Universidad Nacional de Río Cuarto
Facultad de Cs. Exactas, Fco-Qcas y Naturales - Dpto de Computación
Asignatura: **Introducción a la Algorítmica y Programación** (Cód . 3300)
Año 2022
Integrantes:
· ***Celina Gillo Mayer***
· ***Joaquín Pablo Tissera***

Proyecto Final

Sea enunciado el siguiente problema:

> Una estación meteorológica registra a diario una serie de datos como la temperatura máxima, la mínima, la humedad media, la presión atmosférica, la dirección predominante del viento y su velocidad, también la precipitación acumulada. Estos valores se obtienen a diario. La estación analiza la información para obtener series de datos que le permitan alimentar modelos de predicción del tiempo.

Nos solicitan que desarrollemos un programa para poder almacenar los registros diarios y procesar los valores almacenados para la elaboración de los pronósticos del tiempo.

Para el tratamiento de los datos el programa a desarrollar debe poder crear y mantener un archivo con los registros diarios. Los datos de un mismo año están en un único archivo.

El programa a desarrollar deberá presentarle al usuario un menú de opciones con las siguientes directivas:

1. Dar de alta un registro diario
2. Suprimir un registro diario
3. Modificar un registro, buscando por la fecha
4. Mostrar todos los registros activos
5. Buscar registro de un día específico, mostrando todos los parámetros
6. Listar el o los días de máxima temperatura en lo que va del año
7. Listar el o los días de máxima precipitación en lo que va del año
8. Listar las fechas de mayor a menor velocidad del viento
9. Realizar una copia de seguridad del archivo del año en curso
10. Salir

En dónde:
· Las opciones de alta, suprimir, modificar y mostrar se ejecutarán directamente sobre el archivo. La opción suprimir consiste en aplicar borrado lógico.
· Para la **búsqueda** de datos de un día dado, se pasarán los registros diarios del archivo a un arreglo y la búsqueda se realizará sobre el arreglo. Para buscar se debe implementar una función que devuelva el índice del arreglo dónde se encuentra el registro buscado y sino está que devuelva -1. Esta función debe ser recursiva.
· Para listar el día o días con **máxima temperatura**, se creará una lista (Lista Simplemente Encadenada) de registros con la misma máxima temperatura (en caso que haya más de un día con la misma máxima temperatura) y se mostrará las fechas de los días en que ocurrieron esas máximas temperaturas.

· Para listar el día o días con **máxima precipitación**, se ordenará el arreglo por **precipitaciones** de menor a mayor (usar un método estable) y se informarán las fechas y las precipitaciones correspondientes de los diez primeros días de mayores precipitaciones ordenados de mayor a menor.
· Para listar las fechas de mayor a menor **velocidad de viento**, se pasarán los registros diarios del archivo a un arreglo y se ordenará por velocidades del viento mayor a menor (usar un método estable distinto al anterior) y se informarán las fechas y el viento correspondiente de los diez primeros días de mayores velocidades ordenados de mayor a menor.

Para el manejo de **arreglos** y de **listas** deberá implementar las acciones y funciones desarrolladas a lo largo del año (función Vacia(), función Llena(), acción Insertar(), acción Ordenar(), acción Mostrar(), etcétera).

La opción de **copia de seguridad** consiste en crear un archivo nuevo (con otro nombre) con sólo los registros no borrados del archivo original.

A cumplir:
· Para cada opción del menú se debe diseñar un módulo adecuado.

· Todas las opciones del menú deben efectuarse sobre registros activos, los borrados no deben ser utilizados.

· El programa comenzará solicitando el nombre del archivo con el cuál va a trabajar, si el archivo no existe lo creará.

Se hace entrega bajo las siguientes condiciones:

● El proyecto final debe resolverse en grupos de 2 alumnos. Se aceptarán grupos de 3 alumnos en caso que sea debidamente justificado. La fecha límite de conformación de grupos es el 07 de Noviembre.

● Los días 07, 09 y 14 de Noviembre se dispondrán para consultas generales sobre el proyecto final, en los horarios habituales de clases prácticas. En las clases del 07 y el 09 los grupos deberán mostrar los Diseños (algoritmos) que dan solución al problema planteado.

● **Consultas en Laboratorio**: 09 Noviembre de 11 a 13 y 16 a 18hs en el Laboratorio 101 del Pab 2 (UNRC). Por EVELIA se informarán otros días de consulta posibles.

● **IMPORTANTE**: Quienes no formen parte de algún grupo a la fecha límite serán considerados en condición de libre.

● El tiempo estimado de realización del Trabajo Práctico Integrador es de aproximadamente 20 hs en total.

● El proyecto final debe incluir:
    ○ 1) Un documento en formato PDF con el Diseño (algoritmo).
    ○ 2) Un archivo de texto con la Implementación (programa fuente en C).

