Programación 2
Curso 2016/2017

Práctica 1
La granja del tío Owen (parte 1)

Esta práctica consiste en la realización de un programa que permita gestionar la granja del tío Owen, gestionando la recolección de productos agrícolas en los diferentes campos de cultivo mediante androides.

1. Normas generales

1.1. Entrega

1. El último día para entregar esta práctica es el **viernes 3 de marzo, 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

Poco antes de conocer a C-3PO, y a su cabezota amigo R2-D2, el joven Luke Skywalker ayudaba a su tío Owen a gestionar las *granjas de humedad* en Tatooine, mientras soñaba con ser piloto de cazas espaciales. Lo que sucedió después con Luke es bien conocido.[1]

Esta práctica simulará la gestión de una granja, con *campos de cultivo* que generan productos agrícolas, que son recolectados por diferentes androides. Cada granja estará formada por campos de cultivo, y dispondrá de varios androides (de diferentes modelos) que se irán distribuyendo por los campos según un algoritmo que se describe más adelante.

2.1. Funcionamiento de la práctica

Inicialmente, la práctica mostrará el siguiente menú al usuario:

```
----========== Farm manager ==========-----
1- List farm info
2- Add field
3- Add android
4- Start working day
q- Quit
```

Y pedirá al usuario que elija una opción con el mensaje:

```
Option:
```

Si la opción elegida no es ninguna de las que aparecen en el menú (p.ej. un "`7`", una "`a`" o un "`;`"), se emitirá el error `ERR_UNKNOWN_OPTION`, llamando a la función `error` con ese parámetro.[2]

[1] Véase *La guerra de las galaxias: Episodio IV.*, varias veces si es necesario, para más información.

[2] En la web de la asignatura se publicará un fichero fuente, `uncleOwen.cc`, en el que estarán definidos los errores que debe emitir el programa, y la función de error, además de registros, constantes y funciones que se deben usar para hacer la práctica.

Cuando el usuario del programa elige una opción correcta, se debe ejecutar el código asociado a esa opción, y luego se volverá a mostrar el menú y a pedir otra opción, hasta que el usuario decida salir del programa.

2.2. Androides

Los androides que trabajan en las granjas son de varios modelos, y cada uno tiene su propio número de serie (puede haber varios androides del mismo modelo), su velocidad de recolección (en productos/hora), y un indicador de estado, que puede tomar dos valores: trabajando y parado.

Los posibles modelos de androide[3] son:

x-75	androide de última generación y muy rápido
fx-7	androide fiable y rápido
JK9	modelo viejo y lento, pero jamás se estropea
XAR-25	androide ligero y rápido, poco fiable
HC-2	modelo pesado y muy lento

Para recoger todos los datos de un androide se utilizará un registro como el siguiente:

```
typedef struct {
   string model;
   int speed;
   int serialNumber;
   Status status;
} Android;
```

Los posibles valores de "`status`" son `ST_WORKING` y `ST_IDLE` (de momento), que se mostrarán en pantalla con los valores 0 y 1 (que son los que se muestran al imprimir directamente el campo `status`).

A la hora de mostrar por pantalla un androide del modelo `JK9`, con número de serie 103, velocidad 78 y que no está trabajando, se mostrará con el siguiente formato:

```
[JK9 sn=103 s=78 st=1]
```

Para añadir un nuevo androide a la granja, en la opción `Add android`, se debe llamar a la función `createAndroid`, en la que se debe pedir al usuario la velocidad con el siguiente mensaje:

```
Enter android speed:
```

Si la velocidad es menor o igual a 0 se debe emitir el error `ERR_WRONG_SPEED` y no hacer nada más en esa opción (se volvería a mostrar el menú).

A continuación se pedirá el modelo de androide mostrando el mensaje:

```
Enter android model:
```

[3] Los nombres de los modelos y sus características son inventados, no están basados en el universo de *Star Wars*.

Si el modelo no es correcto (debe ser uno de los de la tabla anterior) se emitirá el error `ERR_WRONG_MODEL` volviendo al menú principal.

Si todo es correcto se debe asignar un número de serie al androide, que se corresponde con el valor de `nextSerialNumber`. Este valor debe incrementarse a continuación, de manera que cada androide que se añada a la granja tendrá un número de serie diferente y correlativo. La variable `nextSerialNumber` se declara en la función `main` y se pasa como parámetro por referencia a la función `createAndroid`.

Finalmente se pondrá al androide en el estado `ST_IDLE`.

Una vez tenemos creado el androide, lo añadiremos al vector de androides de la granja.

2.3. Campos de cultivo

Los campos de cultivo tendrán un nombre, una cantidad de productos a recolectar, y un vector de androides trabajando en él. Se representarán con el siguiente registro:

```
typedef struct {
   string name;
   int products;
   vector<Android> androids;
} Field;
```

Cuando el usuario elige la opción `Add field` del menú, el programa debe llamar a la función `createField`, que creará el campo (inicialmente con 0 productos y sin androides) y pedirá su nombre con este mensaje:

```
Enter field name:
```

Una vez creado el campo se añadirá al vector de campos de la granja. Si el nombre del campo está repetido, debe emitirse el error `ERR_NAME` y volver al menú principal (y no añadirlo a la granja, obviamente).

