



**FACULTAD
DE INGENIERIA**

Universidad de Buenos Aires

95.11 - Algoritmos y Programación I

Trabajo práctico Final

Ono, Karen Tamara onokarent@gmail.com
Fernández Rivera, Jorge jorgefn12@gmail.com

Fecha de entrega: 6/8/2018

Indice

1. Introducción	2
2. Desarrollo	2
2.1. Compilación del programa	2
2.2. Ejecución del programa	2
3. Funcionamiento del programa	3
3.1. Validación de argumentos	4
3.2. Crear y cargar listas	5
4. Alternativas consideradas	5
5. Archivos de prueba utilizados	6
6. Resultados de ejecución	6
7. Fuga de memoria	11
8. Problemas encontrados	11
9. Bibliografía	12

1. Introducción

En el presente trabajo práctico se pone en práctica los conocimientos adquiridos sobre: arreglos/vec-tores, memoria dinámica, argumentos en línea de órdenes (CLA), estructuras, archivos, modularización, punteros a función y tipos de dato abstractos con el objetivo de implementar un intérprete que nos permita ejecutar instrucciones de un lenguaje de máquina inventado.

2. Desarrollo

2.1. Compilación del programa

Para compilar el programa, en la terminal de GNU/Linux solo se deberá ingresar:

```
make
```

Dado que se creo un archivo Makefile para lograr que la compilación del mismo sea más simple. Al ingresar este comando en la terminal, este archivo Makefile compila el programa con el siguiente prototipo

```
$ (CC) $(CFLAGS) $(OBJECTS) -o simpletron
```

Donde:

- `$(CC)` = `gcc`: es el compilador utilizado
- `$(CFLAGS)` = `-ansi -Wall -pedantic`
 - La opción `-ansi -pedantic` devuelve avisos de las violaciones del estándar ANSI.
 - `-Wall` nos mostrará todos los avisos que produzca el compilador, no solamente los errores. Los avisos nos indican dónde y/o porqué podría surgir algún error en nuestro programa. Si `-Wall` no produce avisos, es debido a que el programa es completamente válido.
- La opción `-o` especifica el fichero de salida para el código, normalmente es el último argumento en la línea de comandos. Si el mismo se omite, creará un archivo por defecto llamado `a.out`.
- `simpletron` es el nombre del ejecutable que se crea

Esta compilación muestra algunos warnings que tienen que ver con el encapsulamiento de los tda y de algunos casteos.

2.2. Ejecución del programa

Para ejecutar el programa deberá seguir el siguiente prototipo, respetando el orden de los argumentos:

```
./simpletron -m N -f FMT [Archivo1 Archivo2]
```

Donde:

- | | |
|-----------------------------|---|
| -m N, -memoria N | No es obligatorio. Indica la cantidad de memoria que se quiere reservar. Si no se ingresa el argumento, se guardan 50 palabras por defecto
-m N, -m: se guardan N cantidad de palabras. El mismo debe ser un entero positivo.
-m, -memoria: se guardan 50 palabras |
| -f FMT, -formato FMT | No es obligatorio. Indica cómo debe imprimirse el archivo de salida, como un texto o un archivo binario.
-f bin: el formato de salida será un binario
-f txt: el formato de salida será un texto |
| [Archivo1...] | Se pueden ingresar más de un archivo. El nombre del archivo debe ser antecedido por <code>t</code> : en el caso de que se trate de un archivo de texto ó <code>b</code> : en el caso de que sea un archivo binario |

El listado de archivos a procesar se espera que se ingresen luego de verificar que ninguno de los anteriores argumentos haya sido ingresado de acuerdo a las especificaciones

En el caso de que el usuario necesita ayuda ejecutando el programa puede ingresar:

`./run -h` ó `./run -help`

3. Funcionamiento del programa

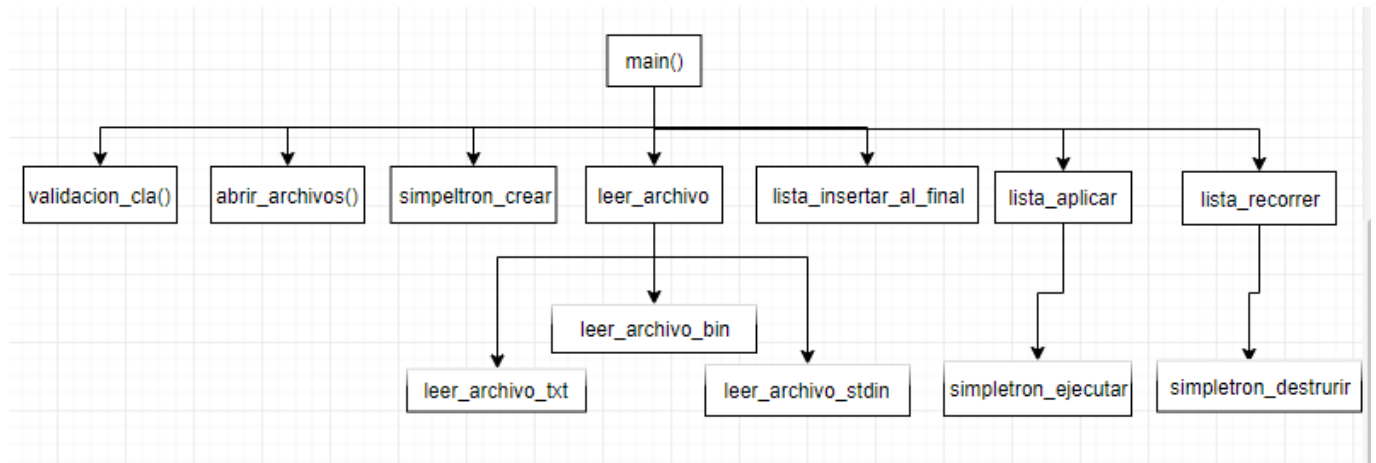


Figura 1: Diagrama de la estructura funcional del programa

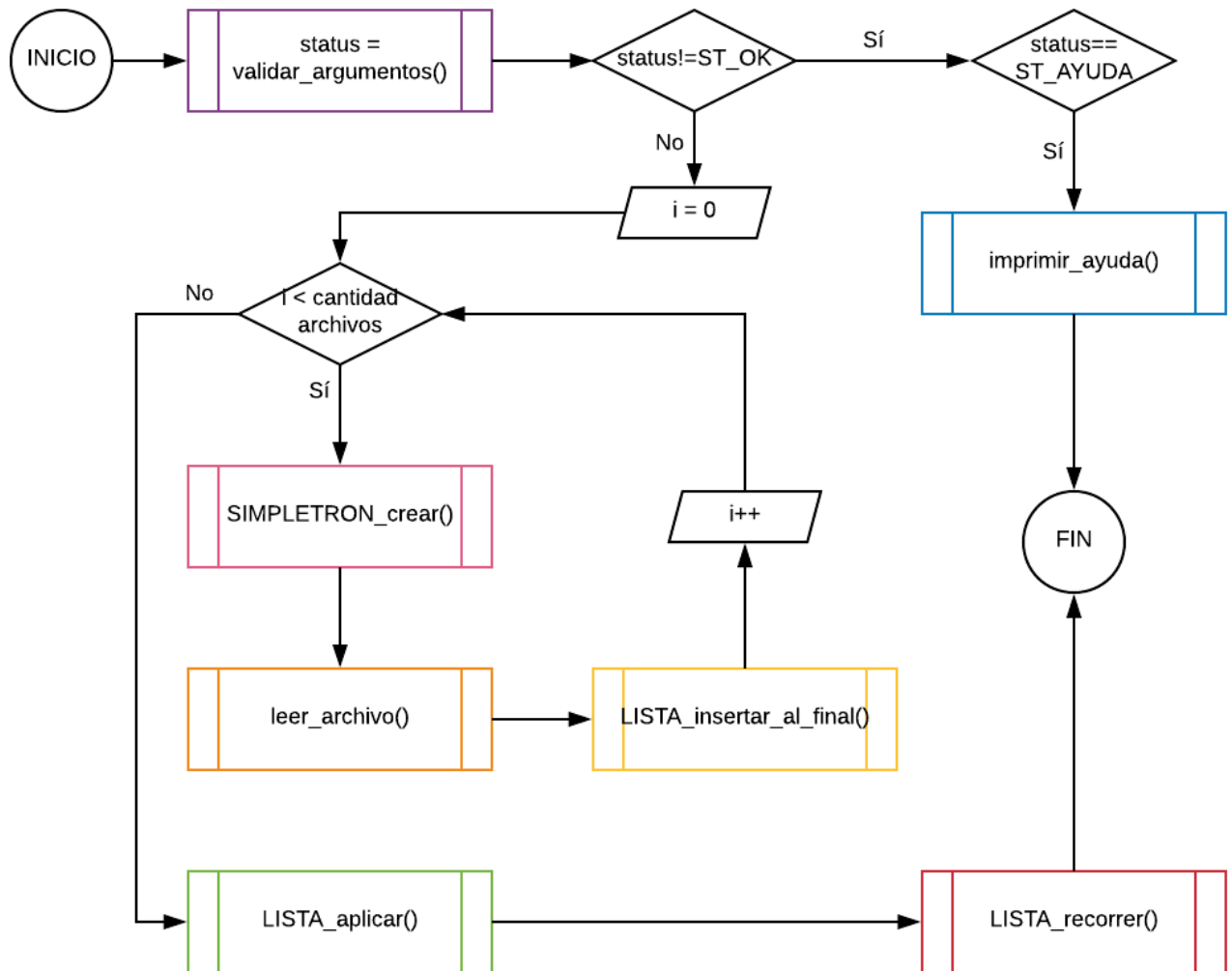


Figura 2: Diagrama de flujo del programa. Se omiten funciones específicas y otras muy ramificadas como las de destrucción y cierre de archivos

3.1. Validación de argumentos

La función `validar_argumentos()` verifica lo siguiente:

Ayuda: en el caso de que el usuario pide ayuda, se pide que solo se ingrese el flag de ayuda. Se llama a la función `imprimir_ayuda()` y luego el programa finaliza

Memoria: se pueden dar los siguientes casos:

- `./simpletron`: se toma por default una memoria de 50
- `./simpletron -m a100`: la función devuelve un error. No se acepta que se ingresen caracteres alfabéticos.
- `./simpletron -m 150a`: la función devuelve un error. No se acepta que se ingresen caracteres alfabéticos.
- `./simpletron -m 150,1`: la función devuelve un error. No se acepta que se ingresen cifras decimales.

- `./simpletron -m -50`: la función devuelve un error. La memoria debe ser positiva y mayor a cero.

se verifica que si el usuario ingresó el flag de memoria, sea seguido de un entero positivo.

Formato: se verifica que el formato ingresado sea `txt` ó `bin`, caso contrario la función devuelve un mensaje de error.

Archivos: el usuario debe especificar el formato en el que será leído el archivo de entrada ingresando antes del nombre del archivo `t` : para que el archivo sea leído como un texto o `b` : para que el archivo sea procesado como un archivo binario.

