

Informe del Trabajo Práctico 0

Alumno: Gavrilov Vsevolod

Padrón: 96252

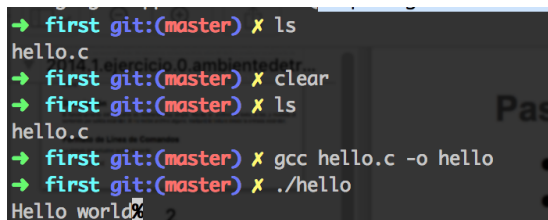
Paso 1: Comenzando

- Valgrind es una herramienta que permite analizar dinámicamente los programas. Es de ayuda para, por ejemplo, detectar leaks de memoria.
- `sizeof` es una función que retorna la cantidad de bytes ocupados por una variable o necesarios para una variable del tipo especificado. La salida siempre depende de la arquitectura, por ejemplo en una arquitectura de 32 bits un `int` ocupará 4 bytes y un `char` - 1 byte.

```
sizeof(char); // 1
sizeof(int); // 4
```

- La afirmación “El `sizeof()` de una `struct` de C es igual a la suma del `sizeof()` de cada uno de los elementos de la misma” es falsa, ya que en un `struct` también se tienen en cuenta los `padding` entre los elementos de `structs`. Se puede ver en este ejemplo:

```
struct S {
    char c; /* 1 byte */
           /* 1 byte de padding */
    short s; /* 2 bytes */
};
```



```
→ first git:(master) X ls
hello.c
→ first git:(master) X clear
→ first git:(master) X ls
hello.c
→ first git:(master) X gcc hello.c -o hello
→ first git:(master) X ./hello
Hello world!
```

Figure 1: Compilación y ejecución normal

```
→ first git:(master) X valgrind ./hello
==2520== Memcheck, a memory error detector
==2520== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==2520== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==2520== Command: ./hello
==2520==
--2520-- run: /usr/bin/dsymutil "./hello"
warning: no debug symbols in executable (-arch x86_64)
Hello world==2520==
==2520== HEAP SUMMARY:
==2520==    in use at exit: 26,252 bytes in 188 blocks
==2520==   total heap usage: 272 allocs, 84 frees, 32,492 bytes allocated
==2520==
==2520== LEAK SUMMARY:
==2520==    definitely lost: 0 bytes in 0 blocks
==2520==   indirectly lost: 0 bytes in 0 blocks
==2520==    possibly lost: 2,064 bytes in 1 blocks
==2520==   still reachable: 0 bytes in 0 blocks
==2520==    suppressed: 24,188 bytes in 187 blocks
==2520== Rerun with --leak-check=full to see details of leaked memory
==2520==
==2520== For counts of detected and suppressed errors, rerun with: -v
==2520== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 2: Compilación y ejecución con valgrind

Paso 2: SERCOM - Error de compilación

Los errores resultantes fueron:

```
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
```

Son errores del compilador. Se deben a que el compilador al pasar por las líneas de error encontró funciones que todavía no están declaradas: `ztrcpy` y `malloc` en este caso.

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2016-03-09 23:04:55	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 2.		Bajar Todo _stdouterr

Figure 3: Subida fallida

```

CC p2.o
p2.c: In function 'main':
p2.c:10: error: implicit declaration of function 'strcpy'
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

```

Figure 4: Resultado de compilación

Paso 3: SERCOM - Normas de programación y código de salida

Comandos Ejecutados

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2016-03-09 23:39:43	0:00:00	Sí			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)'	2016-03-09 23:39:43	0:00:00	No	Se esperaba terminar con un código de retorno 0 pero se obtuvo 1.		Bajar Todo stdouterr...

Figure 5: Generación del ejecutable exitosa

Una de las pruebas realizadas por el SERCOM es la verificación del estilo del código. En la captura de pantalla se ve que la mayoría de los errores se debe a que faltan o sobran espacios en blanco. El único error que no tiene que ver con espacios alienta a usar la función *snprintf* en vez de *strcpy*.

