Tarea #1
Lenguajes de Programación INF-253
Playing God

This is actually happening

Mauricio Araya
maray@inf.utfsm.cl

Rodolfo Castillo
rodolfo.castillo.13@sansano.usm.cl

Benjamín Gautier
benjamin.gautier.14@sansano.usm.cl

Andrés Tapia
andres.tapia@sansano.usm.cl

Marcelo Jara
marcelo.jara.13@sansano.usm.cl

Segundo semestre académico, 2017

1 Objetivos

- Reforzar los conocimientos y el manejo de conceptos de un lenguaje Imperativo.
- Experimentar con el uso de punteros a funciones.
- Que el estudiante use ligado de funciones dinámico (en *runtime*).

2 Tarea

La tarea consiste en implementar una simulación de comportamiento de células en un ambiente acotado. El concepto es similar a un autómata celular[1], sin embargo, se diferencian en que el último utiliza el estado de las células vecinas para evolucionar, mientras que la simulación requerida evolucionará en función de eventos universales.

2.1 Definiciones

A continuación se presentarán conceptos claves para la realización de la tarea.

[1] https://en.wikipedia.org/wiki/Cellular_automaton

Universo Al igual que un autómata celular, el universo estará representado por una grilla cuadricular de dimensiones definidas previamente a la ejecución del simulador. Esta cuadrícula será estática, es decir, no aumentará ni disminuirá su tamaño mientras la simulación esté siendo ejecutada.

Evento universal Los eventos universales son los estímulos que cada célula presente en el universo recibirá para modificar su estado (posicionamiento en el universo). Existen 5 eventos definidos y a falta de creatividad de los creadores, son identificados como `a, b, c, d, e`.

Estado universal Se define como un estado global, es decir, un *snapshot* al estado actual del universo. En otras palabras, el conjunto de los estados individuales de cada célula en el universo (un estado universal puede ser considerado como una instancia de universo).

Característica genética El nombre puede sonar complejo, sin embargo, puede verlo como una personalidad. Una característica genética decide cómo reaccionar (i.e. cómo alterar el estado de una célula) ante un evento universal. En el universo existen sólo 4 características genéticas, sin embargo, estas son un subconjuto que puede ser escogido dinámicamente antes del comienzo de la simulación. Para normalizar su identificación, se identificarán como `A, T, C, G`.

Célula Una célula estará definida por su posición `(x, y)` en el universo y una 5-tupla con las características genéticas que tiene asociadas a cada evento universal `a, b, c, d, e` respectivamente, e.g. `(A, A, T, C, C)` indica que si se recibe el evento universal `b`, la célula reaccionará con la característica genética `A`.

Identificación genética Es un valor numérico identificador único por cada célula en el universo. Se puede notar como `ID(C)`, con `C` una célula.

Precedencia genética Se dice que una célula `A` tiene mayor precedencia genética que una célula `B`, si `ID(A) < ID(B)`.

Apareamiento El apareamiento es un proceso de procreación que requiere de 2 células y siempre generará una tercera.

2.2 Reglas de simulación

Las reglas de la simulación son:

1. Un cambio en el estado universal sólo puede ser consecuencia de un evento universal. Además, siempre se debe considerar el estado universal previo en su totalidad para hacer la simulación y generar uno nuevo.

2. Un evento universal debe ser comunicado a cada célula presente en el universo y esta debe actuar (modificar su estado) según la característica genética correspondiente.

3. Una célula deja de existir si al terminar de procesar un evento, está posicionada fuera de los límites del universo.

4. Al finalizar el procesamiento de un evento, se deben verificar las colisiones de células, esto es, células que estén presentes en la misma celda. Si este fenómeno ocurre, las 2 células con mayor precedencia genética se aparean y las células restantes (de existir) mueren.

5. La generación de células por apareamiento sigue las siguientes reglas:

   (a) La nueva célula tiene la misma posición que sus padres.

   (b) La nueva célula tiene la mayor identificación genética que haya sido asociada hasta el momento.

   (c) Las características genéticas de la nueva célula es una combinación bien definida de las características genéticas de sus padres. Sean `A` y `B` las células procreadoras tales que `ID(A) < ID(B)` y `C` la célula generada. Las 3 primeras características genéticas de `A` serán ubicadas en las posiciones 1, 3 y 5 de la 5-tupla de `C`, mientras que las 2 últimas características genéticas de `B` serán ubicadas en las posiciones 2 y 4 de la 5-tupla de `C`.

6. La simulación debe ser siempre determinista, esto es, dada una configuración inicial del universo y un conjunto de eventos universales, el estado final del universo debe ser siempre el mismo. **Hint:** no use funciones impredecibles en la implementación de las características genéticas (i.e. `random`).