3.2. Crear y cargar listas

Se usan listas polimórficas simplemente enlazadas para albergar las distintas instancias del `simpletron`. En cada nodo se inserta un puntero a dicho `simpletron` que incluye una memoria de palabras correspondiente al archivo en particular.

Todas las palabras de cada archivo se cargan a la memoria de cada `simpletron` usando un TDA Vector con sus primitivas. El procesamiento de la carga de palabras será diferente basado en los argumentos pasados por línea de comandos.

4. Alternativas consideradas

3.1 Alternativas consideradas en `validar_argumentos()`:

En primer lugar se discutió si se debían tomar argumentos posicionales o no. Durante trabajos anteriores se creía que la primera opción comparada con la segunda, no poseía ventajas substanciales. Sin embargo, para implementar la segunda opción de forma correcta, se requerirían de muchas más validaciones para detectar flags, incoherencias, rechazar argumentos inválidos, etc. Si no se implementa de forma correcta, el ingreso de argumentos se siente anti-natural e inconsistente. Las ventajas de usar argumentos posicionales, son la simplicidad y legibilidad del código, un ingreso de argumentos más consistente y mayor velocidad de procesamiento. Considerando que el tipo de programa que desarrollamos, no presta tanta atención a la interacción con el usuario, sino al procesamiento de datos, nos decidimos por ésta última opción.

