso de desarrollo del sistema de administración hotelera en C#, detallando los principales problemas encontrados, los cambios realizados durante su implementación y las dificultades técnicas o de diseño que se presentaron en las distintas etapas del proyecto.

2. Problemas encontrados durante el desarrollo

- Errores de lectura y escritura al manejar archivos de texto con caracteres especiales como '|' y '\', lo que causó datos corruptos al guardar clientes y habitaciones.
- Problemas iniciales con la ruta de almacenamiento de archivos, ya que en algunos entornos el programa no tenía permisos para escribir directamente en el Escritorio.
- Errores en el cálculo del precio total de las estancias cuando se realizaban cambios de habitación sin finalizar la reserva anterior.
- Dificultad para mantener sincronizados los números de habitación cuando se reemplazaba el listado completo de habitaciones.
- Problemas de formato con los decimales y símbolos de moneda debido a diferencias regionales de configuración (punto o coma).
- Validaciones incompletas que permitían crear clientes sin habitación asignada o con días negativos de estancia.
- Error ocasional al generar las facturas debido a nombres de archivo con caracteres inválidos (como los espacios o símbolos en el nombre del cliente).

3. Cambios realizados durante el desarrollo

- Se creó una clase central de persistencia 'DataStore' para manejar toda la lectura y escritura en archivos de texto.

- Se implementó un método de escape ('Esc' y 'UnEsc') para garantizar la correcta lectura y escritura de campos con caracteres reservados.
- Se añadió control de errores al cargar habitaciones y clientes, de modo que los registros corruptos se omiten sin detener el programa.
- Se agregó una opción para generar facturas individuales además de un consolidado general ('Facturas.txt').
- Se incorporó un sistema de autenticación básica para el módulo de administración con usuario y contraseña predeterminados.
- Se implementó un menú principal con tres módulos (Habitaciones, Clientes y Administración) para una navegación más clara.
- Se ajustaron los cálculos de cancelación para devolver el 75% del monto no utilizado.
- Se añadió un control para detectar habitaciones ocupadas antes de editarlas o eliminarlas del sistema.

4. Dificultades técnicas y de desarrollo

- La falta de un motor de base de datos obligó a diseñar un sistema de persistencia manual basado en archivos de texto.
- Las pruebas con distintos formatos regionales (configuración de moneda y fechas) ocasionaron errores al convertir precios y fechas.
- La validación y mantenimiento de integridad referencial entre clientes y habitaciones exigió varias iteraciones y ajustes lógicos.

El desarrollo del sistema presentó una serie de desafíos técnicos y de diseño que fueron resueltos mediante la modularización del código, la implementación de controles de validación y la mejora progresiva del manejo de persistencia en archivos. El resultado final es una aplicación estable, funcional y fácilmente extensible, capaz de gestionar un pequeño hotel con módulos diferenciados para habitaciones, clientes y administración.

Manual de técnico

Sistema de Reservas de Hotel

Resumen

El presente manual técnico documenta el desarrollo del Sistema de Reservaciones de Hotel, diseñado para gestionar de forma eficiente el registro, control y asignación de habitaciones a clientes. El sistema fue implementado utilizando estructuras selectivas, repetitivas, funciones, procedimientos y manejo de archivos, con el fin de automatizar procesos esenciales como la reserva, cancelación y facturación. Además, integra algoritmos de ordenamiento y validaciones para asegurar la integridad de los datos.

Introducción

El Sistema de Reservaciones de Hotel es una aplicación informática básica desarrollada como práctica de aplicación de técnicas de programación estructurada. Su objetivo principal es permitir al usuario registrar habitaciones, clientes y operaciones de reserva de manera automatizada, garantizando la persistencia de datos mediante el uso de archivos.

Arquitectura del sistema

Lenguaje de programación: C#

Paradigma: Programación estructurada con modularidad mediante funciones y procedimientos.

Tipo de aplicación: Aplicación de consola

Estructuras empleadas:

- Arreglos y registros para almacenar habitaciones y clientes.
- Cadenas para manejo de texto.
- Estructuras selectivas (*if*, *switch*).
- Estructuras repetitivas (*for*, *while*).
- Procedimientos y funciones modulares.
- Algoritmo de ordenamiento por tipo o número de habitación.

Requisitos del sistema

Hardware mínimo:

- Procesador: Intel Core i3 o superior
- Memoria RAM: 4 GB
- Espacio en disco: 100 MB libres
- Monitor con resolución mínima 1366×768

Software:

- Sistema operativo Windows 10 o superior
- Microsoft Visual Studio (versión 2019 o posterior)
- .NET Framework 4.7.2 o superior

Estructura funcional del sistema

Registro de habitaciones

- Captura número, tipo, precio y estado.
- Guarda datos en un arreglo de registros.

