

# TP 0: El Ambiente de Trabajo

## Taller de Programación

Agustina Barbetta 96528

15 de Marzo  
1er. Cuatrimestre 2016

### 1. Introducción

Este trabajo práctico apunta a repasar todos aquellos temas de C y programación estructurada que fueron aprendidos en materias anteriores. El mismo, no hará foco en la programación propiamente dicha, sino en el correcto entendimiento y uso de las herramientas de desarrollo, especialmente SERCOM. El desarrollo se encuentra dividido en pasos, cada uno de los cuales cuenta con una sección especial en el presente informe.

### 2. Pasos

A continuación se documentan los pasos del trabajo práctico.

#### 2.1. Comenzando

- **Preparar un ambiente de trabajo local:** Para la realización de este trabajo utilizaré Ubuntu 15.10 con Valgrind y GDB instalados.
- **Compilación y prueba (ambos en forma local) de una aplicación ejemplo, al estilo “hola mundo”:** Escribí el programa y lo compilé como hago usualmente con `$ gcc -Wall -g p1.c -o p1`, luego lo corrí con `./p1` y obtuve la respuesta esperada.
- **Instalar Valgrind (en forma local) y probar la misma aplicación:** Esta vez, corrí el programa con `$ valgrind --leak-check=full ./p1`.
- **Documentación:**
  - **Ver capturas de pantalla de la ejecución del aplicativo (con y sin Valgrind). ¿Qué es Valgrind?** Las capturas se encuentran en el anexo correspondiente a este paso. Valgrind es una herramienta de software libre que ayuda a detectar errores de manejo de memoria, como por ejemplo; el acceso a memoria fuera del heap, el uso de variables no inicializadas, la ausencia o incorrecta liberación de memoria, etc.
  - **¿Qué representa sizeof()? ¿Cuál sería el valor de salida de sizeof(char) y sizeof(int)?** sizeof es un operador de tiempo de compilación que puede ser utilizado para calcular el tamaño de cualquier objeto. Las expresiones `sizeof(char)` y `sizeof(int)` producen un número entero correspondiente al tamaño en bytes del tipo especificado. Suponiendo que la arquitectura es de 32 bits, los chars ocupan un byte, mientras que los enteros ocupan 4. La salida de estas expresiones será 1 y 4 respectivamente.
  - **El sizeof() de una struct de C es igual a la suma del sizeof() de cada uno de los elementos de la misma”. Explique la validez o invalidez de dicha afirmación.** Cuando se aplica a una estructura, el resultado de esta expresión es el número de bytes en el objeto, incluyendo cualquier alineación. Una *alineación de datos* significa poner los datos a una dirección de memoria igual a un múltiplo del tamaño de la palabra, lo que aumenta el rendimiento del sistema debido a la forma en que la CPU accede a la memoria. Para alinear los datos, puede que sea necesario insertar algunos bytes vacíos entre el final del último dato y el comienzo del siguiente, llamamos *padding* a este espacio.

## 2.2. SERCOM - Error de compilación

Se entregó el código fuente del enunciado vía SERCOM y el mismo falló como se esperaba. A continuación se analiza el resultado:

- **Los errores reportados por SERCOM:** Los errores reportados fueron los siguientes:

```
CC p2.o
p2.c: In function 'main':
p2.c:10: error: implicit declaration of function 'ztrcpy'
p2.c:14: error: implicit declaration of function 'malloc'
cc1: warnings being treated as errors
p2.c:14: error: incompatible implicit declaration of built-in
function 'malloc'
make: *** [p2.o] Error 1
```

La salida indica que, para la función *main*:

- En la línea 10 del .c se llama a una función *ztrcpy* de la cual el compilador no encuentra declaración (De aquí el error *implicit declaration of function*).
- En la línea 14 del .c se llama a la función *malloc* pero el archivo de cabecera que contiene su declaración (*stdlib.h*) no está incluido.
- Por último, el compilador avisa que el *malloc* utilizado es incompatible con su declaración; `int malloc()`, distinta a la de *stdlib.h*; `void *malloc(size_t)`, que se intenta utilizar aquí.

Las capturas de pantalla se encuentran en el anexo correspondiente a este paso.

- **¿Fueron errores del compilador o del linker?** Las advertencias fueron del compilador, quien no encontró declaraciones para las funciones *ztrcpy* y *malloc*. Sin embargo, el compilador logra construir el archivo objeto. Es el linker quien no encuentra la definición de estas funciones y no puede construir el ejecutable.

## 2.3. SERCOM - Normas de programación y código de salida

Se corrigieron los errores mencionados anteriormente y se volvió a entregar el código. Se generó el ejecutable con éxito pero falló el chequeo de normas de codificación.

- **Observar la falla reportada por la prueba 1, indicando el código de retorno inesperado**  
*Se esperaba terminar con un código de retorno 1 pero se obtuvo 2.*
- **Documentación:** Las capturas de pantalla se encuentran en el anexo correspondiente a este paso.
  - **Ver captura de pantalla indicando la correcta generación del ejecutable**
  - **Ver captura de pantalla mostrando los problemas de estilo detectados** Los errores encontrados fueron los siguientes:

```
./p3.c:5: Extra space after ( in function call
[whitespace/parens] [4]
./p3.c:11: Extra space after ( in function call
[whitespace/parens] [4]
./p3.c:11: Extra space before ) [whitespace/parens] [2]
./p3.c:11: Almost always, snprintf is better than strcpy
[runtime/printf] [4]
./p3.c:12: Extra space after ( in function call
[whitespace/parens] [4]
./p3.c:12: Extra space before ) [whitespace/parens] [2]
./p3.c:13: Missing space before ( in if( [whitespace/parens] [5]
./p3.c:17: Missing space before ( in while(
[whitespace/parens] [5]
./p3.c:20: Missing space before ( in if( [whitespace/parens] [5]
./p3.c:21: Extra space after ( in function call
```

```

[ whitespace/parens ] [4]
./p3.c:21: Extra space before ) [ whitespace/parens ] [2]
Done processing ./p3.c
Total errors found: 11

```

La salida indica que, para el archivo *p3.c*:

