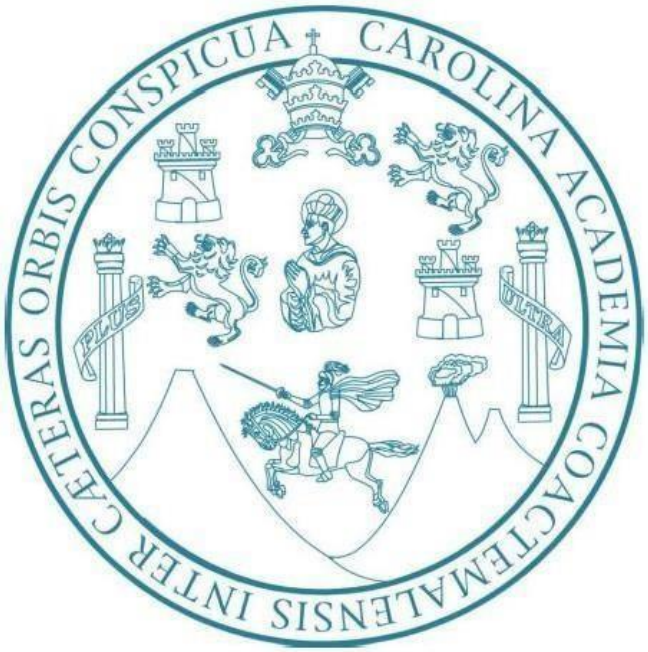
Facultad de Ingeniería
Lenguajes formales y de programación
B-
Cat. Inga. Zulma Aguirre
Tutor académico: Jonatan Leonel García Arana



Manual Técnico

Práctica #1

Introducción

El programa está diseñado para gestionar un inventario utilizando un menú interactivo. Se implementa en Fortran, un lenguaje de programación eficiente para cálculos numéricos y manipulación de datos. El programa permite cargar un inventario inicial, aplicar movimientos a este inventario desde un archivo de texto, y generar informes de inventario antes y después de aplicar dichos movimientos.

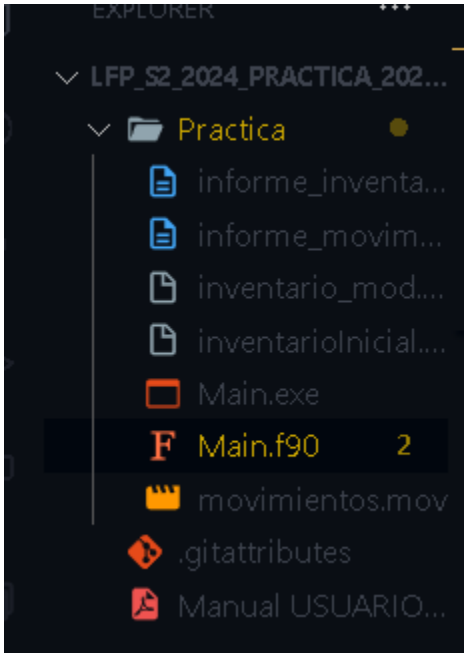
Objetivos

Objetivo General
Desarrollar y documentar un programa en Fortran que permita la gestión eficiente de un inventario, facilitando la carga de datos iniciales, el procesamiento de movimientos de stock, y la generación de informes detallados en formato de texto.

Objetivos Específicos

1. Implementar subrutinas en Fortran para gestionar la creación, modificación, y eliminación de ítems en el inventario.
2. Desarrollar un sistema de carga de datos a partir de archivos de texto que sigan un formato predefinido, asegurando la validación de los mismos.
3. Crear una subrutina para procesar comandos en línea que permita la actualización automática del inventario basado en instrucciones específicas.
4. Generar informes de inventario que incluyan detalles de los ítems y los movimientos realizados, asegurando la precisión y claridad en la presentación de los datos.
5. Garantizar que el programa funcione de manera eficiente en los entornos de hardware y software especificados, cumpliendo con los requisitos mínimos establecidos.

1. Estructura del código



1.1 **Main.f90:** El código cuenta de una sola clase Main.f90, que acá es donde podemos encontrar toda la lógica implementada para nuestro programa.

