Programación 2
Curso 2016/2017

Práctica 2
La granja del tío Owen (parte 2)

Esta práctica consiste en la ampliación y mejora de la primera parte del programa para gestionar la granja del tío Owen.

1. Normas generales

1.1. Entrega

1. El último día para entregar esta práctica es el **miércoles 5 de abril, hasta las 23:59**. No se admitirán entregas fuera de plazo.

2. La práctica debe tener sólo un fichero llamado "`uncleOwen.cc`".

1.2. Detección de plagios/copias

En el Grado en Ingeniería Informática, la Programación es una materia fundamental, que se aprende programando y haciendo las prácticas de las diferentes asignaturas (y otros programas, por supuesto). Si alguien pretende aprobar las asignaturas de programación sin programar (copiando), obviamente tendrá serios problemas en otras asignaturas o cuando intente trabajar. Concretamente, en Programación 2 es muy difícil que alguien que no ha hecho las prácticas supere el examen de teoría, por lo que copiar una práctica es una de las peores decisiones que se puede tomar.

La práctica debe ser un trabajo original de la persona que la entrega; en caso de detectarse indicios de copia de una o más prácticas se tomarán las medidas disciplinarias correspondientes, informando a la dirección de la EPS por si hubiera lugar a otras medidas disciplinarias adicionales.

1.3. Otras normas

1. La práctica se debe entregar exclusivamente a través del servidor de prácticas del departamento de Lenguajes y Sistemas Informáticos, al que se puede acceder desde la página principal del departamento (`www.dlsi.ua.es`, "Entrega de prácticas") o directamente en la url `http://pracdlsi.dlsi.ua.es`.

   - No se admitirán entregas por otros medios (correo electrónico, Campus Virtual, etc.).
   - El usuario y contraseña para entregar prácticas es el mismo que se utiliza en el Campus Virtual.
   - La práctica se puede entregar varias veces, pero sólo se corregirá la última entrega (las anteriores entregas no se borran).

2. El programa debe poder ser compilado sin errores con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas; si la práctica no se puede compilar su calificación será 0. Se recomienda compilar y probar la práctica con el autocorrector inmediatamente antes de entregarla.

3. La corrección de la práctica se hará de forma automática, por lo que es imprescindible respetar estrictamente los textos y los formatos de salida que se indican en este enunciado.

4. Al principio de todos los ficheros fuente entregados se debe incluir un comentario con el nombre y el NIF (o equivalente) de la persona que entrega la práctica, como en el siguiente ejemplo:

```
// NIF   12345678X  GARCIA GARCIA, JUAN MANUEL
```

5. El cálculo de la nota de la práctica y su influencia en la nota final de la asignatura se detallan en las transparencias de la presentación de la asignatura.

2. Introducción

Esta práctica consiste en la ampliación y mejora de la práctica anterior, con mejoras en la distribución y gestión de androides, la simulación detallada, por cada hora de trabajo, de la jornada (que pasará a tener 10 horas en vez de 5), la importación de datos desde ficheros, el almacenaje y recuperación de los datos de la granja mediante ficheros binarios, y la introducción de algunos datos mediante argumentos desde la línea de órdenes.

3. Funcionamiento de la práctica

Para implementar las mejoras requeridas por Luke, es necesario realizar cambios en todos los aspectos de la práctica anterior, además de añadir opciones para gestionar ficheros, y argumentos para usar el programa desde la línea de órdenes. Los cambios más importantes son:

- En la práctica anterior, se simulaba la jornada de trabajo completa (5 horas), pero en esta práctica se simulará hora a hora la jornada (que además pasa a tener 10 horas).
- Los androides se enviarán a mantenimiento cuando alcancen un número determinado de horas trabajadas.
- Cuando en un campo se terminen de recoger todos los productos, los androides del campo se devolverán a la granja, que podrá redistribuirlos por los demás campos para que trabajen en las horas restantes de la jornada de trabajo.
- Si un campo tiene suficientes androides para recoger todos los productos que le quedan en la siguiente hora, no se le asignarán más androides.

- Cambia la forma de introducir los datos de los nuevos productos por recolectar en los campos, en esta práctica se utilizará un `string` con datos de varios campos y sus productos.

3.1. Androides

Con el objetivo de evitar averías en la medida de lo posible, Luke ha decidido que los androides dejarán de trabajar cuando alcancen un cierto número de horas trabajadas, se volverán a la granja y la jornada siguiente se quedarán sin trabajar para realizarles un mantenimiento preventivo. Para implementar esta característica es necesario añadir un nuevo campo al registro `Android`:

```
int hoursWorked;
```

que servirá para contabilizar las horas trabajadas por el androide. Esto implica que la función `printAndroid` debe imprimir también el número de horas trabajadas (`hw`), como en el siguiente ejemplo:

```
[x-75 sn=101 s=100 hw=8 st=3]
```

Cuando, al terminar una hora de trabajo, el androide haya alcanzado el número de horas fijadas para el mantenimiento, el campo en el que está trabajando el androide le pondrá el estado `ST_MAINTENANCE0` y lo devolverá a la granja.

