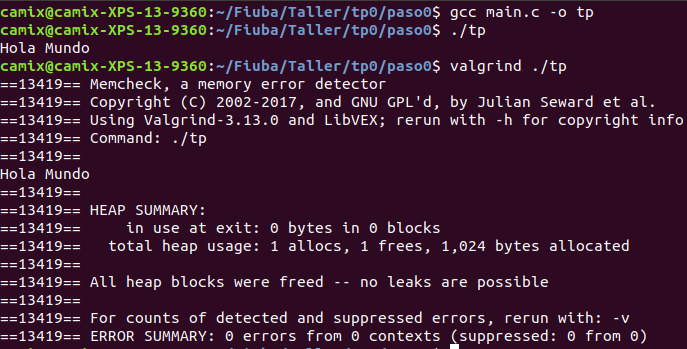
**Paso 0**

**a) Capturas de pantalla de la ejecución del aplicativo (con y sin Valgrind).** 

**b) ¿Para qué sirve ​ Valgrind​ ? ¿Cuáles son sus opciones más comunes?**

Valgrind es una herramienta que sirve para *debuggear* el manejo de la memoria.

Memcheck, la herramienta predeterminada de Valgrind, se encarga de detectar errores en la memoria tales como el acceso a memoria que no está reservada, formas incorrectas de liberar la memoria (doble free, que no haya ninguno, etc), el uso de valores que no fueron inicializados, y demás.

Massif sirve para saber cuánta memoria de heap usa un programa. También (aunque no es su configuración default) se le puede pedir información sobre la memoria que se ocupa en stack.

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

La función sizeof() recibe el nombre de un tipo de dato y devuelve el tamaño que este ocupa en bytes.

La salida de sizeof(char) y sizeof(int) siempre será 1 y 4 respectivamente. La memoria que se necesita para almacenar un char es 1 byte, mientras que la memoria que se requiere para almacenar un int es 4 bytes.

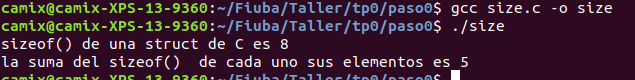
**d- ¿El ​ sizeof() de una struct de C es igual a la suma del sizeof() ​ de cada uno sus elementos?**

No necesariamente. Si bien podría llegar a pasar es muy fácil encontrar algún caso donde, a causa del padding, no se cumpla esta relación.

A continuación presento un ejemplo donde se define una struct en C, se imprime su sizeof() y la suma de los sizeof() de cada uno de sus elementos.

|  |
| --- |
| #include <stdio.h>  struct alumno {  char c;  int i;  }typedef alumno\_t;  int main(void) {  int suma = sizeof(char) + sizeof(int);  int real = sizeof(alumno\_t);  printf("sizeof()​ de una struct de C es %d \n", real);  printf("la suma del sizeof() ​ de cada uno sus elementos es %d \n", suma);  return 0;  } |

Luego de compilar y ejecutar el código de arriba la salida fue la siguiente:



Con lo cual se concluye que no siempre la afirmación “el ​ sizeof() de una struct de C es igual a la suma del sizeof() ​ de cada uno sus elementos” es falsa.

**e) Investigar la existencia de los archivos estándar: STDIN, STDOUT, STDERR. Explicar**

**brevemente su uso y cómo redirigirlos en caso de ser necesario (caracteres > ​ y ​ < ​ ) y como conectar la salida estándar de un proceso a la entrada estándar de otro con un pipe​ (carácter |​ ).**

El STDIN (entrada estándar) es un archivo que se usa para recibir, durante la ejecución de un proceso, datos que van apareciendo con el tiempo y no están estáticos en un archivo. Análogo a esto es el uso del STDOUT (salida estándar) que en lugar de recibir un flujo de datos los emite. Por último, todo proceso tiene un tercer archivo, STDERR, que es destinado a emitir los errores en tiempo de ejecución.

"*STDOUT, generalmente, tiene un búfer asociado para hacerlo más rápido, pero si el programa termina por un error y el búfer no se vacía, entonces se pierde la información. En cambio, STDERR generalmente no tiene un búfer asociado, entonces tenemos la seguridad de que los mensajes se envían siempre y sin retrasos*."

Apuntes de la cátedra 75.41, segundo cuatrimestre 2018, curso Buchwald, FIUBA.