- En la línea 5 hay un espacio de más, se debería corregir por `int main(int argc, char *argv[])`.
- En la línea 11 hay espacios de más antes y después de cada paréntesis, además de una recomendación para utilizar `snprintf` en lugar de `strcpy`. La corrección podría ser `strcpy(nombre, argv[1]);`.
- En la línea 12 se repite el problema anterior de los paréntesis, se debería corregir por `fp = fopen(nombre, r");`.
- En la línea 13 se pide dejar un espacio entre el `if` y el paréntesis de su condición, se debería corregir por `if ( fp == NULL ) return 2;`.
- En la línea 17 se pide dejar un espacio entre el `while` y el paréntesis de su condición, se debería corregir por `while ( !feof(fp) )`.
- En la línea 20 se repite el problema anterior del paréntesis faltante, se debería corregir por `if ( c != EOF )`.
- En la línea 21 se repite el problema anterior de los espacios extras antes y después de los paréntesis, se debería corregir por `printf("%c", (char) c);`.
- **Ver captura de pantalla indicando el error reportado en la prueba 1.** El error reportado es: *Se esperaba terminar con un código de retorno 1 pero se obtuvo 2*. Se produce porque la función `fopen()` devuelve `null` al no encontrar el archivo de nombre *no-existo*. El programa termina devolviendo 2 según `if ( fp == NULL ) return 2;`.

## 2.4. SERCOM - Pérdida de memoria

Se corrigió el `.c` de acuerdo a las normas de codificación y se reemplazaron los números mágicos por constantes. Además, se modificó el código de retorno 2 para el caso de archivo inexistente por 1. Por último, se volvió a entregar el código. Se observa que la primera prueba es exitosa y la salida de normas de codificación sólo muestra la sugerencia de `snprintf`.

- **Observar el resultado de la prueba 2. A nivel funcional ésta terminó correctamente, pero Valgrind reporta problemas.** *Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.*
- **Documentación:** Las capturas de pantalla se encuentran en el anexo correspondiente a este paso.
  - **Ver captura de pantalla indicando la nueva salida del chequeo de normas de codificación.**
  - **Ver captura de pantalla indicando la correcta finalización de la prueba 1.**
  - **Ver captura de pantalla indicando los problemas reportados por Valgrind.** La salida de Valgrind especifica que:
    - En la línea 20 del `.c` hay una llamada a `malloc` debido a la cual se pierden 4 bytes en un bloque de memoria. Al ejecutar `buffer = malloc(sizeof(int));` se reservan 4 bytes del heap y se devuelve un puntero al primero de ellos (Vale aclarar que el puntero se asigna a la variable `buffer` de tipo `char*`, la cual vive en el stack), pero este bloque de memoria nunca se libera al finalizar su uso. Para solucionar esta pérdida se debe agregar `free(buffer);` antes del retorno de la función.
    - En la línea 17 del `.c` hay una llamada a `fopen` debido a la cual se abre el archivo `archivo-corto.txt` (cuyo nombre se recibe por parámetro) y nunca se vuelve a cerrar. Para solucionar este error se debe agregar `fclose(fp);` antes del retorno de la función.

## 2.5. SERCOM - Escrituras fuera de rango

Se corrigieron los errores reportados por Valgrind y se volvió a entregar el código. Se observa que las pruebas 1, 2 y 4 son correctas.

- **Observar que la prueba 3 presenta un error poco claro. Intentar determinar el origen de la falla (observar en detalle el `strcpy` y la línea de comandos ejecutada)** El error es: *Se esperaba terminar con un código de retorno 0 pero se obtuvo 134. La salida estándar no coincide con lo esperado (archivo `__stdout__.diff`).* A continuación se explica el problema.
- **Documentación:** Las capturas de pantalla se encuentran en el anexo correspondiente a este paso.

- **Ver captura de pantalla de la salida de Valgrind sobre la prueba 2.**
- **Explicar en detalle el problema reportado. ¿Podría solucionarse utilizando `strncpy` en lugar de `strcpy`? ¿Podría ayudar Valgrind a su diagnóstico? (Ver captura de pantalla del mismo).** La falla se debe a que se declara un arreglo de 20 chars, para luego copiar el nombre del archivo con `strcpy`, pero el nombre del .txt excede la longitud de este arreglo. `strcpy` no te detiene en el límite del arreglo, por lo que termina sobrescribiendo bytes de memoria adyacentes.

Si se usara `strncpy` se podrían determinar los n chars a copiar en el arreglo, evitando así un *buffer overflow*. Sin embargo, el nombre del archivo estaría incompleto, la función `fopen` devolvería NULL y el programa finalizaría con un error de código 1.

Valgrind puede ayudar a su diagnóstico con el siguiente mensaje de error descriptivo: `strcpy_chk: buffer overflow detected ***: program terminated`, el cual se encuentra presente en la salida del SERCOM.

- **Explicar de qué se trata un *segmentation fault* y un *buffer overflow*.**
  - **Segmentation fault:** Es un tipo específico de error causado por el acceso a memoria que "no nos pertenece". Es un mecanismo de ayuda que impide la corrupción de memoria y la introducción de errores difíciles de encontrar. Se puede provocar al tratar de acceder a una variable liberada, escribiendo en una porción de memoria de solo lectura, etc.
  - **Buffer overflow:** Se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada (*buffer*). Si dicha cantidad de datos es superior a la memoria asignada al *buffer*, los bytes extras se almacenan en zonas de memoria adyacentes, sobrescribiendo su contenido original.
- **Indicar el contenido de los archivos de entrada utilizados en las pruebas 2 y 4.**
  - Para la prueba 2, `archivo-corto.txt` contiene:

La estructura de estos cuentos (y de todos los relativos a Holmes) es similar: Sherlock está en su casa de Baker Street, muchas veces en compañía de su amigo, cuando de repente aparece un personaje que viene a plantearle un problema para el que necesita ayuda. Otras veces esta noticia llega a él a través del periódico. Los casos son resueltos por la lógica y el razonamiento del famoso detective. Son cuentos de misterio, donde interviene la intriga y la aventura, unido al análisis psicológico de sus personajes.
  - Para la prueba 2, `archivo-largo.txt` contiene:

Rene Geronimo Favalaro (La Plata, Argentina, 12 de julio de 1923 - Buenos Aires, Argentina, 29 de julio de 2000) fue un prestigioso médico cirujano torácico argentino, reconocido mundialmente por ser quien realizó el primer *bypass* cardíaco en el mundo. Estudió medicina en la Universidad de La Plata y una vez recibido, previo paso por el Hospital Policlínico, se mudó a la localidad de Jacinto Arauz para reemplazar temporalmente al médico local, quien tenía problemas de salud. A su vez, leía bibliografía médica actualizada y empezó a tener interés en la cirugía torácica. A fines de la década de 1960 empezó a estudiar una técnica para utilizar la vena safena en la cirugía coronaria. A principios de la década de 1970 fundó la fundación que lleva su nombre. Se desempeñó en la Conadep, condujo programas de televisión dedicados a la medicina y escribió libros. Durante la crisis del 2000, su fundación tenía una gran deuda económica y le solicitó ayuda al gobierno sin recibir respuesta, lo que lo indujo a suicidarse. El 29 de julio de 2000, después de escribir una carta al Presidente De la Rúa criticando al sistema de salud, se quitó la vida de un disparo al corazón.