Las horas que puede trabajar cada modelo de androide estarán almacenadas en el array `ANDROIDDATA`:[1]

```
const int NUMDROIDMODELS=25;

typedef struct {

  string modelName;
  int    hoursMaintenance;

} AndroidData;

const AndroidData ANDROIDDATA[NUMDROIDMODELS] =
  {
    "x-75", 8,
    "fx-7", 10,
    "JK9", 15,
    "XAR-25", 5,
    ...
```

Por ejemplo, cuando un androide modelo `JK9` alcance las 15 horas trabajadas, el campo en el que trabaje lo pondrá en `ST_MAINTENANCE0` y lo devolverá a la granja. Al terminar la jornada, la granja pondrá al androide en el estado `ST_MAINTENANCE1` para que la siguiente jornada no trabaje y se pueda reparar. Finalmente, al acabar esa jornada, la granja pondrá a todos los androides en estado `ST_MAINTENANCE1` en estado `ST_IDLE` de manera que puedan trabajar en la jornada siguiente; en ese momento, el valor de `hoursWorked` se pondrá a 0 para iniciar una nueva cuenta.

[1] Este array está completo en el fichero `uncleOwen-p2.cc`, aquí sólo se muestra una parte.

3.2. Campos de cultivo

En el registro de los campos, `Field`, se añadirá un nuevo campo para llevar la cuenta de las horas de jornada restantes:

```
int hoursLeft;
```

Es necesario modificar la función `printField` para que imprima también el número de horas de trabajo restantes, junto al número de productos, como en este ejemplo (en el que quedan 3 horas):

```
{Field: Lichoogs(4055 3)
  [x-75 sn=100 s=100 hw=7 st=0]
  [fx-7 sn=104 s=35 hw=7 st=0]
}
```

Ese contador de horas restantes inicialmente valdrá 10 (se debe usar la constante `DAILY_WORKING_HOURS`, definida en el fichero `uncleOwen-p2.cc`), que es la duración de la nueva jornada de trabajo, y se decrementará al acabar cada hora de trabajo. Todos los campos empezarán a recolectar en la primera hora, y la jornada de trabajo en un campo determinado finalizará cuando se hayan trabajado todas las horas, o bien cuando se hayan recolectado todos los productos.

Al acabar una hora de trabajo, en la que se han recolectado productos y actualizado las horas de trabajo del campo y de los androides, se debe revisar todos los androides y enviar a mantenimiento a aquellos que hayan alcanzado su límite de horas, tal y como se explica en el apartado anterior.

A continuación, si la jornada de trabajo en el campo ha terminado (porque se han recolectado todos los productos o bien se han trabajado todas las horas), todos los androides que queden trabajando se devuelven a la granja con estado `ST_IDLE`, listos para ser distribuidos por los campos (en esa misma jornada si el campo ha terminado antes de recolectar sus productos, o en la siguiente jornada).[2] También se cambiará el valor de `hoursLeft` del campo para dejarlo preparado para la siguiente jornada de trabajo.

La implementación de estos cambios se hará en la función `collectField`, que simulará la recolección de productos en el campo durante una hora de la jornada de trabajo,[3] y que tendrá el siguiente prototipo:

```
vector<Android> collectField(Field &field);
```

Dicha función devolverá un vector de androides con los androides que tengan que volver a la granja, bien sea por mantenimiento o porque se haya terminado la jornada de trabajo del campo;[4] obviamente, esos androides se borrarán del vector de androides del campo. La granja recogerá los androides del vector que devuelve `collectField` y los añadirá al final de su vector de androides. Antes de guardar un androide en el vector que va a devolver, la función `collectField` tiene que asignarle un estado adecuado (según sea un androide que va a mantenimiento o que puede utilizarse en otro campo). Obviamente, si el campo no va a devolver ningún androide a la granja, devolverá un vector vacío.

[2] Si en un campo se han acabado de recolectar todos los productos pero no se ha acabado la jornada, los androides que devuelve el campo a la granja se pueden asignar a otro campo en la siguiente hora.

[3] Por tanto, el cálculo de productos recolectados por los androides es necesario ajustarlo, porque sólo se simula el trabajo de una hora, no el de toda la jornada como en la práctica 1.

[4] Los androides que siguen trabajando en el campo permanecen en él, no pueden ser distribuidos a otros campos.

