

95.11 - Algoritmos y Programación I

Trabajo práctico N° 4

Nombre	Correo Electrónico	Padrón
Belenky Ivan	ivanbelenky@gmail.com	99716
Guarino Paulo	pauloguarino97@gmail.com	99404
Nastasi Franco	franconastasi92@gmail.com	100002

Fecha de entrega: 07/07/2017

Trabajo Práctico Integrador

““““Simulador””””

95.11/75.02 Algoritmos y Programación I

1. Objetivo del Trabajo Práctico

El objetivo del trabajo final del curso es la implementación de una primera aproximación a un simulador de un sistema con un dado conjunto de sensores.

Para ello, el presente trabajo consiste en la realización de una aplicación en modo consola, escrita en lenguaje ANSI-C89, configurable a través de archivos y la línea de comandos.

2. Alcance del Trabajo Práctico

Mediante el presente TP se busca que el estudiante profundice y aplique conocimientos sobre los siguientes temas:

- Directivas al preprocesador C
- Programas en modo consola
- Tipos enumerativos
- Funciones
- Salida de datos con formato
- Modularización
- Arreglos/Vectores
- Memoria dinámica
- Arg. en línea de comandos
- Estructuras
- Archivos
- Punteros a función
- Modularización
- Tipos de Dato Abstracto, particularmente Lista polimórfica simplemente enlazada.

3. Introducción

Como se mencionó anteriormente, el objetivo es diseñar e implementar un simulador, de un sistema con un conjunto de sensores. Entendamos entonces, antes que nada, qué es un simulador.

3.1. Simulador

Un simulador es una entidad que intenta reproducir de manera fiel el comportamiento de otra, aquella que intenta simular. Un simulador es una herramienta fundamental en el desarrollo de gran cantidad de proyectos. Por ejemplo, se simulan los cohetes, para saber si van a ser correctamente controlados, se simulan los componentes electrónicos, para ver si se diseñaron correctamente, se simulan los edificios para ver si se caen o no, etc. En general, se utilizan para validar diseños de soluciones, y por ello son fundamentales en ingeniería.

3.2. Sistema

Un sistema es definido por la Real Academia Española (RAE)¹ como un *conjunto de cosas que relacionadas entre sí ordenadamente contribuyen a determinado objeto*, y con esa definición nos es suficiente. Nosotros lo modelaremos como un conjunto de sensores.

3.3. Sensor

Un sensor es un elemento que mide variables físicas, como la aceleración en los acelerómetros, la velocidad de giro en los giróscopos, la posición en la tierra para un Sistema Global de Navegación Satelital (GNSS, por sus siglas en inglés. GPS es un tipo particular de GNSS).

4. Especificaciones

En este trabajo, dado el alcance del curso, no es posible implementar un modelo realmente fiel al sistema a simular y el simulador es incompleto. Pero eso no quita la posibilidad de aproximarnos a ese ideal.

A grandes rasgos, el trabajo consta del desarrollo de una aplicación que lee los datos de configuración de un archivo, en el que se indican los sensores del modelo. Luego, cíclicamente, se simulan mediciones de los sensores, se almacenan en una lista y se procesan.

El ejecutable, que llamaremos `simulador.exe` para los que la hagan en Windows y `simulador` para los demás, se debe invocar de la siguiente forma:

¹RAE: <https://www.rae.es>

```
[*NIX] ./simulador --mdl modelo [--fmt formato] [--out salida]
[DOS]  simulador.exe --mdl modelo [--fmt formato] [--out salida]
```

donde

modelo es la ruta del archivo que contiene los datos del modelo

formato determina el formato de la salida. Por ahora, sólo acepta `csv` y `bin`.

salida es la ruta del archivo de salida. El argumento es opcional. Debe aceptar `-` como nombre de archivo, en cuyo caso se elige la salida estándar, `stdout`.

4.1. Modelo

El archivo de configuración del modelo está dado por líneas que cumplen con el siguiente formato:

```
SENSOR sub_id [param1 [param2 [param3 ... [paramN]]]]
```

donde **SENSOR** es el tipo de sensor, **sub_id** es un número identificador, y **param1..N** son parámetros (números reales) del sensor. Los posibles tipos de sensores son IMU, GPS, y US, por lo menos. Estos sensores se detallan más adelante.

Además, puede haber comentarios en el archivo, que se distinguen por ser líneas que comienzan con el carácter `#`.

Ejemplo de 'modelo': rover.mdl

```
# Modelo del UGV: Rovercito
IMU 1 12.3 4.56 7.8E-3 9E-6 100
GPS 1
# Ultrasonidos: distancia_minima distancia_maxima error ruido
US 1 0.1 6.5 0.01 0.0002
US 2 0.1 6.5 0.01 0.0002
US 3 0.1 6.5 0.01 0.0002
```

4.1.1. Sensores

Definiremos un sensor a través de un T.D.A. Éste debe permitir la identificación del tipo de sensor, tener un subidentificador, los parámetros del sensor, una primitiva para adquirir datos a partir del mismo (además de las típicas de creación, clonación, destrucción, etc.), y puede agregarse un puntero a función que sepa destruir el sensor en cuestión. Se recomienda que el T.D.A. utilice una estructura, `sensor_t`, que defina al menos los siguientes campos:

- `id` que identifica el *tipo* de sensor

- `sub_id` con el que se numera el sensor
- `adquirir_datos` un puntero a función con el que se simulará la medición
- `params` un arreglo donde se almacenan los parámetros del sensor
- `n_params` (opcional) un número que indique la cantidad de parámetros

Cada uno de los sensores que se implementen, debe tener una primitiva, `XXX_adquirir_datos()`, que recibe el sensor y retorna un mensaje con las muestras de la medición.

Este mensaje es el elemento a almacenar en la lista y su definición es la siguiente:

```
struct mensaje {
    int id, sub_id;
    size_t largo;
    unsigned char * datos;
};
```

que puede o no implementarse como un T.D.A.

Entonces, dado que no es necesario conocer la física ni funcionamiento de los sensores, es posible llenar los datos del mensaje de manera aleatoria, siguiendo esta idea:

```
/* versión simplificada sin validaciones y algún error */
mensaje_t * US_adquirir_datos(sensor_t * sens)
{
    mensaje_t * mensaje;
    /* completar */
    mensaje = malloc(...);
    mensaje->datos = malloc(2*..); /* ojo el hardcode */

    for /* cada elemento del vector datos */
        mensaje->datos[i] = /* número aleatorio */
    ...
    return mensaje;
}
```

Si se quisiera hacer un sensor fiel a uno real, debería modificarse esta función, manteniendo lo demás.

