Informe de Trabajo Practico Nº 0

12 de marzo de 2016

75.42 Taller de Programación I

Cátedra Veiga

Facultad de Ingeniera - UBA

Autor: Martín Stancanelli

Padrón: 95188

Objetivo

El objetivo del trabajo practico es familiarizarse con la herramienta SERCOM provista por la cátedra para la corrección automática de trabajos prácticos y ejecución de pruebas.

Además se busca que los alumnos tomen un contacto temprano con el entorno de trabajo que se va a utilizar a lo largo de la materia (Linux, gcc, C, etc...).

Configuración del entorno

El entorno de trabajo elegido para la realización del trabajo practico fue una PC con SO Linux Mint 17.3 y compilador gcc 4.8 configurado para utilizar el estándar de C99.

Desarrollo de pasos

Paso 1

Para el primer paso se desarrollo en el entorno indicado en la sección anterior un pequeño programa hola mundo para testear que el entorno este correctamente configurado.

Luego se instalo el programa *Valgrind* y se probo la ejecución de nuestro hola mundo.

En la figura 1 podemos ver la corrida del hola mundo y en la figura 2 podemos ver la corrida con *Valgrind*

```
mastanca@mastanca-desktop ~/workspace/taller-1c-2016-tp0/Debug $ ./taller-1c-2016-tp0
Hello, World!
mastanca@mastanca-desktop ~/workspace/taller-1c-2016-tp0/Debug $
```

Figura 1: Ejecución de hello world

```
mastanca@mastanca-desktop ~/workspace/taller-1c-2016-tp0/Debug $ valgrind ./taller-1c-2016-tp0
==32684== Memcheck, a memory error detector
==32684== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==32684== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==32684== Command: ./taller-1c-2016-tp0
==32684==
Hello, World!
==32684==
==32684== HEAP SUMMARY:
==32684==     in use at exit: 0 bytes in 0 blocks
==32684==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==32684==
==32684== All heap blocks were freed -- no leaks are possible
==32684==
==32684== For counts of detected and suppressed errors, rerun with: -v
==32684== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
mastanca@mastanca-desktop ~/workspace/taller-1c-2016-tp0/Debug $
```

Figura 2: Ejecución de hello world con Valgrind

- ¿Qué representa sizeof()? ¿Cuál sería el valor de salida de sizeof(char) y sizeof(int)?

sizeof() es una función que devuelve la representación en bytes de un tipo de dato. La representación depende de la arquitectura y la implementación, sin embargo C y C++ determinan un mínimo de bits para la representación de tipos, por ejemplo para char son 8bits y para int son 16bits.

Entones la salida de sizeof(char) es 1, y la salida de sizeof(int) es 4, representados ambos en bytes.

- "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.

El valor del sizeof de un struct no es la suma de los sizeof de sus elementos, ya que el compilador agrega padding por una cuestión de performance. Así por ejemplo el sizeof de un struct cuyas componentes son 1 char y 1 int nos dio 8 y la suma de los sizeof de las partes 5 en el siguiente ejemplo .

```
#include <stdio.h>

int main (int argc, const char * argv[]) {

    struct Books {
        int id;
        char authorInitial;
    };

    int sizeInt = sizeof(int);
    int sizeChar = sizeof(char);
    struct Books book1;
    book1.id = 25;
    book1.authorInitial = 'A';

    printf("%s", "Sum␣of␣the␣sizeofs:␣");
    int sumOfSizeOfs = sizeof(book1.id) + sizeof(book1
        .authorInitial);
    printf("%d\n", sumOfSizeOfs);

    printf("%s", "Sum␣of␣the␣struct:␣");
    printf("%d\n", sizeof( book1 ) );

    return 0;
}
```

Paso 2

Para este segundo paso se entrego el código provisto por el enunciado. Tal como predecía el enunciado las pruebas fallaron ya que el SERCOM no pudo generar la aplicación, como podemos ver en la figura 3.

```
  CC  main.o
main.c: In function 'main':
main.c:17: error: implicit declaration of function 'ztrcpy'
main.c:20: error: implicit declaration of function 'malloc'
cc1: warnings being treated as errors
main.c:20: error: incompatible implicit declaration of built-in function 'malloc'
make: *** [main.o] Error 1
```