- **Indicar la línea de comandos utilizada para la ejecución de la prueba 3.** El comando para ejecutar esta prueba es: `$ ./p5 soy-un-archivo-con-nombre-largo.txt`

## 2.6. SERCOM - Entrada estándar

Se corrigió el `.c` utilizando `argv[1]` directamente en el *fopen*, eliminando *buffer* y el *strcpy*. Se volvió a realizar la entrega y se verificó que las cuatro primeras pruebas fueran exitosas. Cabe aclarar que las normas de codificación fueron correctas, debido a la eliminación del *strcpy*. Se observa que la prueba 5 falla, debido a que el aplicativo aún no soporta un requerimiento funcional del enunciado (leer desde la entrada estándar cuando no se especifica un *file* en la línea de comandos).

- **Documentación:** Las capturas de pantalla se encuentran en el anexo correspondiente a este paso.
  - Ver captura de pantalla con la correcta salida del chequeo de normas de codificación.
  - Ver captura de pantalla con el resultado de la prueba 5.

## 2.7. SERCOM - Entrega exitosa

Se agrega la funcionalidad faltante. El código fuente se simplifica si se tiene en cuenta la existencia de *stdin*, variable del tipo `FILE*` automáticamente abierta y disponible al iniciar el aplicativo.

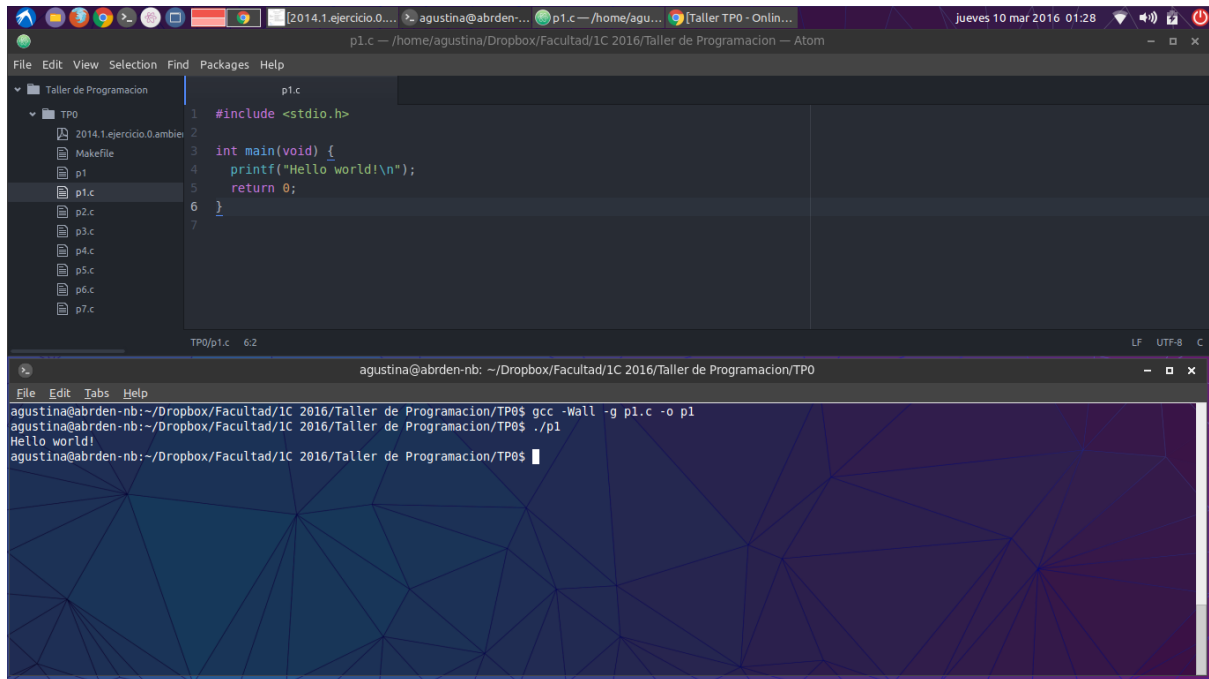
- **Investigar el uso de los caracteres `>` y `<` en la línea de comando.** El caracter `>`, utilizado como `$ ./prog > salida.txt`, guarda la salida del programa ejecutado en el archivo de texto indicado. Si el archivo no existe, lo crea. Análogamente, el caracter `<` utilizado como `$ ./prog < entrada.txt`, toma el contenido del archivo de texto indicado como entrada. Si el archivo no existe, se produce un error.
- **Documentación:** Las capturas de pantalla se encuentran en el anexo correspondiente a este paso.
  - Ver captura de pantalla mostrando la entrega exitosa, en color verde.
  - Ver captura de pantalla mostrando la ejecución local de la prueba 5 sin el uso del teclado. Se utiliza el comando `$ ./p7 < entrada.txt`.
  - Ver captura de pantalla mostrando la ejecución local de la prueba 2, pero redireccionando la salida estándar a un archivo denominado 'salida.txt'. Se utiliza el comando `$ ./p7 archivo-corto.txt > salida.txt`.

Se vuelve a realizar la entrega y, finalmente, se observa que todas las pruebas (y la entrega) son exitosas.

### 3. Apéndices

A continuación se presentan todas las capturas de pantalla tomadas durante la realización del trabajo práctico.