1.2 **Main.exe**: En este archivo lo único que obtendremos es la compilación de nuestra aplicación, no es posible su lectura ya que es de lenguaje máquina.

1.3 **Inventario_mod.mod**: Archivo utilizado por fortran para poder almacenar versiones del código.

1.4 **Archivos de lectura:** En estos podemos encontrar los que son el archivo. Inv que este contendrá nuestro inventario inicial para nosotros poder cargarlo a nuestro programa. Archivo .mov, en el tendremos los movimientos que le ordenaremos que haga en el programa al inventario como puede ser eliminar stock de cierta ubicación y de cierto equipo, así mismo como agregar stock para x ubicación.

2. Clase principal y sus primordiales características

La clase Main.f90 tiene 3 divisiones principales, la que podemos observar en la imagen de arriba es que utilizamos el modulo llamado “inventario_mod”, en el lo que se encuentran son las subrutinas (en otros lenguajes conocidos como métodos o funciones) donde cada uno tiene su función especifica. Tenemos los type que su trabajo es almacenar las variables de forma global en el código para su utilización (podría llamarse lo que es un constructor en lenguajes de programación como Java). Uno lo utilizamos para la lectura de nuestro archivo. inv y el otro para los movimientos generados por él .mov

```
type :: inventario_t
    character(len=1024) :: nombre
    integer :: cantidad
    real :: precioUnitario
    character(len=1024) :: ubicacion
end type inventario_t

type :: movimiento_t
    character(len=1024) :: nombre
    integer :: cantidad
    character(len=1024) :: ubicacion
    character(len=1024) :: tipo
end type movimiento_t
```

2.1 Tenemos la subrutina “Generar informe “donde podemos encontrar la generación del informe. En ella trabajamos los parámetros para poder crear un nuevo archivo con los movimientos que hacemos durante el programa.

```
subroutine generar_informe(inventarios, num_equipos, movimientos, num_movimientos)
    implicit none
    type(inventario_t), intent(inout) :: inventarios(:)
    type(movimiento_t), intent(in) :: movimientos(:)
    integer, intent(in) :: num_equipos, num_movimientos
    integer :: ios, i, j
    character(len=256) :: filename_inventario, filename_movimientos
    real :: valor_total


    filename_movimientos = 'informe_movimientos.txt'

    ! Generar informe de inventario inicial
    open(30, file=trim(filename_movimientos), status='replace', action='write', iostat=ios)
    if (ios /= 0) then
        print *, 'Error al abrir el archivo de informe de inventario, codigo de error:', ios
        return
    end if

    write(30, '(A)') '-------------------------------'
    write(30, '(A)') '      Informe de Movimientos      '
    write(30, '(A)') '-------------------------------'
    write(30, '(A)') ''
    write(30, '(A)') 'Equipo          Cantidad   Precio Unitario   Valor Total   Ubicacion'
    write(30, '(A)') '-----------------------------------------------------------------------'

    do i = 1, num_equipos
        valor_total = inventarios(i)%cantidad * inventarios(i)%precioUnitario
        write(30, '(A20, I10, 1X, A, F10.2, 1X, A, F10.2, A20)') trim(inventarios(i)%nombre), inventarios(i)%cantidad, 'Q', inventarios(i)%precioUnitario, 'Q', valor_total, trim(inventarios(i)%ubicacion)
    end do

    close(30)

    ! Aplicar movimientos al inventario
    do i = 1, num_movimientos
        do j = 1, num_equipos
            if (trim(movimientos(i)%nombre) == trim(inventarios(j)%nombre) .and. trim(movimientos(i)%ubicacion) == trim(inventarios(j)%ubicacion)) then
                if (trim(movimientos(i)%tipo) == 'Agregar') then
                    inventarios(j)%cantidad = inventarios(j)%cantidad + movimientos(i)%cantidad
                else if (trim(movimientos(i)%tipo) == 'Eliminar') then
                    inventarios(j)%cantidad = inventarios(j)%cantidad - movimientos(i)%cantidad
                end if
                exit
            end if
        end do
    end do
end subroutine generar_informe
```