Figura 3: Salida error SERCOM

Los errores generados son errores del linker, ya que estamos haciendo llamadas a funciones no incluidas previamente con una directiva #include

```c
#include <stdio.h>
#include <string.h>

int main( int argc, char *argv[] )
{
  char nombre[20];
  char *buffer;
  FILE *fp;

  ztrcpy( nombre, argv[1] );
  fp = fopen( nombre, "r" );
  if( fp == NULL ) return 2;

  buffer = malloc( sizeof(int) ); /* buffer innecesario
     */

  while( !feof(fp) )
  {
    int c = fgetc(fp);
    if( c != EOF )
      printf( "%c", (char) c );
  }

  return 0;
}
```

Figura 4: Código fuente paso 2

Paso 3

Se corrigieron los errores de compilación incluyendo la librería stdlib.h y cambiando la llamada a ztrcpy por strcpy. A continuación se realizo una segunda entrega del código, la cual pudo ser compilada por el SERCOM pero fallo al validar las normas de codificación como podemos ver en las figuras 5 y 6.

Las normas de codificación fallaron ya que había espacios en blanco de mas entre los argumentos de las funciones y los paréntesis. Además la salida nos sugiere el uso de la función snprintf en vez de strcpy, la cual es casi siempre peor.

Comandos Ejecutados

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2016-03-11 18:39:39	0:00:00	Si			Bajar Todo stdouterr
1	Verificar Normas Codificación	python ./cpplint.py --extensions=h,hpp,c,cpp --filter=`cat filter_options` `find -regextype posix-egrep -regex '.*\.(h|hpp|c|cpp)'`	2016-03-11 18:39:39	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo stdouterr

Figura 5: Generación del ejecutable y error normas de codificación

```
./main.c:13:  Extra space after ( in function call  [whitespace/parens] [4]
./main.c:18:  Extra space after ( in function call  [whitespace/parens] [4]
./main.c:18:  Extra space before )  [whitespace/parens] [2]
./main.c:18:  Almost always, snprintf is better than strcpy  [runtime/printf] [4]
./main.c:19:  Extra space after ( in function call  [whitespace/parens] [4]
./main.c:19:  Extra space before )  [whitespace/parens] [2]
./main.c:20:  Missing space before ( in if(  [whitespace/parens] [5]
./main.c:22:  Missing space before ( in while(  [whitespace/parens] [5]
./main.c:25:  Missing space before ( in if(  [whitespace/parens] [5]
./main.c:26:  Extra space after ( in function call  [whitespace/parens] [4]
./main.c:26:  Extra space before )  [whitespace/parens] [2]
Done processing ./main.c
Total errors found: 11
```

Figura 6: Errores normas de codificación

Además del error ya mencionado también fallo la prueba 1, ya que esta esperaba que el programa retorne un 1 al no encontrar el archivo pasado como parámetro y retornó un 2. Los códigos de retorno habituales para un programa son un 0 en caso de salida exitosa o un 1 en caso de algún error, aunque en casos especiales se pueden utilizar códigos de error distintos para indicar distintos tipos de errores.

Pruebas Realizadas

1- Archivo inexistente.

Comando	./tp no-existo
Inicio / Fin	2016-03-11 18:39:39 / 2016-03-11 18:39:40

#	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.		
2	Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:00	Si			Bajar Todo valgrind.out

Figura 7: Falla prueba 1

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main( int argc, char *argv[] )
{
  char nombre[20];
  char *buffer;
  FILE *fp;

  strcpy( nombre, argv[1] );
  fp = fopen( nombre, "r" );
  if( fp == NULL ) return 2;

  buffer = malloc( sizeof(int) ); /* buffer innecesario
     */

  while( !feof(fp) )
  {
    int c = fgetc(fp);
    if( c != EOF )
      printf( "%c", (char) c );
  }