3.3. Granja

En esta práctica, la recolección de productos de la granja se va a simular hora a hora, por lo que el código de la práctica anterior se debe adaptar. La función que realiza esta tarea es:

```
void collectFarm(Farm &farm,int &hour);
```

Se le pasará como parámetro la granja y `hour`, una variable de la función `main` que inicialmente valdrá 1 y servirá para controlar hora a hora la jornada de trabajo. La función tiene que:

1. Mostrar la hora de trabajo (que inicialmente valdrá 1 y se incrementará al final de la función):

   ```
   ================================================
   Hour: 1
   ```

2. Mostrar el mensaje de comienzo (`---- start ----`) y la información de la granja.

3. Distribuir los androides entre los campos, como en la práctica anterior pero con una mejora: si un campo ya tiene suficientes androides asignados para recolectar los productos que le quedan en la siguiente hora, no se le asignan más androides, es decir, no se usa ese campo para buscar el número mínimo de androides (es como si no tuviera productos que recolectar). El algoritmo de distribución de androides quedaría así:[5]

   a) De entre todos los campos que tienen productos que recolectar, *y en los que la capacidad total (para una hora de trabajo) de los androides ya asignados al campo es inferior a la cantidad de productos por recolectar*, se obtiene el número mínimo de androides que hay en cualquiera de los campos; denominaremos m a dicho número mínimo de androides. Como en la práctica anterior, si no hay ningún campo con productos por recolectar, el algoritmo termina (no se puede obtener m). Si un campo no tiene productos, no se le asignarán androides.

   b) De entre los campos (que no tengan suficientes androides ya) con m androides, se elige el que tenga más productos que recolectar (debe haber al menos uno).

   c) Se elige el androide más rápido de los disponibles en la granja (los que tengan un estado `ST_IDLE`); si hay dos o más de la misma velocidad, se coge el que aparezca primero en el vector de androides. Si no quedan androides por distribuir en la granja, el algoritmo termina.

   d) El androide elegido se elimina del vector de la granja, se pone a trabajar (cambiando su estado a `ST_WORKING`) y se añade al final del vector de androides del campo elegido.

   e) Estos pasos se repiten hasta que no quedan androides por distribuir. Si al empezar el algoritmo no hay productos por recolectar en ningún campo, no se podrá obviamente asignar ningún androide a ningún campo, y el algoritmo terminará sin hacer nada.

[5] La mejora en el algoritmo está marcada en cursiva.

4. Mostrar el mensaje de distribución y la información de la granja.
5. Recolectar los productos de cada campo, llamando a `collectField`, y guardando al final del vector de androides de la granja los androides que devuelva el campo; una vez hecho esto, el vector que devuelve el campo se debe borrar.
6. Incrementar la hora de trabajo (el parámetro `hour`)
7. Mostrar el mensaje de fin (`---- end ----`) y la información de la granja.
8. Si ya se han superado las horas de la jornada, se debe realizar el mantenimiento de los androides de la granja, y poner el parámetro `hour` otra vez a 1 para la siguiente jornada. El mantenimiento consiste en:
   a) Si un androide está en estado `ST_MAINTENANCE1`, se pondrá en estado `ST_IDLE` (que es en el que están el resto de androides), y se pondrá el campo `hoursWorked` a 0 para la siguiente jornada.
   b) Si un androide está en estado `ST_MAINTENANCE0`, simplemente se pasará al estado `ST_MAINTENANCE1` para que la jornada siguiente no trabaje y se quede en la granja para realizar el mantenimiento.

La función que gestionará la simulación de la jornada de trabajo hora a hora se debe llamar `startWorkingHour` (que sustituye a la función `startWorkingDay` de la práctica anterior), y tendrá el siguiente prototipo:

```
void startWorkingHour(Farm &f,int &hour);
```

El parámetro `hour` será la variable de la función `main` que se usa para controlar la hora de la jornada, y que también se pasará a `collectFarm`, como se explica más arriba. Desde el menú de la aplicación, en la función `main`, se debe llamar a `startWorkingHour`.

