INF-253 Lenguajes de Programación
Tarea 2: C

Profesor: José Luis Martí, Jorge Díaz Matte, Rodrigo Salas Fuentes
Ayudante Cátedras: Gonzalo Severín Muñoz, Jhossep Martínez Velásquez
Ayudante Coordinador: Álvaro Rojas Valenzuela
Ayudante Tareas: Ricardo Barrida Vera, Bryan González Ramírez, Bastián Salomón Ávalos, Cristian Tapia Llantén, Cristóbal Tirado Morales

8 de abril de 2024

The factory must grow...

1. Factory Of Numbers

El director ZoddaxX, de la empresa C-Games, te contrata para crear un modelo pre-alfa de un juego. Este consiste en un mundo unidimensional que permite construir fábricas operacionales de números. En este hay unas máquinas que hacen las operaciones, entradas y salidas de números. Lo interesante es que hay un robot en el mundo, el cual se mueve constantemente en una línea, donde recoge números, y los deja en otra parte.

Para ello te dan las características del juego y un código base en C para que programes el programa.

2. Plan de Juego

Factory-of-Numbers es una simulación. Cada unidad de tiempo, conocido como *tick*, será un paso donde todos los elementos del mundo interactúan con otros. Como programador del proyecto, el usuario tendrá tres acciones disponibles: construir, destruir y simular uno o más ticks del juego.

El juego es un sandbox, donde uno puede crear una fábrica lineal, se extrae un número de un punto y son pasado por manufacturas aritméticas. Luego son transportados por unos robots transportadores y son finalmente dejados en una salida.

Como tal, no existe un objetivo final. Por lo que es importante la libertad de construir cualquier fábrica en el juego. Ahora se presentarán todos los elementos del juego.

2.1. Estructuras

Las estructuras del juego son los elementos a simular, estos pueden guardar uno o más números, realizar una operación, o trasladar números de una estructura a otras. Estos son tres:

2.1.1. I/O

Es una estructura física en el mundo. Puede tener dos estados, si es una entrada, o si es una salida. Cuando es entrada, este genera copias infinitas del número 1, permitiendo que el robot recoja el número y lo transporte a otra parte. Cuando es salida, **este recibe un número, muestra por pantalla cuál fue y lo destruye**. La estructura estará definida por:

```
typedef struct io
{
    int *objeto;
    char entrada_o_salida;
} io;
```

Esta estructura no puede ser construida o destruida por el jugador.

2.1.2. Manufacturador

Esta estructura se encarga de realizar las operaciones aritméticas de dos números, y las entrega en una salida. Este tiene dos inventarios diferentes, uno para operar y uno para almacenar el resultado. El operando se definirá al momento de construcción. Y la estructura estará definida por:

```
typedef struct manufactura
{
    int *inventario[10];
    int tamanio;
    int *salida[10];
    int tamanio_salida;
    int *(*operar)(int *a, int *b);
} manufactura;
```

Las listas son de tamaño fijo y pueden almacenar hasta 10 números por espacio. Las otras variables permitirán mantener la cuenta de cuantos números actuales tiene cada espacio. Para

operar, se requiere siempre dos números en su inventario. Y el resultado se debe guardar en el primer espacio disponible en *salida*. Claramente, no puede operar si no hay espacio en este último.

Para operar, tiene que esperar su función de paso. Y las operaciones disponibles son la suma, resta, multiplicación y división. Estos están definidos por las siguientes funciones:

```
void step_craft(void *estructura, int x);
int *operar_suma(int *a, int *b);
int *operar_resta(int *a, int *b);
int *operar_division(int *a, int *b);
int *operar_multiplicacion(int *a, int *b);
```

- **Lógica de Paso:**
  El manufacturador, por cada tick de juego, intentará llamar la función de operar. Tiene que verificar si tiene al menos dos números en su *inventario* y un espacio disponible en *salida*. Sin importar si logra o no, se esperará el siguiente tick.
- **Operandos:**
  El operando reciben las direcciones de los últimos dos números del inventario. Operan el número retornando una dirección del resultado para ser guardado después en el primer espacio disponible de izquierda a derecha en la salida. En caso de la división por 0, el número debe ser el mayor posible representable en un **int**. Y en caso de *overflow*, este debe dejar que ocurra.
- **Manejo de Inventario:**
  Cuando el robot interactúe con esta estructura, este **Siempre extraerá el último número disponible** o **Insertará en la primera posición disponible**.