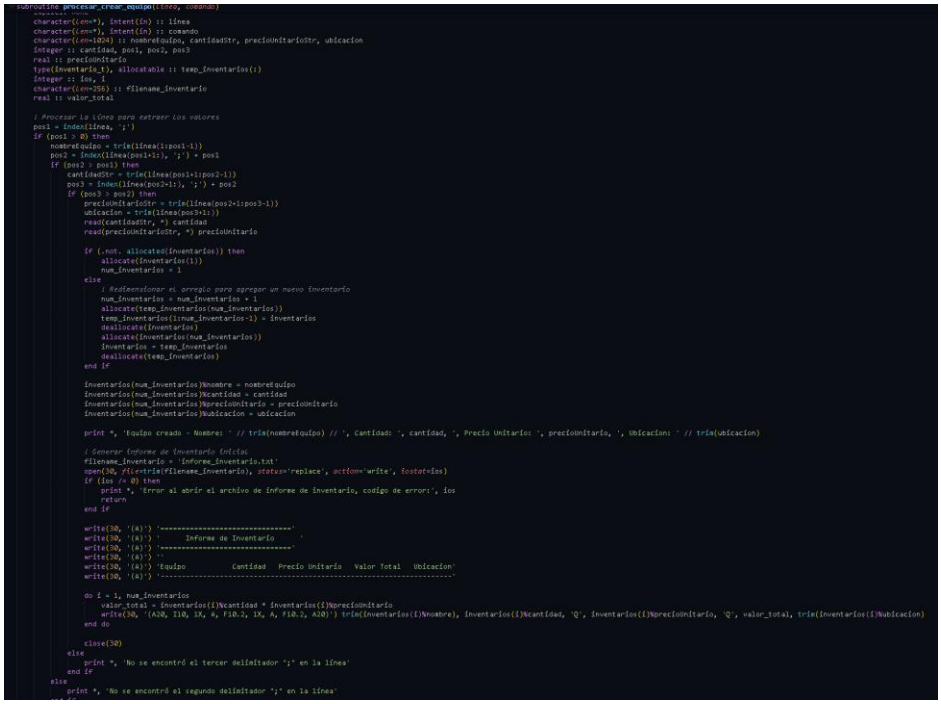
2.2 Subrutina eliminar equipo reduce la cantidad de un ítem en el inventario, manejamos las diferentes verificaciones si existe suficiente stock o si existe en cierta localidad el ítem a eliminar. También registra el movimiento de eliminación en el sistema.

```
subroutine procesar_eliminar_equipo(linea)
    implicit none
    character(len=*), intent(in) :: linea
    character(len=1024) :: nombreEquipo, cantidadStr, ubicacion
    integer :: cantidad, pos1, pos2, i
    logical :: encontrado, ubicacion_existe
    type(movimiento_t), allocatable :: temp_movimientos(:)

    pos1 = index(linea, ';')
    if (pos1 > 0) then
        nombreEquipo = trim(linea(1:pos1-1))
        pos2 = index(linea(pos1+1:), ';') + pos1
        if (pos2 > pos1) then
            cantidadStr = trim(linea(pos1+1:pos2-1))
            ubicacion = trim(linea(pos2+1:))
            read(cantidadStr, *) cantidad

            ubicacion_existe = .false.
            do i = 1, num_inventarios
                if (trim(inventarios(i)%ubicacion) == ubicacion) then
                    ubicacion_existe = .true.
                    exit
                end if
            end do

            if (.not. ubicacion_existe) then
                print *, '================================================='
                print *, 'Error: La ubicacion especificada no existe en el inventario.'
                print *, 'Ubicación: ', trim(ubicacion)
                print *, '================================================='
                return
            end if

            encontrado = .false.
            do i = 1, num_inventarios
                if (trim(inventarios(i)%nombre) == trim(nombreEquipo) .and. trim(inventarios(i)%ubicacion) == trim(ubicacion)) then
                    encontrado = .true.
                    if (inventarios(i)%cantidad >= cantidad) then
                        inventarios(i)%cantidad = inventarios(i)%cantidad - cantidad
                        print *, 'Stock eliminado - Nombre: ', trim(nombreEquipo), ', Cantidad: ', cantidad, ', Ubicacion: ', trim(ubicacion)
                        print *, 'Inventario actual de ', trim(nombreEquipo), ': ', inventarios(i)%cantidad
                    else
                        print *, '================================================='
                        print *, 'Error: Cantidad insuficiente en la ubicacion especificada, tiene que ser una cantidad menor del inventario.'
                        print *, 'Equipo: ', trim(nombreEquipo), ', Ubicacion: ', trim(ubicacion)
                        print *, '================================================='
                    end if
                    exit
                end if
            end do
            if (.not. encontrado) then
                print *, '================================================='
                print *, 'Error: El equipo no existe en la ubicacion especificada.'
                print *, 'Equipo: ', trim(nombreEquipo), ', Ubicacion: ', trim(ubicacion)
                print *, '================================================='
            end if
```