#### 3.1. Comenzando



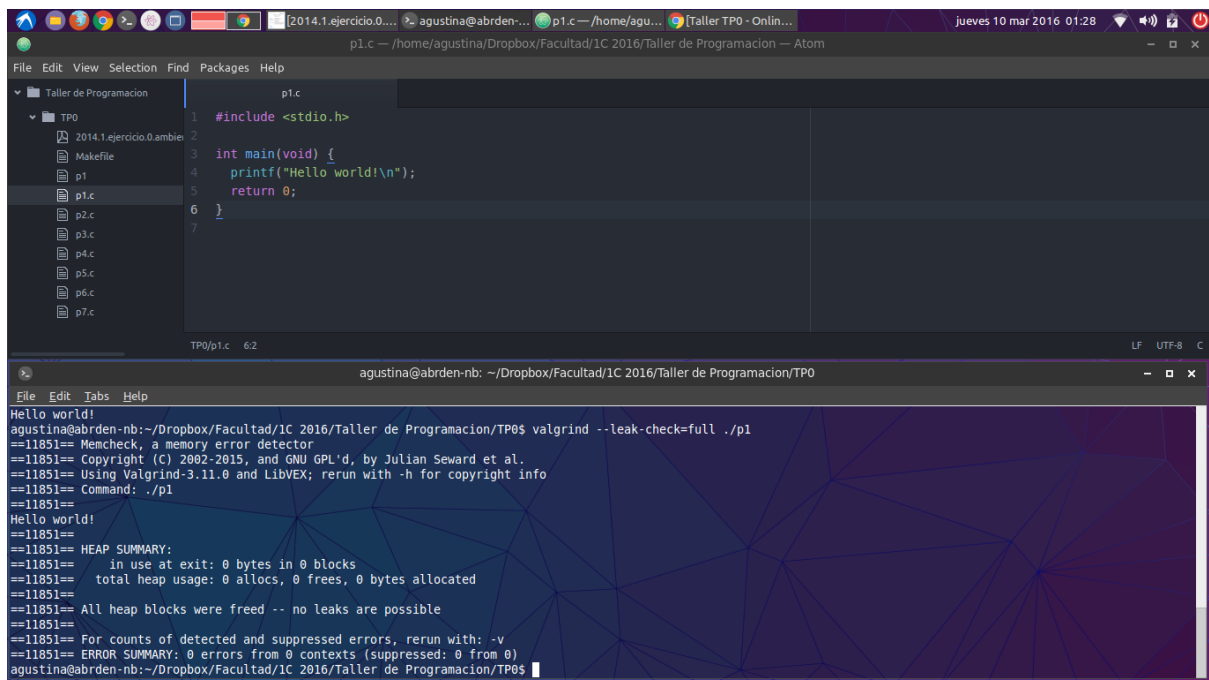
The screenshot shows the Atom text editor with a file named `p1.c` open. The code in the editor is:

```
1 #include <stdio.h>
2
3 int main(void) {
4     printf("Hello world!\n");
5     return 0;
6 }
7
```

Below the editor is a terminal window. The terminal shows the following commands and output:

```
agustina@abrdn-nb: ~/Dropbox/Facultad/1C 2016/Taller de Programacion/TP0
agustina@abrdn-nb:~/Dropbox/Facultad/1C 2016/Taller de Programacion/TP0$ gcc -Wall -g p1.c -o p1
agustina@abrdn-nb:~/Dropbox/Facultad/1C 2016/Taller de Programacion/TP0$ ./p1
Hello world!
agustina@abrdn-nb:~/Dropbox/Facultad/1C 2016/Taller de Programacion/TP0$
```

Figura 1: Captura de pantalla de la ejecución del aplicativo



The screenshot shows the same Atom editor with the `p1.c` file. The terminal window now shows the execution of the program using Valgrind:

```
agustina@abrdn-nb:~/Dropbox/Facultad/1C 2016/Taller de Programacion/TP0$ valgrind --leak-check=full ./p1
Hello world!
==11851== Memcheck, a memory error detector
==11851== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==11851== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==11851== Command: ./p1
==11851==
Hello world!
==11851==
==11851== HEAP SUMMARY:
==11851==   in use at exit: 0 bytes in 0 blocks
==11851==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==11851==
==11851== All heap blocks were freed -- no leaks are possible
==11851==
==11851== For counts of detected and suppressed errors, rerun with: -v
==11851== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
agustina@abrdn-nb:~/Dropbox/Facultad/1C 2016/Taller de Programacion/TP0$
```