● El proyecto final se entregará mediante la Sección de Actividades del aula virtual.
    ○ Se fija el lunes 14 de Noviembre como fecha límite para la entrega del algoritmo (Diseño) con la resolución del proyecto. El algoritmo debe respetar las convenciones y estar debidamente comentado.
    ○ Se fija el miércoles 16 de noviembre como fecha límite para la entrega del código fuente (segunda parte) con la resolución del proyecto. El código fuente debe respetar las convenciones y estar debidamente comentado.
    ○ El responsable del grupo será el encargado de adjuntar el algoritmo (Diseño) y el código fuente (implementación) en el aula virtual.

● El proyecto es obligatorio y debe aprobarse (igual que un parcial). No tiene recuperatorio.

● El grupo realizará defensa del proyecto en caso que la cátedra lo considere necesario (en fecha a definir en cada caso).

A continuación presentaremos el Algoritmo en pseudocódigo:

```
Algoritmo EstaciónMetereológica
Léxico
// Constantes y tipos estructurados
   Max = 1000
   regdiario = < ddmmyyyy e Z, tmax e Z, tmin e Z, HUM e (0..100), PNM e Z, DV e
(0..360), FF e Z, PP e (0..1000), borrado e Lógico >
   TData =< a e arreglo de [1...Max] de regdiario, cant e (0...Max)>
   Tnodo = < info e regdiario, next e puntero a Tnodo>

// Declaración de funciones y acciones del léxico global
/* Chequea si un archivo está vacío, ya sea por EOF o porque todos los registros están
borrados */
   Función ArchVacio(dato nom e Cadena) → Lógico
   Léxico local
      f e Archivo de regdiario
      r e regdiario
   Inicio
      Abrir(nom, f, l)
      Si ( EOF(f) ) entonces
         Cerrar(f)
         ← Verdadero
      sino
         Mientras ( no ( EOF(f) ) ) hacer
            Leer(f, r)
            Si ( no( r.borrado ) ) entonces  // Apenas haya alguno presente, sale falso
               Cerrar(f)
               ← Falso
```

```
            Fsi
        Fmientras
        Si ( EOF(f) ) entonces
            Cerrar(f)
            ← Verdadero
        Fsi
    Fsi
Ffunción

/* Le pide el año el usuario. Usada en las opciones que se requiera para que coincida con
los años que ya estén cargados y no avanzar en vano. */
Acción PedirAño(resultado año e Z)
Léxico local
    msg e Cadena
Inicio
    msg ← "Año: "
    Salida: msg
    Entrada: año
    // Def del año
    Si ( año < 1000 o año > 9999 ) entonces
        Repetir
            msg ← "Ingresar un año válido"
            Salida: msg
            Entrada: año
        Hasta que ( año >= 1000 y año <= 9999 )
    Fsi
Facción

// Recibe el año como dato y dentro de la acción pide el día y el mes.
// Convierte los tres datos individuales en uno solo que corresponderá al formato ddmmyyyy.
Acción PedirFecha(dato aA e Z resultado fecha e Z)
Léxico local
    Función EsBisiesto(checkA e Z) → Lógico
    Inicio
        ← ( checkA mod 4 = 0 y checkA mod 100 <> 0 ) o ( checkA mod 100 = 0 y checkA
mod 400 = 0 )
    Ffunción
    msg e Cadena
    dA, mA e Z
Inicio
    msg ← "Mes: "
    Salida: msg
    Entrada: mA
    // Def de mes
    Si ( mA < 1 o mA > 12 ) entonces
        Repetir
            msg ← "Ingresar un mes válido"
            Salida: msg
```

```
            Entrada: mA
        Hasta que ( mA >= 1 y mA <= 12 )
    Fsi
    msg ← "Día: "
    Salida: msg
    Entrada: dA
    // Def de días
    Si ( dA < 1 o dA > 31 ) entonces
        Repetir
            msg ← "Ingresar un día válido"
            Salida: msg
            Entrada: dA
        Hasta que ( dA >= 1 y dA <= 31 )
    Fsi
    Según
        ( EsBisiesto(aA) y mA = 2 y dA > 29 ) :
            Repetir
                msg ← "Ingresar un día de febrero válido. El año ingresado es bisiesto: "
                Salida: msg
                Entrada: dA
            Hasta que ( dA >= 1 y dA <= 29 )
        ( no( EsBisiesto(aA) ) y mA = 2 y dA > 28 ) :
            Repetir
                msg ← "Ingresar un día de febrero válido. El año ingresado no es bisiesto: "
                Salida: msg
                Entrada: dA
            Hasta que ( dA >= 1 y dA <= 28 )
        ( (mA=4 o mA=5 o mA=6 o mA=9 o mA=11) y dA = 31 ) :
                Repetir
                    msg ← "Los meses 4, 5, 6, 9 y 11 tienen 30 días. Ingresar un día válido: "
                    Salida: msg
                    Entrada: dA
                Hasta que ( dA >= 1 y dA <= 30 )
    Fsegún
    fecha ← dA * 1000000 + mA * 10000 + aA
Facción

// Función que recibe como parámetro un archivo y devuelve un arreglo
Función ArchArray(dato nom e Cadena) → TData
Léxico local
    registro e TData
    reg e regdiario
    i e Z
    f e Archivo de regdiario
Inicio
    Abrir(nom, f, l)
    Según
        (ArchVacio(nom)) :
```

```
            registro.cant ← 0
        ( no( ArchVacio(nom) ) ):
            i ← 1 // inicial. trat
            Mientras ( no( EOF(f) ) y i <= Max ) hacer
                Leer(f, reg)
                registro.a[i] ← reg
                i ← i+1
            Fmientras
            registro.cant ← i
    Fsegún
    Cerrar(f)
    ← registro
Ffunción
```