Otras primitivas de los sensores en general y en particular, son:

```
/* sensor.h */
status_t SENSOR_crear(sensor_t **);
void SENSOR_destruir(sensor_t **);
status_t SENSOR_set_adquisidora(sensor_t*, /* ptr a func */);
mensaje_t * SENSOR_adquirir_datos(sensor_t *);
```

```
/* ultrasonido.h */
status_t US_crear(sensor_t **, int, int, double *);
void US_destruir(sensor_t **);
mensaje_t * US_adquirir_datos(sensor_t *);
```

Las primitivas anteriores no son exhaustivas, se pueden implementar más.

Los sensores implementados en este trabajo se diferencian por la cantidad de parámetros y la primitiva adquirente de datos, es por ello que se deja al grupo de trabajo definir qué hace cada función en su interior, respetando los prototipos. Por otro lado, cada tipo de sensor debe tener una cantidad diferente de parámetros que los demás.

4.2. Formato

Los formatos de salida pueden ser `csv` o `binarios`, formatos que se utilizaron ampliamente a lo largo del curso. Para ello, se aceptan las cadenas `csv` y `bin` como argumentos. Por ejemplo:

```
[*NIX] ./simulador --mdl rover.mdl --fmt bin --out mensajes.bin
[DOS] simulador.exe --mdl rover.mdl --fmt csv --out -
```

4.2.1. Formato de campos separados por delimitadores

Al utilizar este formato, cada campo de cada mensaje se imprimirá en texto plano utilizando un separador conveniente. De este modo, puede obtenerse una salida como la siguiente, dados los mensajes correspondientes,

mensajes.csv					
LEXEM	largo_1	id_1	sub_id_1	datos_1...	FIN
LEXEM	largo_2	id_2	sub_id_2	datos_2...	FIN
.					
.					
.					
LEXEM	largo_M	id_M	sub_id_M	datos_M,...	FIN

donde `LEXEM` es un arreglo de 5 `chars`² que indica el comienzo de un mensaje y `FIN` es un `char`³ que indica el fin del mensaje.

4.2.2. Formato binario

Al utilizar este formato, cada campo de cada mensaje se almacena en formato binario siguiendo el siguiente orden: `LEXEM`, `largo`, `id`, `sub_id`, `datos[0]`, ..., `datos[...]`, `FIN`.

²LEXEM es a elección del grupo. El del vehículo real es 'C', 'H', 'O', 'R', 'I'.

³FIN es a elección del grupo. El del vehículo real es 'U'.

4.3. Simulador

El simulador es el siguiente algoritmo:

Algoritmo 1: Nuestra aproximación a un simulador

Data: vector de sensores
 lista de mensajes
 N_s := número de sensores
 MAX := mensajes a generar y/o procesar
 $t_0 \leftarrow$ tiempo inicial
while *haya que continuar* **do**
 $N \leftarrow$ un número aleatorio $[0, MAX)$
 for $i \leftarrow 0$ **to** $N-1$ **do**
 $j \leftarrow$ un número aleatorio $[0, N_s)$
 seleccionar sensor j -ésimo
 adquirir datos del sensor seleccionado
 guardar mensaje en una lista
 $M \leftarrow$ un número aleatorio $[0, MAX)$
 for $i \leftarrow 0$ **to** $M-1$ **do**
 if *hay un mensaje en la lista* **then**
 procesar mensaje

Entonces, la aplicación leerá el archivo modelo a partir del cual creará un vector de sensores.

Luego, cíclicamente, accederá a ellos aleatoriamente, N veces, generando N mensajes que se agregan a una lista. Luego, M veces, se toma un mensaje y se lo procesa (se lo imprime).

Finalmente, se debe liberar toda la memoria pedida.

4.4. T.D.A. Lista

Se debe implementar un T.D.A. lista polimórfica simplemente enlazada (lista) para almacenar los mensajes. La misma fue parcialmente implementada en las clases prácticas y enviada en 2 versiones a la lista de correo del curso. Se puede utilizar una de esas versiones, una versión modificada, o una versión totalmente nueva.

Será necesario que la lista posea, entre sus primitivas, una para insertar datos al final de la lista y otra obtener el primer elemento de la lista. De esta forma, será posible insertar el mensaje recibido al final de la lista, tomar el primero cuando sea necesario, procesando los mensajes en orden de llegada.

5. Restricciones

La realización del trabajo se encuentra sujeta a las siguientes restricciones:

- Debe realizarse en grupos de 3 (tres) integrantes.
- No está permitida la utilización de `scanf()`, `fflush()`, `gets()`, la biblioteca `conio.h`, etc.
- Debe recurrirse a la utilización de funciones mediante una adecuada parametrización.
- Deben utilizarse punteros a función en los sensores y en las funciones que procesan los mensajes.
- Se deben utilizar T.D.A. para los sensores y las listas, por lo menos.
- No está permitido en absoluto tener hard-codings:

```
...
if(!strcmp(linea[0], "US"))          /* ¡¡hard-coded!! */
    US_crear(&vector[i], 3,...);     /* ¡¡hard-coded!! */
if(!strcmp(linea[0], "GPS"))        /* ¡¡hard-coded!! */
    GPS_crear(&vector[i], 9,...);    /* ¡¡hard-coded!! */
...
```

sino que debe recurrirse al uso de ETIQUETAS, MACROS, CONSTANTES SIMBÓLICAS, etc.

Los ejemplos no son exhaustivos, sino que existen otros hard-codings y tampoco son aceptados.

- Hay ciertas cuestiones que no han sido especificadas intencionalmente en este Requerimiento, para darle al/la desarrollador/a la libertad de elegir implementaciones que, según su criterio, resulten más convenientes en determinadas situaciones. Por lo tanto, se debe explicitar cada una de las decisiones adoptadas, y el o los fundamentos considerados para las mismas.

6. Entrega del Trabajo Práctico

La fecha de entrega del trabajo práctico es: 26 de junio de 2017 o antes.

No debe hacerse entrega en papel. Deberá realizarse una entrega digital, a través del campus de la materia, de un único archivo cuyo nombre debe seguir el siguiente formato:

YYYYMMDD_apellido1-apellido2-apellido3_entrega-N.tar.gz

donde YYYY es el año (2017), MM el mes y DD el día en que uno de los integrantes sube el archivo, `apellido-1a3` son los apellidos de los integrantes ordenados alfabéticamente, `entrega-N` indica el número de vez que se envía el trabajo (`entrega-1`, `entrega-2`, etc.), y `.tar.gz` es la extensión, que no necesariamente es `.tar.gz`.

