Paso 1: Comenzando.

Preparar el ambiente de trabajo local:

Una vez ya instalado Linux sobre mi computadora, lo primero que hice fue descargarme Eclipse desde su web oficial (www.eclipse.org), el paquete especial para desarrollar C/C++. Una vez con la IDE correctamente instalada, con el compilador correctamente instalado y funcionando procedi a la creación de un pequeño proyecto de prueba para comprobas que todo esta funcionando correctamente.

Compilación y prueba de una aplicación ejemplo:

Abrí la IDE y cree un nuevo proyecto llamado “Hola Mundo”. El proyecto es simplemente un archivo .c con una salida por consola que imprime “Hello, World”. El código es el siguiente:

```
//
//  main.c
//  Hola Mundo
//
//  Created by Gastón Montes on 26/08/14.
//  Copyright (c) 2014 Gastón Montes. All rights reserved.
//

#include <stdio.h>

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

    // insert code here...
    printf("Hello, World!\n");
    return 0;
}
```

Guardo el proyecto y abro una terminal. Me situo donde estan los archivos del proyecto. Compilo el archivo .c con el compilador de C y luego lo corro llamando al archivo resultante de la compilación:

```
MacBook-Air-de-Gaston:Hola Mundo gastonmontes$ gcc main.c
MacBook-Air-de-Gaston:Hola Mundo gastonmontes$ ./a.out
Hello, World!
MacBook-Air-de-Gaston:Hola Mundo gastonmontes$
```

Instalar Valgrind:

Valgrind es una herramienta que nos permite depurar los problemas de memoria y mejorar el rendimiento de los programas. Nos permite hacer un seguimiento del uso de memoria y detectar los siguientes problemas: Uso de memoria no inicializada; Lectura/escritura de memoria liberada; Lectura/escritura fuera de los límites de la memoria dinámica; Leaks (fugas) de memoria; y todos los problemas relacionados al manejo de memoria de un programa.
Descargue el archivo .tar de Valgrind de su web oficial (http://valgrind.org/). descomprimi el archivo con la versión 3.9.0.
Una vez descomprimido, me situo en la carpeta resultante de la descompreción y corro los siguientes comandos:

```
>> cd valgrind
>> ./autogen.sh
>> ./configure
>> make
>> sudo make install
```

Con esto, ya tenemos Valgrind instalado en nuestro ordenador.
Falta probarlo sobre algún código de prueba, para ello vamos a usar el programita “Hola Mundo!” creado anteriormente.
Me situo sobre la carpeta que contiene los archivos fuentes del programa de prueba. Una vez allí, compilo como de costumbre pero para ejecutar el programa anteponemos el comando de valgrind:

```
gcc main.c
valgrind ./a.out
```

Esto hará que valgrind haga un chequeo de errores de memoria y nos tire un resultado:

```
Hello, World!
==28639==
==28639== HEAP SUMMARY:
==28639==     in use at exit: 29,068 bytes in 371 blocks
==28639==   total heap usage: 447 allocs, 76 frees, 35,012 bytes allocated
==28639==
==28639== LEAK SUMMARY:
==28639==    definitely lost: 0 bytes in 0 blocks
==28639==    indirectly lost: 0 bytes in 0 blocks
==28639==      possibly lost: 0 bytes in 0 blocks
==28639==    still reachable: 4,096 bytes in 1 blocks
==28639==         suppressed: 24,972 bytes in 370 blocks
==28639== Rerun with --leak-check=full to see details of leaked memory
==28639==
==28639== For counts of detected and suppressed errors, rerun with: -v
==28639== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 116 from 20)
```

Como vemos los errores son nulos.
Es importante saber que existen parámetros que se le pasan a valgrind para hacer que muestre más información en su salida, ya que con el comando anterior solo nos muestra la mínima información:

```
valgrind --tool=memcheck --leak-check=yes --leak-check=full ./a.out
```

Operador sizeOf()

Es un operador unario que recibe tanto variables como tipos y devuelve el tamaño en bytes que estos ocupan.
Como ejemplo, en un procesador de 32 bits al realizar el sizeOf() de un char y un int obtenemos 1 y 2 respectivamente, que son los bytes necesarios para representar dichos tipos de datos. En cambio si hacemos sizeOf() char * e int * obtendremos 4 en ambos casos, ya que son los bytes que ocupan en memoria cada tipo de dato, ya que ambos son punteros a memoria.
Al utitlizar el operador sizeOf() sobre un tipo de dato struct, lo que obtenemos es la suma de los sizeOf() de cada miembro del struct más una cantidad de bytes que inserta el compilador para hacer que la estructura sea más eficiente.
Por ejemplo: tenemos el siguiente archivo .c:

```c
struct employee
{
    int id;
    char name[2];
};

int main( int argc, char *argv[])
{
    struct employee e1;
    printf("Size of struct: %d\n", sizeof(e1));
    printf("Size of id: %d\n", sizeof(e1.id));
    printf("Size of name: %d\n", sizeof(e1.name));

    return 0;
}
```

Compilamos y corremos el programa y tenemos cómo salida lo siguiente:

```
MacBook-Air-de-Gaston:TP0 gastonmontes$ ./a.out
Size of struct: 8
Size of id: 4
Size of name: 2
```

Claramente, el sizeof() de un struct no es igual a la suma de todos sus componentes.