2.1.3. Robot Transportador

El robot transportador se puede mover solo de **izquierda a derecha**. Este se mueve en espacios libres, y cuando la siguiente celda está ocupada, intentará tomar un número de la estructura con sus garras. Y cuando ya tenga un número, va a intentar insertar el número en la estructura. Este está definido como:

```
typedef struct robot
{
    int * inventario;
    char tiene_inventario;
    int direccion;
} robot;
```

El robot puede tomar y dejar un número a la vez. Este debe estar almacenado en *inventario*. La dirección actual estará dada por la variable *dirección*.

El Robot tiene las siguientes funciones asociadas:

```
void step_robot(void *estructura, int x);
void interactuar_manufactura(manufactura *manf, robot *rob);
void interactuar_io(io * io, robot *rob);
```

- **Lógica de Paso:**
  Por cada tick de juego, dependiendo de la situación, el robot intentará hacer alguna de estas acciones:

- Si en la siguiente es una casilla vacía, este avanzará y esperará al siguiente tick.
- Si en la siguiente hay un robot, este invertirá su dirección y esperará al siguiente tick.
- Si en la siguiente hay una manufactura. Entonces, si tiene el inventario vacío, intentará extraer un número. Si tiene algún número en su inventario, intentará insertar el número. Después, independiente de si tuvo éxito o no, este invertirá su dirección y esperará al siguiente tick.
- Si en el siguiente hay un I/O, y tiene el inventario vacío, intentará extraer un número. Si tiene algún número en su inventario, intentará insertarlo. Después, independiente de si tuvo éxito o no, este invertirá su dirección y esperará al siguiente tick.

- **Interactuar Manufactura:**
  Esta función permite que existan los intercambios de números entre la manufactura y el robot con las condiciones previamente mencionadas.
- **Interactuar I/O:**
  Esta función se encarga del intercambio de números entre la manufactura y el robot. Cuando el robot deja un número en un I/O que es salida, se debe **mostrar por pantalla qué número insertó**.

2.2. El Mundo de Pruebas

El mundo de Factory of Numbers es un arreglo unidimensional, donde cada casilla se puede instalar alguna estructura. Para ello se te facilita dos elementos: Un *puntero doble void* de acceso global y una estructura denominada *casilla*. Estos están definidos como:

```c
extern void **mundo;

typedef struct casilla
{
    char tipo_estructura;
    void *estructura;
    void (*step)(void *estructura, int x);
} casilla;
```

El mundo puede ser de tamaño variable. Al momento de iniciar el juego, este debe ser pedido por el jugador mediante consola, siendo el tamaño mínimo de 5 espacios. Cada puntero debe apuntar a una casilla. Además, la primera y última casilla del mundo debe crear un I/O de entrada y salida respectivamente.

La casilla te permite guardar diferente información sobre la estructura en esa posición. **void *estructura** guarda la dirección de memoria de la estructura; **char tipo_estructura** recordará qué tipo de estructura es. Y la función **step** permitirá llamar la función de paso de la estructura. Esta depende también del tipo de estructura.

Además, se incluyen las siguientes funciones:

```c
void crear_mundo(int tamanio);
void mostrar_mundo();
void borrar_mundo();
void simular(int ticks);
```

- **Crear Mundo**: Debe crear el mundo al momento de iniciar el juego.
- **Mostrar Mundo**: Debe imprimir por pantalla todas las casillas del mundo. (Ver en el punto 3.1)
- **Borrar Mundo**: Debe liberar todas las casillas del mundo, y luego borrar el mundo al momento de salir del juego.
- **Simular**: Debe encargarse en simular el mundo según la cantidad de tick ingresado. (Ver en el punto 3)

3. Flujo de Juego

El programa debe iniciar preguntando el tamaño del mundo al jugador. Después entra a un ciclo infinito, donde primero se muestra por pantalla el mundo y luego se le pregunta por alguna de estas 5 opciones.