Figura 2: Captura de pantalla de la ejecución del aplicativo con Valgrind

### 3.2. SERCOM - Error de compilación



Figura 3: Entrega a SERCOM fallida



Figura 4: Errores reportados por SERCOM

### 3.3. SERCOM - Normas de programación y código de salida

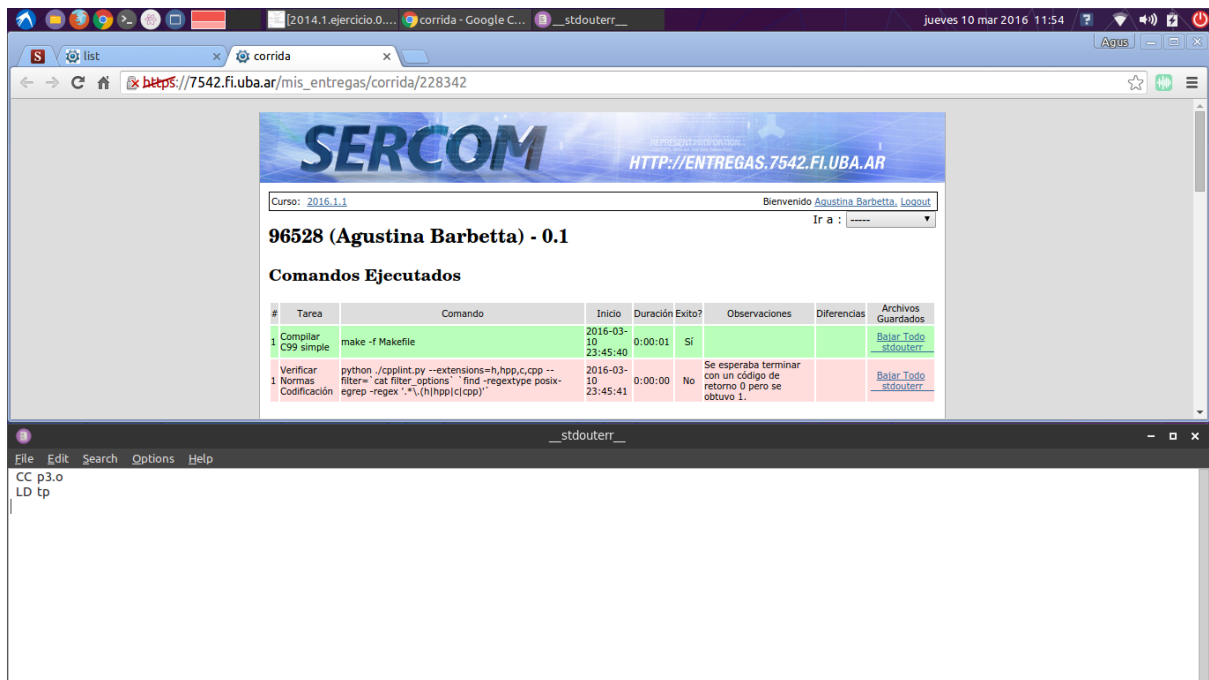


Figura 5: Captura de pantalla indicando la correcta generación del ejecutable



Figura 6: Captura de pantalla mostrando los problemas de estilo detectados



**SERCOM**  
HTTP://ENTREGAS.7542.FI.UBA.AR

Curso: 2016.1.1 Bienvenido Agustina Barbetta, Logout

96528 (Agustina Barbetta) - 0.1

**Comandos Ejecutados**

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2016-03-10 23:45:40	0:00:01	Si			Bajar Todo stdouterr
1	Verificar Normas Codificación	python ./cpplint.py --extensions=h,hpp,c,cpp --filter=cat filter_options --find -regex type posix-egrep -regex '.*\.(h hpp c cpp)'	2016-03-10 23:45:41	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo stdouterr

**Pruebas Realizadas**

**1- Archivo inexistente.**

Comando: ./tp no-existo  
Inicio / Fin: 2016-03-10 23:45:41 / 2016-03-10 23:45:41

#	Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Correr	Prueba normalmente, sin filtros	0:00:00	No	Se esperaba terminar con un código de retorno 1 pero se obtuvo 2.		Bajar Todo stdouterr
2	Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:00	Si			Bajar Todo valgrind.out

**2- Archivo existente.**

Comando: ./tp archivo-corto.txt  
Inicio / Fin: 2016-03-10 23:45:41 / 2016-03-10 23:45:42

#	Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Correr	Prueba normalmente, sin filtros	0:00:00	Si			Bajar Todo stdouterr
2	Valgrind	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo stdouterr

Figura 7: Captura de pantalla indicando el error reportado en la prueba 1

### 3.4. SERCOM - Pérdida de memoria

**SERCOM**  
HTTP://ENTREGAS.7542.FI.UBA.AR

Curso: 2016.1.1 Bienvenido Agustina Barbetta, Logout

96528 (Agustina Barbetta) - 0.1

**Comandos Ejecutados**

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2016-03-11 14:21:25	0:00:00	Si			Bajar Todo stdouterr
1	Verificar Normas Codificación	python ./cpplint.py --extensions=h,hpp,c,cpp --filter=cat filter_options --find -regex type posix-egrep -regex '.*\.(h hpp c cpp)'	2016-03-11 14:21:25	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo stdouterr

**Pruebas Realizadas**

**1- Archivo inexistente.**

Comando: ./tp no-existo  
Inicio / Fin: 2016-03-11 14:21:25 / 2016-03-11 14:21:25

#	Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Correr	Prueba normalmente, sin filtros	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo stdouterr
2	Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:00	Si			Bajar Todo valgrind.out

**2- Archivo existente.**

Comando: ./tp archivo-corto.txt  
Inicio / Fin: 2016-03-11 14:21:25 / 2016-03-11 14:21:25

#	Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Correr	Prueba normalmente, sin filtros	0:00:00	Si			Bajar Todo stdouterr
2	Valgrind	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo stdouterr

Figura 8: Captura de pantalla indicando la nueva salida del chequeo de normas de codificación

Curso: 2016.1.1 Bienvenido Agustina Barbetta, Logout

Ir a : -----

### 96528 (Agustina Barbetta) - 0.1

#### Comandos Ejecutados

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2016-03-11 14:21:25	0:00:00	Sí			Reparar Todo stdouterr
1	Verificar Normas Codificación	python ./cpplint.py --extensions=h,hpp,c,cpp --filter=cat filter_options --find -regex type posix-egrep -regex '^(h hpp c cpp)\$'	2016-03-11 14:21:25	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Reparar Todo stdouterr

#### Pruebas Realizadas

**1- Archivo inexistente.**

Comando: /tp no-existo

Inicio / Fin: 2016-03-11 14:21:25 / 2016-03-11 14:21:26

#	Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Correr	Prueba normalmente, sin filtros	0:00:00	Sí			Reparar Todo stdouterr
2	Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:01	Sí			Reparar Todo valgrind.out

**2- Archivo existente.**

Comando: /tp archivo-corto.txt

Inicio / Fin: 2016-03-11 14:21:26 / 2016-03-11 14:21:26

#	Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Correr	Prueba normalmente, sin filtros	0:00:00	Sí			Reparar Todo stdouterr
2	Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Reparar Todo valgrind.out

**3- Nombre largo.**

Comando: /tp soy-un-archivo-con-nombre-largo.txt

Inicio / Fin: 2016-03-11 14:21:26 / 2016-03-11 14:21:27

#	Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Correr	Prueba normalmente, sin filtros	0:00:00	Sí			Reparar Todo stdouterr
2	Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Reparar Todo valgrind.out