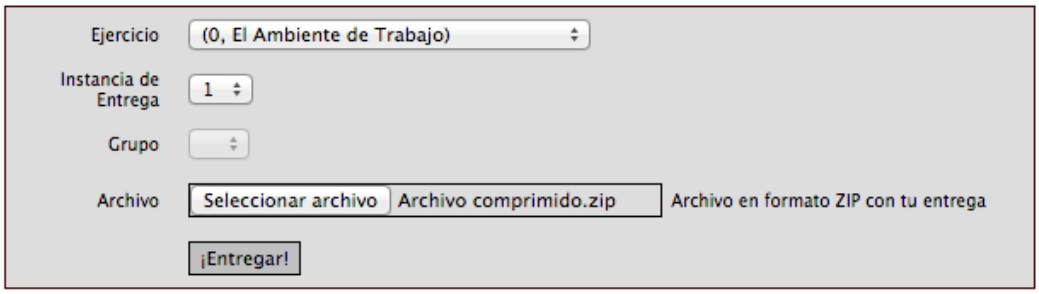
SERCOM - Error de compilación.

Entrega:

Me dirijo al sitio web de la cátedra, voy a la pestaña de SERCOM y me logueo con mi usuario y contraseña.
Una vez dentro del sitio voy a “Mis entregas” y subo el archivo .zip comprimido con los archivos .c que me provee la cátedra para cada paso. Para realizar el comprimido descargue el enunciado, lo descomprimi y volví a comprimir el archivo .c correspondiente a cada paso dejando afuera el archivo pdf que contiene al enunciado y los demás archivos .c que corresponden a los puntos siguientes o anteriores, ya que se especifica que sean solo archivos .h, .cpp y .c salvo caso explícito en el enunciado.

Crear Nueva Entrega

Si aún no lo hizo le recomendamos leer detenidamente la guía de entrega de trabajos de la cátedra.



Acerca del archivo entregable:

- El formato debe ser ZIP, sin excepción, e incluir solamente los elementos requeridos por la entrega.
- Debe incluir el código fuente (.C, .CPP, .H)
    1. Utilizando el caracter TAB o hasta cuatro espacios para el sangrado (indent).
    2. Usando el conjunto de caracteres UTF-8 (sobre todo si hay letars acentuadas).
- No debe incluir código objeto (.O, .OBJ, etc.) o archivos de formato binario salvo expresa indicación en el enunciado.
- No debe incluir Makefile, script de compilación o similar salvo indicación en el enunciado.
- No debe incluir estructura de carpetas. Los archivos fuente deben encontrarse en la raíz del comprimido.

Vemos inmediatamente que el sistema de SERCOM me rechaza la entrega debido a fallas en los test que corre:

Administración de Entregas

Ejercicio	Resultado	Fecha	Duración	Observaciones	Operaciones
0.1 (El Ambiente de Trabajo)	Rechazado	2014-08-26 16:15:24	0:00:00		Corrida Bajar Navegar PDF

Mas especificamente:

Comandos Ejecutados

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2014-08-26 16:15:32	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 2.		Bajar Todo __stdouterr__

Pruebas Realizadas

Si bajo el archivo me tira los errores específicos que se produjeron:

```
  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
```

Veo que en el archivo p2, en la función main, en la línea 10 encontró una declaración de la función ztrcpy no especificada, o sea no reconoce el compilador dicha función.
En la línea 14 del mismo archivo y misma función, el compilador desconoce la declaración de la función malloc.
Ambos errores encontrados son errores de compilación.

SERCOM - Normas de programación y código de salida.

Corrección de errores:

Corregimos los errores del archivo p2.c, cambiando el ztrcpy por la función conocida strcpy e incluyendo la librería en donde se encuentra declarada la función malloc, esta librería se llama stdlib, por ello incluyo su .h.

```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;
}
```

Reentrega:

A continuación, vuelvo a crear un archivo comprimido con el p2.c del proyecto y realizo nuevamente la entrega. Para realizar esto, en la parte de “Administración de entregas” selecciono el botón “Agregar” y subo el nuevo comprimido con la nueva entrega.
Vemos que se agrega una nueva row en “Administración de entregas”. Esta entrega también falló, así que vamos a la parte de corrida y nos fijamos su error:

Comandos Ejecutados

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2014-08-26 17:00:35	0:00:00	Sí			Bajar Todo __stdouterr__
1	Verificar Normas Codificación	python ./cpplint.py --filter=`cat filter_options` `find -regextype posix-egrep -regex '.*\.(h|hpp|c|cpp)'`	2014-08-26 17:00:35	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo __stdouterr__

Vemos que SERCOM compiló correctamente el tp entregado, pero que hubo fallas con la tarea “Verificar Normas Codificación” y el código de retorno fue 1 y no 0 como se esperaba. Vamos a los archivos guardados para obtener una mejor descripción de lo ocurrido:

```
./p2.c:5:  Extra space after ( in function call  [whitespace/parens] [4]
./p2.c:11:  Extra space after ( in function call  [whitespace/parens] [4]
./p2.c:11:  Extra space before )  [whitespace/parens] [2]
./p2.c:11:  Almost always, snprintf is better than strcpy  [runtime/printf] [4]
./p2.c:12:  Extra space after ( in function call  [whitespace/parens] [4]
./p2.c:12:  Extra space before )  [whitespace/parens] [2]
./p2.c:13:  Missing space before ( in if(  [whitespace/parens] [5]
./p2.c:15:  Extra space after ( in function call  [whitespace/parens] [4]
./p2.c:17:  Missing space before ( in while(  [whitespace/parens] [5]
./p2.c:20:  Missing space before ( in if(  [whitespace/parens] [5]
./p2.c:21:  Extra space after ( in function call  [whitespace/parens] [4]
./p2.c:21:  Extra space before )  [whitespace/parens] [2]
Done processing ./p2.c
./__MACOSX/._p2.c:1:  Lines should very rarely be longer than 100 characters  [whitespace/line_length] [4]
./__MACOSX/._p2.c:1:  Missing space after ;  [whitespace/semicolon] [3]
./__MACOSX/._p2.c:1:  Line contains invalid UTF-8 (or Unicode replacement character).  [readability/utf8] [5]
./__MACOSX/._p2.c:1:  Could not find a newline character at the end of the file.  [whitespace/ending_newline] [5]
Done processing ./__MACOSX/._p2.c
Total errors found: 16
```