Para mostrar por pantalla un campo llamado "`Kawees`", con un par de androides trabajando y 147 productos por recoger, lo haremos con el siguiente formato:

```
{Field: Kawees(147)
  [HC-2 sn=102 s=78 st=0]
  [JK9 sn=103 s=35 st=0]
}
```

2.4. Granja

La granja tendrá un nombre, un vector de campos y un vector de androides. El siguiente registro contendrá la información de la granja:

```
typedef struct {
  string name;
  vector<Field> fields;
  vector<Android> androids;
} Farm;
```

La información que se debe mostrar de la granja cuando el usuario elija la opción `List farm info` del menú es, por ejemplo:

```
Farm: west farm
{Field: Kawees(147)
  [HC-2 sn=102 s=78 st=0]
  [JK9 sn=103 s=35 st=0]
}
{Field: Lichoogs(248)
  [x-75 sn=100 s=100 st=0]
}
{Field: Zaanahor(1215)
  [fx-7 sn=101 s=90 st=0]
  [XAR-25 sn=108 s=18 st=0]
}
[x-75 sn=104 s=100 st=1]
[x-75 sn=105 s=100 st=1]
[HC-2 sn=106 s=78 st=1]
[JK9 sn=107 s=35 st=1]
```

Lo primero que aparece es el nombre de la granja ("`west farm`" en el ejemplo), la información de los campos (si hay alguno), con los androides asignados a cada campo, y finalmente la información de los androides que no están trabajando en ningún campo. Si un campo no tiene androides, se mostrará así:

```
{Field: Kawees(0)
}
```

2.5. La jornada de trabajo en la granja

Cuando empieza la jornada de trabajo en la granja (al elegir el usuario la opción `Start working day`), el programa debe hacer lo siguiente (siempre que la granja tenga al menos un campo, en caso contrario se debe emitir el error `ERR_NO_FIELDS` y volver al menú):

1. Pedir al usuario los datos de productos de cada campo. Para ello:
   - se debe pedir el nombre del campo con el mensaje `"Enter field name: "`; Si se introduce una "`q`", se finaliza la introducción de datos y se pasa a la siguiente tarea, la de recolectar los productos;
   - si el nombre no coincide con el de algún campo de la granja, se debe emitir el error `ERR_WRONG_FIELD`, y volver a pedir el nombre (hasta que el usuario introduzca una "`q`" o un nombre correcto);
   - si el nombre es correcto, se debe pedir el número de productos con el mensaje `"Products: "`. Este número debe ser mayor que 0, en caso contrario se emitirá el error `ERR_WRONG_PRODUCTS`, y se volverá a pedir el nombre del campo. Si es correcto se sumará a los productos de ese campo el número de productos indicado por el usuario (los productos no recogidos en anteriores jornadas permanecen en el campo hasta que se recojan);
   - estos pasos se repetirán hasta que el usuario introduzca una "`q`" como nombre de campo.
2. Recolectar los productos de la granja (llamando a la función `collectFarm`), para lo que se debe hacer lo siguiente:
   a) Imprimir por pantalla el mensaje:[4]

[4] Usando las constantes definidas en el fichero "`uncleOwen.cc`"

```
---- start ----
```

y a continuación mostrar la información de la granja al comienzo de la recolección (como en la opción `List farm info` del menú).

b) Distribuir los androides de la granja entre los campos, según el algoritmo que se explica más adelante. Inicialmente, los campos no tienen androides, porque al terminar la jornada de trabajo anterior se han devuelto a la granja, pero sí pueden tener productos que no se han podido recolectar. Si un campo no tiene productos que recolectar, no se le asignará ningún androide (el algoritmo lo tiene en cuenta).

c) Una vez distribuidos los androides, se debe imprimir por pantalla el mensaje:

```
---- distribution ----
```

y a continuación mostrar la información de la granja.

d) Para cada campo de la granja, se simulará la jornada de trabajo de los androides en dicho campo. Para ello se llamará a la función `collectField`, que calculará los productos recolectados por los androides, multiplicando la velocidad de cada androide, en productos/hora, por el número de horas de la jornada, que está fijado en 5 horas.[5] El número total de productos recolectados por los androides en las 5 horas se restará del número de productos del campo. Obviamente, si los androides tienen capacidad para recolectar más productos que los que había en el campo, el número de productos del campo se quedará en 0 (no puede ser un número negativo).

Después de simular la recolección de productos de cada campo, cuando termina la jornada de trabajo, los androides del campo se devolverán a la granja (cambiando antes su estado a `ST_IDLE`), añadiéndolos al final del vector de androides de la granja. Al terminar, el vector de androides de cada campo debe estar vacío.

e) Finalmente, se mostrará el mensaje:

```
---- end ----
```

y a continuación la información de la granja, para que se pueda apreciar la labor de recolección de los androides.