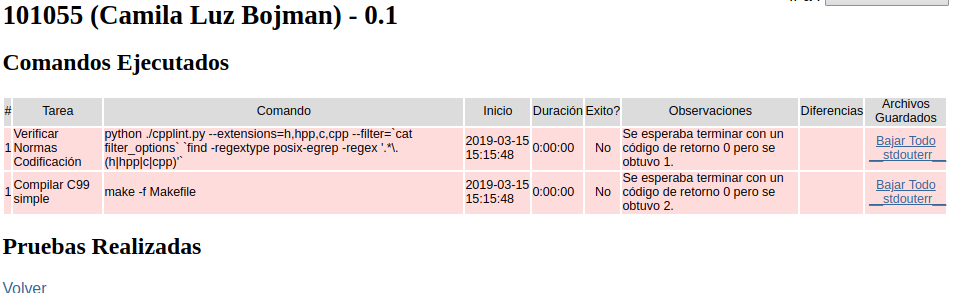
Finalmente a los archivos STDIN, STDOUT Y STDERR se los suele asociar respectivamente a los números 0, 1 y 2.

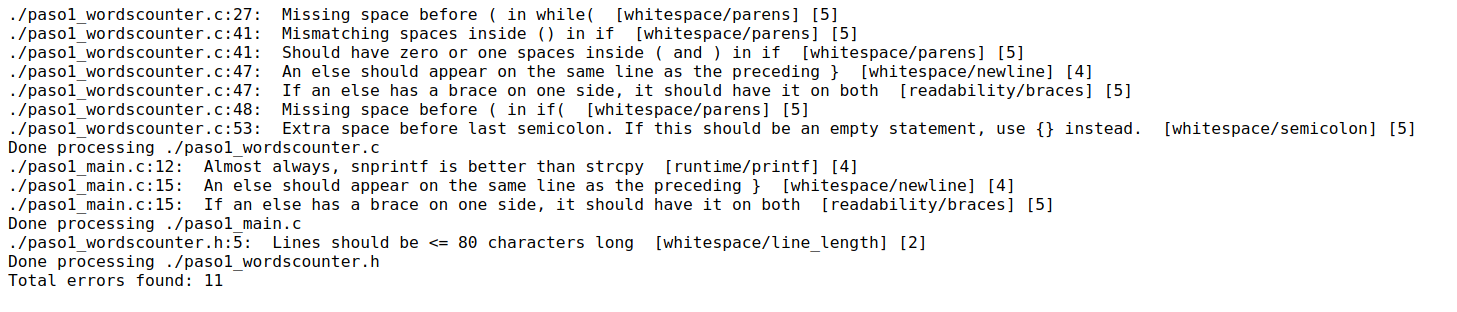
Al escribir ./mi\_programa > archivo en la consola se está redirigiendo el STDOUT del programa. Es decir, la instrucción que se está dando sería “ejecuta el programa ./mi\_programa usando como salida estándar el archivo archivo“. Se obtiene exactamente el mismo resultado al ejecutar el comando ./mi\_programa 1 > archivo ya que en el caso anterior el 1 estaba implícito. En cambio, si la sintaxis es ./mi\_programa 2 > archivo entonces se redirecciona el archivo 2, el STDERR, que pasa a ser el archivo archivo. Para redireccionar el archivo de entrada lo más común es ejecutar ./mi\_programa < archivo. Existen múltiples formas de combinar estas sintaxis y redirigir más de un archivo a la vez.

Por otro lado, se puede también redirigir la entrada estándar de un programa hacia la salida de otro. Es decir, el mismo archivo que un programa usa para escribir un stream de datos un segundo programa lo utiliza para ir leyéndolo. La forma de hacerlo por consola es “pipeando” por ejemplo echo “hola cómo estas” | ./mi-programa el programa echo va a escribir en un archivo de salida estandar “hola mundo” y ese mismo archivo va a ser el de lectura estándar del programa mi\_programa.

**Paso 1**

Se hace entrega del archivo “paso1.zip” al sercom.



**a) Captura de pantalla mostrando los problemas de estilo detectados. Explicar cada uno.**

Missing space before ( in while( [whitespace/parens]

Debería haber un espacio en blanco entre la declaración del while y la el primer paréntesis que encierra la expresión lógica.