Registro de clientes

- Captura nombre, documento, teléfono y días de estancia.
- Asocia cliente a una habitación.

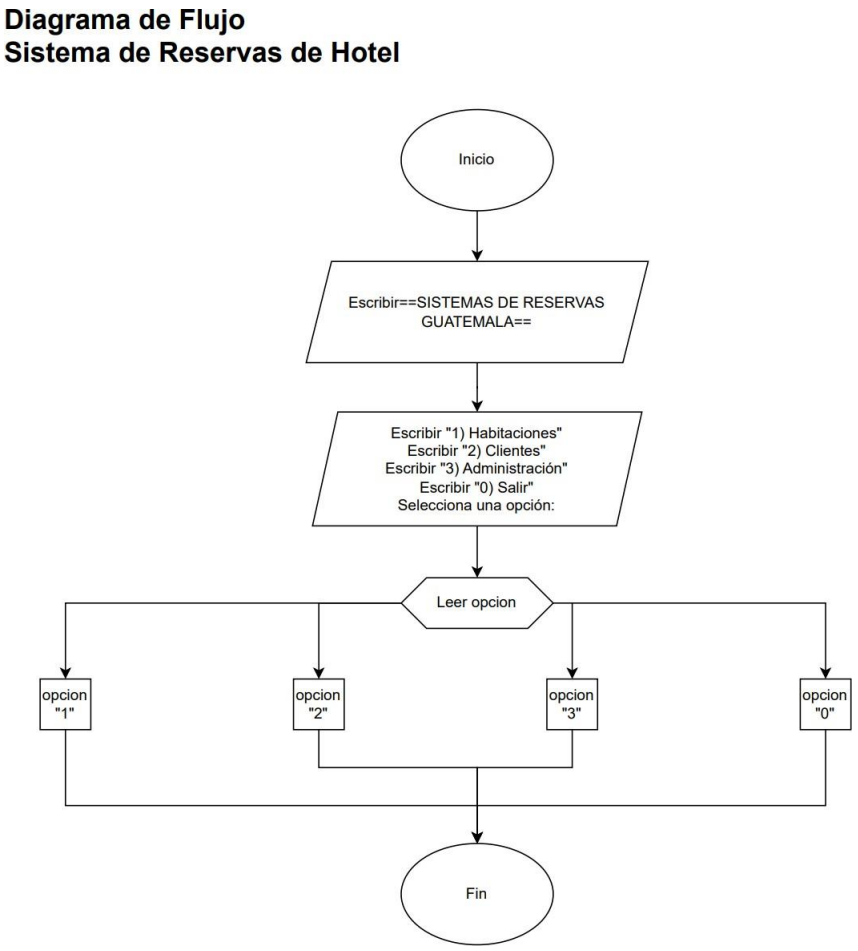
Gestión de operaciones

- Reservar habitación
- Cancelar reservación
- Modificar datos
- Consultar disponibilidad
- Generar factura

Manejo de archivos

- Carga automática de datos al iniciar.
- Guardado de cambios al cerrar el sistema.

Diagrama general



Estructura de datos

Estructura para habitación:

```
struct Habitacion {
    int numero;
    string tipo; // sencilla, doble, suite
    double precio;
```

```
    bool disponible;
}
```

Estructura para cliente:

```
struct Cliente {
    string nombre;
    string documento;
    string telefono;
    int diasEstancia;
    int numHabitacion;
}
```

Procedimientos y funciones clave

Registrar habitación → Permite ingresar nueva habitación.

Registrar Cliente → Permite ingresar cliente y asociar habitación.

Reservar → Asigna cliente a habitación disponible.

Cancelar → Libera habitación ocupada.

Consultar Disponibilidad → Muestra habitaciones libres.

Generar Factura → Calcula monto según días de estancia.

Guardar Archivo / Cargar Archivo → Persistencia de datos.

Control de errores y validaciones

Validación de tipos de habitación válidos.

Validación de disponibilidad antes de reservar.

Verificación de campos vacíos.

Control de errores en lectura/escritura de archivos.

Mantenimiento y respaldo

verificación de integridad de archivos de datos antes de ejecutar.

Realizar copias de respaldo periódicamente.

Actualizar tarifas o tipos de habitación modificando los registros.