Un ejemplo de la salida de una jornada de trabajo sería:

```
---- start ----
Farm: west farm
{Field: Kawees(200)
}
{Field: Lichoogs(500)
}
{Field: Zaanahor(1200)
}
[x-75 sn=100 s=100 st=1]
[x-75 sn=101 s=100 st=1]
[HC-2 sn=102 s=78 st=1]
[JK9 sn=103 s=35 st=1]
---- distribution ----
Farm: west farm
```

[5] Este número podría cambiar en las siguientes prácticas, debes tenerlo en cuenta.

```
{Field: Kawees(200)
  [HC-2 sn=102 s=78 st=0]
}
{Field: Lichoogs(500)
  [x-75 sn=101 s=100 st=0]
}
{Field: Zaanahor(1200)
  [x-75 sn=100 s=100 st=0]
  [JK9 sn=103 s=35 st=0]
}
---- end ----
Farm: west farm
{Field: Kawees(0)
}
{Field: Lichoogs(0)
}
{Field: Zaanahor(525)
}
[HC-2 sn=102 s=78 st=1]
[x-75 sn=101 s=100 st=1]
[x-75 sn=100 s=100 st=1]
[JK9 sn=103 s=35 st=1]
```

2.6. Algoritmo de distribución de androides

El algoritmo para distribuir los androides entre los campos de la granja debe implementarse en la función `distributeAndroids`, y es el siguiente:[6]

1. Inicialmente, todos los campos tienen 0 o más productos que recolectar, y ningún androide (al terminar la jornada anterior todos los androides de los campos volvieron a la granja).

2. Se elige el campo que tenga menor número de androides y, en caso de empate, mayor número de productos, y se le asigna el androide más rápido de la granja (de entre los que estén en estado `ST_IDLE`). Para ello:

   a) De entre todos los campos que tienen productos que recolectar, se obtiene el número mínimo de androides que hay en cualquiera de los campos; denominaremos m a dicho número mínimo de androides. 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 con m androides, se elige el que tenga más productos que recolectar (debe haber al menos uno).[7]

   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.

[6] Este algoritmo no es óptimo, y cambiará en las siguientes prácticas.

[7] Si hay más de un campo con el mismo número de productos, se toma el que aparezca primero en el vector.

3. El paso **??** se repite 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.

3. Implementación

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

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

   `uncleOwen.cc` : debes basarte en este fichero para realizar tu práctica. 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`).

   b) Una tabla con los modelos de androide disponibles, y algunas constantes: número de modelos, y tipos enumerados para el error y los estados del androide (`ST_WORKING` o `ST_IDLE`).

   c) Una función para emitir mensajes de error.

   d) Los prototipos de las funciones que se debe implementar[8].

   e) Una función `main` que debes completar.

   `autocorrectorP2p1.tgz` : autocorrector para probar la práctica con algunas pruebas. La corrección automática de la práctica se realizará con un corrector similar.

2. Las funciones que se debe implementar son (están incluidas en el fichero `uncleOwen.cc`):[9]

```
void printAndroid(const Android &android); // prints android info
void printField(const Field &field); // prints field info
void printFarm(const Farm &farm); // prints farm info


// Creates a new android asking the data to the user given nextSerialNumber, and
// adds the android to the farm
void createAndroid(Farm &farm, int &nextSerialNumber);

// creates a new field in the farm with the name provided by the user
void createField(Farm &farm);

// distributes the farm's androids among its fields
void distributeAndroids(Farm &farm);

// simulates the collection of products in a field by its androids
void collectField(Field &field);
```

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

[9] Tendrás que implementar más funciones además de las que aparecen aquí.

```
// collects the products in the farm's fields
void collectFarm(Farm &farm);

// asks for products data in the farm's fields, then collects them
void startWorkingDay(Farm &farm);
```

3. 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 el número de productos, 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).

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. Ten en cuenta especialmente estas recomendaciones:[10]

   - El sangrado del código debe ser adecuado, siguiendo uno de los estilos recomendados en la guía de estilo. Usa 2 o 3 espacios en vez de tabuladores.
   - Los ficheros fuente deben estar adecuadamente documentados, con comentarios donde se considere necesario, en aquellos puntos del código que sean más *oscuros* (pero sin poner comentarios obvios).[11]
   - En Programación 2 no se pueden utilizar variables globales de ninguna clase, los únicos símbolos globales permitidos son las funciones, los tipos definidos por el programador y las constantes. Tampoco se permite usar `break` para salir de bucles, ni usar `return` para salir de funciones en mitad del código.[12]
   - Los nombres de constantes, variables y funciones deben ser adecuados, y deben informar sobre su contenido.
   - Las funciones deben tener un tamaño adecuado; si una función es demasiado grande (p.ej. no cabe en una pantalla normal), probablemente se debería dividir en dos o más funciones. Nunca se debe duplicar (*copy/paste*) código.

[10] Recuerda una cita famosa de John Woods: *Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.*

[11] Muchos programadores consideran que los comentarios no deberían existir, si el código está bien escrito no es necesario poner comentarios.

[12] El objetivo de estas normas es afianzar el aprendizaje, en asignaturas de cursos superiores sí se permite usar estas características del lenguaje.