Figura 9: Captura de pantalla indicando la correcta finalización de la prueba 1

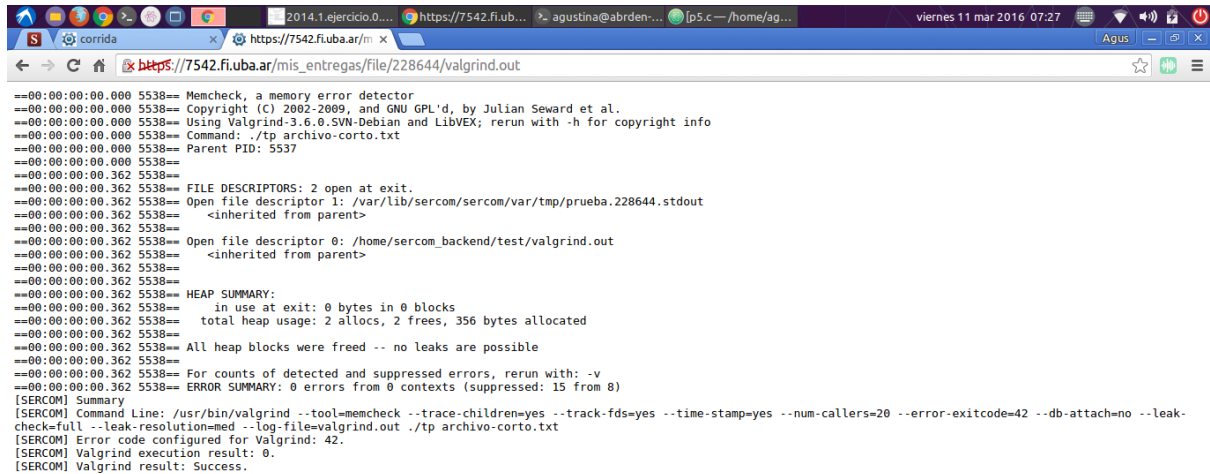
```

==00:00:00.000 5076== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==00:00:00.000 5076== Command: ./tp archivo-corto.txt
==00:00:00.000 5076== Parent PID: 5075
==00:00:00.000 5076==
==00:00:00.000 5076== FILE DESCRIPTORS: 3 open at exit.
==00:00:00.000 5076== Open file descriptor 2: archivo-corto.txt
==00:00:00.000 5076==   at 0x48E1B1E: __open_nocancel (syscall-template.S:82)
==00:00:00.000 5076==   by 0x488B8C7: _IO_file_fopen@@GLIBC_2.1 (fileops.c:336)
==00:00:00.000 5076==   by 0x487F0EC: _fopen_internal (iofopen.c:93)
==00:00:00.000 5076==   by 0x487FE4B: fopen@@GLIBC_2.1 (iofopen.c:107)
==00:00:00.000 5076==   by 0x8048603: main (p4.c:17)
==00:00:00.000 5076== Open file descriptor 1: /var/lib/sercom/sercom/var/tmp/prueba.228570.stdout
==00:00:00.000 5076==   <inherited from parent>
==00:00:00.000 5076== Open file descriptor 0: /home/sercom_backend/test/valgrind.out
==00:00:00.000 5076==   <inherited from parent>
==00:00:00.000 5076==
==00:00:00.000 5076== HEAP SUMMARY:
==00:00:00.000 5076==   in use at exit: 356 bytes in 2 blocks
==00:00:00.000 5076==   total heap usage: 2 allocs, 0 frees, 356 bytes allocated
==00:00:00.000 5076==
==00:00:00.000 5076== 4 bytes in 1 blocks are definitely lost in loss record 1 of 2
==00:00:00.000 5076==   at 0x47EBF20: malloc (vg_replace_malloc.c:236)
==00:00:00.000 5076==   by 0x804861A: main (p4.c:20)
==00:00:00.000 5076==
==00:00:00.000 5076== LEAK SUMMARY:
==00:00:00.000 5076==   definitely lost: 4 bytes in 1 blocks
==00:00:00.000 5076==   indirectly lost: 0 bytes in 0 blocks
==00:00:00.000 5076==   possibly lost: 0 bytes in 0 blocks
==00:00:00.000 5076==   still reachable: 352 bytes in 1 blocks
==00:00:00.000 5076==   suppressed: 0 bytes in 0 blocks
==00:00:00.000 5076== Reachable blocks (those to which a pointer was found) are not shown.
==00:00:00.000 5076== To see them, rerun with: --leak-check=full --show-reachable=yes
==00:00:00.000 5076== For counts of detected and suppressed errors, rerun with: -v
==00:00:00.000 5076== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 15 from 8)
[SERCOM] Summary
[SERCOM] Command Line: /usr/bin/valgrind --tool=memcheck --trace-children=yes --track-fds=yes --time-stamp=yes --num-callers=20 --error-exitcode=42 --db-attach=no --leak-check=full --leak-resolution=med --log-file=valgrind.out ./tp archivo-corto.txt
[SERCOM] Error code configured for Valgrind: 42.
[SERCOM] Valgrind execution result: 42.
[SERCOM] Valgrind result: Failure.

```

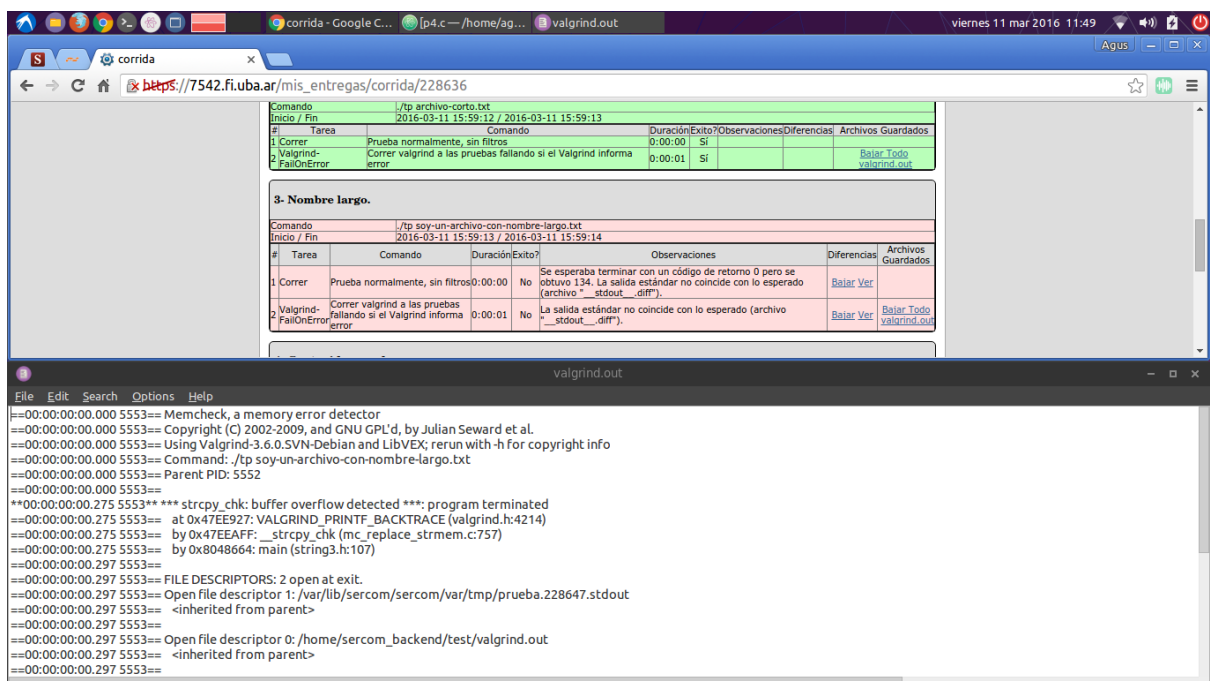
Figura 10: Captura de pantalla indicando los problemas reportados por Valgrind