1. **Construir:** El jugador debe elegir si va a construir un Robot o una Manufactura, y en cuál posición.
2. **Destruir:** Se pregunta por alguna coordenada de alguna estructura en el mundo para ser destruida. (No se pueden destruir los I/O)
3. **Simular:** Se pregunta por cuántos ticks se van a simular. Cada tick va a llamar la función de paso por cada casilla del mundo, de izquierda a derecha. Todas las estructuras tienen la misma prioridad de simulación.
4. **Información extra:** Se pregunta por unas coordenadas para mostrar por pantalla información sobre la estructura en la casilla. Esto incluye:
   - **Robot:** Dirección actual e inventario.
   - **Manufactura:** Los contenidos de inventario y la salida. Y qué tipo de operación realiza.
   - **I/O:** Si es extractor o salida.
   - **Vacío:** Decir que está vacía la casilla.
5. **Salir:** Esta opción es para salir del juego.

Como tal, no hay condición de victoria, por lo que el juego puede continuar infinitamente, o hasta que el jugador decida salir del programa.

3.1. Interfaz de Usuario

La interfaz no tiene que seguir un formato específico, pero sí tiene que mostrar información suficiente para que el jugador entienda qué es lo que le piden y el estado actual del mundo sin la necesidad de adivinar.

3.2. Ejemplo de ejecución:

```
Factory Of Numbers
Ingrese largo: 5
----------------
|E1|  |  |  |S |
----------------
MENU PRINCIPAL
1.- Construir 2.- Destruir
3.- Simular 4.- Salir
Ingresa una Accion: 1
Ingrese Posicion: 2
Elige una opcion para Construir
1.- Robot
2.- Manufactura
>> 1
Direccion Inicial
0.- Izquierda
1.- Derecha
>> 0
----------------
|E1|RI|  |  |S |
----------------
MENU PRINCIPAL
1.- Construir 2.- Destruir
3.- Simular 4.- Salir
Ingresa una Accion: 1
Ingrese Posicion: 3
Elige una opcion para Construir
1.- Robot
2.- Manufactura
>> 2
Elige una operacion
1.- Sumar
2.- Restar
3.- Mutiplicar
4.- Dividir
>> 1
 ----------------
|E1|RI|M+|  |S |
----------------
MENU PRINCIPAL
1.- Construir 2.- Destruir
3.- Simular 4.- Salir
Ingresa una Accion: 4
Saliendo...
```

4. Datos Relevantes

- Junto con el enunciado, se le entregan múltiples archivos de código para su implementación. Estos son:
    - Io.h

- Maufactura.h
- Robot.h
- Mundo.h
- FactoryOfNumbers.c

No se pueden editar o eliminar las variables, y firmas de las funciones que incluye los archivos **.h** (excepto las comentadas como opcional). Sí se puede editar en su totalidad el archivo **FactoryOfNumbers.c**. Este último es donde debe contener la función **main**.

- Se puede agregar otras definiciones y funciones que estime conveniente.
- **Es uso obligatorio de la variable void **mundo** para el manejo del mundo del juego.
- Puede asumir que el input del usuario va a ser siempre correcto.
- **Toda consulta se deben realizar en el foro en la sección Tareas, disponible en Aula**.

5. Sobre la Entrega

- El código debe estar indentado y ordenado.
- Se deberá entregar los siguientes archivos:
  - Io.h, Manufactura.h, Robot.h, Mundo.h
  - Manufactura.c, Robot.c, Mundo.c
  - FactoryOfNumbers.c
  - Makefile
  - Readme.txt
- Las funciones deberán ir comentadas, explicando clara y brevemente lo que realiza, los parámetros que recibe y los que devuelve (en caso de que devuelva algo). Se deja libertad al formato del comentario.
- Debe estar presente el archivo MAKEFILE para que se efectúe la revisión, este debe compilar TODOS los archivos.
- De no existir orden en el código, se realizarán descuentos.
- Se utilizará Valgrind para detectar las fugas de memoria.
- El trabajo es individual obligatoriamente.
- **La entrega debe realizarse en un archivo comprimido en tar.gz y debe llevar el nombre: Tarea2LP_RolAlumno.tar.gz**.
- **El archivo README.txt debe contener nombre y rol del alumno, e instrucciones detalladas para la correcta utilización y compilación del programa**.
- La entrega será vía aula y el plazo máximo de entrega es hasta el **26 de abril a las 23:55 hora aula**.

