Algoritmo Nodo

```
Definir Estudiante, Libro, Prestamo como tipo de dato
Definir Nodo como tipo de dato

Procedimiento ConstructorNodoEstudiante(valorEs: Estudiante)
    nodo <- Nuevo Nodo
    nodo.valorEs <- valorEs
    nodo.valorLib <- Nulo
    nodo.valorPr <- Nulo
    nodo.siguiente <- Nulo
Fin Procedimiento

Procedimiento ConstructorNodoLibro(valorLib: Libro)
    nodo <- Nuevo Nodo
    nodo.valorLib <- valorLib
    nodo.valorEs <- Nulo
    nodo.valorPr <- Nulo
    nodo.siguiente <- Nulo
Fin Procedimiento

Procedimiento ConstructorNodoPrestamo(valorPr: Prestamo)
    nodo <- Nuevo Nodo
    nodo.valorPr <- valorPr
    nodo.valorEs <- Nulo
    nodo.valorLib <- Nulo
    nodo.siguiente <- Nulo
Fin Procedimiento

Función ObtenerValor() retorna Estudiante
```

```
    Retornar nodo.valorEs
Fin Función

Función ObtenerValorLib() retorna Libro
    Retornar nodo.valorLib
Fin Función

Función ObtenerValorPr() retorna Prestamo
    Retornar nodo.valorPr
Fin Función

Función ObtenerSiguiente() retorna Nodo
    Retornar nodo.siguiente
Fin Función

Procedimiento EstablecerValor(valorEs: Estudiante)
    nodo.valorEs <- valorEs
Fin Procedimiento

Procedimiento EstablecerValorLib(valorLib: Libro)
    nodo.valorLib <- valorLib
Fin Procedimiento

Procedimiento EstablecerValorPr(valorPr: Prestamo)
    nodo.valorPr <- valorPr
Fin Procedimiento

Procedimiento EstablecerSiguiente(siguiente: Nodo)
    nodo.siguiente <- siguiente
```

```
    Fin Procedimiento
Fin Algoritmo
```

Algoritmo Estudiante

```
    Definir nombre, apellido, cedula, carrera como cadena de caracteres

    Procedimiento ConstructorEstudiante(nombre: cadena, apellido: cadena, cedula: cadena,
carrera: cadena)
        estudiante <- Nuevo Estudiante
        estudiante.nombre <- nombre
        estudiante.apellido <- apellido
        estudiante.cedula <- cedula
        estudiante.carrera <- carrera
    Fin Procedimiento

    Función ObtenerNombre() retorna cadena
        Retornar estudiante.nombre
    Fin Función

    Función ObtenerApellido() retorna cadena
        Retornar estudiante.apellido
    Fin Función

    Función ObtenerCedula() retorna cadena
        Retornar estudiante.cedula
    Fin Función

    Función ObtenerCarrera() retorna cadena
        Retornar estudiante.carrera
```

```
    Fin Función

    Función ToString() retorna cadena
        Retornar "Nombre: " + ObtenerNombre() + "\r\n" +
                 "Apellido: " + ObtenerApellido() + "\r\n" +
                 "Cédula: " + ObtenerCedula() + "\r\n" +
                 "Carrera: " + ObtenerCarrera() + "\r\n"
    Fin Función
Fin Algoritmo
```

Algoritmo Libro

```
    Definir titulo, nombreAutor, apellidoAutor, genero como cadena de caracteres
    Definir disponible como booleano

    Procedimiento ConstructorLibro(titulo: cadena, nombreAutor: cadena, apellidoAutor: cadena,
genero: cadena, disponible: booleano)
        libro <- Nuevo Libro
        libro.titulo <- titulo
        libro.nombreAutor <- nombreAutor
        libro.apellidoAutor <- apellidoAutor
        libro.genero <- genero
        libro.disponible <- disponible
    Fin Procedimiento

    Función ObtenerTitulo() retorna cadena
        Retornar libro.titulo
    Fin Función

    Función ObtenerNombreAutor() retorna cadena
```

```
    Retornar libro.nombreAutor
  Fin Función

  Función ObtenerApellidoAutor() retorna cadena
    Retornar libro.apellidoAutor
  Fin Función

  Función ObtenerGenero() retorna cadena
    Retornar libro.genero
  Fin Función

  Función ObtenerDisponible() retorna booleano
    Retornar libro.disponible
  Fin Función

  Procedimiento EstablecerDisponible(disponible: booleano)
    libro.disponible <- disponible
  Fin Procedimiento

  Función ToString() retorna cadena
    Retornar "Título: " + ObtenerTitulo() + "\r\n" +
        "Autor: " + ObtenerNombreAutor() + " " + ObtenerApellidoAutor() + "\r\n" +
        "Género: " + ObtenerGenero() + "\r\n"
  Fin Función
Fin Algoritmo
```

Algoritmo Prestamo

```
  Definir Estudiante, Libro como tipo de dato
  Definir fechaPrestamo como tipo de dato DateTime
```

```
    Procedimiento ConstructorPrestamo(estudiante: Estudiante, libro: Libro, fechaPrestamo:
DateTime)
        prestamo <- Nuevo Prestamo
        prestamo.estudiante <- estudiante
        prestamo.libro <- libro
        prestamo.fechaPrestamo <- fechaPrestamo
    Fin Procedimiento

    Función ObtenerEstudiante() retorna Estudiante
        Retornar prestamo.estudiante
    Fin Función

    Función ObtenerLibro() retorna Libro
        Retornar prestamo.libro
    Fin Función

    Función ObtenerFechaPrestamo() retorna DateTime
        Retornar prestamo.fechaPrestamo
    Fin Función

    Procedimiento EstablecerFechaPrestamo(fechaPrestamo: DateTime)
        prestamo.fechaPrestamo <- fechaPrestamo
    Fin Procedimiento

    Función ToString() retorna cadena
        Retornar ObtenerEstudiante().getNombre() + "-" + ObtenerEstudiante().getApellido() + "-" +
            ObtenerEstudiante().getCedula() + "-" + ObtenerEstudiante().getCarrera() + "-" +
            ObtenerLibro().getTitulo() + "-" + ObtenerLibro().getnombreAutor() + "-" +
            ObtenerLibro().getApellidoAutor() + "-" + ObtenerLibro().getGenero() + "-" +
```

```
        ObtenerFechaPrestamo()
    Fin Función
Fin Algoritmo
```