### 3.5. SERCOM - Escrituras fuera de rango



```
==00:00:00.000 5538== Memcheck, a memory error detector
==00:00:00.000 5538== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==00:00:00.000 5538== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==00:00:00.000 5538== Command: ./tp archivo-corto.txt
==00:00:00.000 5538== Parent PID: 5537
==00:00:00.000 5538==
==00:00:00.362 5538==
==00:00:00.362 5538== FILE DESCRIPTORS: 2 open at exit.
==00:00:00.362 5538== Open file descriptor 1: /var/lib/sercom/sercom/var/tmp/prueba.228644.stdout
==00:00:00.362 5538== <inherited from parent>
==00:00:00.362 5538== Open file descriptor 0: /home/sercom_backend/test/valgrind.out
==00:00:00.362 5538== <inherited from parent>
==00:00:00.362 5538==
==00:00:00.362 5538== HEAP SUMMARY:
==00:00:00.362 5538==   in use at exit: 0 bytes in 0 blocks
==00:00:00.362 5538==   total heap usage: 2 allocs, 2 frees, 356 bytes allocated
==00:00:00.362 5538==
==00:00:00.362 5538== All heap blocks were freed -- no leaks are possible
==00:00:00.362 5538==
==00:00:00.362 5538== For counts of detected and suppressed errors, rerun with: -v
==00:00:00.362 5538== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 15 from 8)
[SERCOM] Summary
[SERCOM] Command Line: /usr/bin/valgrind --tool=memcheck --trace-children=yes --track-fds=yes --time-stamp=yes --num-callers=20 --error-exitcode=42 --db-attach=no --leak-check=full --leak-resolution=med --log-file=valgrind.out ./tp archivo-corto.txt
[SERCOM] Error code configured for Valgrind: 42.
[SERCOM] Valgrind execution result: 0.
[SERCOM] Valgrind result: Success.
```

Figura 11: Captura de pantalla de la salida de Valgrind sobre la prueba 2



#	Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Correr	Prueba normalmente, sin filtros	0:00:00	Sí			
2	Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:01	Sí			Bajar Todo valgrind.out

#	Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Correr	Prueba normalmente, sin filtros	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 134. La salida estándar no coincide con lo esperado (archivo "stdout_diff").	Bajar Ver	
2	Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:01	No	La salida estándar no coincide con lo esperado (archivo "stdout_diff").	Bajar Ver	Bajar Todo valgrind.out

```
File Edit Search Options Help
==00:00:00.000 5553== Memcheck, a memory error detector
==00:00:00.000 5553== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==00:00:00.000 5553== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==00:00:00.000 5553== Command: ./tp soy-un-archivo-con-nombre-largo.txt
==00:00:00.000 5553== Parent PID: 5552
==00:00:00.000 5553==
==00:00:00.275 5553== *** strcpv_chk: buffer overflow detected ***: program terminated
==00:00:00.275 5553== at 0x47EE927: VALGRIND_PRINTF_BACKTRACE (valgrind.h:4214)
==00:00:00.275 5553== by 0x47EEAFF: __strcpy_chk (mc_replace_strmem.c:757)
==00:00:00.275 5553== by 0x8048664: main (string3.h:107)
==00:00:00.297 5553==
==00:00:00.297 5553== FILE DESCRIPTORS: 2 open at exit.
==00:00:00.297 5553== Open file descriptor 1: /var/lib/sercom/sercom/var/tmp/prueba.228647.stdout
==00:00:00.297 5553== <inherited from parent>
==00:00:00.297 5553== Open file descriptor 0: /home/sercom_backend/test/valgrind.out
==00:00:00.297 5553== <inherited from parent>
==00:00:00.297 5553==
```

Figura 12: Prueba 3 presenta un error poco claro

### 3.6. SERCOM - Entrada estándar



Figura 13: Captura de pantalla con la correcta salida del chequeo de normas de codificación