  return 0;
}
```

Figura 8: Código fuente paso 3

Paso 4

En este paso se corrigieron los errores de codificación que fueron marcados por el SERCOM en el paso anterior, se reemplazaron los números mágicos por constantes declaradas con la directiva #define y se corrigió el código de error incorrecto que hacia fallarla prueba 1.

Se volvió a realizar una entrega del código, observando ahora que la prueba 1 pasaba exitosamente (figura 9) y el chequeo de las normas de codificación solo mostraba un warning sugiriendo el uso de la función snprintf (figura 10).

Sin embargo la prueba 2 fallo, ya que la corrida de Valgrind sobre el código detectaba errores (ver figura 11).

Los fallos en Valgrind se deben a la apertura de archivos que luego no fueron cerrados y a la llamada a un malloc que no esta seguida de un free en algún momento. Los archivos que no se cierran no pueden garantizar que todo lo que se mando a escribir a ellos sea realmente escrito, ya que puede haber quedado en algún buffer y además quedan marcados como en uso. El malloc que no se libera con free es memoria que se pierde ya que queda reservada pero nunca mas es usada.

```
./main.c:21:  Almost always, snprintf is better than strcpy  [runtime/printf] [4]
Done processing ./main.c
Total errors found: 1
```

Figura 9: Salida correcta de normas de codificación

Pruebas Realizadas

1- Archivo inexistente.

Comando	./tp no-existo
Inicio / Fin	2016-03-11 19:07:30 / 2016-03-11 19:07:31

#	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

Figura 10: Prueba 1 correcta

```
==00:00:00:00.000 6129== Memcheck, a memory error detector
==00:00:00:00.000 6129== Copyright (C) 2002-2009, and GNU GPL'd, by
    Julian Seward et al.
==00:00:00:00.000 6129== Using Valgrind-3.6.0.SVN-Debian and LibVEX;
    rerun with -h for copyright info
==00:00:00:00.000 6129== Command: ./tp archivo-corto.txt
==00:00:00:00.000 6129== Parent PID: 6128
==00:00:00:00.000 6129==
==00:00:00:00.371 6129==
==00:00:00:00.371 6129== FILE DESCRIPTORS: 3 open at exit.
==00:00:00:00.371 6129== Open file descriptor 2: archivo-corto.txt
==00:00:00:00.371 6129==    at 0x48E1B1E: __open_nocancel (syscall-
    template.S:82)
==00:00:00:00.371 6129==    by 0x488BBC7: _IO_file_fopen@@GLIBC_2.1 (
    fileops.c:336)
==00:00:00:00.371 6129==    by 0x487FDEC: __fopen_internal (iofopen.c
    :93)
==00:00:00:00.371 6129==    by 0x487FE4B: fopen@@GLIBC_2.1 (iofopen.c
    :107)
==00:00:00:00.371 6129==    by 0x8048603: main (main.c:22)
==00:00:00:00.371 6129==
==00:00:00:00.371 6129== Open file descriptor 1: /var/lib/sercom/sercom/
    var/tmp/prueba.228754.stdout
==00:00:00:00.371 6129==    <inherited from parent>
==00:00:00:00.371 6129==
==00:00:00:00.371 6129== Open file descriptor 0: /home/sercom_backend/
    test/valgrind.out
==00:00:00:00.371 6129==    <inherited from parent>
==00:00:00:00.371 6129==
==00:00:00:00.371 6129==
==00:00:00:00.371 6129== HEAP SUMMARY:
==00:00:00:00.371 6129==     in use at exit: 356 bytes in 2 blocks
==00:00:00:00.371 6129==   total heap usage: 2 allocs, 0 frees, 356
    bytes allocated
==00:00:00:00.371 6129==
==00:00:00:00.372 6129== 4 bytes in 1 blocks are definitely lost in loss
     record 1 of 2
==00:00:00:00.372 6129==    at 0x47EBF20: malloc (vg_replace_malloc.c
    :236)
==00:00:00:00.372 6129==    by 0x804861A: main (main.c:24)
==00:00:00:00.372 6129==
==00:00:00:00.372 6129== LEAK SUMMARY:
==00:00:00:00.372 6129==    definitely lost: 4 bytes in 1 blocks
==00:00:00:00.372 6129==    indirectly lost: 0 bytes in 0 blocks
==00:00:00:00.372 6129==      possibly lost: 0 bytes in 0 blocks
==00:00:00:00.372 6129==    still reachable: 352 bytes in 1 blocks
==00:00:00:00.372 6129==         suppressed: 0 bytes in 0 blocks
==00:00:00:00.372 6129== Reachable blocks (those to which a pointer was
    found) are not shown.
==00:00:00:00.372 6129== To see them, rerun with: --leak-check=full --
    show-reachable=yes
==00:00:00:00.372 6129==
==00:00:00:00.372 6129== For counts of detected and suppressed errors,
    rerun with: -v
==00:00:00:00.372 6129== 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 11: Salida Valgrind paso 4