El archivo comprimido debe contener los siguientes elementos:

- La correspondiente documentación de desarrollo del TP, siguiendo la numeración siguiente, incluyendo:
 1. Carátula del TP. Incluir una dirección de correo electrónico.
 2. Enunciado del TP.
 3. Estructura funcional de los programas desarrollados.
 4. Explicación de cada una de las alternativas consideradas y las estrategias adoptadas.
 5. Resultados de la ejecución (corridas) de los programas, captura de las pantallas, bajo condiciones normales e inesperadas de entrada.
 6. Reseña sobre los problemas encontrados en el desarrollo de los programas y las soluciones implementadas para subsanarlos.
 7. Bibliografía (ver aparte).
 8. Indicaciones sobre la compilación de lo entregado para generar la aplicación.

NOTA: Si la compilación del código fuente presenta mensajes de aviso (warning), notas o errores, los mismos deben ser comentados en un apartado del informe.

NOTA: El Informe deberá ser redactado en *correcto* idioma castellano.

- Códigos fuentes en formato de texto plano (`.c` y `.h`), *debidamente documentados*.

NOTA: Todos los integrantes del grupo deben subir el *mismo* archivo.

NOTA: Se debe generar y subir un único archivo (comprimido) con todos los elementos de la entrega digital. **NO usar RAR.** La compresión RAR no es un formato libre, en tanto sí se puede utilizar *ZIP*, *GUNZIP*, u otros (soportados, por ejemplo, por la aplicación de archivo *TAR*).

Si no se presenta cada uno de estos items, será rechazado el TP.

7. Bibliografía

Debe incluirse la referencia a toda bibliografía consultada para la realización del presente TP: libros, artículos, URLs, etc., citando:

- Denominación completa del material (Título, Autores, Edición, Volumen, etc.).
- Código ISBN del libro (opcional: código interbibliotecario).
- URL del sitio consultado. No poner Wikipedia.org o stackexchange.com, sino que debe incluirse un enlace al artículo, hilo, etc. consultado.

Utilizando L^AT_EX, la inclusión de citas/referencias es trivial. Los editores de texto gráficos de las suites de ofimática, como LibreOffice Write o MS Word, admiten plugins que facilitan la inclusión.

Ejemplo de referencias

- [1] B.W. Kernighan y D.M. Ritchie. *The C Programming Language*. 2.^a ed. Prentice-Hall software series. Prentice Hall, 1988. ISBN: 9780131103627.
- [2] P. Deitel y H. Deitel. *C How to Program*. 7.^a ed. Pearson Education, 2012. ISBN: 9780133061567.

1. Consideraciones sobre la realización del programa

El objetivo de este programa es simular la adquisición de datos mediante distintos tipos de sensores los cuales poseen una cierta cantidad de parámetros. Para la realización de este trabajo práctico se implementaron 2 TDA's principales. Uno de ellos, el TDA sensor, se usó para simular el comportamiento de los sensores mediante una abstracción de sus características de funcionamiento. Junto a este TDA se utilizó el código de los archivos IMU.c GPS.c y ultrasonido.c para especificar el tipo de sensor. El otro TDA se encuentra en el archivo lista.c, el cual fue usado para almacenar los datos cuya adquisición se simulaba con el TDA sensor mencionado anteriormente.

El algoritmo para realizar la simulación se encuentra en simulador.c y consiste de 3 funciones principales: *crear_arreglo_sensor_modelo* que se encarga de leer un archivo y, a partir de éste, genera un arreglo de punteros a estructuras de sensores donde el tipo de sensor depende de la línea que haya leído, ignorando las líneas que comienzan con #. La segunda función es *generar_mensajes* que a partir de los sensores cargados por la función anterior genera una lista de mensajes donde la cantidad de datos de cada mensajes es un número aleatorio. Cabe aclarar que se utilizó la función *rand* para generar los datos y que como puede devolver un número entre 0 y 32767 [1], se decidió utilizar el resto de lo que devuelve y distintas macros para limitar la cantidad de datos generados y la cantidad de iteraciones en todo el programa. La tercera función es la que se encarga de procesar los mensajes y que se llamó *procesar_mensajes*. Esta función utiliza un arreglo de puntero a funciones para procesar los datos dentro de los mensajes dependiendo del formato con el cual el usuario haya deseado procesar la información.

Al comenzar la ejecución del programa la función *validar_arg* realiza todas las validaciones necesarias para el correcto funcionamiento del programa y obtiene las posiciones de cada uno de los argumentos introducidos mediante CLA, lo que permite abrir el archivo donde se encuentran los modelos de los sensores, conocer en que formato de salida el usuario quiere observar los datos simulados, pudiendo elegir entre *csv* y *bin*, y conocer si el usuario quiere imprimir por pantalla los datos o imprimirlos sobre un archivo. Por defecto se ha establecido que en caso de que el usuario no introduzca el parámetro “--ftm” los datos se conocerán en formato *csv* mientras que si no introduce el parámetro “--out” o el nombre del archivo de salida es “_”, los datos se imprimirán por pantalla. Se implementó una función *str_tolower* para que los comandos puedan ser introducidos en mayúscula y en minúscula.

Luego mediante la función *crear_arreglo_sensor_modelo*, que utiliza a la función *split* y *leer_linea_modelo*, se lee el archivo que el usuario introdujo por CLA donde se encuentran los modelos de los sensores y se genera un arreglo dinámico de punteros a *sensor_t* donde cada uno de los elementos de ese arreglo apunta a una estructura que se corresponde con alguno de los modelos de sensor leídos.

Luego la función *generar_mensajes* itera una cantidad de veces aleatoria el proceso de generar un mensaje con una cantidad de datos aleatorios a partir de un sensor del arreglo que devolvió *crear_arreglo_sensor_modelo*. Este sensor también es elegido aleatoriamente.

Por último la función *procesar_mensajes* se encarga de leer la lista e imprimir los datos en el formato y en la salida especificada por el usuario. Por cada mensaje procesado, esta función se encarga de eliminar, liberando la memoria que sea necesaria, dicho mensaje de la lista mediante la función *pop_lista*.

Este último proceso en el que se generan y procesan mensajes se repite una cantidad de veces a partir de un ciclo *for* y termina cuando se alcanza un número de iteraciones dada por la macro CANT_ITERACIONES.