- Por cada día de atraso se descontarán 20 puntos (10 puntos dentro la primera hora).
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.

6. Clasificación

6.1. Entrega

Para la clasificación de su tarea, la entrega debe cumplir con los requerimientos mínimos que le permitan obtener 30 puntos base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

6.1.1. Entrega Mínima

Para obtener el puntaje mínimo, el programa debe cumplir con los siguientes requisitos:

- Iniciar el mundo con un tamaño fijo de al menos 5 celdas, asignando espacio en memoria *Heap* en la variable **void **mundo** con acceso global. Y cada puntero debe apuntar a una *Celda*.
  - El *mundo* debe incluir las dos estructuras I/O con sus atributos en la primera y última posición del array.
  - El *mundo* debe inicializarse y borrarse a través de *malloc* y *free*, no puede haber leaks de memoria.
  - Se requiere las funciones *crear_mundo, mostrar_mundo, borrar_mundo*.
- Poder construir las estructuras *Robot y Manufactura* en cualquier posición del tablero con *Casilla* como mediador para acceder a estas.
  - Las estructuras deben ser creadas usando *malloc*. No puede haber leaks de memoria.
  - No requiere la función de paso terminada.
- Una interfaz de usuario mínima que permita visualizar el estado del mundo para su prueba. Esto incluye que cada estructura debe tener al menos un carácter único que lo represente.
  - Implica mostrar el mundo, permitir construir alguna estructura y ver el mundo luego de ser colocada.

6.1.2. Entrega

Luego de cumplir con la Entrega Mínima, puede obtener más puntaje cumpliendo con los siguientes puntos (puede haber puntaje parcial por cada punto):

- **Funciones y Lógica del programa:** (50 pts total)
  - *Función Step Manufactura*. La estructura logra operar dos números y guardarlo en la salida según en su enunciado. (Max 10 pts)
  - *Función Step Robot*. El robot cumple con los 4 casos presentes en su enunciado. (Max 12 pts)

- *Interacción entre Robot y I/O*. El robot interactúa con I/O según en su enunciado. (Max 5 pts)
- *Interacción entre Robot y Manufactura*. El robot interactúa con la manufactura según en su enunciado. (Max 10 pts)
- *Simulación*. El juego puede simular todos ticks pedidos respetando el orden de simulación según en su enunciado (Max 6 pts)
- *Uso de función operar y step*. Usa ambos punteros a función presente en las estructuras *Casilla* y *Manufactura*. (Max 7 pts)

- **Interfaz de Usuario y Flujo de Juego:** (20 pts total)
  - *Crear Mundo* para más tamaños. (Max 3 pts)
  - *Construir*, El jugador pude construir las dos estructuras posibles, eligiendo las características iniciales. (Max 4 pts)
  - *Eliminar*, El jugador puede eliminar las estructura previamente creadas en cualquier posición del mundo. (Max 5 pts)
  - *Información extra*, El jugador puede y recibe toda la información sobre una estructura del mundo a elección. (Max 5 pts)
  - *Interfaz*, Los menús y el mundo son fácilmente legibles y navegables. (Max 3 pts)

6.2. Descuentos

- Falta de orden (-20 pts)
- Código no compila (-100 pts)
- Warning (-5 pts c/u)
- Falta de comentarios (-10 pts c/u, Max 30 pts)
- Falta de README (-20 pts)
- Falta de Makefile (-100 pts)
- Falta de alguna información obligatoria en el README (-5 pts c/u)
- Día de atraso (-20 pts por día, -10 dentro de la primera hora)
- Porcentaje de leak de memoria ((1 - 5) % -3 puntos (6 - 50 %) -15 puntos (51 - 100 %) -30 puntos)
- Mal nombre en algún archivo entregado (-5 pts c/u)

En caso de existir nota negativa, esta será reemplazada por un 0.