La función `startWorkingHour` tiene que hacer lo siguiente:

1. Si la granja no tiene campos, debe emitir el error `ERR_NO_FIELDS`, como en la práctica anterior.
2. Si el valor de `hour` es 1, se trata del comienzo de la jornada, por lo que debe pedir al usuario las nuevas cantidades de los productos de los campos, para añadirlos a los que hayan quedado por recolectar en la jornada anterior. Para que a Luke le resulte más fácil introducir los datos de los productos, el programa pedirá los productos con el mensaje:

   ```
   Products:
   ```

   y Luke introducirá una cadena con los productos de uno o más campos, como la del siguiente ejemplo:

   ```
   "Kawees":  2000  "Lichoogs":5000    "Zaanahor" : 12000
   ```

   Dicha cadena tendrá los datos de uno o más campos; primero aparecerá el nombre del campo entre comillas[6] (si no existe el campo se debe emitir el error `ERR_WRONG_FIELD`), seguido del símbolo "`:`", y seguido de un

[6] El caracter `"` no aparecerá dentro del nombre del campo, no es necesario comprobarlo.

número entero sin signo (que podría ser 0, en cuyo caso se emitiría el error `ERR_WRONG_PRODUCTS`). La cadena puede tener datos de uno o más campos, y el resto de caracteres serán un número indeterminado de espacios en blanco (no es necesario comprobarlo, la sintaxis será correcta). Si se produce algún error procesando los datos de un campo (`ERR_WRONG_FIELD` o `ERR_WRONG_PRODUCTS`), se ignorará esa pareja de datos y se seguirá procesando la cadena buscando datos de otros campos.

Una vez leída la cadena de datos de los productos, la función `processProducts` es la que debe procesar esa cadena y emitir errores, extraer la información de campos y productos, e incrementar el número de productos de los campos. Esta función se utilizará también en otra parte de la práctica (cuando se importen datos de productos de un fichero).

3. Finalmente, debe llamar a la función `collectFarm`.

3.4. Ejemplo

Suponiendo que se introducen en la granja dos campos, `Aamis` y `Boonu`, 4 androides (dos `x-75` con velocidad 100, un `JK9` con velocidad 150 y un `XAR-25` con velocidad 300), y se introducen 3000 productos en el campo `Aamis` y 1000 en el `Boonu`, la salida de la práctica simulando la jornada de trabajo sería:

```
================================================
Hour: 1
---- start ----
Farm: west farm
{Field: Aamis(3000 10)
}
{Field: Boonu(1000 10)
}
[x-75 sn=100 s=100 hw=0 st=1]
[x-75 sn=101 s=100 hw=0 st=1]
[JK9 sn=102 s=150 hw=0 st=1]
[XAR-25 sn=103 s=300 hw=0 st=1]
---- distribution ----
Farm: west farm
{Field: Aamis(3000 10)
  [XAR-25 sn=103 s=300 hw=0 st=0]
  [x-75 sn=100 s=100 hw=0 st=0]
}
{Field: Boonu(1000 10)
  [JK9 sn=102 s=150 hw=0 st=0]
  [x-75 sn=101 s=100 hw=0 st=0]
}
---- end ----
Farm: west farm
{Field: Aamis(2600 9)
  [XAR-25 sn=103 s=300 hw=1 st=0]
  [x-75 sn=100 s=100 hw=1 st=0]
}
{Field: Boonu(750 9)
  [JK9 sn=102 s=150 hw=1 st=0]
  [x-75 sn=101 s=100 hw=1 st=0]
```

```
}
==============================================
Hour: 2
---- start ----
Farm: west farm
{Field: Aamis(2600 9)
  [XAR-25 sn=103 s=300 hw=1 st=0]
  [x-75 sn=100 s=100 hw=1 st=0]
}
{Field: Boonu(750 9)
  [JK9 sn=102 s=150 hw=1 st=0]
  [x-75 sn=101 s=100 hw=1 st=0]
}
---- distribution ----
Farm: west farm
{Field: Aamis(2600 9)
  [XAR-25 sn=103 s=300 hw=1 st=0]
  [x-75 sn=100 s=100 hw=1 st=0]
}
{Field: Boonu(750 9)
  [JK9 sn=102 s=150 hw=1 st=0]
  [x-75 sn=101 s=100 hw=1 st=0]
}
---- end ----
Farm: west farm
{Field: Aamis(2200 8)
  [XAR-25 sn=103 s=300 hw=2 st=0]
  [x-75 sn=100 s=100 hw=2 st=0]
}
{Field: Boonu(500 8)
  [JK9 sn=102 s=150 hw=2 st=0]
  [x-75 sn=101 s=100 hw=2 st=0]
}
==============================================
Hour: 3
---- start ----
Farm: west farm
{Field: Aamis(2200 8)
  [XAR-25 sn=103 s=300 hw=2 st=0]
  [x-75 sn=100 s=100 hw=2 st=0]
}
{Field: Boonu(500 8)
  [JK9 sn=102 s=150 hw=2 st=0]
  [x-75 sn=101 s=100 hw=2 st=0]
}
---- distribution ----
Farm: west farm
{Field: Aamis(2200 8)
  [XAR-25 sn=103 s=300 hw=2 st=0]
  [x-75 sn=100 s=100 hw=2 st=0]
}
{Field: Boonu(500 8)
  [JK9 sn=102 s=150 hw=2 st=0]
  [x-75 sn=101 s=100 hw=2 st=0]
```

```
}
---- end ----
Farm: west farm
{Field: Aamis(1800 7)
  [XAR-25 sn=103 s=300 hw=3 st=0]
  [x-75 sn=100 s=100 hw=3 st=0]
}
{Field: Boonu(250 7)
  [JK9 sn=102 s=150 hw=3 st=0]
  [x-75 sn=101 s=100 hw=3 st=0]
}
================================================
Hour: 4
---- start ----
Farm: west farm
{Field: Aamis(1800 7)
  [XAR-25 sn=103 s=300 hw=3 st=0]
  [x-75 sn=100 s=100 hw=3 st=0]
}
{Field: Boonu(250 7)
  [JK9 sn=102 s=150 hw=3 st=0]
  [x-75 sn=101 s=100 hw=3 st=0]
}
---- distribution ----
Farm: west farm
{Field: Aamis(1800 7)
  [XAR-25 sn=103 s=300 hw=3 st=0]
  [x-75 sn=100 s=100 hw=3 st=0]
}
{Field: Boonu(250 7)
  [JK9 sn=102 s=150 hw=3 st=0]
  [x-75 sn=101 s=100 hw=3 st=0]
}
---- end ----
Farm: west farm
{Field: Aamis(1400 6)
  [XAR-25 sn=103 s=300 hw=4 st=0]
  [x-75 sn=100 s=100 hw=4 st=0]
}
{Field: Boonu(0 10)
}
[JK9 sn=102 s=150 hw=4 st=1]
[x-75 sn=101 s=100 hw=4 st=1]
```