Vamos que la mayoría de los errores son de espacios mal introducidos o espacios faltantes. También podemos ver las pruebas realizadas por SERCOM, las que fallaron y las que fueron exitosas:

Pruebas Realizadas

1- Archivo inexistente.

Comando	./tp no-existo
Inicio / Fin	2014-08-26 17:00:35 / 2014-08-26 17:00:36

#	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	Sí			Bajar Todo valgrind.out

2- Archivo existente.

Comando	./tp archivo-corto.txt
Inicio / Fin	2014-08-26 17:00:36 / 2014-08-26 17:00:37

#	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	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo valgrind.out

3- Nombre largo.

Comando	./tp soy-un-archivo-con-nombre-largo.txt
Inicio / Fin	2014-08-26 17:00:37 / 2014-08-26 17:00:37

#	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	0:00:00	No	La salida estándar no coincide con lo esperado (archivo "__stdout__.diff").	Bajar Ver	Bajar Todo valgrind.out

4- Contenido grande.

Comando	./tp archivo-largo.txt
Inicio / Fin	2014-08-26 17:00:37 / 2014-08-26 17:00:38

#	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	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo valgrind.out

5- Entrada estandar.

Comando	./tp
Inicio / Fin	2014-08-26 17:00:38 / 2014-08-26 17:00:38

#	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 139. 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	La salida estándar no coincide con lo esperado (archivo "__stdout__.diff").	Bajar Ver	Bajar Todo valgrind.out

Vamos que en la primer prueba, al correr hubo una falla y el retorno no fue el esperado, se esperaba un código de retorno 0 pero se obtuvo un 1. Sin embargo, al correr con valgrind la prueba dió correcta, o sea que valgrind no encontró errores para esta prueba específica.

SERCOM - Pérdida de memoria.

Corregimos los errores:

Eliminamos los spacings incorrectos, reemplazamos los números mágicos por constantes, corregimos el código de retorno y volvemos a entregar:

```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;
}
```

snprintf.

Vemos que la compilación fue correcta nuevamente, pero que volvió a fallar la verificación de normas de codificación: nos indica que es mejor utilizar snprintf a utilizar strcpy. Algo que por el momento vamos a ignorar:

Comandos Ejecutados

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2014-08-26 17:14:02	0:00:00	Sí			Bajar Todo __stdouterr__
1	Verificar Normas Codificación	python ./cpplint.py --filter=`cat filter_options` `find -regextype posix-egrep -regex '.*\.(h|hpp|c|cpp)'`	2014-08-26 17:14:02	0:00:01	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo __stdouterr__

```
./p2.c:16:  Almost always, snprintf is better than strcpy  [runtime/printf] [4]
Done processing ./p2.c
./__MACOSX/._p2.c:1:  Lines should very rarely be longer than 100 characters  [whitespace/line_length] [4]
./__MACOSX/._p2.c:1:  Missing space after ;  [whitespace/semicolon] [3]
./__MACOSX/._p2.c:1:  Line contains invalid UTF-8 (or Unicode replacement character).  [readability/utf8] [5]
./__MACOSX/._p2.c:1:  Could not find a newline character at the end of the file.  [whitespace/ending_newline] [5]
Done processing ./__MACOSX/._p2.c
Total errors found: 5
```

Prueba 1 superada.

Observamos que ahora la prueba 1 fue superada tanto al probar normalmente como al probar con valgrind:

Pruebas Realizadas

1- Archivo inexistente.

Comando	./tp no-existo
Inicio / Fin	2014-08-26 17:14:03 / 2014-08-26 17:14:03

#	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	0:00:00	Sí			Bajar Todo valgrind.out

2- Archivo existente.

Comando	./tp archivo-corto.txt
Inicio / Fin	2014-08-26 17:14:03 / 2014-08-26 17:14:04

#	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	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo valgrind.out

3- Nombre largo.

Comando	./tp soy-un-archivo-con-nombre-largo.txt
Inicio / Fin	2014-08-26 17:14:04 / 2014-08-26 17:14:04

#	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	0:00:00	No	La salida estándar no coincide con lo esperado (archivo "__stdout__.diff").	Bajar Ver	Bajar Todo valgrind.out

4- Contenido grande.

Comando	./tp archivo-largo.txt
Inicio / Fin	2014-08-26 17:14:04 / 2014-08-26 17:14:05

#	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	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo valgrind.out

5- Entrada estandar.