Conclusiones

El Sistema de Reservaciones de Hotel cumple con los objetivos propuestos de aplicar estructuras de programación y manejo de archivos. La documentación técnica permite su mantenimiento, comprensión y ampliación futura.

Manual de usuario

Sistema de Reservas de Hotel

Resumen

El presente manual de usuario tiene como objetivo guiar al usuario en el uso correcto del *Sistema de Reservaciones de Hotel*, desarrollado en lenguaje C#. A través de instrucciones claras y precisas, se describen los pasos necesarios para realizar las principales operaciones del sistema: registrar habitaciones, registrar clientes, efectuar reservaciones, cancelar operaciones y generar facturas. El manual incluye también una descripción general de las funciones y mensajes de error comunes, con el propósito de facilitar la comprensión y el manejo adecuado de la aplicación.

Introducción

El Sistema de Reservaciones de Hotel es una aplicación diseñada para optimizar el proceso de registro y control de habitaciones en establecimientos hoteleros. Mediante un entorno sencillo e intuitivo, el sistema permite registrar información de clientes y habitaciones, realizar reservaciones y consultar disponibilidad, todo de forma automatizada.

Este manual tiene como propósito ofrecer una guía práctica para los usuarios que interactúan con el sistema, detallando los procedimientos paso a paso y brindando información para resolver posibles inconvenientes durante su utilización.

Requisitos del sistema

Hardware mínimo:

- Procesador: Intel Core i3 o superior
- Memoria RAM: 4 GB
- Espacio en disco: 100 MB libres
- Monitor con resolución mínima 1366×768

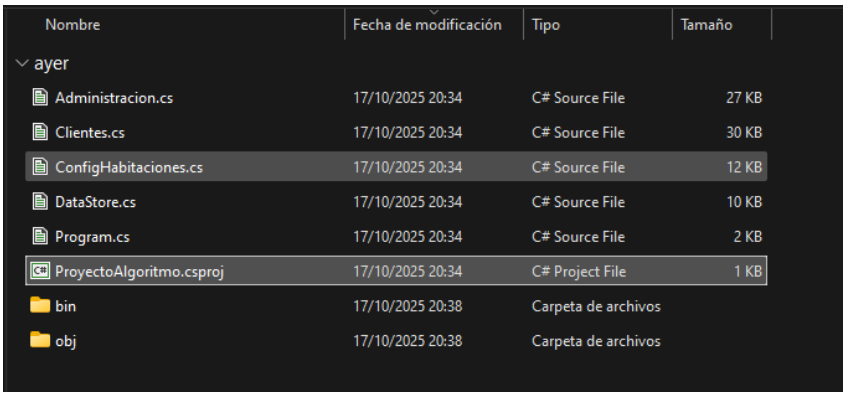
Software:

- Sistema operativo Windows 10 o superior
- Microsoft Visual Studio (versión 2019 o posterior)
- .NET Framework 4.7.2 o superior

Instrucciones de instalación

Descargue o copie la carpeta del sistema SistemaReservacionesHotel.

Asegúrese de que el archivo ejecutable (SistemaReservaciones.csproj) se encuentre en la carpeta principal.



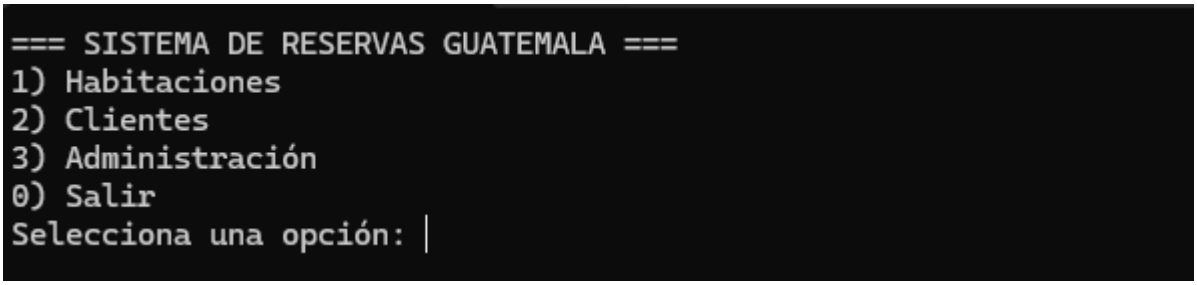
Haga doble clic sobre el ejecutable para iniciar el sistema.

Si el sistema solicita permisos, haga clic en “Permitir”.

Espere unos segundos a que carguen los archivos iniciales de datos.