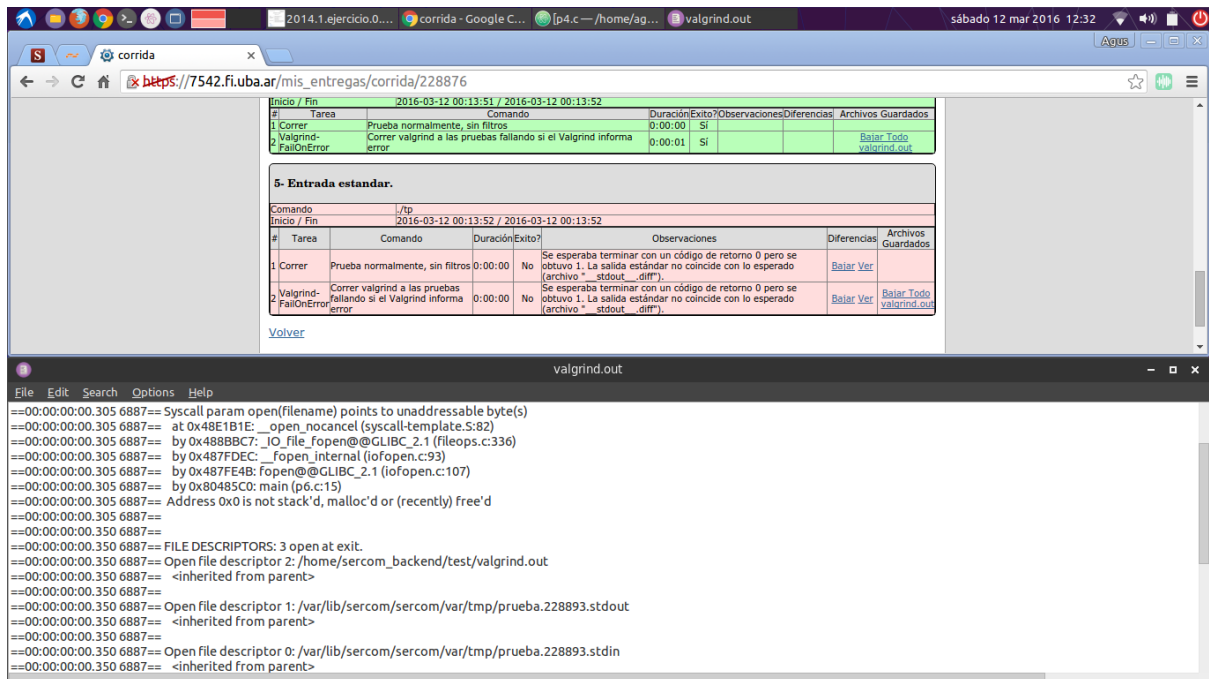


Figura 14: Captura de pantalla con el resultado de la prueba 5

### 3.7. SERCOM - Entrega exitosa



Figura 15: Captura de pantalla mostrando la entrega exitosa, en color verde

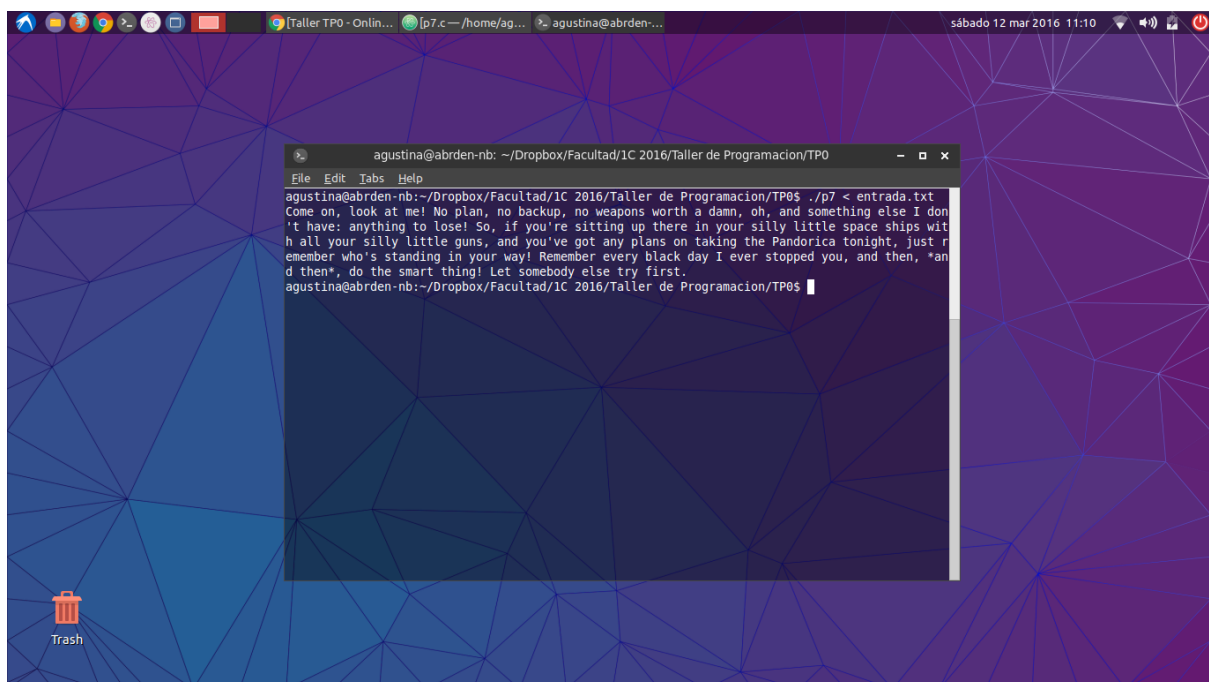


Figura 16: Captura de pantalla mostrando la ejecución local de la prueba 5 sin el uso del teclado

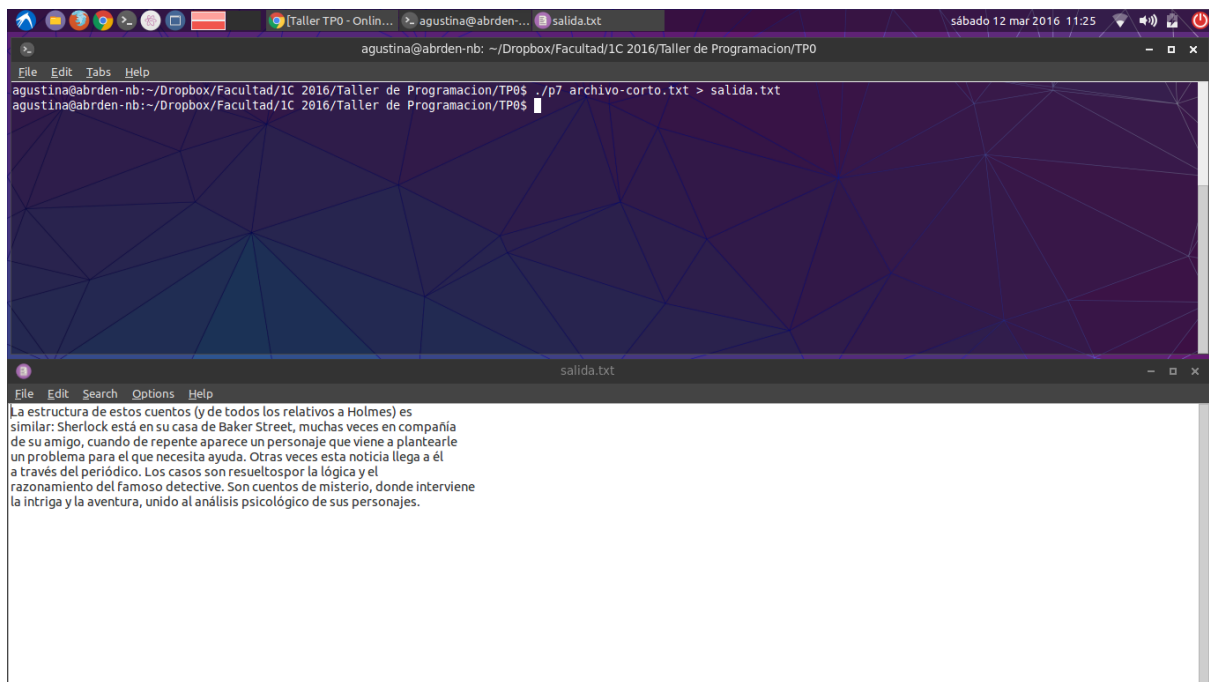


Figura 17: Captura de pantalla mostrando la ejecución local de la prueba 2, pero redireccionando la salida estándar a un archivo denominado 'salida.txt'