Algoritmo ListaEstudiantes

```
    Definir Nodo como tipo de dato
    Definir cabeza como Nodo
    Definir tamano como entero

    Procedimiento ConstructorListaEstudiantes()
        cabeza <- Nulo
        tamano <- 0
    Fin Procedimiento

    Función estaVacia() retorna booleano
        Retornar cabeza = Nulo
    Fin Función

    Procedimiento agregarEstudiante(valor: Estudiante)
        nuevo <- Nuevo Nodo
        nuevo.valor <- valor
        Si cabeza = Nulo Entonces
            cabeza <- nuevo
        Sino
            aux <- cabeza
            Mientras aux.siguiente ≠ Nulo Hacer
                aux <- aux.siguiente
            Fin Mientras
            aux.siguiente <- nuevo
        Fin Si
```

```
    tamano <- tamano + 1
  Fin Procedimiento

  Función buscarEstudiante(cedula: cadena) retorna Estudiante
    aux <- cabeza
    Mientras aux ≠ Nulo Hacer
      Si aux.valor.getCedula() = cedula Entonces
        Retornar aux.valor
      Fin Si
      aux <- aux.siguiente
    Fin Mientras
    Retornar Nulo
Función existeEstudiante(cedula: cadena) retorna booleano
    actual <- cabeza
    Mientras actual ≠ Nulo Hacer
      Si actual.valor.getCedula() = cedula Entonces
        Retornar Verdadero
      Fin Si
      actual <- actual.siguiente
    Fin Mientras
    Retornar Falso
  Fin Función

  Procedimiento eliminarEstudiante(cedula: cadena)
    Si cabeza ≠ Nulo Entonces
      Si cabeza.valor.getCedula() = cedula Entonces
        cabeza <- cabeza.siguiente
        encontrado <- Verdadero
      Sino
```

```
            aux <- cabeza
            Mientras aux.siguiente ≠ Nulo Hacer
               Si aux.siguiente.valor.getCedula() = cedula Entonces
                  aux.siguiente <- aux.siguiente.siguiente
                  encontrado <- Verdadero
                  Romper
               Fin Si
               aux <- aux.siguiente
            Fin Mientras
         Fin Si (encontrado)
            Tamano-- ;
   Fin Procedimiento
Fin Algoritmo
```

Algoritmo ListaLibros

```
   Definir Nodo como tipo de dato
   Definir cabeza como Nodo
   Definir tamaño como entero

   Procedimiento ConstructorListaLibros()
      cabeza <- Nulo
      tamaño <- 0
   Fin Procedimiento

   Función getCabeza() retorna Nodo
      Retornar cabeza
   Fin Función

   Función getTamaño() retorna entero
      Retornar tamaño
```

```
Fin Función

Procedimiento agregarLibro(libro: Libro)
    nuevo <- Nuevo Nodo
    nuevo.valorLib <- libro
    Si cabeza = Nulo Entonces
        cabeza <- nuevo
    Sino
        aux <- cabeza
        Mientras aux.siguiente ≠ Nulo Hacer
            aux <- aux.siguiente
        Fin Mientras
        aux.siguiente <- nuevo
    Fin Si
    tamaño <- tamaño + 1
Fin Procedimiento

Función buscarLibro(titulo: cadena) retorna Libro
    aux <- cabeza
    Mientras aux ≠ Nulo Hacer
        Si aux.valorLib.getTitulo() = titulo Entonces
            Retornar aux.valorLib
        Fin Si
        aux <- aux.siguiente
    Fin Mientras
    Retornar Nulo
Fin Función
Función existeLibro(titulo: cadena) retorna booleano
    actual <- cabeza
```

```
    Mientras actual ≠ Nulo Hacer
        Si actual.valorLib.getTitulo() = titulo Entonces
            Retornar Verdadero
        Fin Si
        actual <- actual.siguiente
    Fin Mientras
    Retornar Falso
Fin Función

Procedimiento eliminarLibro(titulo: cadena)
    Si cabeza ≠ Nulo Entonces
        Si cabeza.valorLib.getTitulo() = titulo Entonces
            cabeza <- cabeza.siguiente
            tamaño <- tamaño - 1
        Sino
            aux <- cabeza
            Mientras aux.siguiente ≠ Nulo Hacer
                Si aux.siguiente.valorLib.getTitulo() = titulo Entonces
                    aux.siguiente <- aux.siguiente.siguiente
                    tamaño <- tamaño - 1
                    Romper
                Fin Si
                aux <- aux.siguiente
            Fin Mientras
        Fin Si
    Fin Si
Fin Procedimiento
Función mostrarLibros() retorna cadena
    listaLibros <- "" // Se crea una cadena vacía
```

```
    aux <- cabeza // Se crea un nodo auxiliar
    Mientras aux ≠ Nulo Hacer
      libro <- aux.valorLib // Se obtiene el libro del nodo
      listaLibros <- listaLibros + libro.ToString() + "\r\n" // Se concatenan los datos del libro
      aux <- aux.siguiente // Se avanza al siguiente nodo
    Fin Mientras
    Retornar listaLibros // Se retorna la cadena con los títulos de los libros
  Fin Función

  Función mostrarLibrosGenero(genero: cadena) retorna cadena
    listaLibros <- "" // Se crea una cadena vacía
    aux <- cabeza // Se crea un nodo auxiliar
    Mientras aux ≠ Nulo Hacer
      libro <- aux.valorLib // Se obtiene el libro del nodo
      Si libro.getGenero() = genero Entonces // Si el género del libro coincide con el género ingresado
        listaLibros <- listaLibros + libro.ToString() + "\r\n" // Se concatenan los datos del libro
      Fin Si
      aux <- aux.siguiente // Se avanza al siguiente nodo
    Fin Mientras
    Retornar listaLibros // Se retorna la cadena con los títulos de los libros
  Fin Función
Fin Algoritmo
```