Descripción general del sistema

Al iniciar el sistema, se mostrará el menú principal con las siguientes opciones:



El usuario debe seleccionar la opción deseada ingresando el número correspondiente o haciendo clic sobre el botón, según la interfaz de uso.

Funciones principales

Registro de habitación

Permite ingresar los datos de cada habitación: número, tipo (sencilla, doble o suite), precio y estado (disponible/ocupada).

- Ingrese los valores solicitados.
- Presione **Enter** para confirmar.
- El sistema confirmará el registro exitoso.

Registro de cliente

Registra los datos personales del cliente (nombre, documento, teléfono, días de estancia).

- Asegúrese de ingresar información completa.
- El sistema verificará que exista una habitación disponible antes de continuar.

Reservación

Asigna una habitación disponible al cliente.

- Seleccione el número de habitación.
- Confirme la acción.
- El sistema mostrará el mensaje: “Reservación realizada con éxito”.

Cancelación

Libera una habitación previamente reservada.

- Ingrese el número de habitación.
- Confirme la cancelación.
- El sistema cambiará el estado a “Disponible”.

Consulta de disponibilidad

Muestra una lista de habitaciones libres.

- Solo lectura, no requiere ingreso de datos.

Generación de factura

Calcula el monto total en base al número de días de estancia y el precio de la habitación.

- El sistema imprimirá o mostrará el total a pagar con desglose de datos.

Mensajes de error comunes

Código	**Descripción**	**Solución**
E001	Habitación no disponible	Seleccione otra habitación.
E002	Datos incompletos	Verifique que todos los campos estén llenos.
E003	Cliente no registrado	Registre primero al cliente.
E004	Archivo no encontrado	Verifique la ruta del archivo del sistema.

Preguntas frecuentes (FAQ)

¿Se pueden modificar los precios de las habitaciones?

Sí, desde la opción de Modificar datos o editando el archivo correspondiente.

¿Qué pasa si cierro el sistema sin guardar?

El sistema guarda automáticamente la información actual al cerrar correctamente.

¿Puedo usar el sistema en otra computadora?

Sí, siempre que tenga instalado .NET Framework 4.7.2 o superior.

Recomendaciones de uso

No apague el equipo mientras el sistema guarda datos.

Realice respaldos periódicos de los archivos.

Utilice nombres de cliente y número de habitación sin caracteres especiales.

Revise la carpeta de archivos antes de ejecutar el programa.

Glosario de términos

Término	**Definición**
Habitación disponible	Espacio libre que puede ser reservado por un cliente.

Reserva	Acción de asignar una habitación a un cliente por un tiempo determinado.
Factura	Documento generado con el monto a pagar por la estancia.
Cliente	Persona que solicita y paga el servicio de hospedaje.

Propuesta Económica

Sistema de Reservas de Hotel

Resumen

La presente propuesta económica tiene como finalidad detallar los costos estimados del desarrollo e implementación del Sistema de Reservaciones de Hotel, una aplicación diseñada para optimizar los procesos de registro y control de habitaciones, clientes y facturación en establecimientos hoteleros. El sistema fue desarrollado en lenguaje C# bajo el paradigma de programación estructurada, con almacenamiento de datos mediante archivos.

El documento incluye el desglose de costos directos e indirectos, el tiempo estimado de desarrollo, y un análisis costo-beneficio que evidencia la viabilidad del proyecto desde el punto de vista técnico y financiero.

Introducción

La automatización de procesos en los servicios de hospedaje es una necesidad creciente, especialmente en pequeños hoteles que buscan mejorar su control operativo y atención al cliente. El Sistema de Reservaciones de Hotel surge como una alternativa económica, eficiente y adaptable a las necesidades de administración básica.

Esta propuesta económica detalla los costos relacionados con su desarrollo, pruebas, mantenimiento inicial y recursos necesarios para garantizar la correcta operación del sistema.

Desglose de costos

Costos de desarrollo

Concepto	**Descripción**	**Costo estimado (Q)**
Análisis y diseño del sistema	Levantamiento de requerimientos y diseño de estructura	350.00
Programación y codificación	Desarrollo del sistema en C#	500.00
Pruebas y depuración	Corrección de errores y validaciones	150.00
Elaboración de manuales	Manual técnico y de usuario	100.00
Subtotal		**1,100.00**

Costos de infraestructura

Concepto	**Descripción**	**Costo estimado (Q)**
Equipo de cómputo	Computadora para desarrollo y pruebas	250.00
Software de desarrollo	Visual Studio y herramientas asociadas	100.00

Energía eléctrica y mantenimiento	Insumos durante el desarrollo	75.00
Subtotal		**425.00**