Comando	./tp
Inicio / Fin	2014-08-26 17:14:05 / 2014-08-26 17:14:06

#	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 139. 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

Prueba 2 - Error de valgrind

Como podemos ver en la captura anterior, la prueba 2 corrió correctamente en la prueba normal, pero al hacer la misma prueba con valgrind falló. Los errores son los detallados a continuación:

```
==00:00:00:00.000 19969== Memcheck, a memory error detector
==00:00:00:00.000 19969== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==00:00:00:00.000 19969== Using Valgrind-3.5.0-Debian and LibVEX; rerun with -h for copyright info
==00:00:00:00.000 19969== Command: ./tp archivo-corto.txt
==00:00:00:00.000 19969== Parent PID: 19968
==00:00:00:00.000 19969==
==00:00:00:00.404 19969==
==00:00:00:00.404 19969== FILE DESCRIPTORS: 3 open at exit.
==00:00:00:00.404 19969== Open file descriptor 2: archivo-corto.txt
==00:00:00:00.404 19969==    at 0x41116FE: __open_nocancel (in /lib/libc-2.10.1.so)
==00:00:00:00.404 19969==    by 0x40BB7E7: _IO_file_fopen@@GLIBC_2.1 (fileops.c:335)
==00:00:00:00.404 19969==    by 0x40AF9AC: __fopen_internal (iofopen.c:93)
==00:00:00:00.404 19969==    by 0x40AFA0B: fopen@@GLIBC_2.1 (iofopen.c:107)
==00:00:00:00.404 19969==    by 0x8048603: main (p2.c:17)
==00:00:00:00.404 19969==
==00:00:00:00.404 19969== Open file descriptor 1: /var/lib/sercom/sercom/var/tmp/prueba.199176.stdout
==00:00:00:00.404 19969==    <inherited from parent>
==00:00:00:00.405 19969==
==00:00:00:00.405 19969== Open file descriptor 0: /home/sercom/test/valgrind.out
==00:00:00:00.405 19969==    <inherited from parent>
==00:00:00:00.405 19969==
==00:00:00:00.405 19969==
==00:00:00:00.405 19969== HEAP SUMMARY:
==00:00:00:00.405 19969==     in use at exit: 356 bytes in 2 blocks
==00:00:00:00.405 19969==   total heap usage: 2 allocs, 0 frees, 356 bytes allocated
==00:00:00:00.405 19969==
==00:00:00:00.405 19969== 4 bytes in 1 blocks are definitely lost in loss record 1 of 2
==00:00:00:00.405 19969==    at 0x4024C1C: malloc (vg_replace_malloc.c:195)
==00:00:00:00.405 19969==    by 0x804861A: main (p2.c:20)
==00:00:00:00.405 19969==
==00:00:00:00.405 19969== LEAK SUMMARY:
==00:00:00:00.405 19969==    definitely lost: 4 bytes in 1 blocks
==00:00:00:00.406 19969==    indirectly lost: 0 bytes in 0 blocks
==00:00:00:00.406 19969==      possibly lost: 0 bytes in 0 blocks
==00:00:00:00.406 19969==    still reachable: 352 bytes in 1 blocks
==00:00:00:00.406 19969==         suppressed: 0 bytes in 0 blocks
==00:00:00:00.406 19969== Reachable blocks (those to which a pointer was found) are not shown.
==00:00:00:00.406 19969== To see them, rerun with: --leak-check=full --show-reachable=yes
==00:00:00:00.406 19969==
==00:00:00:00.406 19969== For counts of detected and suppressed errors, rerun with: -v
==00:00:00:00.406 19969== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 14 from 7)
[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.
```

Donde claramente nos informa que hubo 2 allocs y 0 frees, algo que esta mal, todo lo que se hace alloc debe ser liberado.

SERCOM - Escrituras fuera de rango.

Corrección

Agrego el cierre del archivo y libero el buffer para eliminar los errores detallados anteriormente:

```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;
}
```

Luego realizo la reentrega del TP.

snprintf

Volvemos a ver la misma falla en la tarea de normas de codificación. Seguimos ignorandola.

Prueba 1 superada

Observamos que la prueba 1 volvió a ser superada como detallamos anteriormente.

Prueba 2 superada

Observamos que ahora la prueba 2 también fue superada:

Pruebas Realizadas

1- Archivo inexistente.

Comando	./tp no-existo
Inicio / Fin	2014-08-26 17:25:19 / 2014-08-26 17:25:20

#	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

2- Archivo existente.

Comando	./tp archivo-corto.txt
Inicio / Fin	2014-08-26 17:25:20 / 2014-08-26 17:25:20

#	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	0:00:00	Sí			Bajar Todo valgrind.out

3- Nombre largo.

Comando	./tp soy-un-archivo-con-nombre-largo.txt
Inicio / Fin	2014-08-26 17:25:20 / 2014-08-26 17:25:21

#	Tarea	Comando	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Correr	Prueba normalmente, sin filtros	0:00:01	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	0:00:00	No	La salida estándar no coincide con lo esperado (archivo "__stdout__.diff").	Bajar Ver	Bajar Todo valgrind.out

4- Contenido grande.

Comando	./tp archivo-largo.txt
Inicio / Fin	2014-08-26 17:25:21 / 2014-08-26 17:25:22

#	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

5- Entrada estandar.

Comando	./tp
Inicio / Fin	2014-08-26 17:25:22 / 2014-08-26 17:25:22

#	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 139. 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	La salida estándar no coincide con lo esperado (archivo "__stdout__.diff").	Bajar Ver	Bajar Todo valgrind.out