La estructura funcional del programa desarrollado es la siguiente:

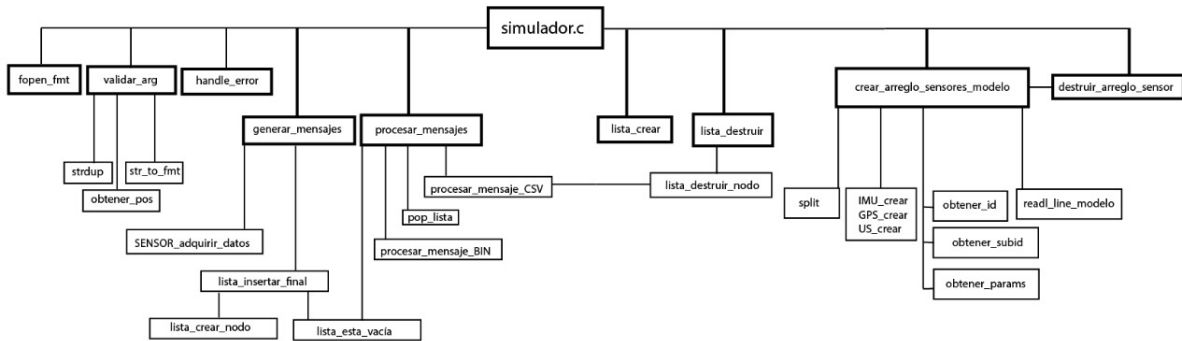


Figura 1: Estructura funcional de simulador.c

No hubo grandes inconvenientes para lograr que la aplicación funcione correctamente pero sí hubo que modificar algunas funciones y revisar exhaustivamente el código por pérdidas de memoria que se encontraron ejecutando la aplicación mediante el programa *valgrind*. El hecho de que ciertas funciones llamadas en el *main* dependían de valores generados aleatoriamente complicaba esta tarea, ya que, en consecuencia, la cantidad de bloques de memoria perdidos también era aleatoria, por lo que no había forma seguro de corroborar que un cambio introducido había reducido la pérdida. Sin embargo, la estructura del programa permitía modificar el código de manera tal que se supriman algunas de las funciones que eran invocadas en el *main* sin alterar el funcionamiento de las otras. De esta manera, por ejemplo, se podían quitar los llamados a las funciones *generar_mensajes* y *procesar_mensajes* y así comprobar si existía pérdida de memoria en la validación de argumentos y en la creación y destrucción del arreglo de sensores y de las listas, que no dependían de parámetros aleatorios. Por otro lado, la función *split* original tuvo que ser modificada sustancialmente porque el arreglo de cadenas dependía de memoria que fuera de la función no podía ser liberada a menos que se modificase el código del resto del programa.

2. Compilación y ejecución del programa

A continuación se muestra la compilación del programa junto con las advertencias que informó el compilador:

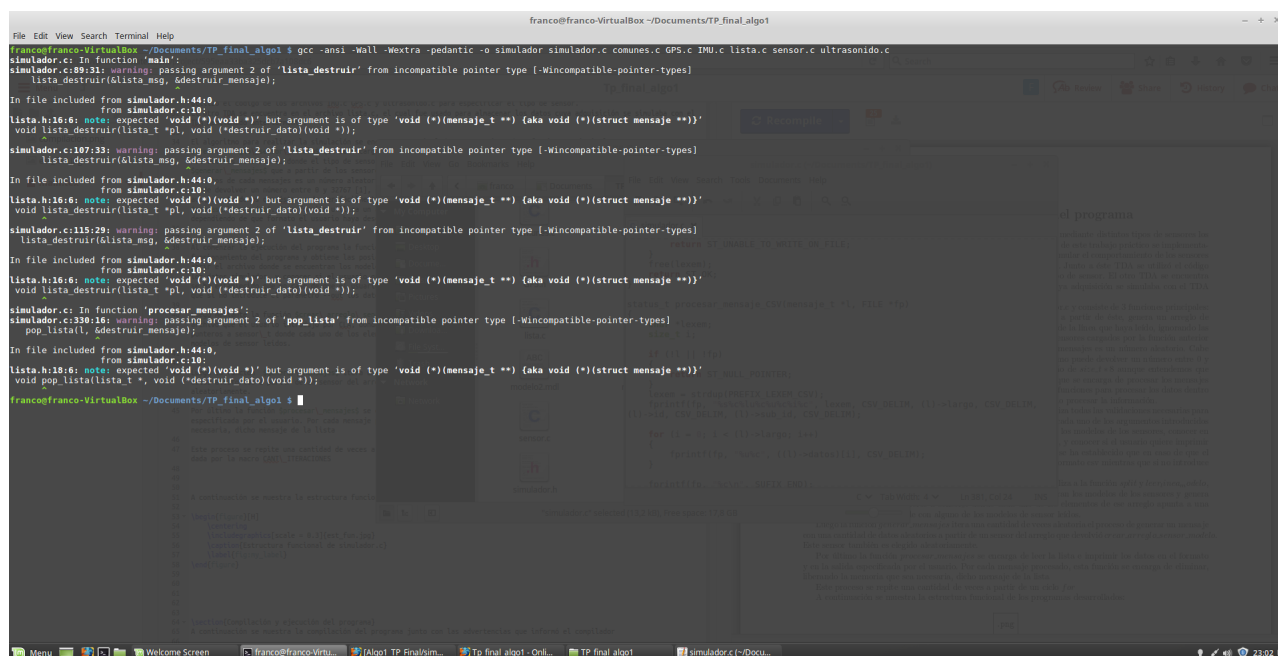


Figura 2: Compilación de simulador.c

Se observa que las cuatro advertencias que indica el compilador se deben a la aparente incompatibilidad entre lo que la función *lista_destruir* definida en *lista.h* espera recibir como argumento (un puntero a función cuyo argumento sea de tipo puntero a *void*) y el tipo de argumento que recibe al ser invocada en el *main* (un puntero a función cuyo argumento es de tipo puntero doble a la estructura *mensaje_t*). Sin embargo, el propósito de que el argumento de la función que recibe *lista_destruir* como puntero sea de tipo puntero a *void* es que pueda recibir cualquier tipo de puntero y de esa manera el usuario que quiera trabajar con el TDA lista no tenga que modificar el código de las funciones asociadas a ella. Es decir que la función está pensada para destruir la lista con todos sus datos independientemente del tipo de dato que contengan los nodos.