Después de 4 horas de trabajo, en el campo `Boonu` no quedan productos, y por tanto se devuelven a la granja los androides `JK9` y uno de los `x-75`, que en la siguiente hora se asignarán al campo `Aamis`.

```
================================================
Hour: 5
---- start ----
Farm: west farm
{Field: Aamis(1400 6)
  [XAR-25 sn=103 s=300 hw=4 st=0]
```

```
  [x-75 sn=100 s=100 hw=4 st=0]
}
{Field: Boonu(0 10)
}
[JK9 sn=102 s=150 hw=4 st=1]
[x-75 sn=101 s=100 hw=4 st=1]
---- distribution ----
Farm: west farm
{Field: Aamis(1400 6)
  [XAR-25 sn=103 s=300 hw=4 st=0]
  [x-75 sn=100 s=100 hw=4 st=0]
  [JK9 sn=102 s=150 hw=4 st=0]
  [x-75 sn=101 s=100 hw=4 st=0]
}
{Field: Boonu(0 10)
}
---- end ----
Farm: west farm
{Field: Aamis(750 5)
  [x-75 sn=100 s=100 hw=5 st=0]
  [JK9 sn=102 s=150 hw=5 st=0]
  [x-75 sn=101 s=100 hw=5 st=0]
}
{Field: Boonu(0 10)
}
[XAR-25 sn=103 s=300 hw=5 st=2]
```