2.3 Subrutina agregar stock actualiza el inventario agregando stock a un ítem existente. Si el ítem no existe, muestra un mensaje de error.

```
subroutine procesar_agregar_stock(linea)
    implicit none
    character(len=*), intent(in) :: linea
    character(len=1024) :: nombreEquipo, cantidadStr, ubicacion
    integer :: cantidad, pos1, pos2, i
    logical :: encontrado, ubicacion_existe
    type(movimiento_t), allocatable :: temp_movimientos(:)

    pos1 = index(linea, ';')
    if (pos1 > 0) then
        nombreEquipo = trim(linea(1:pos1-1))
        pos2 = index(linea(pos1+1:), ';') + pos1
        if (pos2 > pos1) then
            cantidadStr = trim(linea(pos1+1:pos2-1))
            ubicacion = trim(linea(pos2+1:))
            read(cantidadStr, *) cantidad

            ubicacion_existe = .false.
            do i = 1, num_inventarios
                if (trim(inventarios(i)%ubicacion) == ubicacion) then
                    ubicacion_existe = .true.
                    exit
                end if
            end do

            if (.not. ubicacion_existe) then
                print *, '================================================='
                print *, 'Error: La ubicacion especificada no existe en el inventario.'
                print *, 'Ubicación: ' // trim(ubicacion)
                print *, '================================================='
                return
            end if

            encontrado = .false.
            do i = 1, num_inventarios
                if (trim(inventarios(i)%nombre) == trim(nombreEquipo) .and. trim(inventarios(i)%ubicacion) == trim(ubicacion)) then
                    if (inventarios(i)%cantidad + cantidad <= max_inventario) then
                        inventarios(i)%cantidad = inventarios(i)%cantidad + cantidad
                        encontrado = .true.
                        print *, 'Stock agregado - Nombre: ' // trim(nombreEquipo) // ', Cantidad: ', cantidad, ', Ubicacion: ' // trim(ubicacion)
                        print *, 'Inventario actual de ' // trim(nombreEquipo) // ': ', inventarios(i)%cantidad
                    else
                        print *, '================================================='
                        print *, 'Error: No se puede agregar al inventario. Excede el maximo permitido.'
                        print *, '================================================='
                    end if
                    exit
                end if
            end do
            if (.not. encontrado) then
                print *, '================================================='
                print *, 'Error: El equipo no existe en el inventario.'
                print *, '================================================='
            end if

            ! Registrar el movimiento
            num_movimientos = num_movimientos + 1
```

2.4 Subrutina procesar_crear_equipo procesa la creación de un nuevo ítem en el inventario a partir de una línea de texto que contiene la información del equipo. La subrutina descompone la línea, valida los datos, y agrega el equipo al inventario.



2.5 Subrutina opcion1 carga un archivo de inventario inicial. El archivo debe seguir un formato específico, con datos separados por punto y coma.

```
subroutine opcion1()
    character(len=1024) :: linea
    integer :: ios
    character(len=256) :: filename
    print *, '================================================='
    print *, 'Has seleccionado la Opcion 1'

    do
        print *, '================================================='
        print *, 'Por favor, ingrese la ruta del archivo de inventario:'
        read *, filename
        print *, '================================================='
        print *, 'Intentando abrir el archivo:', trim(filename)
        print *, '================================================='
        open(unit = 20, file = filename, status = 'old', action = 'read', iostat = ios)
        if (ios /= 0) then
            if (ios == 2) then
                print *, 'El archivo no existe. Intente de nuevo.'
            else
                print *, 'Error al abrir el archivo, código de error:', ios
            end if
        else
            print *, 'Archivo abierto exitosamente'
            print *, '-------------------------------------------------'
            exit
        end if
    end do

    do
        read(20, '(A)', iostat = ios) linea
        if (ios /= 0) then
            if (ios == -1) then
                print *, '-------------------------------------------------'
                print *, 'Fin del archivo alcanzado'
                exit
            else
                print *, 'Error al leer la línea, código de error:', ios
                exit
            end if
        else
            call procesar_linea(trim(linea))
        end if
    end do

    close(20)
end subroutine opcion1
```