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define LARGO_FILE 20
#define POS_NOMBRE_ARCHIVO 1
#define ARCHIVO_NO_ENCONTRADO 1
#define SALIDA_NORMAL 0

int main(int argc, char *argv[])
{
  char nombre[LARGO_FILE];
  char *buffer;
  FILE *fp;

  strcpy(nombre, argv[POS_NOMBRE_ARCHIVO]);
  fp = fopen(nombre, "r");
  if ( fp == NULL ) return ARCHIVO_NO_ENCONTRADO;

  buffer = malloc(sizeof(int)); /* buffer innecesario */

  while ( !feof(fp) )
  {
    int c = fgetc(fp);
    if ( c != EOF )
      printf("%c", (char) c);
  }

  return SALIDA_NORMAL;
}
```

Figura 12: Código paso 4

Paso 5

Se agrego una llamada a free y una a fclose y se volvió a realizar una entrega. Esta vez las pruebas 1, 2 y 4 fueron correctas, pero la 3 arrojo un código de error extraño (134).

Según lo apreciado el programa estaría tratando de leer mas allá del espacio reservado de memoria. Al leer la salida de la ejecución de valgrind sobre el código actual pareciera que estamos tratando de leer de la dirección 0x0. Utilizando strncpy podría solucionarse el problema, ya que el tercer argumento de dicha función es hasta cuantos caracteres voy a leer, evitando así pasarme si la entrada es mas grande.

```
==00:00:00:00.000 6224== Memcheck, a memory error detector
==00:00:00:00.000 6224== Copyright (C) 2002-2009, and GNU GPL'd, by
    Julian Seward et al.
==00:00:00:00.000 6224== Using Valgrind-3.6.0.SVN-Debian and LibVEX;
    rerun with -h for copyright info
==00:00:00:00.000 6224== Command: ./tp archivo-corto.txt
==00:00:00:00.000 6224== Parent PID: 6223
==00:00:00:00.000 6224==
==00:00:00:00.378 6224==
==00:00:00:00.378 6224== FILE DESCRIPTORS: 2 open at exit.
==00:00:00:00.378 6224== Open file descriptor 1: /var/lib/sercom/sercom/
    var/tmp/prueba.228772.stdout
==00:00:00:00.378 6224==    <inherited from parent>
==00:00:00:00.378 6224==
==00:00:00:00.378 6224== Open file descriptor 0: /home/sercom_backend/
    test/valgrind.out
==00:00:00:00.378 6224==    <inherited from parent>
==00:00:00:00.378 6224==
==00:00:00:00.378 6224==
==00:00:00:00.378 6224== HEAP SUMMARY:
==00:00:00:00.378 6224==     in use at exit: 0 bytes in 0 blocks
==00:00:00:00.378 6224==   total heap usage: 2 allocs, 2 frees, 356
    bytes allocated
==00:00:00:00.378 6224==
==00:00:00:00.378 6224== All heap blocks were freed -- no leaks are
    possible
==00:00:00:00.378 6224==
==00:00:00:00.378 6224== For counts of detected and suppressed errors,
    rerun with: -v
==00:00:00:00.378 6224== 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 13: Salida Valgrind paso 5