Costos indirectos

Concepto	**Descripción**	**Costo estimado (Q)**
Transporte y logística	Desplazamientos y gastos operativos	50.00
Materiales de documentación	Impresiones, papelería y encuadernado	75.00
Subtotal		**125.00**

Costo total estimado del proyecto

Categoría	**Total (Q)**
Costos de desarrollo	1,100.00
Costos de infraestructura	425.00
Costos indirectos	125.00
Total general del proyecto	**Q1,650.00**

Análisis costo-beneficio

El Sistema de Reservaciones de Hotel representa una inversión de bajo costo y alta eficiencia. Su desarrollo local evita gastos por licencias externas y permite su adaptación a las necesidades del usuario final.

Beneficios principales:

Reducción del tiempo de registro y control manual.

Disminución de errores humanos en asignaciones.

Mejora de la atención al cliente.

Capacidad de escalabilidad futura (versión gráfica o web).

Conclusiones

La propuesta económica del Sistema de Reservaciones de Hotel demuestra la factibilidad financiera y técnica del proyecto. Con un costo total de aproximadamente Q1,650.00, se obtiene una solución funcional, confiable y adaptable a las necesidades de cualquier hotel pequeño o mediano.

La implementación de este sistema representa una inversión inteligente que optimiza procesos y mejora la eficiencia operativa.

Matriz de riesgos

Sistema de Reservas de Hotel

Introducción

La gestión de riesgos constituye una etapa fundamental en el ciclo de vida del software, ya que permite anticipar problemas que podrían comprometer los objetivos del proyecto.

En el caso del Sistema de Reservaciones de Hotel, se realizó un análisis preventivo de los posibles riesgos que podrían surgir durante el desarrollo, implementación y uso del sistema. Este documento presenta una matriz que clasifica los riesgos según su tipo, probabilidad, impacto y estrategias de mitigación.

Matriz de riesgos del proyecto

	Riesgo identificado	**Tipo**	**Probabilidad**	**Impacto**	**Nivel de riesgo**	**Acción preventiva**	**Plan de contingencia**
1	Falla en el código fuente	Técnico	2	3	6 (Alto)	Revisar y depurar cada módulo antes de la integración final	Restaurar copia de seguridad funcional
2	Pérdida de archivos del sistema	Técnico	3	3	9 (Crítico)	Realizar respaldos automáticos	Recuperar copia en directorio alterno
3	Errores en la captura de datos	Operativo	2	2	4 (Medio)	Implementar validaciones de entrada	Depuración manual de registros

4	Incompatibilidad del sistema con versiones antiguas de .NET	Técnico	1	3	3 (Medio)	Verificar versión de framework antes de instalación	Instalar versión compatible
5	Desconocimiento del usuario final	Humano	2	2	4 (Medio)	Capacitación previa al uso del sistema	Elaborar guía de uso resumida
6	Fallo eléctrico o interrupción del equipo	Operativo	1	3	3 (Medio)	Usar UPS y guardar con frecuencia	Restaurar datos desde último guardado
7	Eliminación accidental de archivos	Humano	2	3	6 (Alto)	Restringir acceso a archivos críticos	Recuperar desde respaldo automático
8	Retraso en la entrega del proyecto	Administrativo	2	2	4 (Medio)	Planificar cronograma realista	Ajustar tiempos con prioridad en módulos clave
9	Errores en el cálculo de facturas	Lógico	2	3	6 (Alto)	Revisar funciones de cálculo antes de compilar	Aplicar pruebas unitarias de validación
10	Falla de hardware durante pruebas	Técnico	1	3	3 (Medio)	Verificar funcionamiento de equipo antes de uso	Reemplazar componente dañado

Análisis general

La evaluación evidencia que los riesgos más críticos corresponden al ámbito técnico, especialmente aquellos vinculados a la pérdida de archivos y errores en el código fuente.

Estos representan los mayores impactos potenciales en el proyecto y deben ser atendidos con copias de seguridad, revisión modular y pruebas constantes.

Los riesgos de tipo humano y operativo presentan un nivel medio, mitigable mediante capacitación, buenas prácticas de respaldo y control de accesos.

Conclusiones

El análisis de riesgos demuestra que el Sistema de Reservaciones de Hotel es técnicamente viable y que los posibles incidentes pueden controlarse mediante una adecuada planificación, respaldos periódicos y capacitación de los usuarios.

La matriz elaborada permite implementar estrategias preventivas que garantizan la estabilidad, confiabilidad y continuidad del sistema en su entorno operativo.