/* Recibe el nombre externo del archivo y el año que quiere cargar. Si la fecha que puso ni bien arrancar la acción no está cargada o está borrada continúa con la carga. Sino vuelve al menú. */

```
Acción Alta(dato añoEntry e Z, nom e Cadena)
Léxico local
    Acción PedirTemp(resultado tM e Z)
    Léxico local
        msg e Cadena
    Inicio
        Entrada: tM
        Si ( tM < -100 o tM > 100 ) entonces
            Repetir
                msg ← "Ingresar una temperatura válida. Entre -100 y 100 grados centígrados."
                Salida: msg
                Entrada: tM
            Hasta que ( tM >= -100 y tm <= 100 )
        Fsi
    Facción
    Acción PedirPresAtm(resultado prA e Z)
    Léxico local
        msg e Cadena
    Inicio
        Entrada: prA
        Si ( prA < 900 o prA > 3500 ) entonces
            Repetir
                msg ← "Ingresar una presión atmosférica válida. Entre 900 y 3500."
                Salida: msg
                Entrada: prA
            Hasta que ( prA >= 900 y prA <= 3500 )
        Fsi
    Facción
    Acción PedirVelocViento(resultado wA e Z)
    Léxico local
        msg e Cadena
```

```
    Inicio
      Entrada: wA
      Si ( wA <= 0 ) entonces
        Repetir
          msg ← "Ingresar una velocidad de viento válida. Debe ser positivo."
          Salida: msg
          Entrada: wA
        Hasta que ( wA > 0 )
      Fsi
    Facción
    fA e Archivo de regdiario
    rA, captReg e regdiario
    iguales e Lógico
    msg e Cadena
  Inicio  // De la acción Alta
    Abrir(nom, fA, a)
    PedirFecha(añoEntry, rA.ddmmyyyy)
    IrPos(fA, 0)
    iguales ← Falso
    // Recorro hasta llegar al final o hasta encontrar que sean iguales. Si ya hay uno igual
pero está borrado, lo ignora y lo cargará sin problemas
    Mientras ( no ( EOF(fA) ) y no ( iguales ) ) hacer
      Leer (fA, captReg)
      iguales ← ( rA.ddmmyyyy = captReg.ddmmyyyy y no(captReg.borrado) )
    Fmientras
    Si ( no ( iguales ) ) entonces  // No hay una fecha igual, puede continuar
      msg ← "Ingresar la temperatura máxima de la fecha"
      Salida: msg
      PedirTemp(rA.tmax)
      msg ← "Ingresar la temperatura mínima de la fecha"
      Salida: msg
      PedirTemp(rA.tmin)
      msg ← "Ingresar el porcentaje de humedad promedio de la fecha"
      Salida: msg
      Entrada: rA.HUM
      msg ← "Ingresar el promedio de la presión atmosférica de la fecha"
      Salida: msg
      PedirPresAtm(rA.PNM)        // entre 900 y 3500
      msg ← "Ingresar la dirección del viento de la fecha"
      Salida: msg
      Entrada: rA.DV
      msg ← "Ingresar la velocidad máxima del viento de la fecha"
      Salida: msg
      PedirVelocViento(rA.FF)       // positivo
      msg ← "Ingresar la precipitación pluvial acumulado el día de la fecha"
      Salida: msg
      Entrada: rA.PP
      rA.borrado ← Falso
```

```
      Escribir(fA, rA)
    sino
      msg ← "Ya existe un registro presente con la fecha indicada. Vuelve al menú."
      Salida: msg
    Fsi
    Cerrar(fA)
  Facción
```