Definir clase ListaPrestamos

Definir atributos cabeza y tamano

Definir método constructor ListaPrestamos

Inicializar cabeza como nulo

Inicializar tamano como 0

Fin constructor

```
Definir método getCabeza
   Retornar cabeza
Fin método

Definir método getTamano
   Retornar tamano
Fin método

Definir método agregarPrestamo con parámetro prestamo
   Crear un nodo nuevo con el valor prestamo y asignarlo a nuevo
   Si cabeza es nulo
      Asignar nuevo a cabeza
   Sino
      Crear un nodo auxiliar y asignarlo a cabeza
      Mientras auxiliar tenga un siguiente
         Asignar el siguiente de auxiliar a auxiliar
      Fin Mientras
      Asignar nuevo como siguiente de auxiliar
   Fin Si
   Incrementar tamano en 1
Fin método

Definir método eliminarPrestamo con parámetro cedula
   Crear un nodo auxiliar y asignarlo a cabeza
   Crear un nodo anterior y asignarlo como nulo
   Mientras auxiliar no sea nulo
      Si la cédula del estudiante en el valor de prestamo en auxiliar es igual a cedula
         Si anterior es nulo
```

```
            Asignar el siguiente de auxiliar a cabeza
          Sino
            Asignar el siguiente de auxiliar como siguiente de anterior
          Fin Si
          Decrementar tamano en 1
          Retornar
        Fin Si
        Asignar auxiliar a anterior
        Asignar el siguiente de auxiliar a auxiliar
      Fin Mientras
    Fin método

    Definir método buscarPrestamo con parámetro cedula
      Crear un nodo auxiliar y asignarlo a cabeza
      Mientras auxiliar no sea nulo
        Si la cédula del estudiante en el valor de prestamo en auxiliar es igual a cedula
          Retornar el valor de prestamo en auxiliar
        Fin Si
        Asignar el siguiente de auxiliar a auxiliar
      Fin Mientras
      Retornar nulo
    Fin método
  Definir método mostrarPrestamos
    Definir listaPrestamos como Cadena de caracteres con valor vacío
    Crear un nodo auxiliar y asignarlo a cabeza
    Mientras auxiliar no sea nulo
      Obtener el valor de prestamo en auxiliar y asignarlo a prestamo
      Obtener el estudiante en prestamo y asignarlo a estudiante
      Obtener el libro en prestamo y asignarlo a libro
```

Calcular la diferencia de días entre la fecha actual y la fecha de préstamo y asignarla a difFechas

Si difFechas es mayor a 3

Concatenar estudiante, libro y "Estado: Suspendido/a" con un salto de línea y asignarlo a listaPrestamos

Asignar el siguiente de auxiliar a auxiliar

Sino

Concatenar estudiante, libro y "Estado: Habilitado/a" con un salto de línea y asignarlo a listaPrestamos

Asignar el siguiente de auxiliar a auxiliar

Fin Si

Fin Mientras

Retornar listaPrestamos

Fin método

Definir método mostrarSancionados

Definir listaSancionados como Cadena de caracteres con valor vacío

Crear un nodo auxiliar y asignarlo a cabeza

Mientras auxiliar no sea nulo

Obtener el valor de prestamo en auxiliar y asignarlo a prestamo

Obtener el estudiante en prestamo y asignarlo a estudiante

Obtener el libro en prestamo y asignarlo a libro

Calcular la diferencia de días entre la fecha actual y la fecha de préstamo y asignarla a difFechas

Si difFechas es mayor a 3

Concatenar estudiante, libro y "Estado: Suspendido/a" con un salto de línea y asignarlo a listaSancionados

Asignar el siguiente de auxiliar a auxiliar

Sino

Asignar el siguiente de auxiliar a auxiliar

Fin Si

```
    Fin Mientras
    Retornar listaSancionados
Fin método
Fin clase
```