A pesar de las advertencias, se verificó que el programa ejecuta correctamente tanto las funciones que precisan del puntero a la función que destruye el dato del nodo, como esta última función en sí, logrando liberar toda la memoria asignada a los datos de la lista. Si se modificaba el tipo de argumento que recibía *lista_destruir* y se la adaptaba al caso particular que se usa en todas sus llamadas dentro del *main* (puntero doble a *mensaje_t*), el programa también habría funcionado correctamente pero la estructura *lista_t* habría perdido su utilidad más allá de esta aplicación en particular en la que se trabaja con la estructura *mensaje_t*, y por lo tanto dejaría de tener sentido como TDA.

El hecho de que el argumento que recibe al ser llamada la función sea un puntero a función cuyo argumento es un puntero doble en vez de simple debería ser irrelevante, ya que un puntero doble sigue siendo un puntero, es decir, una dirección de memoria, y por lo tanto debería ser compatible con el argumento de tipo puntero a *void* como se estableció en la definición. De hecho se probó cambiar el tipo de dato que recibe la función *destruir_mensaje* (cuya dirección de memoria es la que recibe *lista_destruir*) por puntero simple a *mensaje_t* en vez puntero doble, y las advertencias al compilar persistieron.

Se tomaron capturas de pantalla de los resultados de la ejecución de *simulador* para distintos argumentos por línea de comandos y de la ejecución de la aplicación mediante el programa *valgrind*:

```
paulo@pulonet:~/Documents/TP4/Franco$ clear
paulo@pulonet:~/Documents/TP4/Franco$ ./simulador --mdl modelo.mdl --out salida.txt --fmt csv
paulo@pulonet:~/Documents/TP4/Franco$ ./simulador --mdl modelo.mdl --OUT salida.bin --fmt BIN
paulo@pulonet:~/Documents/TP4/Franco$ ./simulador --mdl modelo.mdl --OUT salida.bin --fmt X
Error: Argumento de formato no válido
paulo@pulonet:~/Documents/TP4/Franco$ ./simulador --mdl archivoinexistente.mdl --OUT salida.bin --fmt CSV
Error: No se pudo abrir archivo
paulo@pulonet:~/Documents/TP4/Franco$ ./simulador --mdl modelo.mdl
HOUSE,23,GPS,1,71,86,53,32,60,45,96,16,49,26,70,97,24,45,39,54,28,83,91,68,94,30,81,D
HOUSE,2,GPS,1,51,11,D
HOUSE,17,GPS,1,38,62,12,76,16,97,88,13,93,4,62,19,74,12,43,20,3,D
HOUSE,28,IMU,1,38,93,68,32,75,50,41,98,53,4,21,95,94,83,59,71,99,56,59,13,1,64,27,72,90,91,68,10,D
HOUSE,22,IMU,1,11,33,63,79,18,38,81,59,88,35,63,9,82,58,45,41,81,44,50,92,9,3,D
HOUSE,29,US,2,76,99,80,44,61,75,14,72,61,29,4,79,19,37,38,60,24,53,21,6,63,66,0,96,63,50,89,72,53,D
HOUSE,5,US,1,29,48,94,25,10,D
HOUSE,3,GPS,1,34,82,20,D
HOUSE,1,IMU,1,92,D
HOUSE,23,US,2,52,52,57,73,11,72,92,11,69,55,61,10,79,66,59,93,48,8,39,25,18,60,17,D
HOUSE,27,US,3,37,95,8,29,23,12,33,27,69,7,38,41,51,49,62,6,62,72,85,29,32,30,29,40,69,54,10,D
HOUSE,3,US,1,14,29,61,D
HOUSE,29,GPS,1,42,84,49,76,12,70,35,2,12,86,52,26,44,66,99,29,47,31,12,76,23,33,83,33,15,54,99,96,67,D
HOUSE,30,GPS,1,10,97,35,38,61,6,25,64,70,63,68,96,7,34,47,88,82,30,0,10,53,34,93,38,1,0,38,98,19,51,D
HOUSE,13,US,3,48,19,19,10,77,96,74,99,59,42,48,18,28,D
HOUSE,15,IMU,1,62,26,59,73,31,45,18,70,99,70,60,97,90,11,33,D
HOUSE,15,GPS,1,4,43,69,34,39,95,33,99,89,33,17,18,29,76,32,D
HOUSE,20,US,1,57,90,33,28,60,32,98,20,29,88,83,14,12,43,19,55,64,53,94,60,D
HOUSE,21,US,2,1,72,15,71,53,91,4,12,79,13,2,13,41,15,45,40,87,27,80,71,41,D
HOUSE,26,US,1,12,99,30,17,94,42,56,39,44,28,6,67,33,98,71,45,77,85,47,42,78,62,88,18,50,67,D
HOUSE,25,GPS,1,60,95,87,73,95,69,90,89,12,98,80,8,78,39,75,11,37,99,56,66,36,56,9,14,18,D
HOUSE,21,US,3,20,68,36,93,28,31,32,53,78,2,44,19,66,94,52,26,73,91,1,84,28,D
HOUSE,13,US,1,46,88,49,55,55,19,56,40,92,24,28,85,53,D
HOUSE,2,US,3,6,38,D
HOUSE,14,US,1,9,90,49,61,16,22,52,69,58,80,22,3,79,62,D
HOUSE,14,US,3,17,24,43,57,16,67,85,53,72,97,23,31,87,47,D
```