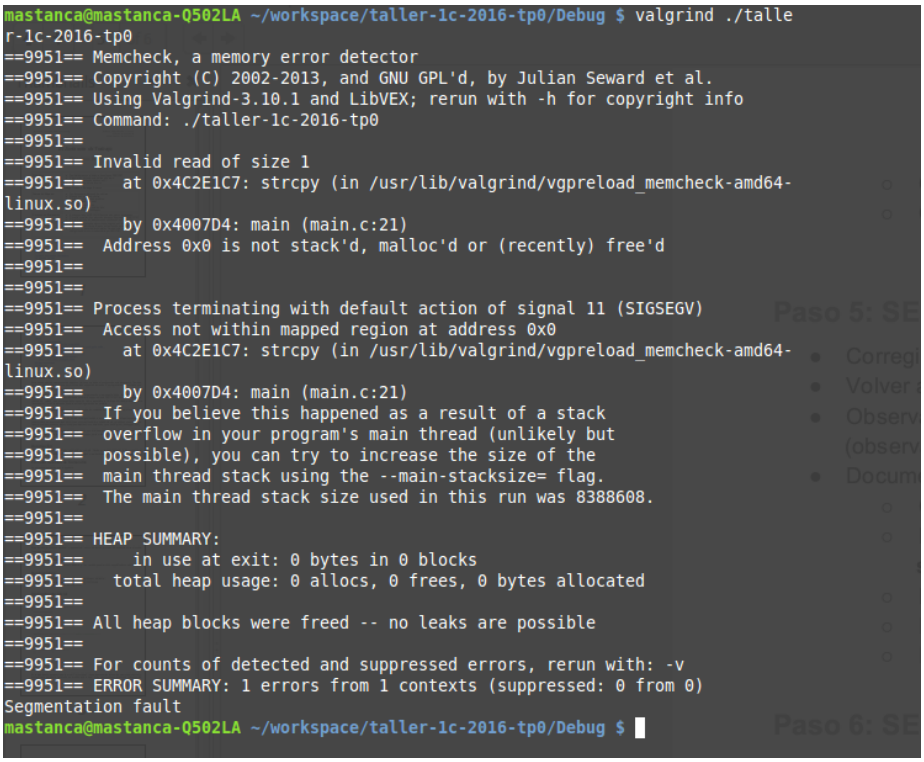


Figura 14: Salida valgrind paso 5

A continuación de explican los errores mas comunes de accesos a memoria

- Segmentation fault: Se produce cuando un programa trata de acceder a una posición de memoria a la cual no tiene acceso, o en una forma no permitida.
- Buffer overflow: Se genera cuando se escribe en un buffer y el tamaño de lo que tratamos de escribir excede el tamaño del buffer, pisando la memora adyacente a este.

Para la prueba 2 se ingreso el archivo texto corto y para la 4 el texto largo. El comando ejecutado para la prueba 3 fue ./tp soy-un-archivo-con-nombre-largo.txt

La estructura de estos cuentos (y de todos los relativos a Holmes) es similar: Sherlock esta 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 el a traves del periodico. Los casos son resueltospor la logica y el razonamiento del famoso detective. Son cuentos de misterio, donde interviene la intriga y la aventura, unido al analisis psicologico de sus personajes.

Figura 15: Contenido archivo prueba 2

Rene Geronimo Favaloro (La Plata, Argentina, 12 de julio de 1923 - Buenos Aires, Argentina, 29 de julio de 2000) fue un prestigioso médico cirujano toracico argentino, reconocido mundialmente por ser quien realizo el primer bypass cardiaco en el mundo. Estudio medicina en la Universidad de La Plata y una vez recibido, previo paso por el Hospital Policlinico, se mudo a la localidad de Jacinto Arauz para reemplazar temporalmente al medico local, quien tenia problemas de salud. A su vez, leia bibliografia medica actualizada y empezo a tener interes en la cirugia toracica. A fines de la decada de 1960 empezo a estudiar una tecnica para utilizar la vena safena en la cirugia coronaria. A principios de la decada de 1970 fundo la fundacion que lleva su nombre. Se desempeno en la Conadep, condujo programas de television dedicados a la medicina y escribio libros.
Durante la crisis del 2000, su fundacion tenia una gran deuda economica y le solicito 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 Rua criticando al sistema de salud, se quito la vida de un disparo al corazon.