Algoritmo MenuPrincipal

```
    Procedimiento ConstructorMenuPrincipal()
        // Inicialización del formulario
        MostrarFormulario()
    Fin Procedimiento

    Procedimiento btnRegistroAlum_Click(sender: objeto, e: EventArgs)
        frm <- Nuevo frmAlumnoRegistro
        MostrarFormulario(frm) // Mostrar el formulario
    Fin Procedimiento

    Procedimiento btnRegistroPrest_Click(sender: objeto, e: EventArgs)
        frm <- Nuevo frmRegistrarPrestamo
        MostrarFormulario(frm)
    Fin Procedimiento

    Procedimiento button4_Click(sender: objeto, e: EventArgs)
        frm <- Nuevo frmRegistroLibro
        MostrarFormulario(frm)
    Fin Procedimiento

    Procedimiento btnSalir_Click(sender: objeto, e: EventArgs)
        CerrarFormulario()
    Fin Procedimiento
```

```
    Procedimiento btnLibrosDisp_Click(sender: objeto, e: EventArgs)
        LeerArchivoLibrosDisponible()
        frm <- Nuevo frmLibrosDisponibles
        MostrarFormulario(frm)
    Fin Procedimiento
Procedimiento LeerArchivoLibrosDisponible
    Probar
        ruta <- "LibrosIngresados.txt"
        lineas <- LeerTodasLasLineasDelArchivo(ruta)
        Para cada linea en lineas Hacer
            datos <- SepararCadenaPorEspacios(linea)
            datos[3] <- ConvertirATexto(datos[3])
            Si datos[3] = "True" Entonces
                libro <- Nuevo Libro(datos[0], datos[1], datos[2], datos[3], Verdadero)
                DatosListaLibros.lista.agregarLibro(libro)
            Sino
                libro <- Nuevo Libro(datos[0], datos[1], datos[2], datos[3], Falso)
                DatosListaLibros.lista.agregarLibro(libro)
            Fin Si
        Fin Para
    Atrapar Excepcion ex
        MostrarMensaje("Error al leer el archivo de libros disponibles: " + ex.Mensaje)
    Fin Probar

Procedimiento btnEstudiantesBaneados_Click(sender: objeto, e: EventArgs)
    frm <- Nuevo frmEstudiantesBaneados
    MostrarFormulario(frm)
```

```
Procedimiento btnPrestamosAct_Click(sender: objeto, e: EventArgs)
    frm <- Nuevo frmPrestamosActivos
    MostrarFormulario(frm)

Procedimiento button2_Click(sender: objeto, e: EventArgs)
    frm <- Nuevo frmRenovaciones
    MostrarFormulario(frm)

Procedimiento btnDevolucion_Click(sender: objeto, e: EventArgs)
    frm <- Nuevo frmDevoluciones
    MostrarFormulario(frm)
Fin Algoritmo
```

Algoritmo frmAlumnoRegistro

```
    Clase estática DatosLista
        lista <- Nueva ListaEstudiantes // Se crea una lista de estudiantes
    Fin Clase

    Procedimiento ConstructorfrmAlumnoRegistro()
        // Inicialización del formulario
        MostrarFormulario()
    Fin Procedimiento

    Cadena nombre, apellido, cedula, carrera

    Procedimiento txtCedula_KeyPress(sender: objeto, e: KeyPressEventArgs)
        Si No (e.KeyChar >= 48 Y e.KeyChar <= 57 O e.KeyChar = 8) Entonces // Si el caracter
presionado no es un número o la tecla de retroceso
            e.Handled <- Verdadero // No se escribe el caracter
```

```
        MostrarMensaje("Solo se permiten números", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation) // Mensaje de advertencia

        Retornar // Salir del evento

    Fin Si

  Fin Procedimiento


  Procedimiento btnBuscar_Click(sender: objeto, e: EventArgs)

    Si ValidarCampos() Entonces

      AgregarDatos()

      LimpiarDatos()

      EscribirArchivo()

      MostrarMensaje("Se ha registrado el estudiante")

    Sino

      Retornar

    Fin Si

  Fin Procedimiento

 Procedimiento EscribirArchivo

  ruta <- "EstudiantesRegistrados.txt" // Ruta del archivo

  Si ExisteArchivo(ruta) Entonces // Verifica si el archivo en la ruta especificada existe

    agregar <- Nuevo StreamWriter(ruta) // Se crea un objeto de tipo StreamWriter para agregar información al archivo

    agregar.EscribirLinea(nombre + " " + apellido + " " + cedula + " " + carrera) // Se escriben los datos del estudiante recién registrado en el archivo

    agregar.Cerrar() // Se cierra el archivo

  Sino // Si el archivo no existe

    escribir <- Nuevo TextWriter(ruta) // Se crea un objeto de tipo TextWriter para escribir en el archivo

    escribir.EscribirLinea(nombre + " " + apellido + " " + cedula + " " + carrera) // Se escriben los datos del estudiante recién registrado en el archivo

    escribir.Cerrar() // Se cierra el archivo

  Fin Si
```

```
Fin Algoritmo


Procedimiento LimpiarDatos()
   LimpiarTexto(txtNombre)
   LimpiarTexto(txtApellido)
   LimpiarTexto(txtCedula)
   SeleccionarIndice(cboCarrera, -1) // Seleccionar el índice -1 del combo box para que quede en blanco
Fin Procedimiento


Función ValidarCampos() retorna booleano
   Si txtNombre.Text = "" O txtApellido.Text = "" O txtCedula.Text = "" O cboCarrera.SelectedIndex = -1 Entonces // Si alguno de los campos está vacío
      MostrarMensaje("Debe llenar todos los campos", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation) // Se muestra un mensaje de advertencia
      Retornar Falso // No se puede continuar
   Sino
      Retornar Verdadero // Todos los campos están llenos
   Fin Si
Fin Función


   Procedimiento AgregarDatos()
      nombre <- txtNombre.Text
      apellido <- txtApellido.Text
      cedula <- txtCedula.Text
      carrera <- cboCarrera.Text


      estudiante <- Nuevo Estudiante(nombre, apellido, cedula, carrera) // Se crea un objeto de tipo estudiante con los datos ingresados
      txtListEstudiantes.Text <- txtListEstudiantes.Text + estudiante.ToString() + "\r\n"
```

```
    DatosLista.lista.agregarEstudiante(estudiante) // Se agrega el objeto a la lista de estudiantes
  Fin Procedimiento
  Procedimiento txtApellido_KeyPress(sender: objeto, e: KeyPressEventArgs)
    Si No (EsLetra(e.KeyChar) O e.KeyChar = 8 O e.KeyChar = 32) Entonces
      e.Handled <- Verdadero
      MostrarMensaje("Solo se permiten letras", "Advertencia", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation)
      Retornar
    Fin Si
  Fin Procedimiento

  Procedimiento txtNombre_KeyPress(sender: objeto, e: KeyPressEventArgs)
    Si No (EsLetra(e.KeyChar) O e.KeyChar = 8 O e.KeyChar = 32) Entonces
      e.Handled <- Verdadero
      MostrarMensaje("Solo se permiten letras", "Advertencia", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation)
      Retornar
    Fin Si
  Fin Procedimiento

  Procedimiento btnVolverMenu_Click(sender: objeto, e: EventArgs)
    CerrarFormulario() // Cerrar el formulario actual
  Fin Procedimiento
Fin Algoritmo
```