Figura 3: Ejecución con distintos comandos

```
HOUSE,1,GPS,1,20,D
HOUSE,14,US,3,72,61,71,63,51,62,22,2,59,4,37,81,93,16,D
HOUSE,0,IMU,1,D
HOUSE,9,GPS,1,28,78,66,44,11,94,30,40,66,D
HOUSE,3,US,3,39,93,62,D
HOUSE,28,US,3,76,76,46,36,80,84,69,25,52,49,66,51,88,89,60,68,19,79,13,30,73,43,71,39,75,62,30,68,D
HOUSE,17,US,3,64,53,61,62,89,94,98,58,19,51,60,37,54,48,79,15,16,D
HOUSE,18,US,2,29,29,19,25,0,58,0,14,89,20,90,26,84,43,39,98,84,33,D
HOUSE,23,US,1,5,48,3,42,54,3,21,69,19,72,15,1,1,86,26,53,45,78,67,86,98,9,12,D
HOUSE,1,US,3,51,D
HOUSE,1,US,1,37,D
HOUSE,24,GPS,1,42,29,87,36,84,90,10,5,62,82,73,63,35,59,41,88,56,19,7,94,69,16,58,55,D
HOUSE,2,US,3,87,59,D
HOUSE,9,US,3,43,93,50,31,81,86,73,91,44,D
HOUSE,1,US,1,17,D
HOUSE,16,IMU,1,28,43,48,37,14,7,31,83,24,90,38,45,52,78,4,3,D
HOUSE,24,US,3,96,49,31,77,88,4,21,32,44,46,49,94,59,29,90,59,66,4,67,98,40,91,88,78,D
HOUSE,28,GPS,1,8,93,95,7,93,43,57,24,72,45,80,93,77,24,92,78,71,51,59,61,62,26,17,29,76,57,20,16,D
HOUSE,17,US,2,8,96,2,55,4,47,98,61,71,70,58,3,16,87,28,60,17,D
HOUSE,7,GPS,1,76,64,73,54,81,55,30,D
HOUSE,15,GPS,1,98,79,36,58,75,90,13,79,37,63,92,60,86,2,64,D
HOUSE,29,US,1,44,14,58,95,25,87,59,50,41,92,57,24,83,85,22,14,73,81,90,64,46,21,53,10,14,14,48,16,78,D
HOUSE,10,US,1,74,16,68,69,93,55,80,95,97,24,D
HOUSE,5,US,2,8,38,95,22,11,D
HOUSE,20,US,2,27,75,86,81,37,0,47,85,68,77,39,78,51,55,47,20,48,54,52,43,D
HOUSE,0,GPS,1,D
HOUSE,4,US,2,36,86,24,59,D
HOUSE,12,IMU,1,23,77,79,61,10,68,61,57,5,82,34,44,D
HOUSE,9,US,1,51,59,57,99,14,61,95,17,38,D
HOUSE,26,US,2,74,82,18,85,32,22,61,61,54,22,24,22,36,33,80,18,20,24,30,57,76,42,67,27,8,28,D
HOUSE,1,US,1,66,D
HOUSE,27,US,2,93,0,41,78,84,16,39,97,70,14,73,44,50,7,24,20,27,49,50,84,77,44,51,56,52,32,79,D
HOUSE,2,US,1,49,5,D
HOUSE,21,US,2,99,74,85,15,13,34,37,27,8,81,29,15,6,49,94,7,52,78,36,96,82,D
paulo@pulonet:~/Documents/TP4/Franco$
```

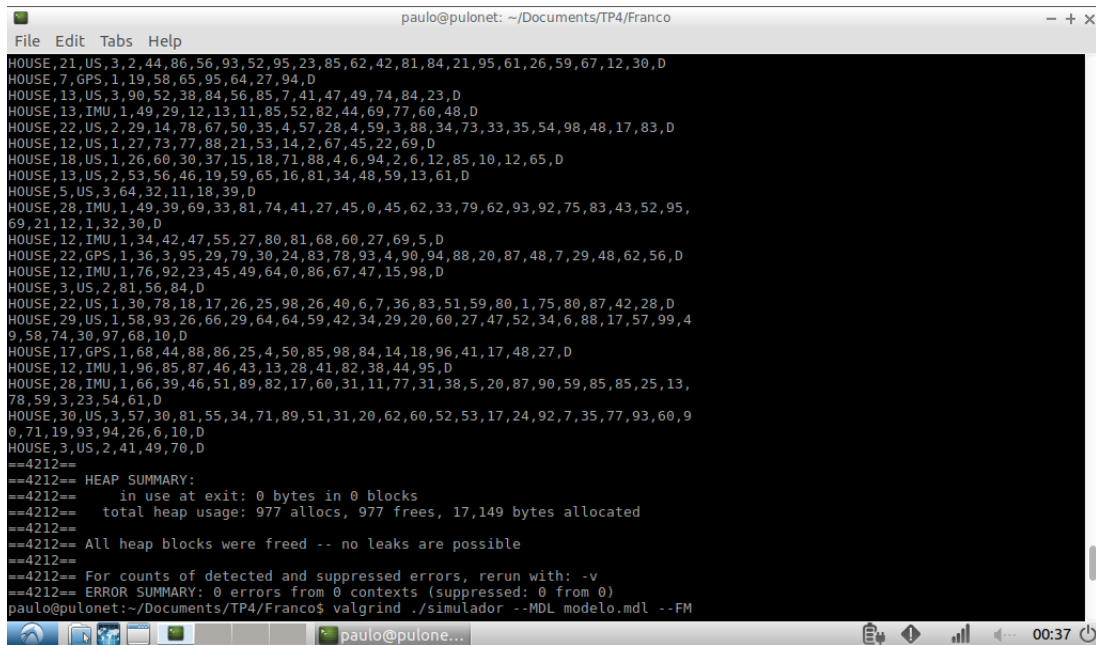
Figura 4: Ejecución con distintos comandos

```
paulo@pulonet: ~/Documents/TP4/Franco
File Edit Tabs Help
HOUSE,2,US,1,49,5,D
HOUSE,21,US,2,99,74,85,15,13,34,37,27,8,81,29,15,6,49,94,7,52,78,36,96,82,D
paulo@pulonet:~/Documents/TP4/Franco$ ./simulador --mdl modelo.mdl --out - --fmt CSV
HOUSE,10,US,1,68,73,17,19,70,65,17,85,4,9,D
HOUSE,10,US,1,61,16,66,35,62,17,98,3,20,52,D
HOUSE,17,GPS,1,0,4,76,93,26,83,63,94,8,33,66,78,98,83,16,54,92,D
HOUSE,5,US,2,54,72,27,41,34,D
HOUSE,15,GPS,1,89,17,44,17,58,96,74,86,41,52,69,57,98,29,90,D
HOUSE,3,US,3,88,47,23,D
HOUSE,8,US,2,27,4,98,51,83,39,37,80,D
HOUSE,27,IMU,1,49,74,96,59,23,22,45,64,74,66,21,73,95,11,37,2,52,37,78,47,81,57,3,79,8,86,18,D
HOUSE,6,IMU,1,48,24,67,75,73,78,D
HOUSE,27,US,2,23,14,22,89,36,95,84,99,84,39,3,73,69,50,54,78,53,85,38,92,3,35,10,4,12,30,79,D
HOUSE,20,IMU,1,29,32,84,43,54,25,31,1,62,83,38,1,86,63,22,37,70,0,42,55,D
HOUSE,22,IMU,1,11,25,97,15,37,79,46,74,39,27,7,23,22,13,1,54,15,63,37,5,16,23,D
HOUSE,22,IMU,1,12,90,90,7,98,80,41,9,57,90,76,95,21,74,69,61,1,28,36,75,42,37,D
HOUSE,5,US,3,0,18,66,68,94,D
HOUSE,6,GPS,1,58,77,48,65,75,80,D
HOUSE,0,US,1,D
HOUSE,21,US,2,12,33,71,86,54,84,39,83,20,14,77,10,96,86,62,66,52,31,60,90,89,D
HOUSE,7,US,3,38,84,94,70,43,30,8,D
HOUSE,14,IMU,1,93,16,32,48,52,23,83,72,38,60,34,86,98,97,D
HOUSE,18,US,1,28,13,92,69,84,60,7,20,6,78,64,89,38,61,35,32,29,68,D
HOUSE,1,US,1,91,D
HOUSE,27,GPS,1,40,67,77,89,20,79,69,85,71,38,69,83,98,41,42,28,57,31,66,18,66,50,47,86,34,28,30,D
HOUSE,30,GPS,1,11,80,26,31,9,15,51,88,84,88,12,22,9,47,72,50,41,0,8,72,67,26,91,17,74,29,4,54,59,5,D
HOUSE,23,IMU,1,38,66,2,47,33,5,88,69,93,0,44,54,99,16,4,41,17,12,65,84,91,56,53,D
HOUSE,22,US,1,57,23,97,15,64,68,53,82,22,52,68,27,40,37,72,92,81,26,92,98,31,85,D
HOUSE,11,IMU,1,50,51,86,59,56,55,97,66,79,94,81,D
HOUSE,1,GPS,1,37,D
HOUSE,21,US,2,65,79,83,37,71,16,16,15,14,47,0,81,94,3,84,33,62,41,88,11,7,D
HOUSE,13,US,3,40,14,72,78,92,10,16,89,27,47,72,64,71,D
HOUSE,0,GPS,1,D
HOUSE,27,US,2,79,39,89,26,94,73,59,56,14,99,19,73,19,76,65,33,49,43,25,59,12,15,86,11,39,2,82,D
HOUSE,23,IMU,1,73,88,66,64,77,92,58,2,3,66,69,3,85,42,22,61,60,7,10,3,33,69,67,D
```