Figura 16: Texto usado como entrada en prueba 4

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define LARGO_FILE 20
#define POS_NOMBRE_ARCHIVO 1
#define ARCHIVO_NO_ENCONTRADO 1
#define SALIDA_NORMAL 0

int main(int argc, char *argv[])
{
  char nombre[LARGO_FILE];
  char *buffer;
  FILE *fp;

  strcpy(nombre, argv[POS_NOMBRE_ARCHIVO]);
  fp = fopen(nombre, "r");
  if ( fp == NULL ) return ARCHIVO_NO_ENCONTRADO;

  buffer = malloc(sizeof(int)); /* buffer innecesario */

  while ( !feof(fp) )
  {
    int c = fgetc(fp);
    if ( c != EOF )
      printf("%c", (char) c);
  }

  fclose(fp);
  free(buffer);

  return SALIDA_NORMAL;
}
```

Figura 17: Código paso 5

Paso 6

Para corregir el problema reportado en el paso 5 se elimino el buffer innecesario y se puso directamente el argv1 en la llamada a fopen.

Seguido de la corrección se realizo una nueva entrega. En esta ocasión todas las pruebas a excepción de la ultima pasaron, ya que aun no esta implementada la funcionalidad de la salida estándar (figura 19).

```
Done processing ./main.c
Total errors found: 0
```

Figura 18: Salida normas verificación paso 6

5- Entrada estandar.

Comando	./tp
Inicio / Fin	2016-03-11 19:56:02 / 2016-03-11 19:56:02

#	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. 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	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1. La salida estándar no coincide con lo esperado (archivo "__stdout__.diff").	Bajar Ver	Bajar Todo valgrind.out

Figura 19: Resultado prueba 5

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define LARGO_FILE 20
#define POS_NOMBRE_ARCHIVO 1
#define ARCHIVO_NO_ENCONTRADO 1
#define SALIDA_NORMAL 0

int main(int argc, char *argv[])
{
  char *buffer;
  FILE *fp;

  fp = fopen(argv[POS_NOMBRE_ARCHIVO], "r");
  if ( fp == NULL ) return ARCHIVO_NO_ENCONTRADO;

  buffer = malloc(sizeof(int)); /* buffer innecesario */

  while ( !feof(fp) )
  {
    int c = fgetc(fp);
    if ( c != EOF )
      printf("%c", (char) c);
  }

  fclose(fp);
  free(buffer);

  return SALIDA_NORMAL;
}
```

Figura 20: Código fuente paso 6

Paso 7

En este ultimo paso se agrego la funcionalidad de leer desde stdin.

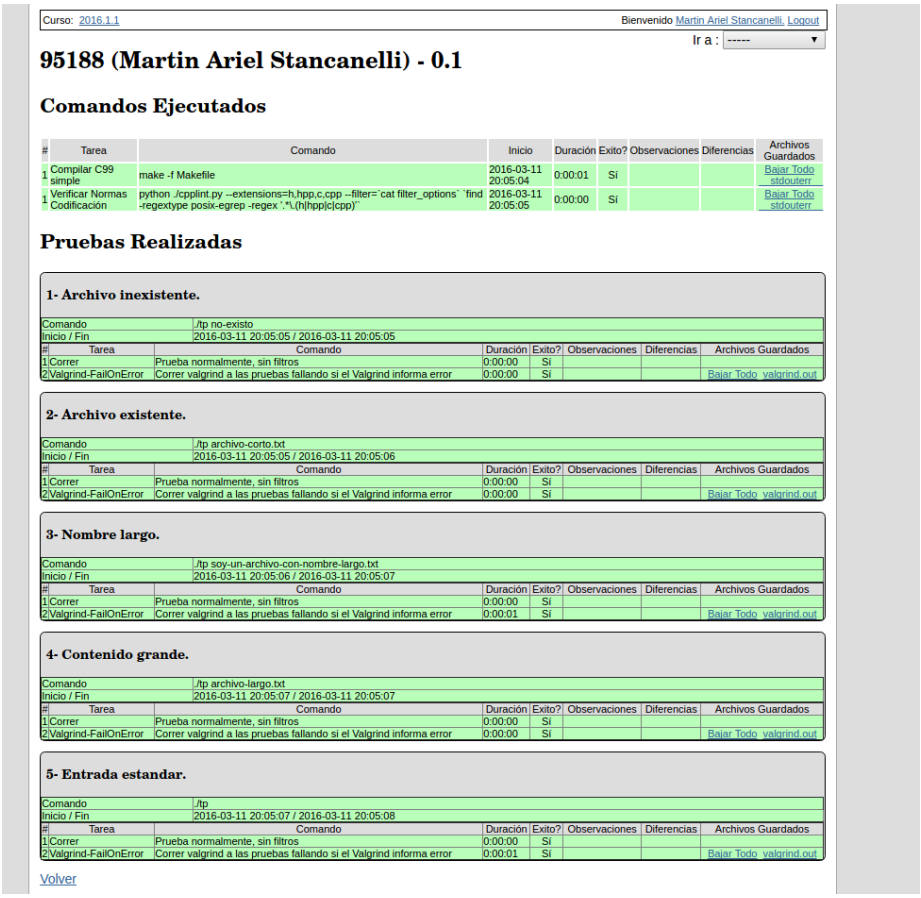
Podemos apreciar que para redireccionar stdin y stdout se utilizan los caracteres < y >.