Algoritmo frmLibrosDisponibles

```
  Procedimiento ConstructorfrmLibrosDisponibles()
    // Inicialización del formulario
    MostrarFormulario()
```

```
Fin Procedimiento

Cadena genero

Procedimiento btnVolverMenu_Click(sender: objeto, e: EventArgs)
    CerrarFormulario()
Fin Procedimiento

Procedimiento btnBuscarLibro_Click(sender: objeto, e: EventArgs)
    LimpiarTexto(txtLibrosDisponibles)
    datosLibros <- DatosListaLibros.lista.mostrarLibros()
    txtLibrosDisponibles.Text <- datosLibros
Fin Procedimiento

Procedimiento btnFiltrar_Click(sender: objeto, e: EventArgs)
    Si ValidarCampos() Entonces
        LimpiarTexto(txtLibrosDisponibles)
        genero <- cboGenero.Text
        datosLibros <- DatosListaLibros.lista.mostrarLibrosGenero(genero)
        txtLibrosDisponibles.Text <- datosLibros
        SeleccionarIndice(cboGenero, -1)
    Sino
        Retornar
    Fin Si
Fin Procedimiento

Función ValidarCampos() retorna booleano
    Si cboGenero.Text = "" Entonces
```

```
        MostrarMensaje("Debe seleccionar un género para filtrar los libros", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        Retornar Falso
      Sino
        Retornar Verdadero
      Fin Si
    Fin Función

    Procedimiento txtTitulo_KeyPress(sender: objeto, e: KeyPressEventArgs)
      Si No (EsLetra(e.KeyChar) O e.KeyChar = 8 O e.KeyChar = 32) Entonces
        e.Handled <- Verdadero
        MostrarMensaje("Solo se permiten letras", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        Retornar
      Fin Si
    Fin Procedimiento
Fin Algoritmo
```

Algoritmo frmPrestamosActivos

```
    Clase estática DatosListaPrestamos
      lista <- Nueva ListaPrestamos
    Fin Clase

    Procedimiento ConstructorfrmPrestamosActivos()
      // Inicialización del formulario
      MostrarFormulario()
    Fin Procedimiento

    Procedimiento btnVolverMenu_Click(sender: objeto, e: EventArgs)
      CerrarFormulario()
```

```
    Fin Procedimiento


    Procedimiento LeerArchivoPrestamos()
        Intentar
            ruta <- "PrestamosActivos.txt"
            lineas <- LeerTodasLasLineasDelArchivo(ruta)
            Para cada linea en lineas Hacer
                datos <- SepararCadenaPorGuion(linea)
                estudiante <- Nuevo Estudiante(datos[0], datos[1], datos[2], datos[3])
                libro <- Nuevo Libro(datos[4], datos[5], datos[6], datos[7], Falso)
                fechaPrestamo <- ConvertirAFecha(datos[8])
                prestamo <- Nuevo Prestamo(estudiante, libro, fechaPrestamo)
                DatosListaPrestamos.lista.agregarPrestamo(prestamo)
            Fin Para
        Atrapar Excepcion ex
            MostrarMensaje("Error al leer el archivo de préstamos activos: " + ex.Mensaje, "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        Fin Intentar
    Fin Procedimiento


    Procedimiento frmPrestamosActivos_Load(sender: objeto, e: EventArgs)
        LeerArchivoPrestamos()
        txtPrestamosActivos.Text <- DatosListaPrestamos.lista.mostrarPrestamos()
    Fin Procedimiento
Fin Algoritmo
```

Algoritmo frmRegistrarPrestamo

```
    Procedimiento ConstructorfrmRegistrarPrestamo()
        // Inicialización del formulario
        MostrarFormulario()
```

```
Fin Procedimiento


Cadena cedula, titulo


Procedimiento btnControlUsuario_Click(sender: objeto, e: EventArgs)
    Si ValidarCampos() Entonces
        LeerArchivoEstudiantes()
        LeerArchivoLibros()
        UsuarioRegistrado()
    Sino
        Retornar
    Fin Si
Fin Procedimiento


Función ValidarCampos() retorna booleano
    Si txtCedula.Text = "" O txtLibro.Text = "" Entonces
        MostrarMensaje("Debe ingresar la cédula del estudiante y el título del libro para poder registrar el préstamo", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        Retornar Falso
    Sino
        Retornar Verdadero
    Fin Si
Fin Función


Procedimiento LeerArchivoEstudiantes()
    Intentar
        ruta <- "EstudiantesRegistrados.txt"
        lineas <- LeerTodasLasLineasDelArchivo(ruta)
        Para cada linea en lineas Hacer
```

```
            datos <- SepararCadenaPorEspacios(linea)
            estudiante <- Nuevo Estudiante(datos[0], datos[1], datos[2], datos[3])
            DatosLista.lista.agregarEstudiante(estudiante)
        Fin Para
    Atrapar Excepcion ex
        MostrarMensaje("No existe un archivo con los estudiantes", "Advertencia",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
    Fin Intentar
  Fin Procedimiento
Procedimiento LeerArchivoLibros
  Intentar
    ruta <- "LibrosIngresados.txt" // Ruta del archivo
    lineas <- LeerTodasLasLineasDelArchivo(ruta)
    Para cada linea en lineas Hacer
      datos <- SepararCadenaPorEspacios(linea)
      libro <- Nuevo Libro(datos[0], datos[1], datos[2], datos[3], Verdadero)
      DatosListaLibros.lista.agregarLibro(libro)
    Fin Para
  Atrapar Excepcion ex
    MostrarMensaje("No existe un archivo con los libros", "Advertencia", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation)
  Fin Intentar
Fin Algoritmo

Procedimiento UsuarioRegistrado()
  cedula <- txtCedula.Text
  titulo <- txtLibro.Text
  Si DatosLista.lista.existeEstudiante(cedula) Y DatosListaLibros.lista.existeLibro(titulo) Y
DatosListaLibros.lista.buscarLibro(titulo).getDisponible() Entonces
    MostrarMensaje("Préstamo Registrado")
```