Después de 5 horas de trabajo, el androide `XAR-25` pasa a mantenimiento y vuelve a la granja.

```
================================================
Hour: 6
---- start ----
Farm: west farm
{Field: Aamis(750 5)
  [x-75 sn=100 s=100 hw=5 st=0]
  [JK9 sn=102 s=150 hw=5 st=0]
  [x-75 sn=101 s=100 hw=5 st=0]
}
{Field: Boonu(0 10)
}
[XAR-25 sn=103 s=300 hw=5 st=2]
---- distribution ----
Farm: west farm
{Field: Aamis(750 5)
  [x-75 sn=100 s=100 hw=5 st=0]
  [JK9 sn=102 s=150 hw=5 st=0]
  [x-75 sn=101 s=100 hw=5 st=0]
}
{Field: Boonu(0 10)
}
[XAR-25 sn=103 s=300 hw=5 st=2]
---- end ----
Farm: west farm
```

```
{Field: Aamis(400 4)
  [x-75 sn=100 s=100 hw=6 st=0]
  [JK9 sn=102 s=150 hw=6 st=0]
  [x-75 sn=101 s=100 hw=6 st=0]
}
{Field: Boonu(0 10)
}
[XAR-25 sn=103 s=300 hw=5 st=2]
================================================
Hour: 7
---- start ----
Farm: west farm
{Field: Aamis(400 4)
  [x-75 sn=100 s=100 hw=6 st=0]
  [JK9 sn=102 s=150 hw=6 st=0]
  [x-75 sn=101 s=100 hw=6 st=0]
}
{Field: Boonu(0 10)
}
[XAR-25 sn=103 s=300 hw=5 st=2]
---- distribution ----
Farm: west farm
{Field: Aamis(400 4)
  [x-75 sn=100 s=100 hw=6 st=0]
  [JK9 sn=102 s=150 hw=6 st=0]
  [x-75 sn=101 s=100 hw=6 st=0]
}
{Field: Boonu(0 10)
}
[XAR-25 sn=103 s=300 hw=5 st=2]
---- end ----
Farm: west farm
{Field: Aamis(50 3)
  [x-75 sn=100 s=100 hw=7 st=0]
  [JK9 sn=102 s=150 hw=7 st=0]
  [x-75 sn=101 s=100 hw=7 st=0]
}
{Field: Boonu(0 10)
}
[XAR-25 sn=103 s=300 hw=5 st=2]
================================================
Hour: 8
---- start ----
Farm: west farm
{Field: Aamis(50 3)
  [x-75 sn=100 s=100 hw=7 st=0]
  [JK9 sn=102 s=150 hw=7 st=0]
  [x-75 sn=101 s=100 hw=7 st=0]
}
{Field: Boonu(0 10)
}
[XAR-25 sn=103 s=300 hw=5 st=2]
---- distribution ----
Farm: west farm
```

```
{Field: Aamis(50 3)
  [x-75 sn=100 s=100 hw=7 st=0]
  [JK9 sn=102 s=150 hw=7 st=0]
  [x-75 sn=101 s=100 hw=7 st=0]
}
{Field: Boonu(0 10)
}
[XAR-25 sn=103 s=300 hw=5 st=2]
---- end ----
Farm: west farm
{Field: Aamis(0 10)
}
{Field: Boonu(0 10)
}
[XAR-25 sn=103 s=300 hw=5 st=2]
[x-75 sn=100 s=100 hw=8 st=2]
[x-75 sn=101 s=100 hw=8 st=2]
[JK9 sn=102 s=150 hw=8 st=1]
```

Después de 8 horas ya se han recolectado todos los productos de los dos campos, y todos los androides vuelven a la granja, pero los `x-75` deben pasar a mantenimiento, y el `JK9` se queda en estado `ST_IDLE`. Las siguientes horas (9 y 10) no se hace nada (no quedan productos por recolectar), y no se muestran por brevedad; tampoco se muestra el menú de la aplicación por el mismo motivo.[7]

4. Otros cambios menores

A continuación se describen algunos cambios menores con respecto a la práctica anterior:

- En la función `createField` solamente debe añadirse la inicialización del campo `hoursLeft`, nada más.
- La función `createAndroid` debe usar el vector `ANDROIDDATA` para buscar los modelos de androide, y el vector `droidModels` de la práctica anterior debe eliminarse. Además, se debe inicializar el campo `hoursWorked`.
- El menú de la aplicación quedaría de la siguiente manera:

```
-----========== Farm manager ==============
1- List farm info
2- Add field
3- Add android
4- Start working hour (1)
5- Write farm data
6- Read farm data
7- Import androids
8- Import products
q- Quit
Option:
```

[7] Este ejemplo completo será incluído en el autocorrector de la práctica.

En la opción 4, el número entre paréntesis indica la hora de la jornada de trabajo que se va a simular (el valor de la variable `hour` de la función `main`). Para conseguir esto se debe pasar esa variable como parámetro a la función que muestra el menú.

5. Ficheros

5.1. Almacenamiento y recuperación de datos en fichero binario

Las opciones 5 y 6 del menú permiten almacenar los datos de la granja en un fichero binario, y recuperarlos. Estas opciones se pueden invocar en cualquier momento desde el menú, incluso en mitad de una jornada de trabajo.

En el fichero binario se almacenará la granja, usando el siguiente registro:

```
const int MAXNAME=10;

typedef struct {
  char name[MAXNAME];
  int nextSerialNumber;
  int hour;
  unsigned int numFields;
} FarmBin;
```

Los campos `hour` y `nextSerialNumber` se corresponden con las variables del mismo nombre de la función `main`; el nombre de la granja debe recortarse si es necesario para que se pueda guardar en el vector `name`. El último campo indica el número de campos de cultivo que tiene la granja.

A continuación de este registro, en el fichero binario se almacenarán los datos de los `numFields` campos que tiene la granja, usando un registro como el siguiente:

```
typedef struct {
  char name[MAXNAME];
  unsigned int products;
  int hoursLeft;
  unsigned int numAndroids;
} FieldBin;
```

A continuación del registro de un campo se almacenarán los datos de los androides de ese campo, utilizando el siguiente registro:

```
typedef struct {
  char model[MAXNAME];
   int speed;
   int serialNumber;
   Status status;
   int hoursWorked;
} AndroidBin;
```

Cuando ya se han almacenado los datos de un campo y sus androides, se almacenan los del siguiente campo de la granja. Finalmente, se almacenarán los datos de los androides que se encuentren en el vector de androides de la granja y que, por lo tanto, no están asignados a ningún campo.