/* Al invocarse se pregunta si el archivo está vacío o si todos los registros están borrados. En caso afirmativo no hay nada para suprimir. En caso negativo le pregunta si quiere suprimir: Si quiere entonces le pide la fecha que quiere borrar. Si la encuentra pone el borrado lógico en verdadero, sino le avisa que no se encontró y vuelve al menú princpal. Si no quiere suprimir vuelve al menú principal. */

```
  Acción Suprimir(dato añoEntry e Z, nom e Cadena)
  Léxico local
    // Devuelve V si encuentra la fecha y está presente. sino devuelve F.
    Función PropEC(dato regArch e regdiario, fechaPedida e Z) → Lógico
    Inicio
      ← ( regArch.ddmmyyyy = fechaPedida y no( regArch.borrado ) )
    Ffunción
    fecha e Z
    rA e regdiario
    msg e Cadena
    rta, enc e Lógico
    f e ARCHIVO de regdiario
  Inicio
    Abrir(nom, f, a)
    IrPos(f, 0)
    Si ( ArchVacio(nom) ) entonces
      msg ← "No se puede suprimir porque el archivo está vacío o todos los registro están
borrados. Vuelve al menú principal."
      Salida: msg
    sino
      msg ← "¿Desea suprimir un registro?"
      Salida: msg
      Entrada: rta
      Si ( rta ) entonces
        msg ← "¿De qué fecha es el registro que quiere suprimir?"
        Salida: msg
        PedirFecha(añoEntry, fecha)        // fecha quedo con formato ddmmyyyy
        Repetir
          Leer (f, rA)
        Hasta que ( EOF(f) o PropEC(rA, fecha) )
        Si ( PropEC(rA, fecha) ) entonces
          rA.borrado ← Verdadero
          IrPos(f, PosicionActual(f)-1)
          Escribir(f, rA)
          msg ← "Suprimido con éxito"
```

```
            Salida: msg
        sino
            msg ← "No se encontró la fecha para suprimir"
            Salida: msg
        Fsi
    sino
        msg ← "No se quiso suprimir nada. Vuelve al menú principal"
        Salida: msg
    Fsi
  Fsi
  Cerrar(f)
Facción

// Modificar un registro, buscando por la fecha
Acción ModificarRegistro(dato añoEntry e Z, nom e cadena)
Lexico local
  Acción PedirTemp(resultado tM e Z)
  Léxico local
    msg e Cadena
  Inicio
    Entrada: tM
    Si ( tM < -100 o tM > 100 ) entonces
      Repetir
        msg ← "Ingresar una temperatura válida. Entre -100 y 100 grados centígrados."
        Salida: msg
        Entrada: tM
      Hasta que ( tM >= -100 y tm <= 100 )
    Fsi
  Facción
  Acción PedirPresAtm(resultado prA e Z)
  Léxico local
    msg e Cadena
  Inicio
    Entrada: prA
    Si ( prA < 900 o prA > 3500 ) entonces
      Repetir
        msg ← "Ingresar una presión atmosférica válida. Entre 900 y 3500."
        Salida: msg
        Entrada: prA
      Hasta que ( prA >= 900 y prA <= 3500 )
    Fsi
  Facción
  Acción PedirVelocViento(resultado wA e Z)
  Léxico local
    msg e Cadena
  Inicio
    Entrada: wA
    Si ( wA <= 0 ) entonces
```

```
            Repetir
               msg ← "Ingresar una velocidad de viento válida. Debe ser positivo."
               Salida: msg
               Entrada: wA
            Hasta que ( wA > 0 )
         Fsi
      Facción
      // Devuelve V si encuentra la fecha y está presente. sino devuelve F.
      Función PropEC(dato reg e regdiario, fechaPedida e Z) → Lógico
      Inicio
         ← ( reg.ddmmyyyy = fechaPedida y no( reg.borrado ) )
      Ffunción
      fecha e Z
      f e ARCHIVO de regdiario
      msj e Cadena
      regMod, regArch e regdiario
   Inicio
      Abrir(nom, f, l)
      Si ( ArchVacio(nom) ) entonces
         msj ← "No se puede modificar porque el archivo está vacío o todos los registros
están borrados. Vuelve al menú principal."
         Salida: msj
      sino
         msj ← "¿De qué fecha es el registro que quiere modificar?"
         Salida: msj
         PedirFecha(añoEntry, fecha)
         Repetir
            Leer(f, regArch)
         Hasta que ( EOF(f) o PropEC(regArch, fecha)
         Según:
            ( PropEC(regArch, fecha) ) :
               IrPos(f, PosiciónActual(f)-1)
               // Modifico todo menos la fecha ni el borrado lógico
               msj ← "Ingresar la temperatura máxima de la fecha"
               Salida: msj
               PedirTemp(regMod.tmax)
               msj ← "Ingresar la temperatura mínima de la fecha"
               Salida: msj
               PedirTemp(regMod.tmin)
               msj ← "Ingresar el porcentaje de humedad promedio de la fecha"
               Salida: msj
               Entrada: regMod.HUM
               msj ← "Ingresar el promedio de la presión atmosférica de la fecha"
               Salida: msj
               PedirPresAtm(regMod.PNM)     // entre 900 y 3500
               msj ← "Ingresar la dirección del viento de la fecha"
               Salida: msj
               Entrada: regMod.DV
```

```
                msj ← "Ingresar la velocidad máxima del viento de la fecha"
                Salida: msj
                PedirVelocViento(regMod.FF)   // positivo
                msj ← "Ingresar la precipitación pluvial acumulado el día de la fecha"
                Salida: msj
                Entrada: regMod.PP
                regMod.ddmmyyyy ← fecha
                regMod.borrado ← Falso
                Escribir(f, regMod)
                msj ← “Se modificó con éxito”
                Salida: msj
            ( no( PropEC(regArch, fecha) ) ) :
                Si ( regArch.ddmmyyyy = fecha y regArch.borrado ) entonces
                    msj ← “La fecha a modificar está borrada”
                    Salida: msj
                sino
                    msj ← “No se encontró la fecha a modificar”
                    Salida: msj
                Fsi
            Fsegun
        Fsi
        Cerrar(f)
    Faccion

// Mostrar todos los registros activos del archivo.
    Acción MostrarArchivo(dato nom e cadena)
    Lexico local
        msj e Cadena
        reg e regdiario
        i e Z
        f e ARCHIVO de regdiario
    Inicio
        Abrir(nom, f, l)
        Según
            ( ArchVacio(nom) ) :
                msj <- “El archivo está vacío o todos sus registros están borrados.”
                Salida: msj
            no( ArchVacio(nom) ) :
                i ← 1
                Mientras ( no ( EOF(f) ) ) hacer
                    Leer(f, reg)
                    Si ( no( reg.borrado ) ) entonces
                        msj ← "Registro nro: "
                        Salida: msg i
                        i ← i + 1
                        Salida: reg
                    Fsi
                Fmientras
```

```
    Fsegún
  Cerrar(f)
Facción
```