EscribirArchivo()

DatosListaLibros.lista.buscarLibro(titulo).setDisponible(Falso)

Sino Si No DatosLista.lista.existeEstudiante(cedula) Entonces

MostrarMensaje("Usuario no Registrado", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

Sino Si No DatosListaLibros.lista.existeLibro(titulo) Entonces

MostrarMensaje("Libro no Disponible", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

Sino Si No DatosListaLibros.lista.buscarLibro(titulo).getDisponible() Entonces

MostrarMensaje("Libro no Disponible", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

Fin Si

LimpiarTexto(txtCedula)

LimpiarTexto(txtLibro)

Fin Procedimiento

Al presionar una tecla en el campo de texto "txtCedula":

Si la tecla no es un número o la tecla de retroceso:

Marcar la tecla como manejada (no se escribe el caracter)

Mostrar un mensaje de advertencia: "Solo se permiten números"

Salir del evento

Al presionar una tecla en el campo de texto "txtLibro":

Si la tecla no es una letra, la tecla de retroceso o la tecla de espacio:

Marcar la tecla como manejada

Mostrar un mensaje de advertencia: "Solo se permiten letras"

Salir del evento

Al cargar el formulario "frmRegistrarPrestamo":

No hacer nada

Al hacer clic en el botón "btnVolverMenu":

Cerrar el formulario

Fin algoritmo

Definir clase frmRegistroLibro como Formulario

Definir clase DatosListaLibros como Estática

Definir atributo lista como ListaLibros

Fin Definir

Definir constructor frmRegistroLibro

Inicializar componente

Fin constructor

Definir atributos titulo, nombreAutor, apellidoAutor, genero como Cadena de caracteres

Definir atributo disponible como Booleano

Definir método BtnAgregarLibro al hacer clic en el botón

Si ValidarCampos es verdadero

AgregarDatos

LimpiarTexto

DatosLibros

Mostrar mensaje: "Se ha registrado el libro"

Sino

Salir del método

Fin Si

Fin método

Definir método AgregarDatos

Asignar valor de txtTitulo a titulo

Asignar valor de txtNombreAutor a nombreAutor

Asignar valor de txtApellidoAutor a apellidoAutor

Asignar valor de cboGenero a genero

Crear objeto libro de tipo Libro con los valores de titulo, nombreAutor, apellidoAutor, genero y disponible

Agregar libro a la lista de libros en DatosListaLibros

Fin método

Definir método ValidarCampos

Si txtTitulo está vacío o txtNombreAutor está vacío o cboGenero no está seleccionado

Mostrar mensaje de advertencia: "Debe llenar todos los campos"

Retornar falso

Fin Si

Retornar verdadero

Fin método

Definir método LimpiarTexto

Borrar contenido de txtNombreAutor

Borrar contenido de txtApellidoAutor

Desseleccionar cboGenero

Borrar contenido de txtTitulo

Fin método

Definir método txtTitulo_KeyPress al presionar una tecla

Si la tecla no es una letra o la tecla de retroceso o la tecla de espacio

Marcar la tecla como manejada

Mostrar mensaje de advertencia: "Solo se permiten letras"

Salir del método

Fin Si

Fin método

Definir método txtAutor_KeyPress al presionar una tecla

Si la tecla no es una letra o la tecla de retroceso o la tecla de espacio

Marcar la tecla como manejada

Mostrar mensaje de advertencia: "Solo se permiten letras"

Salir del método

Fin Si

Fin método

Definir método txtApellidoAutor_KeyPress al presionar una tecla

Si la tecla no es una letra o la tecla de retroceso o la tecla de espacio

Marcar la tecla como manejada

Mostrar mensaje de advertencia: "Solo se permiten letras"

Salir del método

Fin Si

Fin método

Definir método btnVolverMenu_Click al hacer clic en el botón

Cerrar el formulario actual

Fin método

Definir método DatosLibros

Definir ruta como Cadena de caracteres con valor "LibrosIngresados.txt"

Si existe el archivo en la ruta especificada

Crear objeto agregar de tipo StreamWriter para agregar información al archivo

Escribir en el archivo los datos del libro recién registrado: titulo + " " + nombreAutor + " " + apellidoAutor + " " + genero

Cerrar el objeto agregar

Sino

Crear objeto escribir de tipo TextWriter para escribir en el archivo

Escribir en el archivo los datos del libro recién registrado: titulo + " " + nombreAutor + " " + apellidoAutor + " " + genero