```

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

```

Figure 6: Problemas de estilo de código

La primer prueba falló por un error en el código: al no existir un archivo la prueba espera el código 1, pero el programa en este caso devuelve 2.

1- Archivo inexistente.						
Comando		/tp no-existo				
Inicio / Fin		2016-03-09 23:39:43 / 2016-03-09 23:39:44				
#	Tarea	Comando	Duración	Exito?	Observaciones	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

Figure 7: Error reportado en la prueba 1

```
./tp.c:16: Almost always, snprintf is better than strcpy [runtime/printf] [4]
Done processing ./tp.c
Total errors found: 1
```

Figure 8: Problemas de estilo de código después de las correcciones

1- Archivo inexistente.						
Comando		/tp no-existo				
Inicio / Fin		2016-03-09 23:56:21 / 2016-03-09 23:56:22				
#	Tarea	Comando	Duración	Exito?	Observaciones	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

Figure 9: Finalización de la prueba 1

```
==00:00:00:00.000 914== Memcheck, a memory error detector
==00:00:00:00.000 914== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==00:00:00:00.000 914== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==00:00:00:00.000 914== Command: ./tp archivo-corto.txt
==00:00:00:00.000 914== Parent PID: 913
==00:00:00:00.000 914==
==00:00:00:00.355 914==
==00:00:00:00.355 914== FILE DESCRIPTORS: 3 open at exit.
==00:00:00:00.355 914== Open file descriptor 2: archivo-corto.txt
==00:00:00:00.355 914==   at 0x48E1B1E: __open_nocancel (syscall-template.S:82)
==00:00:00:00.355 914==   by 0x488BBC7: _IO_file_fopen@@GLIBC_2.1 (fileops.c:336)
==00:00:00:00.355 914==   by 0x487FDEC: __fopen_internal (iofopen.c:93)
==00:00:00:00.355 914==   by 0x487FE4B: fopen@@GLIBC_2.1 (iofopen.c:107)
==00:00:00:00.355 914==   by 0x8048603: main (tp.c:17)
==00:00:00:00.355 914== Open file descriptor 1: /var/lib/sercom/sercom/var/tmp/prueba.227830.stdout
==00:00:00:00.355 914==   <inherited from parent>
==00:00:00:00.355 914== Open file descriptor 0: /home/sercom_backend/test/valgrind.out
==00:00:00:00.355 914==   <inherited from parent>
==00:00:00:00.355 914==
==00:00:00:00.355 914== HEAP SUMMARY:
==00:00:00:00.355 914==   in use at exit: 356 bytes in 2 blocks
==00:00:00:00.355 914==   total heap usage: 2 allocs, 0 frees, 356 bytes allocated
==00:00:00:00.355 914==
==00:00:00:00.356 914== 4 bytes in 1 blocks are definitely lost in loss record 1 of 2
==00:00:00:00.356 914==   at 0x47EBF20: malloc (vg_replace_malloc.c:236)
==00:00:00:00.356 914==   by 0x804861A: main (tp.c:20)
==00:00:00:00.356 914==
==00:00:00:00.356 914== LEAK SUMMARY:
==00:00:00:00.356 914==   definitely lost: 4 bytes in 1 blocks
==00:00:00:00.356 914==   indirectly lost: 0 bytes in 0 blocks
==00:00:00:00.356 914==   possibly lost: 0 bytes in 0 blocks
==00:00:00:00.356 914==   still reachable: 352 bytes in 1 blocks
==00:00:00:00.356 914==   suppressed: 0 bytes in 0 blocks
==00:00:00:00.356 914== Reachable blocks (those to which a pointer was found) are not shown.
==00:00:00:00.356 914== To see them, rerun with: --leak-check=full --show-reachable=yes
==00:00:00:00.356 914==
==00:00:00:00.356 914== For counts of detected and suppressed errors, rerun with: -v
==00:00:00:00.356 914== 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.
```

Figure 10: Errores de Valgrind

Paso 4: SERCOM - Pérdida de memoria

Lo que se puede ver en los errores de valgrind resultantes (ver la figura 10) es que hay pérdida de memoria. Valgrind nos avisa cordialmente que hay 2 *allocs* y 0 *frees*, y que seguro perdemos 4 bytes de memoria en la línea 20 de la función `main` (donde justamente hay un `alloc` de tamaño de un `int`).

Paso 5: SERCOM - Escrituras fuera de rango

2- Archivo existente.							
Comando		/tp archivo-corto.txt					
Inicio / Fin		2016-03-10 00:11:06 / 2016-03-10 00:11:07					
#	Tarea	Comando		Duración	Exito?	Observaciones	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

Figure 11: Resultado de la prueba 2

El problema de la prueba 3 se debe a que el nombre del archivo pasado al programa supera el buffer definido de 20 caracteres, pero la función `strcpy` no respeta ese límite y sigue escribiendo caracteres lo cual provoca resultados inesperados. Se puede solucionar con la función `strncpy`, ya que sirve para el mismo proposito (copia de strings) pero además de los argumentos de `strcpy` recibe un la cantidad de bytes a copiar.

Valgrind (ver la figura 12) nos da una pista para diagnosticar ese problema: se ve en su mensaje de salida que detectó un buffer overflow y apunta a que ocurre en `string3.h`, de donde se puede sospechar que es un problema que tiene que ver con como estamos manejando strings.

Segmentation fault es un error originado por intentos de acceso a segmentos inaccesibles. Por ejemplo, ocurre al tratar de leer un segmento asignado a otro programa, o tratar de escribir en el segmento de código.

Buffer overflow es un error que ocurre al escribir datos fuera del rango de un buffer. Cabe aclarar que no es un error por si solo, sino que provoca que el programa pueda dar resultados inesperados.

Contenido del archivo corto:

Entrada de la prueba 2

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