Figura 5: Ejecución con distintos comandos

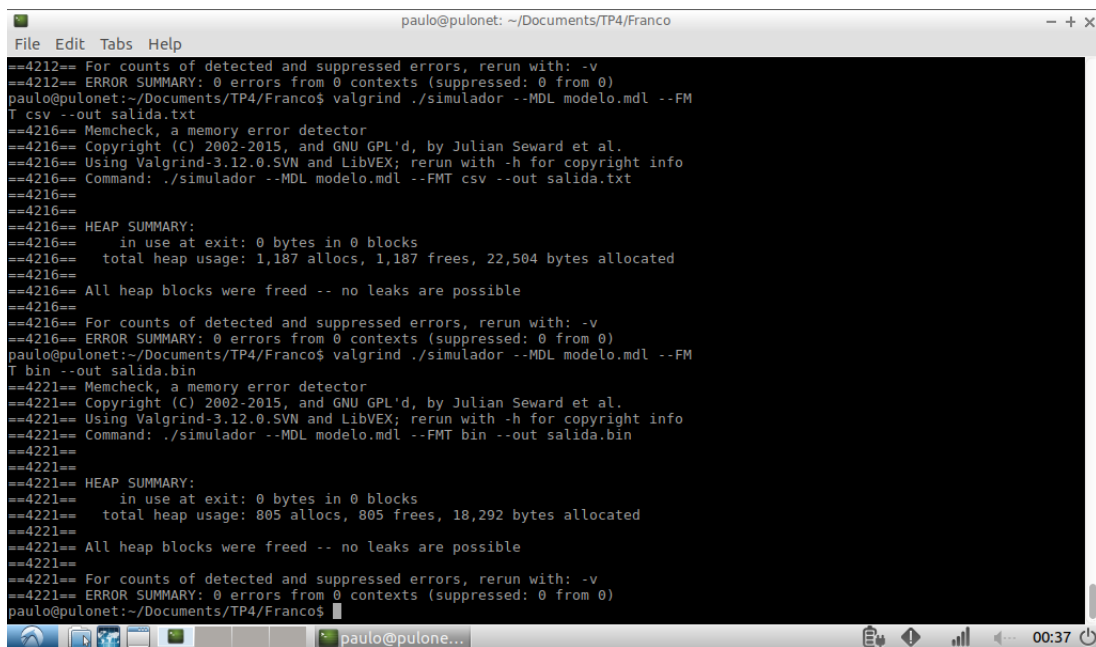
```
paulo@pulonet: ~/Documents/TP4/Franco
File Edit Tabs Help
I/DHOUSE[...DHOUSE
c
HOUSE[...I3#?>
[XTZ2
274Y[...?DHOUSE[...M
[...8Z%;[...N&6[...DV^[...#...]*%DHOUSE[...G[...K' "L
,=DHOUSE
[...FS[...F4[...ZDHOUSE[...A6ZRDHOUSE[...[...]-[...@[...DHOUSE[...DHOUSE
[...ZHZ;DHOUSE[...DHOUSE
[...[...DHOUSE[...ZU7[...DHOUSE[...DHOUSE[...I8IT[...]:#0' F\ G[...DHOUSE[...K,2GF[...75,X[...K 4KDHOUSE[...W-[...YPI*5U6PQ[...[...
DHOUSE[...>Y' S8DHOUSE[...Z+
-N[...]:B+DHOUSE[...SSHhA>]7F
2%8E$DHOUSE
[...c->[...DHOUSE[...[...4[...T]# \828[...DHOUSE[...DHOUSE[...K%R;b[...
2/MWR+0'-X[...[...
ZcD
DHOUSE[...4(3[...DHOUSE
[...Ca[...ET[...DHOUSE[...YSI=)_SUDHOUSE
[...[...]:$[...
HOUSE[...P=T<S[...0
NAJH<0 HVDHOUSE[...DHOUSE[...],IDHOUSE
0=DHOUSE[...
-D;Y[...HOUSE[...T]NODHOUSE[...D. (9H[...[...DHOUSE
[...L8M=7' [...]) DHOUSE[...B[...]:+I[...$+ [...KRD
HOUSE[...SNDHOUSE[...Z[...%# ,0NAQ[...P[...T]DHOUSE[...G[...G]M>GL1RM0c
[...RY8(IR "0[...
V@DHOUSE
[...$[... ,#60,"=DHOUSE[...]:C0S \B[...NR*+.
0:DHOUSE[...
[...9DVHWF+61KDHOUSE[...M0S[...[...F7=<[...HOUSE[...A[...Y]*:;&@,
2_ DHOUSE[...MREaNg6+>SW'[...JDHOUSE[...@I
[...[...K[...C[...G[...paulo@pulonet:~/Documents/TP4/Franco$ ./simulador --MDL modelo.mdl --FMT bin
```