Cerrar el objeto escribir

Fin Si

Fin método

Fin Algoritmo

Definir clase frmRenovaciones como Formulario

Definir constructor frmRenovaciones

Inicializar componente

Fin constructor

Definir atributo cedula como Cadena de caracteres

Definir método btnVOlver_Click al hacer clic en el botón

Cerrar el formulario actual

Fin método

Definir método txtCedula_KeyPress al presionar una tecla

Si la tecla no es un número o la tecla de retroceso

Marcar la tecla como manejada

Mostrar mensaje de advertencia: "Solo se permiten números"

Salir del método

Fin Si

Fin método

Definir método btnRenovar_Click al hacer clic en el botón

Si ValidarCampos es verdadero

ValidarCedula

```
        ActulizarArchivoPrestamos
        Mostrar mensaje: "Renovación exitosa"
        LimpiarTexto
      Sino
        Salir del método
      Fin Si
    Fin método

    Definir método LeerArchivoPrestamos
      Intentar leer el archivo
      Definir ruta como Cadena de caracteres con valor "PrestamosActivos.txt"
      Leer todas las líneas del archivo en la variable lineas
      Para cada línea en lineas
        Dividir la línea en partes usando el carácter "-"
        Crear objeto estudiante de tipo Estudiante con los valores de las partes 0, 1, 2 y 3
        Crear objeto libro de tipo Libro con los valores de las partes 4, 5, 6, 7 y falso
        Convertir la parte 8 a una fecha y asignarla a fechaPrestamo
        Crear objeto prestamo de tipo Prestamo con estudiante, libro y fechaPrestamo
        Agregar prestamo a la lista de préstamos en DatosListaPrestamos
      Fin Para
      Atrapar cualquier excepción y mostrar mensaje de error
    Fin método
  Definir método ValidarCedula
    LeerArchivoPrestamos // Se leen los préstamos activos
    Asignar valor de txtCedula a cedula // Se almacena la cédula ingresada

    Si existe el préstamo con la cédula en DatosListaPrestamos
      Buscar el préstamo con la cédula en DatosListaPrestamos y asignarlo a prestamo
```

Calcular la diferencia de días entre la fecha actual y la fecha de préstamo y asignarla a difFechas

Si difFechas es mayor a 3

Mostrar mensaje de advertencia: "El préstamo no puede ser renovado, ya que está suspendido"

Sino

Actualizar la fecha de préstamo en prestamo con la fecha actual

Fin Si

Sino

Mostrar mensaje de advertencia: "El préstamo no existe"

Fin Si

Fin método

Definir método ValidarCampos

Si txtCedula está vacío

Mostrar mensaje de advertencia: "Debe ingresar la cédula del estudiante"

Retornar falso

Sino

Retornar verdadero

Fin Si

Fin método

Definir método actulizarArchivoPrestamos

Definir ruta como Cadena de caracteres con valor "PrestamosActivos.txt" // Ruta del archivo

Vaciar el contenido del archivo en la ruta especificada

Crear un nodo auxiliar y asignarlo a aux

Mientras aux no sea nulo

Obtener el préstamo en aux y asignarlo a prestamo

Obtener el estudiante en prestamo y asignarlo a estudiante

Obtener el libro en prestamo y asignarlo a libro

Obtener la fecha de préstamo en prestamo y asignarla a fechaPrestamo

Crear una línea concatenando los datos: nombre del estudiante, apellido del estudiante, cédula del estudiante, carrera del estudiante, título del libro, nombre del autor del libro, apellido del autor del libro y fecha de préstamo

Escribir la línea en el archivo en la ruta especificada, seguido de un salto de línea

Avanzar al siguiente nodo en aux

Fin Mientras

Fin método

Definir método LimpiarTexto

Borrar contenido de txtCedula // Se limpia el campo de cédula

Fin método

Definir método frmRenovaciones_Load al cargar el formulario

No hacer nada

Fin método

Fin Clase

Definir clase frmDevoluciones como Formulario

Definir constructor frmDevoluciones

Inicializar componente

Fin constructor

Definir atributos cedula y titulo como Cadenas de caracteres

Definir método btnVOLVer_Click al hacer clic en el botón

Cerrar el formulario actual

Fin método

Definir método btnDevolver_Click al hacer clic en el botón

```
    Si ValidarCampos es verdadero
        ValidarDevolucion
        LimpiarTexto
    Sino
        Salir del método
    Fin Si
Fin método

Definir método LeerArchivoEstudiantes
    Intentar leer el archivo
    Definir ruta como Cadena de caracteres con valor "EstudiantesRegistrados.txt"
    Leer todas las líneas del archivo en la variable lineas
    Para cada línea en lineas
        Dividir la línea en partes usando el carácter " "
        Crear objeto estudiante de tipo Estudiante con los valores de las partes 0, 1, 2 y 3
        Agregar estudiante a la lista de estudiantes en DatosLista
    Fin Para
    Atrapar cualquier excepción y mostrar mensaje de advertencia
Fin método

Definir método LeerArchivoLibros
    Intentar leer el archivo
    Definir ruta como Cadena de caracteres con valor "LibrosIngresados.txt"
    Leer todas las líneas del archivo en la variable lineas
    Para cada línea en lineas
        Dividir la línea en partes usando el carácter " "
        Crear objeto libro de tipo Libro con los valores de las partes 0, 1, 2, 3 y verdadero
        Agregar libro a la lista de libros en DatosListaLibros
    Fin Para
```

```
        Atrapar cualquier excepción y mostrar mensaje de advertencia
    Fin método
Definir método LeerArchivoLibros
    Intentar leer el archivo
    Definir ruta como Cadena de caracteres con valor "LibrosIngresados.txt"
    Leer todas las líneas del archivo en la variable lineas
    Para cada línea en lineas
        Dividir la línea en partes usando el carácter " "
        Crear objeto libro de tipo Libro con los valores de las partes 0, 1, 2, 3 y verdadero
        Agregar libro a la lista de libros en DatosListaLibros
    Fin Para
    Atrapar cualquier excepción y mostrar mensaje de advertencia
Fin método

Definir método LeerArchivoPrestamos
    Intentar leer el archivo
    Definir ruta como Cadena de caracteres con valor "PrestamosActivos.txt"
    Leer todas las líneas del archivo en la variable lineas
    Para cada línea en lineas
        Dividir la línea en partes usando el carácter "-"
        Crear objeto estudiante de tipo Estudiante con los valores de las partes 0, 1, 2 y 3
        Crear objeto libro de tipo Libro con los valores de las partes 4, 5, 6, 7 y falso
        Convertir la parte 8 a una fecha y asignarla a fechaPrestamo
        Crear objeto prestamo de tipo Prestamo con estudiante, libro y fechaPrestamo
        Agregar prestamo a la lista de préstamos en DatosListaPrestamos
    Fin Para
    Atrapar cualquier excepción y mostrar mensaje de error
Fin método
Definir método ValidarDevolucion
```