El caracter > permite redireccionar la salida del programa hacia un archivo. El caracter < permite tomar como entrada desde stdin a un archivo indicado después de dicho caracter.

Para finalizar se realizo una entrega final del código, con la cual todas las pruebas fueron exitosas.

Ejercicio	Resultado	Fecha	Duración	Observaciones	Operaciones
0.1 (El Ambiente de Trabajo)	Aceptado	2016-03-11 20:04:58	0:00:04		Corrida Bajar Navegar PDF

Figura 21: Entrega exitosa



Curso: 2016.1.1 — Bienvenido Martin Ariel Stancanelli, Logout

Ir a : -----

95188 (Martin Ariel Stancanelli) - 0.1

Comandos Ejecutados

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2016-03-11 20:05:04	0:00:01	Si			Bajar Todo stdouterr
1	Verificar Normas Codificación	python ./cpplint.py --extensions=h,hpp,c,cpp --filter=`cat filter_options` `find -regextype posix-egrep -regex '.*\.(h|hpp|c|cpp)'`	2016-03-11 20:05:05	0:00:00	Si			Bajar Todo stdouterr

Pruebas Realizadas

1- Archivo inexistente.

Comando	./tp no-existo					
Inicio / Fin	2016-03-11 20:05:05 / 2016-03-11 20:05:05					
# Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1 Correr	Prueba normalmente, sin filtros	0:00:00	Si			
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 20:05:05 / 2016-03-11 20:05:06					
# Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1 Correr	Prueba normalmente, sin filtros	0:00:00	Si			
2 Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:00	Si			Bajar Todo valgrind.out

3- Nombre largo.

Comando	./tp soy-un-archivo-con-nombre-largo.txt					
Inicio / Fin	2016-03-11 20:05:06 / 2016-03-11 20:05:07					
# Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1 Correr	Prueba normalmente, sin filtros	0:00:00	Si			
2 Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:01	Si			Bajar Todo valgrind.out

4- Contenido grande.

Comando	./tp archivo-largo.txt					
Inicio / Fin	2016-03-11 20:05:07 / 2016-03-11 20:05:07					
# Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1 Correr	Prueba normalmente, sin filtros	0:00:00	Si			
2 Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:00	Si			Bajar Todo valgrind.out

5- Entrada estandar.

Comando	./tp					
Inicio / Fin	2016-03-11 20:05:07 / 2016-03-11 20:05:08					
# Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1 Correr	Prueba normalmente, sin filtros	0:00:00	Si			
2 Valgrind-FailOnError	Correr valgrind a las pruebas fallando si el Valgrind informa error	0:00:01	Si			Bajar Todo valgrind.out

Volver

Figura 22: Pruebas exitosas

```
mastanca@mastanca-Q502LA ~/workspace/taller-1c-2016-tp0/Debug $ ./taller-1c-2016-tp0 < archivo-corto.txt
La estructura de estos cuentos (y de todos los relativos a Holmes) es
similar: Sherlock est en su casa de Baker Street, muchas veces en compa a
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 l
a trav s del peri dico. Los casos son resueltospor la l gica y el
razonamiento del famoso detective. Son cuentos de misterio, donde interviene
la intriga y la aventura, unido al an lisis psicol gico de sus personajes.
```