Y el detalle de valgrind de la prueba 2 es el siguiente:

```
==00:00:00:00.000 20085== Memcheck, a memory error detector
==00:00:00:00.000 20085== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==00:00:00:00.000 20085== Using Valgrind-3.5.0-Debian and LibVEX; rerun with -h for copyright info
==00:00:00:00.000 20085== Command: ./tp archivo-corto.txt
==00:00:00:00.000 20085== Parent PID: 20084
==00:00:00:00.000 20085==
==00:00:00:00.378 20085==
==00:00:00:00.378 20085== FILE DESCRIPTORS: 2 open at exit.
==00:00:00:00.378 20085== Open file descriptor 1: /var/lib/sercom/sercom/var/tmp/prueba.199194.stdout
==00:00:00:00.378 20085==    <inherited from parent>
==00:00:00:00.378 20085==
==00:00:00:00.378 20085== Open file descriptor 0: /home/sercom/test/valgrind.out
==00:00:00:00.378 20085==    <inherited from parent>
==00:00:00:00.378 20085==
==00:00:00:00.378 20085==
==00:00:00:00.378 20085== HEAP SUMMARY:
==00:00:00:00.378 20085==     in use at exit: 0 bytes in 0 blocks
==00:00:00:00.378 20085==   total heap usage: 2 allocs, 2 frees, 356 bytes allocated
==00:00:00:00.378 20085==
==00:00:00:00.378 20085== All heap blocks were freed -- no leaks are possible
==00:00:00:00.378 20085==
==00:00:00:00.378 20085== For counts of detected and suppressed errors, rerun with: -v
==00:00:00:00.378 20085== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 14 from 7)
[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.
```

Donde especifica que no tengo leaks.

Prueba 3 - Falla

Vemos que la prueba 3 dió falla tanto en correr normalmente como correr con valgrind. El error es bastante complejo hasta que entramos al detalle de los archivos guardados de la prueba de valgrind:

```
==00:00:00:00.000 20101== Memcheck, a memory error detector
==00:00:00:00.000 20101== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==00:00:00:00.000 20101== Using Valgrind-3.5.0-Debian and LibVEX; rerun with -h for copyright info
==00:00:00:00.000 20101== Command: ./tp soy-un-archivo-con-nombre-largo.txt
==00:00:00:00.000 20101== Parent PID: 20100
==00:00:00:00.000 20101==
**00:00:00:00.295 20101** *** strcpy_chk: buffer overflow detected ***: program terminated
==00:00:00:00.295 20101==    at 0x4027041: VALGRIND_PRINTF_BACKTRACE (valgrind.h:3720)
==00:00:00:00.295 20101==    by 0x402721F: __strcpy_chk (mc_replace_strmem.c:739)
==00:00:00:00.295 20101==    by 0x8048664: main (string3.h:106)
==00:00:00:00.321 20101==
==00:00:00:00.321 20101== FILE DESCRIPTORS: 2 open at exit.
==00:00:00:00.321 20101== Open file descriptor 1: /var/lib/sercom/sercom/var/tmp/prueba.199197.stdout
==00:00:00:00.321 20101==    <inherited from parent>
==00:00:00:00.321 20101==
==00:00:00:00.321 20101== Open file descriptor 0: /home/sercom/test/valgrind.out
==00:00:00:00.321 20101==    <inherited from parent>
==00:00:00:00.321 20101==
==00:00:00:00.321 20101==
==00:00:00:00.321 20101== HEAP SUMMARY:
==00:00:00:00.321 20101==    in use at exit: 0 bytes in 0 blocks
==00:00:00:00.321 20101==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==00:00:00:00.321 20101==
==00:00:00:00.321 20101== All heap blocks were freed -- no leaks are possible
==00:00:00:00.321 20101==
==00:00:00:00.321 20101== For counts of detected and suppressed errors, rerun with: -v
==00:00:00:00.321 20101== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 14 from 7)
[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 soy-un-archivo-con-nombre-largo.txt
[SERCOM] Error code configured for Valgrind: 42.
[SERCOM] Valgrind execution result: 127.
[SERCOM] Valgrind result: Success.
```

Notamos que en una de las líneas indica que existe un overflow con el buffer pasado a la función de strcpy:

```
**00:00:00:00.295 20101** *** strcpy_chk: buffer overflow detected ***: program terminated
```

La línea de comando de ejecución de la prueba 3 es la siguiente: ./tp soy-un-archivo-con-nombre-largo.txt

Esto se debe a que el nombre de archivo recibido por parámetro es de un largo mayor a 25 caracteres cuando nosotros le ponemos un largo máximo de 20 caracteres al buffer.

Esto se salvaría elevando el número de caracteres del nombre esperado a un número mayor a 50 ya que la función strncpy lo único que hace es recibir un parámetro más indicando la cantidad de caracteres a copiar, si le indicamos 20 simplemente va a copiar el nombre del archivo incompleto aunque no nos daría un overflow.

Un overflow se produce cuando nuestro programa copia en una sobre un área de memoria reservada (buffer) una mayor cantidad de datos que esta puede contener, sobreescribiendo datos continuos no reservados por este buffer.

Un segmentation fault es cuando desde nuestro código se desea acceder a cierta información pero no tiene permisos para hacerlo.

Contenido de archivos:

A continuación detallé el contenido de los archivos utilizados para las pruebas 2 y 4 que dieron satisfactorias:

Archivo corto:

```
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 compaÒÌa
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 Èl
a travÈs del periÛdico. Los casos son resueltospor la lÛgica y el
razonamiento del famoso detective. Son cuentos de misterio, donde interviene
la intriga y la aventura, unido al an·lisis psicolÛgico de sus personajes.
```

Archivo largo:

```
Rene Geronimo Favaloro (La Plata, Argentina, 12 de julio de 1923 - Buenos
Aires, Argentina, 29 de julio de 2000) fue un prestigioso medico 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, despues de escribir una carta al
Presidente De la Rua criticando al sistema de salud, se quito la vida de un
disparo al corazon.
```

SERCOM - Entrada estandar.