Asignar valor de txtCedula a cedula

Asignar valor de txtLibro a titulo

LeerArchivoEstudiantes

LeerArchivoLibros

Si no existe el estudiante con la cédula en DatosLista

Mostrar mensaje de advertencia: "El estudiante no está registrado"

Salir del método

Sino si no existe el libro con el título en DatosListaLibros

Mostrar mensaje de advertencia: "El libro no está registrado"

Salir del método

Sino si no existe el préstamo con la cédula en DatosListaPrestamos

Mostrar mensaje de advertencia: "El estudiante no tiene préstamos activos"

Salir del método

Sino

Eliminar el préstamo con la cédula en DatosListaPrestamos

Actualizar el archivo de préstamos en actulizarArchivoPrestamos

Mostrar mensaje: "Devolución exitosa"

Fin Si

Fin método

Definir método actulizarArchivoPrestamos

Obtener los datos de préstamos en DatosListaPrestamos y asignarlos a datosListaPrestamos

Definir ruta como Cadena de caracteres con valor "PrestamosActivos.txt"

Crear un objeto escribir de tipo TextWriter para escribir en el archivo en la ruta especificada, sobreescribiendo el contenido existente

Si datosListaPrestamos está vacío

Escribir una cadena vacía en el archivo

Cerrar el objeto escribir

```
        Salir del método
    Sino
        Escribir datosListaPrestamos en el archivo, seguido de un salto de línea
        Cerrar el objeto escribir
    Fin Si
Fin método

Definir método ValidarCampos
    Si txtLibro está vacío o txtCedula está vacío
        Mostrar mensaje de advertencia: "Debe llenar todos los campos"
        Retornar falso
    Sino
        Retornar verdadero
    Fin Si
Fin método
Definir método txtCedula_KeyPress al presionar una tecla
    Si la tecla no es un número o la tecla de retroceso
        Marcar la tecla como manejada
        Mostrar mensaje de advertencia: "Solo se permiten números"
        Salir del método
    Fin Si
Fin método

Definir método txtLibro_KeyPress al presionar una tecla
    Si la tecla no es una letra o la tecla de retroceso o la tecla de espacio
        Marcar la tecla como manejada
        Mostrar mensaje de advertencia: "Solo se permiten letras"
        Salir del método
    Fin Si
```

Fin método

Definir método frmDevoluciones_Load al cargar el formulario

No hacer nada

Fin método

Definir método LimpiarTexto

Borrar contenido de txtCedula

Borrar contenido de txtLibro

Fin método

Fin clase

Definir clase frmEstudiantesBaneados

FUNCIÓN frmEstudiantesBaneados()

InicializarComponente()

FIN

FIN FUNCIÓN

FUNCIÓN btnVolverBaneados_Click(sender, e)

Cerrar() // cerrar el formulario

FIN

FIN FUNCIÓN

FUNCIÓN LeerArchivoPrestamos()

INICIO

INTENTAR

ruta <- "PrestamosActivos.txt" // Ruta del archivo

lineas <- LeerTodasLasLíneasDelArchivo(ruta) // Se leen todas las líneas del archivo

```
                PARA CADA línea EN lineas

                    datos <- SepararDatosPorGuion(línea) // Se usa el método Split para separar los datos de cada línea cuando encuentre un guión

                    estudiante <- CrearEstudiante(datos[0], datos[1], datos[2], datos[3]) // Se crea un objeto de tipo estudiante con los datos separados

                    libro <- CrearLibro(datos[4], datos[5], datos[6], datos[7], FALSO) // Se crea un objeto de tipo libro con los datos separados

                    fechaPrestamo <- ConvertirAFecha(datos[8]) // Se convierte la fecha de string a DateTime

                    prestamo <- CrearPrestamo(estudiante, libro, fechaPrestamo) // Se crea un objeto de tipo préstamo con los datos separados

                    AgregarPrestamoALista(prestamo) // Se agrega el objeto a la lista de préstamos

                FIN PARA

            ATRAPAR EXCEPCIÓN ex

                MostrarMensajeDeError("Error al leer el archivo de préstamos activos: " + ex.Mensaje, "Error")

            FIN INTENTAR

        FIN

    FIN FUNCIÓN


    FUNCIÓN frmEstudiantesBaneados_Load(sender, e)

        INICIO

            LeerArchivoPrestamos()

            txt.Sancionados.Texto <- MostrarSancionadosDeLista()

        FIN

    FIN FUNCIÓN

  FIN

 FIN CLASE

FIN
```