2.3 Reglas de implementación

La tarea tiene un formato bien estructurado, por lo que se definirán ciertas interfaces comunes para normalizar su implementación:

1. El **universo** puede ser representado de la manera que estime conveniente.

2. Los **eventos universales** serán los caracteres mencionados en su definición.

3. Las **características genéticas** serán representadas por funciones, que reciben como primer argumento un evento universal y retornan una 4-tupla de enteros indicando la cantidad de celdas que la célula debe moverse en las direcciones *izquierda, abajo, arriba, derecha* respectivamente (aunque sea un valor de retorno, este arreglo será pasado por argumento por valor-resultado). La firma de una característica genética es de la forma:

   ```
   void genetic_behaviour(char, int*);
   ```

Como fue definido, las características genéticas que participarán en la simulación serán definidas antes de la ejecución dinámicamente. Para poder lograr esto, se trabajará con librerías dinámicas.

Para formalizar la interfaz de la característica genética, se entregará un archivo `genetic_behaviour.h` que contendrá la firma de una característica genética.

Algunos recursos que podrían servirle para adentrarse en la materia de cargado dinámico de funciones:

- http://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html
- http://tldp.org/HOWTO/Program-Library-HOWTO/dl-libraries.html

Ejemplo:

```
int movement[4];
genetic_behaviour('c', movement);

// movement[0] = -1
// movement[1] = 0
// movement[2] = 1
// movement[3] = 0
```

El ejemplo anterior indica que luego de ejecutar la característica genética con el evento universal `c`, el contenido de `movement` es `{-1, 0, 1, 0}`, lo que indica que la célula debe moverse una celda a la izquierda y una hacia arriba.

4. Considerando la definición de **célula**, puede ser representada de la manera que estime conveniente.
5. La identificación genética, precedencia y apareamiento pueden ser logradas de la manera que estime conveniente siempre siguiendo las reglas de simulación.

2.4 Input

Existen 2 instancias de *input* para el simulador:

1. **Configuración:** es un archivo que se pasará como argumento al simulador que contendrá el estado inicial del universo y las definiciones de las características genéticas. El formato es estructurado de la siguiente manera:
   - **Línea 1**: Dimensiones del universo separadas por un espacio. El primer número corresponde al ancho y el segundo al alto.
   - **Líneas 2-5**: Identificación de cada característica genética junto al *path* a la librería dinámica (separados por espacio) que contiene su implementación. Para facilitar lectura, siempre se presentarán en el orden `A, T, C, G`.
   - **Línea 6**: Número de células iniciales en el universo (`N`).

- **Líneas 7-(7 + N - 1):** Definición de cada célula. Esta contempla la 5-tupla de características genéticas (representadas por sus identificadores) y la posición inicial. La identificación genética se asignará automáticamente considerando que las primeras células tienen mayor precedencia genéticas que las últimas definidas.

Ejemplo: Archivo `init.uni`

```
25 25
A libw.so
T libx.so
C liby.so
G libz.so
3
A A T C G 3 0
C C C T C 9 1
G C T A A 0 0
```

Note que:

(a) El *path* de cada característica genética es sólo un nombre de archivo. Esto indicaría que la librería se encuentra en el mismo directorio que el ejecutable del simulador, sin embargo, podría ser una ruta completa, e.g.: `A /home/rcastill/LP/libw.so`

(b) Las posiciones parten desde el 0

2. **Eventos:** Una vez la simulación esté corriendo, debe esperar por input de consola, que indicará el evento universal emitido. Los *input* posibles son: `a|b|c|d|e|q`, dónde `a-e` son eventos universales y `q` indica el término de la simulación.

2.5 Output

Como la tarea está diseñada para ser determinista, tanto el *input* como el *output* está bien definido y puede ser normalizado. Existen 2 instancias de *output* para el simulador:

1. **Cada vez que se ingresa un evento universal** el simulador debe imprimir por pantalla el número de células vivas en el universo.

   Ejemplo:

```
a
3
a
3
b
3
c
2
e
3
```

Note que las posiciones impares corresponden al input y las posiciones pares corresponden al output del simulador. **Es muy importante que siga estas reglas**, i.e.: no agregue un indicador de input tal como >, ya que será penalizado.

2. **Cuando se termine la simulación**, es decir, cuando el simulador reciba una `q` por pantalla, el programa deberá escribir a un archivo llamado `universe.txt` el estado final del universo al terminar la simulación. El formato del archivo `universe.txt` tendrá forma matricial, representando cada celda del universo como una celda de la matriz. Una celda tendrá valor 0, si no hay una célula en esa posición o el mayor identificador genético (con menor precedencia genética) de las células presentes en esa posición.