/* Buscar la posición de un registro en un arreglo (según una fecha dada) de manera recursiva. El TData querecibe la función ya contiene los registros de un archivo, el i ingresa con 1 modelando una primera supuesta posición, el parámetro fecha1 recibe la fecha que se quiere buscar. */

```
Función BuscaPosRecursiva(dato registro e TData, i e Z, fecha1 e Z) → Z
{Pre:  i = 1 }
Inicio
  Según:
    ( i = registro.cant + 1 ) :
      pos ← -1
    ( i < registro.cant + 1 ) :
      Si ( registro.a[i].ddmmyyyy = fecha1 ) entonces
        ← i
      sino
        ← BuscaPosRecursiva(registro, i+1, fecha1)
      Fsi
  Fsegún
Ffunción
```

/* Acción que necesita una posición (que se consigue mediante la invocación de la función recursiva previamente implementada), y el arreglo que contiene los registros del archivo. */

```
Acción MostrarDatos_Pos (dato posicion e Z, registro e TData)
Léxico local
  msj e Cadena
Inicio
  Si ( posicion = -1 ) entonces
    msj ← "No se encontró la fecha"
    Salida:msj
  sino
    // Muestra todo menos la fecha y el borrado lógico
    Salida: registro.a[posicion].tmax
    Salida: registro.a[posicion].tmin
    Salida: registro.a[posicion].HUM
    Salida: registro.a[posicion].PNM
    Salida: registro.a[posicion].DV
    Salida: registro.a[posicion].FF
    Salida: registro.a[posicion].PP
  Fsi
Facción

Acción VaciarListaConFict(dato q e puntero a Tnodo)
Léxico local
  aux, aux2 e puntero a Tnodo
Inicio
```

```
    aux ← (^q).next
    aux2 ← aux
    (^q).next ← nil
    Si ( aux <> nil ) entonces
      Mientras ( aux <> nil ) hacer
        aux ← (^aux).next
        Liberar(aux2)
        aux2 ← aux
      Fmientras
    Fsi
  Facción

  Acción MostrarLSETempMax(dato q e puntero a Tnodo)
  Inicio
    Mientras ( (^q).next <> nil ) hacer
      q ← (^q).next
      Salida: (^q).info.ddmmyyyy
      Salida: (^q).info.tmax
    Fmientras
  Facción

/* Recibe el nombre externo del archivo y devuelve una lista con los registros de mayor
temperatura. */
  Acción ListarTempMax(dato nom e cadena, resultado s e puntero a Tnodo)
  Léxico local
    msg e Cadena
    f e Archivo de regdiario
    aux, nuevo e puntero a Tnodo
    maxima, reg e regdiario
  Inicio
    Abrir(nom, f, l)
    Obtener(s)
    (^s).next ← nil
    Según
      (ArchVacio(nom)):
        msg ← "El archivo está vacío"
        Salida: msg
      (no(ArchVacio(nom))):
        Repetir
          Leer(f, reg)
        Hasta que ( no(reg.borrado) )
        // Voy hasta el primero presente. Sí o sí hay uno presente por el no(ArchVacio)
        maxima ← reg
        Obtener(nuevo)
        (^nuevo).info ← maxima
        (^nuevo).next ← nil
        (^s).next ← nuevo
        Mientras ( no(EOF(f)) ) hacer
```

```
            Leer(f, reg)
            Según
               ( maxima.tmax < reg.tmax y no(reg.borrado) ) :
                  maxima ← reg
                  VaciarListaConFict(s)
                  Obtener(s)
                  (^s).next ← nil
                  Obtener(nuevo)
                  (^nuevo).info ← maxima    // maxima.ddmmyyyy y maxima.tmax
                  (^nuevo).next ← nil
                  (^s).next ← nuevo
               ( maxima.tmax = reg.tmax y no(reg.borrado) ) :
                  Obtener(aux)
                  (^aux).info ← reg
                  (^aux).next ← nil
                  (^nuevo).next ← aux
                  nuevo ← aux
            Fsegún
         Fmientras
      Fsegún
      Cerrar(f)
   Facción
```