En las opciones 5 y 6, el programa debe pedir el nombre del fichero con el mensaje:

```
Enter filename:
```

A continuación debe abrir el fichero (para escritura en el caso de la opción 5, para lectura en la opción 6), y escribir/leer el fichero. Si el fichero no se ha podido abrir, se emitirá el error `ERR_OPEN_FILE` y se volverá al menú. En el caso de la opción 6, el programa, una vez abierto el fichero, debe borrar el vector de campos (borrando también los vectores de androides de cada campo) y el vector de androides de la granja, antes de empezar a leer los datos del fichero. Se puede asumir que no habrá errores de lectura. Evidentemente, al leer los datos de la granja se leerán también los valores de las variables `hour` y `nextSerialNumber` de la función `main` que se han almacenado en el registro `FarmBin`.

Las funciones que deben realizar las tareas de escritura y lectura de ficheros binarios deben tener el siguiente prototipo:

```
bool writeFarmData(const Farm &farm,int nextSerialNumber,int hour);
bool readFarmData(Farm &farm, int &nextSerialNumber,int &hour);
```

Ambas funciones deben devolver `false` si no han podido abrir el fichero, y `true` en otro caso. Se puede asumir que no habrá errores de lectura/escritura, luego el único error que puede producirse es el de apertura del fichero.

5.2. Importación de datos de androides y productos

La opción 7 del menú permite añadir androides a la granja leyéndolos de un fichero de texto con el formato del siguiente ejemplo:

```
100 x-75
150 JK9
78 HC-2
```

donde el primer número es la velocidad del androide, y la segunda cadena es el modelo de androide. Si la velocidad o el modelo son incorrectos, se debe emitir el error correspondiente (como en la opción 3 del menú) y pasar a leer la siguiente línea del fichero. Cada androide que se lea correctamente se añadirá al final del vector de androides de la granja, con el número de serie que le corresponda, que se le asignará justo antes de añadirlo al vector.

Como en las opciones 5 y 6, se pedirá el nombre del fichero con el mismo mensaje, y si el fichero no puede abrirse, se mostrará el error `ERR_OPEN_FILE`.

La opción 8 del menú permite añadir productos a los campos de la granja de forma similar a como se hace al principio de la jornada de trabajo. Al principio se pedirá el nombre del fichero (como en las anteriores opciones), se abrirá y se leerá la información del fichero línea a línea. Un ejemplo de fichero sería:

```
"Kawees"    : 1500  "Lichoogs" : 2000  "Zaanahor":  10000
   "Lichoogs"  :3000  "Kawees":500
"Zaanahor":2000
```

El código para procesar cada línea y añadir los productos a los campos de la granja debe ser el mismo que se utiliza para añadirlos al comienzo de la jornada de trabajo, no se debe duplicar código. Como en ese caso, si el nombre del campo o el número de productos es incorrecto se debe emitir el mensaje de error y seguir procesando el resto de la línea y del fichero.

Las funciones que deben realizar las tareas de importación de datos de androides y productos deben tener el siguiente prototipo:

```
bool importAndroids(Farm &farm,int &nextSerialNumber);
bool importProducts(Farm &farm);
```

Ambas funciones deben devolver `false` si no han podido abrir el fichero y `true` en otro caso, aunque se produzcan errores en los datos que se quieren importar, porque por ejemplo un campo no exista, o algún dato de algún androide (velocidad, modelo) sea incorrecto.

6. Argumentos

Algunos datos de la granja se podrán introducir mediante argumentos desde la línea de órdenes. Puede haber tres tipos de argumentos:

-f fieldName permite añadir un nuevo campo a la granja (puede aparecer varias veces).

-a androidsFilename permite añadir androides a la granja a partir de un fichero con los datos de los androides, como en la opción 7 del menú.

-p productsFilename permite añadir productos a los campos de la granja a partir de un fichero con los datos de los productos, como en la opción 8 del menú.