Corrección:

Se elimina el error anterior con el nombre eliminando el buffer y el strcpy, tomando el nombre del archivo directamente desde los parámetros pasados:

```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;
}
```

Luego realizamos la reentrega.

Verificar normas de codificación

Lamentablemente, a pesar de seguir todos los pasos especificados por los docentes, esta prueba se me hizo imposible superarla:

Comandos Ejecutados

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2014-08-26 17:55:04	0:00:00	Sí			Bajar Todo __stdouterr__
1	Verificar Normas Codificación	python ./cpplint.py --filter=`cat filter_options` `find -regextype posix-egrep -regex '.*\.(h|hpp|c|cpp)'`	2014-08-26 17:55:04	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo __stdouterr__

El archivo guardado:

```
Done processing ./p2.c
./__MACOSX/._p2.c:1:  Lines should very rarely be longer than 100 characters  [whitespace/line_length] [4]
./__MACOSX/._p2.c:1:  Missing space after ;  [whitespace/semicolon] [3]
./__MACOSX/._p2.c:1:  Line contains invalid UTF-8 (or Unicode replacement character).  [readability/utf8] [5]
./__MACOSX/._p2.c:1:  Could not find a newline character at the end of the file.  [whitespace/ending_newline] [5]
Done processing ./__MACOSX/._p2.c
Total errors found: 4
```

Igualmente notamos que el error por utilizar el strcpy desapareció del log de archivos gaurdados.

Prueba 5 - Falla

Notamos que ahora solamente la prueba 5 es la que falla:

Pruebas Realizadas

1- Archivo inexistente.

Comando	./tp no-existo
Inicio / Fin	2014-08-26 17:54:30 / 2014-08-26 17:54:31

#	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

2- Archivo existente.

Comando	./tp archivo-corto.txt
Inicio / Fin	2014-08-26 17:54:31 / 2014-08-26 17:54:31

#	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	0:00:00	Sí			Bajar Todo valgrind.out

3- Nombre largo.

Comando	./tp soy-un-archivo-con-nombre-largo.txt
Inicio / Fin	2014-08-26 17:54:31 / 2014-08-26 17:54:32

#	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

4- Contenido grande.

Comando	./tp archivo-largo.txt
Inicio / Fin	2014-08-26 17:54:32 / 2014-08-26 17:54:32

#	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	0:00:00	Sí			Bajar Todo valgrind.out

5- Entrada estandar.

Comando	./tp
Inicio / Fin	2014-08-26 17:54:32 / 2014-08-26 17:54:33

#	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

Notamos que el archivo de valgrind nos dice que hay un error con el parametro de entrada, ya que no le enviamos ninguno:

```
==00:00:00:00.000 20230== Memcheck, a memory error detector
==00:00:00:00.000 20230== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==00:00:00:00.000 20230== Using Valgrind-3.5.0-Debian and LibVEX; rerun with -h for copyright info
==00:00:00:00.000 20230== Command: ./tp
==00:00:00:00.000 20230== Parent PID: 20229
==00:00:00:00.000 20230==
==00:00:00:00.318 20230== Syscall param open(filename) points to unaddressable byte(s)
==00:00:00:00.318 20230==    at 0x41116FE: __open_nocancel (in /lib/libc-2.10.1.so)
==00:00:00:00.318 20230==    by 0x40BB7E7: _IO_file_fopen@@GLIBC_2.1 (fileops.c:335)
==00:00:00:00.318 20230==    by 0x40AF9AC: __fopen_internal (iofopen.c:93)
==00:00:00:00.318 20230==    by 0x40AFA0B: fopen@@GLIBC_2.1 (iofopen.c:107)
==00:00:00:00.318 20230==    by 0x80485C0: main (p2.c:15)
==00:00:00:00.318 20230==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==00:00:00:00.318 20230==
==00:00:00:00.365 20230==
==00:00:00:00.365 20230== FILE DESCRIPTORS: 3 open at exit.
==00:00:00:00.365 20230== Open file descriptor 2: /home/sercom/test/valgrind.out
==00:00:00:00.365 20230==    <inherited from parent>
==00:00:00:00.365 20230==
==00:00:00:00.365 20230== Open file descriptor 1: /var/lib/sercom/sercom/var/tmp/prueba.199221.stdout
==00:00:00:00.365 20230==    <inherited from parent>
==00:00:00:00.365 20230==
==00:00:00:00.365 20230== Open file descriptor 0: /var/lib/sercom/sercom/var/tmp/prueba.199221.stdin
==00:00:00:00.365 20230==    <inherited from parent>
==00:00:00:00.365 20230==
==00:00:00:00.365 20230==
==00:00:00:00.365 20230== HEAP SUMMARY:
==00:00:00:00.365 20230==     in use at exit: 0 bytes in 0 blocks
==00:00:00:00.365 20230==   total heap usage: 1 allocs, 1 frees, 352 bytes allocated
==00:00:00:00.365 20230==
==00:00:00:00.365 20230== All heap blocks were freed -- no leaks are possible
==00:00:00:00.365 20230==
==00:00:00:00.365 20230== For counts of detected and suppressed errors, rerun with: -v
==00:00:00:00.365 20230== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 14 from 7)
[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
[SERCOM] Error code configured for Valgrind: 42.
[SERCOM] Valgrind execution result: 42.
[SERCOM] Valgrind result: Failure.
```

Especificamente en las siguientes líneas:

```
==00:00:00:00.318 20230== Syscall param open(filename) points to unaddressable byte(s)
==00:00:00:00.318 20230==    at 0x41116FE: __open_nocancel (in /lib/libc-2.10.1.so)
==00:00:00:00.318 20230==    by 0x40BB7E7: _IO_file_fopen@@GLIBC_2.1 (fileops.c:335)
==00:00:00:00.318 20230==    by 0x40AF9AC: __fopen_internal (iofopen.c:93)
==00:00:00:00.318 20230==    by 0x40AFA0B: fopen@@GLIBC_2.1 (iofopen.c:107)
==00:00:00:00.318 20230==    by 0x80485C0: main (p2.c:15)
==00:00:00:00.318 20230==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
```

SERCOM - Entrega exitosa

Corrección

Agregamos un if() para verificar que venga el nombre del archivo en los argumentos y salvamos el error anterior:

```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;
}
```

Hacemos la entrega nuevamente y miramos los errores.