/* Recibe el nombre externo del archivo y si tiene datos no borrados los pasa a un arreglo. Ordena de mayor a menor con InsertionSort y muestra hasta 10 como máximo.*/

```
   Accion ListarPrecMax(dato nom e cadena)
   Léxico local
      arregloPrecip e TData
      i, aux e Z
      msj e Cadena
      f e ARCHIVO de regdiario
   Inicio
      Abrir(nom, f, l)
      Según
         ( ArchVacio(nom) ) :
            msj← "El archivo está vacío"
            Salida: msj
         ( no( ArchVacio(f) ) ) :
            arregloPrecip ← ArchArray(nom)
            OrdenarInsertionReves(arregloPrecip)
            aux ← 10
            Para ( i ← 1 , i <= aux y i <= arregloPrecip.cant , i ← i + 1 ) hacer
               Si ( no( arregloPrecip.borrado ) ) entonces
                  Salida: arregloPrecip[i].ddmmyyyy
                  Salida: arregloPrecip[i].PP
               sino
                  aux ← aux + 1
               Fsi
```

```
            Fpara
        Fsegún
        Cerrar(f)
    Facción
```

/* Recibe el nombre externo del archivo y si tiene cosas las pasa a un arreglo. Usa BubbleSort al revés (ordena de mayor a menor) y muestra hasta 10 fechas y vientos como máximo registrados.*/

```
    Acción ListarMayMenVelViento(dato nom e cadena)
    Léxico local
        arregloVientos e TData
        fA e Archivo de regdiario
        msg e Cadena
        i, aux e Z
    Inicio
        Abrir(nom, fA, l)
        Si ( ArchVacio(nom) ) entonces
            msg ← "El archivo está vacío"
            Salida: msg
        sino
            arregloVientos ← ArchArray(nom)
            OrdenarBubbleReves(arregloVientos)
            aux ← 10
            Para ( i ← 1 , i <= aux y i <= arregloVientos.cant , i ← i + 1 ) hacer
                Si ( no( arregloVientos.a[i].borrado ) ) entonces
                    Salida: arregloVientos.a[i].ddmmyyyy
                    Salida: arregloVientos.a[i].FF
                sino
                    aux ← aux + 1
                Fsi
            Fpara
        Fsi
        Cerrar(fA)
    Facción
```

/* Toma el archivo que se está trabajando. Revisa si está vacío o todos borrados lógicamente, si eso pasa no se puede hacer la copia ya que justamente consiste en crear un archivo nuevo con sólo los registros presentes. Sino lo creará preguntándole el nuevo nombre al usuario y concatenándole la hora para que se pueda distinguir con facilidad. */

```
    Acción CopiaSeguridad(dato nom e Cadena)
    Léxico local
```