Respecto al método de procesamiento, las opciones eran recorrer los argumentos con un ciclo y comparando cada uno con un vector de argumentos válidos, o recorrerlos de forma lineal con un cursor. Las ventajas de la primera opción son que probablemente se escriba menos, y sea más simple/legible. La segunda opción nos permite validar argumentos de forma más personalizada, al tener control de los movimientos que realiza el cursor. Por esto último, también puede resultar más rápida. También resulta más escalable ya que disponer de un ciclo `for` no siempre será válido para todos los argumentos que puedan añadirse en el futuro. Se tomó la segunda opción.

3.2 Alternativas consideradas en `ejecutar_codigo()`:

Respecto al aspecto funcional, se decidió trabajar con un elemento a la vez. Es decir, se accede a la memoria del `simpletron` y se obtiene la instrucción, opcode y operando según el `programa_counter` vaya necesitando alguna en específico. Esto es debido a que en el dump, sólo se requieren los últimos registros usados, y también, a que es más eficiente. Por otro lado, esta decisión pudiera considerarse un inconveniente si, por ejemplo, se quisiese un registro de las operaciones ejecutadas (no es el caso para este TP).

Otra cuestión que se tuvo en cuenta es la de añadir la variable `instruccion` a la estructura del `Simpletron`, ya que su presencia haría más legible algunas partes del código de ejecución y de impresión del dump. Con esto, se sacrifica una parte del desempeño del programa, cuya magnitud depende de a dónde se lo implemente.

Se decidió desacoplar las funciones de ejecutar código de las de impresión de dump y de otros mensajes para favorecer la reutilización del código.

También se decidió implementar los punteros a función desde vectores estáticos que vinculan punteros a función con su respectivo código opcode. Otra alternativa pudo haber sido implementar un vector de estructuras que contiene opcode y funciones, o una estructura de vectores de opcode y funciones.

3.3 Alternativas consideradas para las funciones de creación y carga de listas y vectores

Desde el apartado funcional, se utilizó recursividad cuando se pudo para mantener sencillez y naturalidad en la lectura del código. Esto conlleva a una gran pérdida del desempeño. Sin embargo, lo creímos necesario para facilitar el trabajo en grupo y la comprensión de las funciones que en principio parecieran intrincadas.

5. Archivos de prueba utilizados

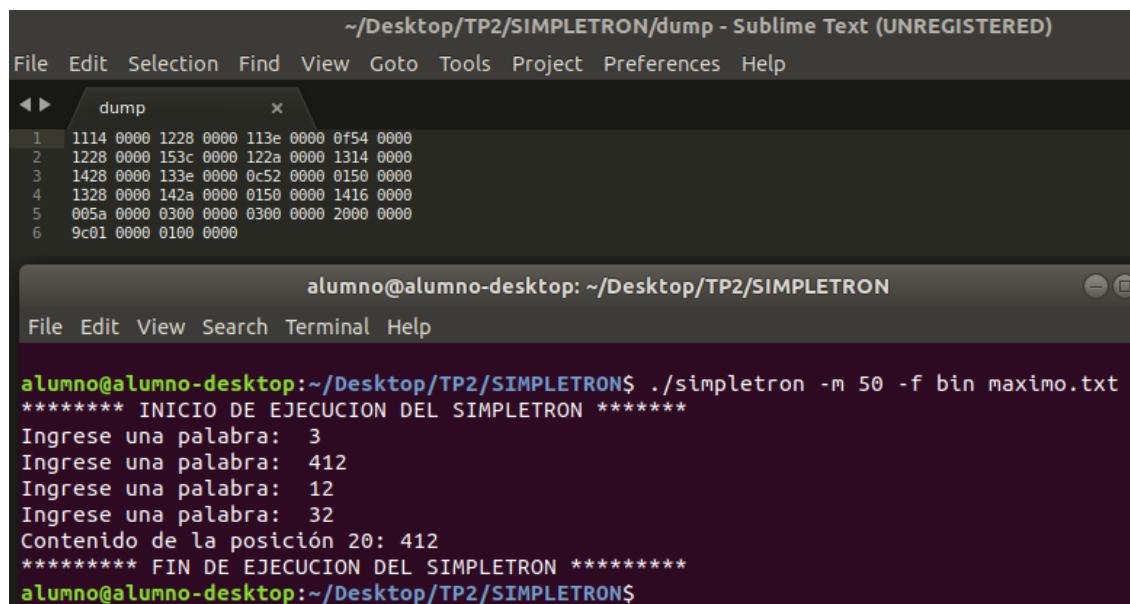
Los archivos de prueba utilizados para la sección Resultados de ejecución son: `suma.txt`, `resta.txt`, `promedio.txt`, `maximo.txt` y `maximo.sac`. Todos ellos se encuentran añadidos junto con los códigos fuente en la entrega de este informe.