Verificar normas de codificación

Lamentablemente, a pesar de seguir todos los pasos especificados por los docentes, esta prueba se me hizo imposible superarla:

Comandos Ejecutados

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2014-08-26 17:55:04	0:00:00	Sí			Bajar Todo __stdouterr__
1	Verificar Normas Codificación	python ./cpplint.py --filter=`cat filter_options` `find -regextype posix-egrep -regex '.*\.(h|hpp|c|cpp)'`	2014-08-26 17:55:04	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo __stdouterr__

El archivo guardado:

```
Done processing ./p2.c
./__MACOSX/._p2.c:1:  Lines should very rarely be longer than 100 characters  [whitespace/line_length] [4]
./__MACOSX/._p2.c:1:  Missing space after ;  [whitespace/semicolon] [3]
./__MACOSX/._p2.c:1:  Line contains invalid UTF-8 (or Unicode replacement character).  [readability/utf8] [5]
./__MACOSX/._p2.c:1:  Could not find a newline character at the end of the file.  [whitespace/ending_newline] [5]
Done processing ./__MACOSX/._p2.c
Total errors found: 4
```

Pruebas exitosas.

Notamos que ahora todas las pruebas fueron exitosas:

Pruebas Realizadas

1- Archivo inexistente.

Comando	./tp no-existo
Inicio / Fin	2014-08-26 17:55:04 / 2014-08-26 17:55:04

#	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	0:00:00	Sí			Bajar Todo valgrind.out

2- Archivo existente.

Comando	./tp archivo-corto.txt
Inicio / Fin	2014-08-26 17:55:04 / 2014-08-26 17:55:05

#	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	0:00:00	Sí			Bajar Todo valgrind.out

3- Nombre largo.

Comando	./tp soy-un-archivo-con-nombre-largo.txt
Inicio / Fin	2014-08-26 17:55:05 / 2014-08-26 17:55:06

#	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

4- Contenido grande.

Comando	./tp archivo-largo.txt
Inicio / Fin	2014-08-26 17:55:06 / 2014-08-26 17:55:06

#	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	0:00:00	Sí			Bajar Todo valgrind.out

5- Entrada estandar.

Comando	./tp
Inicio / Fin	2014-08-26 17:55:06 / 2014-08-26 17:55:07

#	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

Comando stdin y operadores <>

Los comandos de la terminal con los que interactuamos, tienen tres tipos de flujos de entrada/salida: stdin, stdout y srderr.
stdin: entrada estándar del comando, donde recibe los parámetros. Generalmente y por defecto está asociada al teclado.
El operador > redirige la salida a un archivo. Si no existe el archivo se crea y si el archivo existe y no está vacío, el contenido es reemplazado por la salida del comando.
El operador < se utiliza para la entrada.

Ejecución de prueba 5.

Me situo en la carpeta donde se encuentran los archivos fuentes del tp, compilo el archivo correspondiente a la última entrega del tp:

```
MacBook-Air-de-Gaston:TP0 gastonmontes$ gcc p7.c -o tp.out
```

Notar que llame al archivo resultante como tp.out.
Correr este archivo enviandole con el operador < los archivos que quiero que sean de entrada, en mi caso lo hice con los 3 archivos de textos provistos por la cátedra:

```
MacBook-Air-de-Gaston:TP0 gastonmontes$ ./tp.out < soy-un-archivo-con-nombre-lar
go.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.
```

Con el archivo de texto nombrado “archivo-corto.txt”:

```
MacBook-Air-de-Gaston:TP0 gastonmontes$ ./tp.out < 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.
```

Y por último con el archivo de texto “archivo-largo.txt”:

```
MacBook-Air-de-Gaston:TP0 gastonmontes$ ./tp.out < archivo-largo.txt
Rene Geronimo Favaloro (La Plata, Argentina, 12 de julio de 1923 - Buenos
Aires, Argentina, 29 de julio de 2000) fue un prestigioso medico 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, despues de escribir una carta al
Presidente De la Rua criticando al sistema de salud, se quito la vida de un
disparo al corazon.
```

Ejecución prueba 2.

Como ya compilamos en el apartado anterior, entonces simplemente procedemos a realizar la prueba número 2 con una salida en otro archivo .txt llamado salida.txt:

```
MacBook-Air-de-Gaston:TP0 gastonmontes$ ./tp.out archivo-corto.txt > salida.txt
```

En el archivo salida.txt encuentro la siguiente salida:

```
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 compaÒÌa
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 Èl
a travÈs del periÛdico. Los casos son resueltospor la lÛgica y el
razonamiento del famoso detective. Son cuentos de misterio, donde interviene
la intriga y la aventura, unido al an·lisis psicolÛgico de sus personajes.
```

Aclaraciones del alumno

Error en la tarea de verificar normas de programación

El TP entregado falló en la tarea de verificar normas de programación. Esto se dió porque al comprimir el archivo para enviar el .zip, se le agregan otros dos archivos que hacen que la tarea falle.