Figura 6: Ejecución con distintos comandos



```
paulo@pulonet: ~/Documents/TP4/Franco
File Edit Tabs Help
HOUSE, 21, US, 3, 2, 44, 86, 56, 93, 52, 95, 23, 85, 62, 42, 81, 84, 21, 95, 61, 26, 59, 67, 12, 30, D
HOUSE, 7, GPS, 1, 19, 58, 65, 95, 64, 27, 94, D
HOUSE, 13, US, 3, 90, 52, 38, 84, 56, 85, 7, 41, 47, 49, 74, 84, 23, D
HOUSE, 13, IMU, 1, 49, 29, 12, 13, 11, 85, 52, 82, 44, 69, 77, 60, 48, D
HOUSE, 22, US, 2, 29, 14, 78, 67, 50, 35, 4, 57, 28, 4, 59, 3, 88, 34, 73, 33, 35, 54, 98, 48, 17, 83, D
HOUSE, 12, US, 1, 27, 73, 77, 88, 21, 53, 14, 2, 67, 45, 22, 69, D
HOUSE, 18, US, 1, 26, 60, 30, 37, 15, 18, 71, 88, 4, 6, 94, 2, 6, 12, 85, 10, 12, 65, D
HOUSE, 13, US, 2, 53, 56, 46, 19, 59, 65, 16, 81, 34, 48, 59, 13, 61, D
HOUSE, 5, US, 3, 64, 32, 11, 18, 39, D
HOUSE, 28, IMU, 1, 49, 39, 69, 33, 81, 74, 41, 27, 45, 0, 45, 62, 33, 79, 62, 93, 92, 75, 83, 43, 52, 95,
69, 21, 12, 1, 32, 30, D
HOUSE, 12, IMU, 1, 34, 42, 47, 55, 27, 80, 81, 68, 60, 27, 69, 5, D
HOUSE, 22, GPS, 1, 36, 3, 95, 29, 79, 30, 24, 83, 78, 93, 4, 90, 94, 88, 20, 87, 48, 7, 29, 48, 62, 56, D
HOUSE, 12, IMU, 1, 76, 92, 23, 45, 49, 64, 0, 86, 67, 47, 15, 98, D
HOUSE, 3, US, 2, 81, 56, 84, D
HOUSE, 22, US, 1, 30, 78, 18, 17, 26, 25, 98, 26, 40, 6, 7, 36, 83, 51, 59, 80, 1, 75, 80, 87, 42, 28, D
HOUSE, 29, US, 1, 58, 93, 26, 66, 29, 64, 64, 59, 42, 34, 29, 20, 60, 27, 47, 52, 34, 6, 88, 17, 57, 99, 4
9, 58, 74, 30, 97, 68, 10, D
HOUSE, 17, GPS, 1, 68, 44, 88, 86, 25, 4, 50, 85, 98, 84, 14, 18, 96, 41, 17, 48, 27, D
HOUSE, 12, IMU, 1, 96, 85, 87, 46, 43, 13, 28, 41, 82, 38, 44, 95, D
HOUSE, 28, IMU, 1, 66, 39, 46, 51, 89, 82, 17, 60, 31, 11, 77, 31, 38, 5, 20, 87, 90, 59, 85, 85, 25, 13,
78, 59, 3, 23, 54, 61, D
HOUSE, 30, US, 3, 57, 30, 81, 55, 34, 71, 89, 51, 31, 20, 62, 60, 52, 53, 17, 24, 92, 7, 35, 77, 93, 60, 9
0, 71, 19, 93, 94, 26, 6, 10, D
HOUSE, 3, US, 2, 41, 49, 70, D
==4212==
==4212== HEAP SUMMARY:
==4212==   in use at exit: 0 bytes in 0 blocks
==4212==   total heap usage: 977 allocs, 977 frees, 17,149 bytes allocated
==4212==
==4212== All heap blocks were freed -- no leaks are possible
==4212==
==4212== For counts of detected and suppressed errors, rerun with: -v
==4212== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
paulo@pulonet:~/Documents/TP4/Franco$ valgrind ./simulador --MDL modelo.mdl --FM
```

Figura 7: Ejecución con valgrind



```
paulo@pulonet: ~/Documents/TP4/Franco
File Edit Tabs Help
==4212== For counts of detected and suppressed errors, rerun with: -v
==4212== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
paulo@pulonet:~/Documents/TP4/Franco$ valgrind ./simulador --MDL modelo.mdl --FM
T csv --out salida.txt
==4216== Memcheck, a memory error detector
==4216== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==4216== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==4216== Command: ./simulador --MDL modelo.mdl --FMT csv --out salida.txt
==4216==
==4216==
==4216== HEAP SUMMARY:
==4216==   in use at exit: 0 bytes in 0 blocks
==4216==   total heap usage: 1,187 allocs, 1,187 frees, 22,504 bytes allocated
==4216==
==4216== All heap blocks were freed -- no leaks are possible
==4216==
==4216== For counts of detected and suppressed errors, rerun with: -v
==4216== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
paulo@pulonet:~/Documents/TP4/Franco$ valgrind ./simulador --MDL modelo.mdl --FM
T bin --out salida.bin
==4221== Memcheck, a memory error detector
==4221== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==4221== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==4221== Command: ./simulador --MDL modelo.mdl --FMT bin --out salida.bin
==4221==
==4221==
==4221== HEAP SUMMARY:
==4221==   in use at exit: 0 bytes in 0 blocks
==4221==   total heap usage: 805 allocs, 805 frees, 18,292 bytes allocated
==4221==
==4221== All heap blocks were freed -- no leaks are possible
==4221==
==4221== For counts of detected and suppressed errors, rerun with: -v
==4221== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
paulo@pulonet:~/Documents/TP4/Franco$
```

Figura 8: Ejecución con valgrind

3. Repositorio en GitHub

https://github.com/franconastasi/Algo1_TP_Final

4. Bibliografía

- 1 https://www.tutorialspoint.com/c_standard_library/c_function_rand.htm