6. Resultados de ejecución

A continuación se incluyen las capturas de pantalla bajo condiciones normales de ejecución:

```
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron
*** ;Bienvenido a la Simpletron! ***
*** Por favor, ingrese su programa una ***
*** instrucción (o dato) a la vez. Yo ***
*** escribiré la ubicación y un signo de ***
*** pregunta (?). Luego usted ingrese la ***
*** palabra para esa ubicación. Ingrese ***
*** -99999999 para finalizar: ***
00 ? 100009
01 ? 100010
02 ? 200009
03 ? 300010
04 ? 210011
05 ? +0110011
06 ? 450000
07 ? 0
08 ? 0000
09 ? 0
10 ? 00
11 ? 0
12 ? -99999999
***** INICIO DE EJECUCION DEL SIMPLETRON *****
Ingrese una palabra: 214
Ingrese una palabra: 33
Contenido de la posición 11: 247
***** FIN DE EJECUCION DEL SIMPLETRON *****
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$
```

Figura 3: Ejecución del programa suma.txt mediante ingreso por stdin



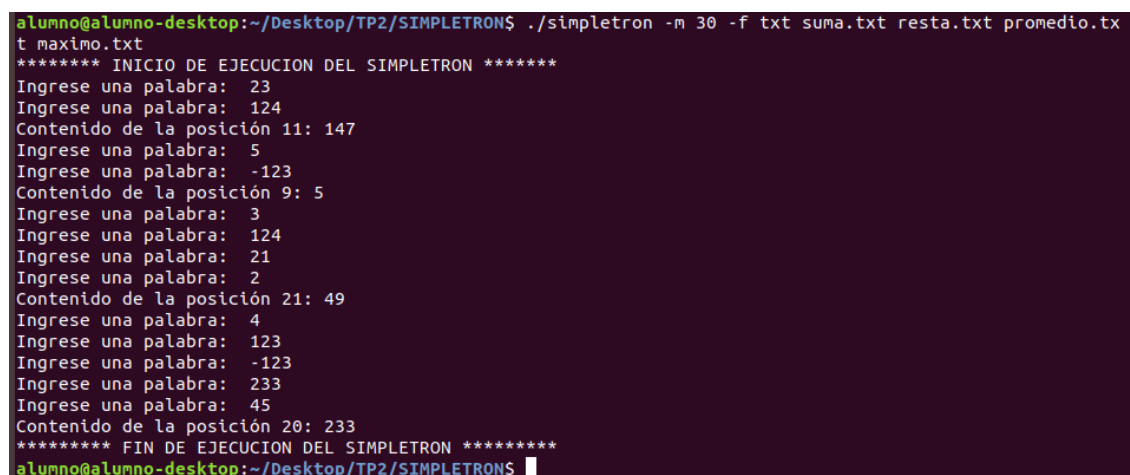
The image shows a Sublime Text editor window titled `~/Desktop/TP2/SIMPLETRON/dump - Sublime Text (UNREGISTERED)`. The editor displays a memory dump with 6 lines of hexadecimal data. Below the editor, a terminal window titled `alumno@alumno-desktop: ~/Desktop/TP2/SIMPLETRON` shows the execution of the `simpletron` program. The terminal output includes the command `./simpletron -m 50 -f bin maximo.txt`, followed by a series of prompts for words and positions, and a final message indicating the end of execution.

```
~/Desktop/TP2/SIMPLETRON/dump - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

1 1114 0000 1228 0000 113e 0000 0f54 0000
2 1228 0000 153c 0000 122a 0000 1314 0000
3 1428 0000 133e 0000 0c52 0000 0150 0000
4 1328 0000 142a 0000 0150 0000 1416 0000
5 005a 0000 0300 0000 0300 0000 2000 0000
6 9c01 0000 0100 0000

alumno@alumno-desktop: ~/Desktop/TP2/SIMPLETRON
File Edit View Search Terminal Help

alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron -m 50 -f bin maximo.txt
***** INICIO DE EJECUCION DEL SIMPLETRON *****
Ingrese una palabra: 3
Ingrese una palabra: 412
Ingrese una palabra: 12
Ingrese una palabra: 32
Contenido de la posición 20: 412
***** FIN DE EJECUCION DEL SIMPLETRON *****
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$
```