Para eliminar este error y por pedido de los ayudantes, el proceso de compresión de los archivos debe hacerse mediante la terminal, para evitar usar compresores que agreguen datos a los .zip.

Para comprimir el archivo usamos el comando zip de la siguiente manera:

zip -r archive_name.zip folder_or_files folder_or_files_2

Para verificar que el archivo tenga solamente los datos que queremos hacemos:

unzip -l archive_name.zip.

En mi caso particular cree un archivo .zip a partir del archivo p7.c:

```
MacBook-Air-de-Gaston:TP0 gastonmontes$ zip -r tp.zip p7.c
  adding: p7.c (deflated 47%)
```

Luego verifico que el archivo comprimido contenga solamente el archivo que queremos:

```
MacBook-Air-de-Gaston:TP0 gastonmontes$ unzip -l tp.zip
Archive:  tp.zip
  Length      Date    Time    Name
 --------    ----    ----    ----
      643  08-26-14 19:49   p7.c
 --------                   -------
      643                   1 file
```

Entrega forzada por los ayudantes.

Debido al error de compresión, los ayudantes me pidieron que les envíe nuevamente el .zip para que ellos puedan forzar la entrega del TP.
Al ingresar al administrador de entrega ahora vemos que la primera row contiene la entrega aceptada por SERCOM:

Administración de Entregas

Ejercicio	Resultado	Fecha	Duración	Observaciones	Operaciones
0.1 (El Ambiente de Trabajo)	Aceptado	2014-08-27 10:35:25	0:00:03	Entrega realizada manualmente por el docente Gabriel Fusca	Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2014-08-26 17:54:58	0:00:04		Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2014-08-26 17:54:19	0:00:04		Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2014-08-26 17:25:10	0:00:04		Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2014-08-26 17:13:52	0:00:04		Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2014-08-26 17:00:28	0:00:03		Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2014-08-26 16:57:21	0:00:01		Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2014-08-26 16:39:42	0:00:01		Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2014-08-26 16:15:24	0:00:00		Corrida Bajar Navegar PDF

Selecciono la opción “Corrida” y noto que la tarea de “Verificar Normas de Codificación” tuvo éxito:

Comandos Ejecutados

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2014-08-27 10:35:35	0:00:00	Sí			Bajar Todo stdouterr
1	Verificar Normas Codificación	python ./cpplint.py --filter=`cat filter_options` `find -regextype posix-egrep -regex '.*\.(h|hpp|c|cpp)'`	2014-08-27 10:35:35	0:00:00	Sí			Bajar Todo stdouterr

Y el log de valgrind de dicha prueba es el siguiente:

```
==00:00:00:00.000 20743== Memcheck, a memory error detector
==00:00:00:00.000 20743== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==00:00:00:00.000 20743== Using Valgrind-3.5.0-Debian and LibVEX; rerun with -h for copyright info
==00:00:00:00.000 20743== Command: ./tp no-existo
==00:00:00:00.000 20743== Parent PID: 20742
==00:00:00:00.000 20743==
==00:00:00:00.376 20743==
==00:00:00:00.376 20743== FILE DESCRIPTORS: 1 open at exit.
==00:00:00:00.376 20743== Open file descriptor 0: /home/sercom/test/valgrind.out
==00:00:00:00.376 20743==    <inherited from parent>
==00:00:00:00.376 20743==
==00:00:00:00.376 20743==
==00:00:00:00.376 20743== HEAP SUMMARY:
==00:00:00:00.376 20743==     in use at exit: 0 bytes in 0 blocks
==00:00:00:00.376 20743==   total heap usage: 1 allocs, 1 frees, 352 bytes allocated
==00:00:00:00.376 20743==
==00:00:00:00.376 20743== All heap blocks were freed -- no leaks are possible
==00:00:00:00.376 20743==
==00:00:00:00.376 20743== For counts of detected and suppressed errors, rerun with: -v
==00:00:00:00.376 20743== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 14 from 7)
[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 no-existo
[SERCOM] Error code configured for Valgrind: 42.
[SERCOM] Valgrind execution result: 1.
[SERCOM] Valgrind result: Success.
```

Las pruebas realizadas también fueron exitosas en su totalidad:

Pruebas Realizadas

1- Archivo inexistente.

Comando	./tp no-existo
Inicio / Fin	2014-08-27 10:35:35 / 2014-08-27 10:35:36

#	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	0:00:00	Sí			Bajar Todo valgrind.out

2- Archivo existente.

Comando	./tp archivo-corto.txt
Inicio / Fin	2014-08-27 10:35:36 / 2014-08-27 10:35:37

#	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

3- Nombre largo.

Comando	./tp soy-un-archivo-con-nombre-largo.txt
Inicio / Fin	2014-08-27 10:35:37 / 2014-08-27 10:35:37

#	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	0:00:00	Sí			Bajar Todo valgrind.out

4- Contenido grande.

Comando	./tp archivo-largo.txt
Inicio / Fin	2014-08-27 10:35:37 / 2014-08-27 10:35:38

#	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

5- Entrada estandar.

Comando	./tp
Inicio / Fin	2014-08-27 10:35:38 / 2014-08-27 10:35:38

#	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	0:00:00	Sí			Bajar Todo valgrind.out

Código devuelto por SERCOM

Una vez que la entrega fue exitosa, SERCOM nos devuelve el código del programa en formato PDF. Lo descargo y hago capturas de pantalla de este código, aunque se entrega impreso por separado junto con el informe:

```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;
}
```