   Ejemplo: Para el estado final de un universo de `4x4`

   ```
   0 0 1 0
   0 4 3 0
   0 0 0 0
   ```

   Como se puede ver en el ejemplo, el archivo `universe.txt` debe representar cada fila separada por un salto de línea y cada columna separada por un espacio (o tabulación).

2.6 Requerimientos

Su repositorio debe contar con:

- Un directorio `src` donde almacenará todo el código fuente necesario de su tarea (.c/.h)
- Un directorio `bin` donde el resultado de la compilación (el archivo binario) será ubicado
- Un arhivo `Makefile` para autmatizar la construcción de su software. Puede optar por bonificación si utiliza `CMake` como herramienta de automatización.
- Un archivo README.md (especificaciones en la sección de **Condiciones de entrega**)
- Archivos necesarios para la ejecución de un caso de prueba, i.e. código fuente (ubicado en el directorio `src`) de 4 o más librerías dinámicas (características genéticas) con sus respectivas reglas de compilación incluidas en el archivo `Makefile` (o `CMakeLists.txt`) y al menos una definición inicial de universo (archivo de configuración) con el que haya hecho pruebas.

3 Ayuda

Se utilizará parte de la ayudantía fijada para el día **11 de octubre** para una explicación detallada de la tarea.

4 Bonus

- **Small prize**: Para ganar un bonus en su puntaje obtenido en la tarea, utilice `CMake`[2] para automatizar la compilación de su software en vez de `Makefile`.
- **Jackpot:** Represente cada estado universal de la simulación de manera visual (evolución completa de la simulación) en una secuencia de imagenes `png`.

[2] https://cmake.org

5 Condiciones de entrega

La entrega de su tarea debe cumplir las siguientes condiciones:

- Deberán trabajar con la herramienta Git, en la plataforma Gitlab que provee el DI. (`https://gitlab.labcomp.cl`)

  Recuerde crear un nuevo repositorio para esta tarea. Una vez el plazo de la tarea haya terminado, los ayudantes obtendrán el trabajo realizado desde sus repositorios, para lo cual deberá **agregar a todos los ayudantes y profesor** (`rcastill`, `bgautier` y `aatapia`, `mijara`, `maray`) como miembros (`master`) de su repositorio.

  Modificaciones después de la entrega significan un atraso de la tarea completa.

  El nombre de su repositorio debe ser nombrado:

  `tarea1LP-2017-2-<user>`

  donde <user> es el nombre de usuario del DI del estudiante.

  Un nombre de repositorio con distinto formato significa menos puntos.

- El repositorio debe contener todos los archivos necesarios para correr su tarea, evite incluir archivos basura.
- Archivo README: Usted puede realizar supuestos respecto a su tarea en el archivo README, serán considerados. Recuerde que los supuestos son para simplificar su tarea, **no para evitar hacer alguna parte de esta**. Además, se pide que se explique la estrategia usada, la cual debe ser concisa y al punto, no se esfuerce con testamentos ya que en lugar de favorecer la nota podrían empeorarla.
- Deberá incluir un archivo `Makefile` en el directorio del código fuente. El comando `make` deberá iniciar la compilación y generación de un archivo ejecutable del programa. Esto es análogo con aquellos que usen `CMake` considerando sus diferencias de uso.
- La tarea debe entregarse antes de las 23:55 hrs el día **25 de octubre de 2017**.

6 Evaluación del código

- Uso correcto del lenguaje.
- Limpieza de recursos alojados en el `heap`. Considere la utilización de `Valgrind` para verificar por su cuenta.
- Uso de punteros a funciones
- Uso de ligado dinámico de funciones.
- Una parte importante dentro de la vida profesional como desarrollador es la adopción de **convenciones**, por lo que se exige, para esta ocasión, usar la **convención del Kernel de Linux**. (`https://github.com/torvalds/linux/blob/master/Documentation/process/coding-style.rst`)
- Orden y modularización.

7 Recordatorios

- Su programa debe poder ejecutar en el ambiente del Laboratorio de Computación (LabComp), el cual será usado para revisar las tareas en todo momento.
- La tarea se realizará individualmente.
- Si existe una moción para cambiar la fecha de entrega de la tarea esta debe ser realizada al menos una semana antes de la fecha original de entrega. La decisión dependerá del avance en general del curso evidenciado en los repositorios git.
- El enunciado está sujeto a cambios. Si existe algún error en su planteamiento, es responsabilidad de los alumnos notificar para su posterior arreglo.