/* Función que simula devolver la hora justa en la que se invocó en formato Dia Mes NroDia hh:mm:ss Año, auxiliar para concatenar el nombre del archivo de la copia */

```
        Función HoraActual () → Cadena
        gA, fOld e Archivo de regdiario
        nomArc e Cadena
        msg, hora e Cadena
        rA e regdiario
```

```
    rta e Lógico
  Inicio
    Abrir(nom, fOld, l)
    Si ( ArchVacio(nom) ) entonces
      msg ← "El archivo el cual se le realizará la copia de seguridad aún no tiene datos
almacenados"
      Salida: msg
    sino
      msg ← "¿Realizar copia de seguridad?"
      Salida: msg
      Entrada: rta
      Si ( rta ) entonces
        msg ← "Ingresar el nombre del archivo para la copia de seguridad (sin el .dat). Se
concatenará la hora en que se realizó luego del nombre."
          Salida: msg
          Entrada: nomArc
          hora ← HoraActual()
          nomArc ← nomArc + "_" + hora + ".dat"
          Abrir(nomArc, gA, e)
          Mientras ( no( EOF(fOld) ) ) hacer
            Leer (fOld, rA)
            Si ( no ( rA.borrado ) ) entonces    // Escribo en el archivo destino todo lo que no
esté borrado del archivo origen
              Escribir (gA, rA)
            Fsi
          Fmientras
          Cerrar(gA)
          msg ← "Copia de seguridad realizada con éxito. El archivo se llama: "
          Salida: msg nomArc
        sino
          msg ← "Vuelve al menú principal"
          Salida: msg
        Fsi
    Fsi
   Cerrar(fOld)
  Facción

/* Acción que se invoca ni bien aranca el programa; si(archivo no existe) entonces lo crea
sino lo abre y lo cierra. Verificará que el archivo realmente exista luego de finalizarse la
acción. */
  Acción Verificar(dato nom e Cadena, dato/resultado fA e Archivo de regdiario)
  Léxico local
    msg e Cadena
  Inicio
    Abrir(nom, fA, a)
    IrPos(fA, 0)
    Si ( EOF(fA) ) entonces
      msg ← "El archivo " + nom + " fue creado."
```

```
        Salida: msg
      sino
        msg ← "Archivo " + nom + " abierto exitosamente."
        Salida: msg
      Fsi
      Cerrar(f)
    Facción

// Para precipitaciones, ordena de mayor a menor
    Acción OrdenarInsertionReves (dato/resultado arr e TData)
    Léxico local
      i, j e Z
      aux e regdiario
    Inicio
      Para ( i ← 2 , i <= arr.cant , i ← i + 1 ) hacer
        aux ← arr.a[i]
        Para ( j ← i - 1 , j > 0 y arr.a[j].PP < aux.PP , j ← j - 1 ) hacer
          arr.a[j+1] ← arr.a[j]
        Fpara
        arr.a[j+1] ← aux
      Fpara
    Facción

// Para velocidad de vientos, ordena de mayor a menor
    Acción OrdenarBubbleReves (dato/resultado a e TData)
    Léxico local
      Acción Intercambiar (dato/resultado x, y e regdiario)
      Léxico local
        tmp e regdiario
      Inicio
        tmp ← x
        x ← y
        y ← tmp
      Facción
      i, j e Z
    Inicio
      Para ( i ← 1 , i <= a.cant , i ← i + 1 ) hacer
        Para ( j ← 1 , j <= a.cant - i - 1 , j ← j + 1 ) hacer
          Si ( a.a[j].FF < a.a[j+1].FF ) entonces
            Intercambiar(a.a[j], a.a[j+1])
          Fsi
        Fpara
      Fpara
    Facción

    Acción DesplegarMenuDeOpciones ()
    Léxico local
      msg ϵ Cadena
```

Inicio
msg ← "· Menú principal
Seleccione una opción para continuar:
1. Alta de un registro diario
2. Suprimir un registro diario
3. Modificar un registro, se busca por la fecha
4. Mostrar todos los registros activos
5. Buscar registro de un día dado y mostrar todos los parámetros
6. Listar el día o días de máxima temperatura en lo que va del año
7. Listar el día o días de máxima precipitación en lo que va del año
8. Listar las fechas de mayor a menor velocidad de viento
9. Realizar una copia de seguridad del archivo del año en curso
10. Salir"
Salida: msg
Facción