Figura 23: Ejecución prueba 5 sin teclado

```
mastanca@mastanca-Q502LA ~/workspace/taller-1c-2016-tp0/Debug $ ./taller-1c-2016-tp0 archivo-corto.txt > salida.txt
mastanca@mastanca-Q502LA ~/workspace/taller-1c-2016-tp0/Debug $ ls
archivo-corto.txt  makefile  objects.mk  salida.txt  sources.mk  src  taller-1c-2016-tp0
```

Figura 24: Redireccionado stdout

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define LARGO_FILE 20
#define POS_NOMBRE_ARCHIVO 1
#define ARCHIVO_NO_ENCONTRADO 1
#define SALIDA_NORMAL 0

int main(int argc, char *argv[])
{
  char *buffer;
  FILE *fp = stdin;

  if (argc > POS_NOMBRE_ARCHIVO)
  {
    fp = fopen(argv[POS_NOMBRE_ARCHIVO], "r");
    if ( fp == NULL ) return ARCHIVO_NO_ENCONTRADO;
  }

  buffer = malloc(sizeof(int)); /* buffer innecesario */

  while ( !feof(fp) )
  {
    int c = fgetc(fp);
    if ( c != EOF )
      printf("%c", (char) c);
  }

  if (argc > POS_NOMBRE_ARCHIVO)
    fclose(fp);

  free(buffer);

  return SALIDA_NORMAL;
}
```

Figura 25: Código fuente final

Se inlcuye un diagrama explicando como fluyen la salida y entrada de los programas a través de los 3 file descriptors básicos de Unix.

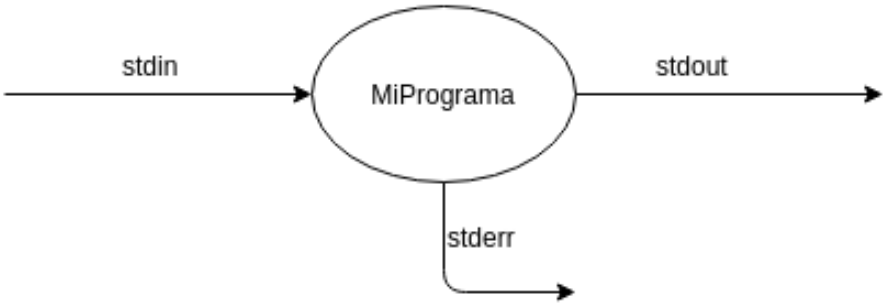


Figura 26: Diagrama file descriptors Unix

Conclusión

El trabajo practico sirvió para tomar conocimiento de los lineamientos que se van a seguir a la hora de la corrección de los trabajos prácticos durante la cursada de la materia.

Además tomamos un primer contacto con el sistema de corrección automática, lo cual es importante para tener un mejor dominio del mismo en futuras entregas.

El poder descargarse las pruebas que son ejecutadas por el sistema para poder correrlas en forma local es de gran ayuda para verificar que lo hecho en el trabajo practico este cumpliendo con las expectativas mínimas del sistema.

Salida entrega final

mar 11, 16 20:41 **main.c** Page 1/1

```
/*
 * main.c

 *
 *  Created on: Mar 10, 2016
 *      Author: mastanca
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define LARGO_FILE 20
#define POS_NOMBRE_ARCHIVO 1
#define ARCHIVO_NO_ENCONTRADO 1
#define SALIDA_NORMAL 0
int main(int argc, char *argv[])
{
  FILE *fp = stdin;
  if (argc > POS_NOMBRE_ARCHIVO)
  {
    fp = fopen(argv[POS_NOMBRE_ARCHIVO], "r");
    if ( fp ≡ NULL ) return ARCHIVO_NO_ENCONTRADO;
  }
  while ( ¬feof(fp) )
  {
    int c = fgetc(fp);
    if ( c ≠ EOF )
      printf("%c", (char) c);
  }
  if (argc > POS_NOMBRE_ARCHIVO)
    fclose(fp);
  return SALIDA_NORMAL;
}
```

mar 11, 16 20:41 **Table of Content** Page 1/1