Figura 4: Ejecución del programa `maximo.txt` por entrada de archivo txt y dump binario

The image shows a terminal window titled `alumno@alumno-desktop: ~/Desktop/TP2/SIMPLETRON`. The terminal output shows the command `./simpletron -m 30 -f txt suma.txt resta.txt promedio.txt maximo.txt` being executed. The program then prompts for words and positions, and displays the content of the files. The output ends with a message indicating the end of execution.

```
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron -m 30 -f txt suma.txt resta.txt promedio.tx
t maximo.txt
***** INICIO DE EJECUCION DEL SIMPLETRON *****
Ingrese una palabra: 23
Ingrese una palabra: 124
Contenido de la posición 11: 147
Ingrese una palabra: 5
Ingrese una palabra: -123
Contenido de la posición 9: 5
Ingrese una palabra: 3
Ingrese una palabra: 124
Ingrese una palabra: 21
Ingrese una palabra: 2
Contenido de la posición 21: 49
Ingrese una palabra: 4
Ingrese una palabra: 123
Ingrese una palabra: -123
Ingrese una palabra: 233
Ingrese una palabra: 45
Contenido de la posición 20: 233
***** FIN DE EJECUCION DEL SIMPLETRON *****
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$
```

Figura 5: Ejecución de programas múltiples por entrada de archivo txt

```

Open [icon] dump
~/Desktop/TP2/SIMPLETRON

REGISTROS:
    acumulador:      0093
program counter:      6
    instruccion: +0450000
    opcode:          45
    operando:         000

MEMORIA:
000: 1409 140A 2809 3C0A 2A0B 160B 5A00 0000    ....(<.*...Z...
010: 0000 0017 007C 0093                      .....|..

REGISTROS:
    acumulador:      0080
program counter:      8
    instruccion: +0450000
    opcode:          45
    operando:         000

MEMORIA:
000: 1409 140A 2809 3E0A 5207 1609 5008 160A    ....(>.R...P...
010: 5A00 0005 FF85 0000                      Z.....

REGISTROS:
    acumulador:      0031
program counter:      16
    instruccion: +0450000
    opcode:          45
    operando:         000

MEMORIA:
000: 1416 2816 3E13 540C 2813 3C14 2A13 1412    ..(>.T.(<.*...
010: 2811 3C12 2A11 5001 2811 4016 2A15 1615    (<.*.P.(@.*...
020: 5A00 0093 0002 0003 0001 0031 0003    Z.....1..

REGISTROS:
    acumulador:      0000
program counter:      16

```

Plain Text

Figura 6: Impresión de dump múltiples en formato txt

A continuación se incluyen las capturas de pantalla bajo condiciones inesperadas de ejecución:

```

alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron -m
La memoria ingresada no es valida, la misma debe ser un entero positivo
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron -f
El formato de archivo de salida no es valido
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron -f bin -m 17
El archivo ingresado no se encuentra
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron asdf
El archivo ingresado no se encuentra
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron suma.txt -
El ingreso de archivos multiples junto con stdin no está soportado
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron -f 3
El formato de archivo de salida no es valido
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron -m 12a
La memoria ingresada no es valida, la misma debe ser un entero positivo
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron
*** ;Bienvenido a la Simpletron!      ***
*** Por favor, ingrese su programa una ***
*** instrucción (o dato) a la vez. Yo  ***
*** escribiré la ubicación y un signo de ***
*** pregunta (?). Luego usted ingrese la ***
*** palabra para esa ubicación. Ingrese ***
*** -99999999 para finalizar:         ***
00 ? a
La palabra ingresada no es valida
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron
*** ;Bienvenido a la Simpletron!      ***
*** Por favor, ingrese su programa una ***
*** instrucción (o dato) a la vez. Yo  ***
*** escribiré la ubicación y un signo de ***
*** pregunta (?). Luego usted ingrese la ***
*** palabra para esa ubicación. Ingrese ***
*** -99999999 para finalizar:         ***
00 ? 653331
La palabra ingresada no es valida
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ █

```

Figura 7: Ejecución de un programa con distintos errores de entrada de argumentos

```

alumno@alumno-desktop: ~/Desktop/TP2/SIMPLETRON
File Edit View Search Terminal Help
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron -m 1 suma.txt
Se superó la memoria solicitada para el simpletron
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ ./simpletron -m 200 -f txt b:maximo.sac
***** INICIO DE EJECUCION DEL SIMPLETRON *****
***** FIN DE EJECUCION DEL SIMPLETRON *****
Se trató de acceder a memoria restringida durante la ejecución
alumno@alumno-desktop:~/Desktop/TP2/SIMPLETRON$ █

```

Figura 8: Se trata de ejecutar un archivo con memoria insuficiente

```
File Edit View Search Terminal Help
alumno@alumno-desktop:~/Desktop/TP2/main$ ./simpletron -m 1 suma.txt
Prototipo de ejecucion: ./simpletron -m N -f FMT [Archivo1 Archivo2 Archivo3..
.]
MEMORIA:
-m N: -m es el flag indicador de memoria y N debe ser un numero entero. No
es obligatorio su ingreso. En caso de no ingresarse N sera 50

FORMATO:
-f FMT: -f es el flag indicador del formato de salida donde FMT debe ser TXT
o BIN. No es obligatorio su ingreso. Por default sera TXT

ARCHIVOS:
Los nombres de archivo de entrada deberan estar anteceditos por 't:' o 'b:' s
i el archivo debiera leerse como un archivo de texto o un archivo binario respe
ctivamente. En caso de omitirse esta especificacion, el archivo sera leído com
o un archivo de texto. En caso de que no se ingresen nombres de archivos, la e
ntrada de datos sera por stdin

alumno@alumno-desktop:~/Desktop/TP2/main$
```

Figura 9: Se trata de ejecutar un archivo binario cuyo lenguaje es incompatible con el programa

7. Fuga de memoria

Para verificar que el programa no tiene fugas de memoria se utiliza valgrind. En la terminal se ejecuta lo siguiente:

```
valgrind -leak-check=yes ./simpletron
```

8. Problemas encontrados

Problemas encontrados al escribir la función `validacio_cla()`:

Al comenzar esta función, se tenía planeado realizar optimizaciones en la ejecución y tratar de reducir el procesamiento al mínimo. El problema que se encontró al usar esta metodología fue que algunos incrementos en las variables de iteración se asignaban en un momento no deseado. La solución que se realizó fue hacer un seguimiento a cada una de las variables y ver que se debían realizar operaciones de post-incremento, pre-incremento, o ninguna de las anteriores, de forma selectiva. Otro problema que se halló fue el de validar correctamente los argumentos para el ingreso de palabras por stdin. Ya que su aparición merecía un procesamiento particular, para la apertura/cierre de archivos y de formatos en los que se puede ingresar palabras. Este inconveniente nos resultó en obtener configuraciones de archivos de salida, de entrada y de formatos, algo inconsistente. La solución adoptada fue la de utilizar flags que detectaban la presencia de un ingreso por stdin y procesar cada sección adecuadamente.

Problemas encontrados al escribir las funciones de creación y carga de listas y vectores:

El estar restringidos a usar funciones primitivas, nos limitó la libertad para realizar un seguimiento al código. En muchas ocasiones nos encontramos sin ideas para la resolución de un bug, que parecía perdido entre todas las líneas de código. Esto es especialmente importante a la hora de manejar punteros, ya que la referencia la perdíamos muy frecuentemente. La única solución que teníamos era la de revisar línea por línea atentamente.

Otro gran problema relacionado con lo anterior tuvo que ver con las fugas de memoria. La estrategia que tuvimos fue la de aprender a usar gdb y valgrind con su respectiva documentación. Luego resultó muy sencillo encontrar exactamente dónde se encontraban las fugas.

9. Bibliografía

[1] *Cómo programar en C, C++ y Java*, Deitel & Deitel, 4^a edición