// Declaración de variables globales
capturarReg e regdiario
arregloPos e TData
listaTempMax e puntero a Tnodo
f e Archivo de regdiario
nombreArchivo, msg e Cadena
opcionEntry, añoPedido, pos, obtDia, obtMes, fechaBuscar e Z
Inicio // Del algoritmo principal
msg ← "Inicio del programa. A continuación se manejará un menú de opciones para el modelo de una estación metereológica. Por favor, inserte el nombre del archivo que usará para el programa (sin el .dat)"
Salida: msg
Entrada: nombreArchivo
nombreArchivo ← nombreArchivo + ".dat"
Verificar(nombreArchivo, f)
Repetir
DesplegarMenuDeOpciones()
Entrada: opcionEntry
Según
(opcionEntry = 1) :
msg ← "Opción 1: Alta de un registro diario"
Salida: msg
msg ← "Ingresar el año correspondiente al registro. En caso que el archivo ya tenga elementos cargados, debe coincidir con el año de los que ya estén cargados en el archivo"
Salida: msg
PedirAño(añoPedido)
Si (ArchVacio(nombreArchivo)) entonces
// Está vacío, no hay problema con poner cualquier año
Alta(añoPedido, nombreArchivo)
sino
// No está vacío. Reviso si los años coinciden:

```
                Abrir(nombreArchivo, f, l)
                Leer(f, capturarReg)
                Cerrar(f)
                obtDia ← capturarReg.ddmmyyyy div 1000000
                obtMes ← capturarReg.ddmmyyyy div 10000 - (obtDia * 100)
                obtAño ← capturarReg.ddmmyyyy - (obtDia*1000000) - (obtMes*10000)
                Si ( obtAño = añoPedido ) entonces
                   Alta(añoPedido, nombreArchivo)
                sino
                   msg ← "El año no coincide con los registros que ya están en el archivo"
                   Salida: msg
                Fsi
             Fsi
          ( opcionEntry = 2 ) :
             msg ← “Opción 2: Suprimir un registro diario”
             Salida: msg
             Si ( ArchVacio(nombreArchivo) ) entonces
                // Está vacío o todo suprimido.
                msg ← "El archivo está vacío o todos sus registros ya están borrados. Vuelve al
menú principal."
                Salida: msg
             sino
                msg ← "Ingresar el año correspondiente al registro que quiere suprimir. Debe
coincidir con el año de los que ya estén cargados en el archivo"
                Salida: msg
                PedirAño(añoPedido)
                Abrir(nombreArchivo, f, l)
                Leer(f, capturarReg)
                Cerrar(f)
                obtDia ← capturarReg.ddmmyyyy div 1000000
                obtMes ← capturarReg.ddmmyyyy div 10000 - (obtDia * 100)
                obtAño ← capturarReg.ddmmyyyy - (obtDia*1000000) - (obtMes*10000)
                Si ( obtAño = añoPedido ) entonces
                   Suprimir(añoPedido, nombreArchivo)
                sino
                   msg ← "El año no coincide con los registros que ya están en el archivo"
                   Salida: msg
                Fsi
             Fsi
          ( opcionEntry = 3 ) :
             msg ← “Opción 3: Modificar un registro, se busca por la fecha”
             Salida: msg
             Si ( ArchVacio(nombreArchivo) ) entonces
                // Está vacío o todo suprimido.
                msg ← "El archivo está vacío o todos sus registros están borrados. Vuelve al
menú principal."
                Salida: msg
             sino
```

```
            // Hay cosas presentes en el archivo, se puede proceder a modificar.
            msg ← "Ingresar el año correspondiente al registro que quiere modificar. Debe
coincidir con el año de los que ya estén cargados en el archivo"
            Salida: msg
            PedirAño(añoPedido)
            Abrir(nombreArchivo, f, l)
            Leer(f, capturarReg)
            Cerrar(f)
            obtDia ← capturarReg.ddmmyyyy div 1000000
            obtMes ← capturarReg.ddmmyyyy div 10000 - (obtDia * 100)
            obtAño ← capturarReg.ddmmyyyy - (obtDia*1000000) - (obtMes*10000)
            Si ( obtAño = añoPedido ) entonces
               ModificarRegistro(añoPedido, nombreArchivo)
            sino
               msg ← "El año no coincide con los registros que ya están en el archivo"
               Salida: msg
            Fsi
         Fsi
      ( opcionEntry = 4 ) :
         msg ← “Opción 4: Mostrar todos los registros activos”
         Salida: msg
         Mostrar(nombreArchivo)
      ( opcionEntry = 5 ) :
         msg ← “Opción 5: Buscar registro de un día dado y mostrar todos los parámetros”
         Salida: msg
         Si ( ArchVacio(nombreArchivo) ) entonces
            msg ← "El archivo está vacío o todos sus registros están borrados. Vuelve al
menú principal."
            Salida: msg
         sino
            msg ← "Ingresar la fecha que quiere buscar"
            Salida: msg
            PedirAño(añoPedido)
            Abrir(nombreArchivo, f, l)
            Leer(f, capturarReg)
            Cerrar(f)
            obtDia ← capturarReg.ddmmyyyy div 1000000
            obtMes ← capturarReg.ddmmyyyy div 10000 - (obtDia * 100)
            obtAño ← capturarReg.ddmmyyyy - (obtDia*1000000) - (obtMes*10000)
            Si ( obtAño = añoPedido ) entonces
               PedirFecha(añoPedido, fechaBuscar)    // devuelve fecha en formato
ddmmyyyy
               arregloPos ← ArchArray(nombreArchivo)
               pos ← BuscaPosRecursiva(arregloPos, 1, fechaBuscar)
               MostrarDatos_Pos (pos, arregloPos)
            sino
               msg ← "El año no coincide con los registros que ya están en el archivo"
               Salida: msg
```

```
            Fsi
        Fsi
    ( opcionEntry = 6 ) :
        msg ← “Opción 6: Listar el día o días de máxima temperatura en lo que va del
año”
        Salida: msg
        ListarTempMax(nombreArchivo, listaTempMax)
        MostrarLSETempMax(listaTempMax)
    ( opcionEntry = 7 ) :
        msg ← “Opción 7: Listar el día o días de máxima precipitación en lo que va del
año”
        Salida: msg
        ListarPrecMax(nombreArchivo)
    ( opcionEntry = 8 ) :
        msg ← “Opción 8: Listar las fechas de mayor a menor velocidad de viento”
        Salida: msg
        ListarMayMenVelViento(nombreArchivo)
    ( opcionEntry = 9 ) :
        msg ← “Opción 9: Realizar una copia de seguridad del archivo del año en curso”
        Salida: msg
        CopiaSeguridad(nombreArchivo)
    ( opcionEntry = 10 ) :
        msg ← “Opción 10: Salir”
        Salida: msg
    ( Otros ) :
        msg ← “Debe ser un número del 1 al 10.”
        Salida: msg
  Fsegún
Hasta que ( opcionEntry = 10 )
msg ← “Salida exitosa."
Salida: msg
Fin
```