2.6 Subrutina opcion2 carga un archivo con instrucciones de movimientos (agregar o eliminar stock) y las procesa.

```
subroutine opcion2()
    implicit none
    integer :: ios
    character(len=256) :: filename
    character(len=1024) :: linea, comando

    print *, '================================================='
    print *, 'Ingrese la ruta del archivo .mov:'
    read(*, '(A)') filename
    print *, '================================================='

    ! Abrir el archivo especificado por el usuario
    open(20, file=trim(filename), status='old', action='read', iostat=ios)
    if (ios /= 0) then
        print *, 'Error al abrir el archivo, codigo de error:', ios
        return
    end if
    do
        read(20, '(A)', iostat = ios) linea
        if (ios /= 0) then
            if (ios == -1) then
                print *, '-------------------------------------------------'
                print *, 'Fin del archivo alcanzado'
                exit
            else
                print *, 'Error al leer la linea, código de error:', ios
                exit
            end if
        else
            comando = trim(linea(1:index(linea, ' ') - 1))
            select case (comando)
                case ('agregar_stock')
                    call procesar_agregar_stock(trim(linea(index(linea, ' ') + 1:)))
                case ('eliminar_equipo')
                    call procesar_eliminar_equipo(trim(linea(index(linea, ' ') + 1:)))
                case default
                    print *, 'Comando no reconocido: ', comando
            end select
        end if
    end do

    close(20)
end subroutine opcion2
```

2.7 Subrutina opcion3 genera y guarda un informe con el estado actual del inventario y los movimientos realizados.

```
    subroutine opcion3(inventarios, num_equipos, movimientos, num_movimientos)
        implicit none
        type(inventario_t), intent(inout) :: inventarios(:)
        type(movimiento_t), intent(inout) :: movimientos(:)
        integer, intent(in) :: num_equipos, num_movimientos

        ! Llamar a la subrutina para generar el informe del inventario y movimientos
        call generar_informe(inventarios, num_equipos, movimientos, num_movimientos)

        print *, '================================================='
        print *, 'Informe del inventario y movimientos generado exitosamente.'
        print *, '================================================='
    end subroutine opcion3

end module inventario_mod
```

2.8 Subrutina leer_linea procesa una línea de texto que contiene un comando para actualizar el inventario. Dependiendo del comando, la línea se descompone para crear un nuevo ítem, agregar stock, o realizar otras operaciones relacionadas con el inventario.

```
subroutine procesar_linea(linea)
    character(len=*), intent(in) :: linea
    character(len=1024) :: comando
    integer :: pos1

    pos1 = index(linea, ' ')
    if (pos1 > 0) then
        comando = trim(linea(1:pos1-1))
        select case (comando)
            case ('crear_equipo')
                call procesar_crear_equipo(linea(pos1+1:), comando)
            case ('agregar_stock')
                call procesar_agregar_stock(linea(pos1+1:))
            case default
                print *, 'Comando no reconocido:', trim(comando)
        end select
    else
        print *, 'Formato de linea no valido:', trim(linea)
    end if
end subroutine procesar_linea


subroutine opcion1()···
```

3. Formato para los archivos de lectura

Se debe de tomar en cuenta que el formato para que tenga una lectura adecuada los archivos de entrada, tanto como él. inv y el .mov son:
INV:

```
crear_equipo <nombre>;<cantidad>;<precio_unitario>;<ubicación>
```

.MOV:

```
agregar_stock <nombre>;<cantidad>;<ubicación>
```

```
eliminar_equipo <nombre>;<cantidad>;<ubicación>
```