```

==00:00:00:00.000 1024== Memcheck, a memory error detector
==00:00:00:00.000 1024== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==00:00:00:00.000 1024== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright
info
==00:00:00:00.000 1024== Command: ./tp soy-un-archivo-con-nombre-largo.txt
==00:00:00:00.000 1024== Parent PID: 1023
==00:00:00:00.000 1024==
**00:00:00:00.330 1024** *** strcpy_chk: buffer overflow detected ***: program terminated
==00:00:00:00.330 1024== at 0x47EE927: VALGRIND_PRINTF_BACKTRACE (valgrind.h:4214)
==00:00:00:00.330 1024== by 0x47EEAFF: __strcpy_chk (mc_replace_strmem.c:757)
==00:00:00:00.330 1024== by 0x8048664: main (string3.h:107)
==00:00:00:00.355 1024==
==00:00:00:00.355 1024== FILE DESCRIPTORS: 2 open at exit.
==00:00:00:00.355 1024== Open file descriptor 1: /var/lib/sercom/sercom/var/tmp/prueba.227851.stdout
==00:00:00:00.355 1024== <inherited from parent>
==00:00:00:00.355 1024==
==00:00:00:00.355 1024== Open file descriptor 0: /home/sercom_backend/test/valgrind.out
==00:00:00:00.355 1024== <inherited from parent>
==00:00:00:00.355 1024==
==00:00:00:00.355 1024== HEAP SUMMARY:
==00:00:00:00.355 1024== in use at exit: 0 bytes in 0 blocks
==00:00:00:00.355 1024== total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==00:00:00:00.355 1024==
==00:00:00:00.355 1024== All heap blocks were freed -- no leaks are possible
==00:00:00:00.355 1024==
==00:00:00:00.355 1024== For counts of detected and suppressed errors, rerun with: -v
==00:00:00:00.355 1024== 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 soy-un-archivo-con-nombre-largo.txt
[SERCOM] Error code configured for Valgrind: 42.
[SERCOM] Valgrind execution result: 127.
[SERCOM] Valgrind result: Success.

```

Figure 12: Resultado de Valgrind de la prueba 3

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.

Contenido del archivo largo

Entrada de la prueba 4

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, despues de escribir una carta al Presidente De la Rua criticando al sistema de salud, se quito la vida de un disparo al corazon.

Comando para la ejecución de la prueba 3

```
./tp soy-un-archivo-con-nombre-largo.txt
```

Paso 6: SERCOM - Entrada estándar

#	Tarea	Comando	Inicio	Duración	Exito?	Observaciones	Diferencias	Archivos Guardados
1	Compilar C99 simple	make -f Makefile	2016-03-10 00:41:51	0:00:00	Sí			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)'\`	2016-03-10 00:41:51	0:00:00	Sí			Bajar Todo stdouterr

Figure 13: Checkeo de normas de codificación exitoso

5- Entrada estandar.							
Comando		./tp					
Inicio / Fin		2016-03-10 00:41:54 / 2016-03-10 00:41:55					
#	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	0:00:01	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

Figure 14: Resultado de la prueba 5

Paso 7: SERCOM - Entrega exitosa

Ejercicio	Resultado	Fecha	Duración	Observaciones	Operaciones
0.1 (El Ambiente de Trabajo)	Aceptado	2016-03-10 00:47:03	0:00:04		Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2016-03-10 00:41:47	0:00:05		Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2016-03-10 00:10:56	0:00:04		Corrida Bajar Navegar PDF
0.1 (El Ambiente de Trabajo)	Rechazado	2016-03-09 23:56:17	0:00:05		Corrida Bajar Navegar PDF

Figure 15: Entrega final exitosa

```

→ Entrega git:(master) X ls
archivo-corto.txt tp.c tp.zip
→ Entrega git:(master) X gcc tp.c -o tp
→ Entrega git:(master) X ./tp < archivo-corto.txt

```

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.

Figure 16: Ejecución redireccionando la entrada

```

→ Entrega git:(master) X ./tp archivo-corto.txt > salida.txt
→ Entrega git:(master) X cat salida.txt

```

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.

Figure 17: Ejecución redireccionando la salida