Tanto la opción `-a` como la opción `-p` sólo pueden aparecer una vez, y después de cada opción (`-f`, `-a` o `-p`) debe aparecer otro argumento, que se asumirá que es un nombre de campo o un nombre de fichero (no se debe comprobar si es o no un nombre adecuado, sea lo que sea lo que aparezca después de una opción se tomará como nombre de campo o de fichero). En caso de que no se cumplan estas condiciones, se producirá el error `ERR_ARGS`. Por ejemplo, si se invoca el programa con estos argumentos debe producirse un error:

```
./uncleOwen -f -a -f Lichoogs -a -f -f Zaanahor -p
```

En este ejemplo, daría error porque después de `-p` no viene un argumento más; no daría error en la combinación `-f -a` (se tomaría `-a` como nombre de campo), ni en `-a -f` (se tomaría `-f` como nombre de fichero). Un ejemplo correcto de ejecución del programa con argumentos sería:

```
./uncleOwen -f Kawees -f Lichoogs -a androids.txt -f Zaanahor -p products.txt
```

Si al acabar de recorrer los argumentos no hay ningún error (es decir, no es necesario emitir el error `ERR_ARGS`), se deben procesar las opciones introducidas por el usuario. Como es posible que aparezcan varias opciones simultáneamente,[8] el orden de ejecución de las acciones asociadas es el siguiente:

1. Si aparece la opción `-a`, se leerán los datos del fichero de androides y se incorporarán a la granja; si se produce algún error, tanto de apertura del fichero como en los datos de los androides, se mostrará el mensaje correspondiente, pero se continuará procesando los argumentos (no se debe emitir el error `ERR_ARGS`).

[8] También es correcto que no aparezca ninguna opción.

2. Se añadirán a la granja todos los campos que se haya especificado desde la línea de órdenes, en el orden en que aparezcan; en el ejemplo anterior, se añadirían en este orden: `Kawees`, `Lichoogs` y `Zaanahor`. Para implementar esta opción se debe usar un `vector<string>` para ir almacenando los nombres de los campos que aparezcan después de cada opción `-f`, y poderlos añadir a la granja una vez se haya verificad que los argumentos son correctos. Si hay nombres de campos duplicados, se emitirá el error correspondiente, pero no se emitirá el error `ERR_ARGS`, y se seguirá procesando los argumentos.

3. Finalmente, si aparece la opción `-p`, se leerán los datos de los productos del fichero y se incorporarán a los campos de la granja. Como en la opción `-a`, si se produce algún error en la lectura se emitirá el mensaje correspondiente, pero la ejecución continuará.

Una vez procesados los argumentos con éxito, se continuará con la ejecución normal del programa, mostrando el menú y ejecutando las opciones que introduzca el usuario. Si se produce el error `ERR_ARGS`, el programa terminará sin mostrar el menú.

7. Implementación

Para implementar la práctica debes tener en cuenta las siguientes observaciones:

1. En la web de la asignatura se publicarán varios ficheros:

   `uncleOwen-p2.cc` : como en la práctica anterior, debes basarte en este fichero para realizar tu práctica, aunque buena parte del código de la práctica anterior se puede reutilizar. Tendrás que añadir más funciones y constantes además de las que aparecen en él. Este fichero contiene lo siguiente:

   a) Tipos definidos que es necesario utilizar, entre los que están los registros (`struct`) para los androides (`Android`), los campos (`Field`) y la granja (`Farm`), actualizados para esta práctica.
   b) Los registros necesarios para escribir/leer el fichero binario.
   c) Una tabla con los datos de los androides disponibles (modelo y horas de trabajo antes del mantenimiento), y algunas constantes: número de modelos, y tipos enumerados para el error y los estados del androide (actualizados para esta práctica).
   d) La función para emitir mensajes de error de la práctica anterior, con un nuevo mensaje de error.
   e) Los prototipos de las funciones que se debe implementar[9].
   f) Una función `main` que debes completar.

   `autocorrectorP2p2.tgz` : autocorrector para probar la práctica con algunas pruebas. La corrección automática de la práctica se realizará con un corrector similar, pero con nuevas pruebas.

[9] Seguramente necesitarás implementar alguna función adicional para que el código sea más sencillo y legible.

2. En la corrección de la práctica se introducirán datos del tipo correcto siempre, aunque con valores que pueden ser incorrectos. Por ejemplo, si se pide la velocidad de un androide, se introducirá siempre un valor entero, que podría ser `-1237` o `0`, pero nunca se introducirá un valor de otro tipo (carácter, string o número real).

3. El formato de los ficheros será siempre correcto, aunque pueden aparecer valores (velocidad, productos, nombre de campo) incorrectos. En los ficheros binarios no habrá errores internos, y se asume que nunca va a haber errores de lectura o escritura.

4. El resto de decisiones en la implementación de la práctica quedan a tu criterio, pero ten en cuenta que el código fuente de la práctica será revisado por tu profesor de prácticas siguiendo la guía de estilo publicada en la web de la asignatura, y parte de la nota de la práctica depende de dicha revisión. Recuerda las recomendaciones que aparecen en el enunciado de la práctica 1.