Mismatching spaces inside () in if [whitespace/parens]

La cantidad de espacios en blanco existentes entre la apertura de paréntesis y la primer letra debería ser igual a la cantidad de espacios en blanco existentes entre la última letra dentro del paréntesis y el paréntesis de cierre. Estéticamente esto significa que la expresión lógica debe estar centrada con respecto a los paréntesis que la encierran.

Should have zero or one spaces inside ( and ) in if [whitespace/parens]

La cantidades de espacios blancos mencionados en el error anterior deben ser 0 o 1 únicamente.

An else should appear on the same line as the preceding } [whitespace/newline]

Los else deben posicionarse en la misma línea que la llave que cierra el if al que hacen referencia.

If an else has a brace on one side, it should have it on both [readability/braces]

Si un else está precedido por una llave, entonces debería existir otra que la sucede. Lo mismo se da en caso contrario: Si un else está sucedido por una llave, entonces debería existir un otra que la antecede.

Missing space before ( in if( [whitespace/parens]

Este error es análogo al primero con la salvedad de que se trata de un if en lugar de un while.

Extra space before last semicolon. If this should be an empty statement, use {} instead. [whitespace/semicolon]

Hay un espacio en blanco extra entre el el último carácter de la sentencia (final real de la sentencia) y el punto y coma que delimita el fin de la sentencia.

Almost always, snprintf is better than strcpy [runtime/printf]

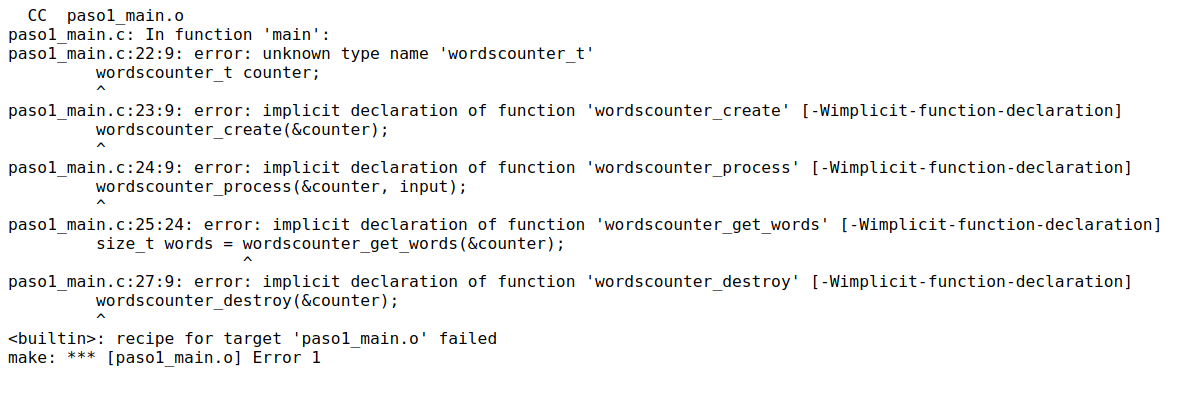
El error nos está indicando que casi siempre es mejor utilizar la función snprintf que strcopy.

A veces nos encontramos con varias funciones de la librería estándar que asimilan el comportamiento que buscamos. En estas ocasiones es una buena práctica es optar por la función que menos posibilidad de emitir en un error tenga.

En este caso en particular snprintf maneja un parámetro para especificar la cantidad de bytes (caracteres) se desean copiar. Por lo tanto es factible que tenga menos probabilidad de caer en un error de buffer overflow.

Lines should be <= 80 characters long [whitespace/line\_length]

El estándar dice que todas las líneas deben ser menores a 80 caracteres. Esto surge de motivos “antiguos” y tal vez obsoletos. No obstante facilita la lectura de cualquier código.

**b) Captura de pantalla indicando los errores de generación del ejecutable. Explicar cada uno e indicar si se trata de errores del compilador o del linker.**

unknown type name 'wordscounter\_t'

En la función main se declara una variable de tipo wordscounter\_t. El error se genera porque la estructura wordscounter\_t no está definida, por lo tanto el compilador no la conoce y no sabe cuánta memoria debe reservar. Este es un error de compilación ya que es el compilador carece de la información suficiente para hacer su tarea.

implicit declaration of function 'wordscounter\_create'

implicit declaration of function 'wordscounter\_process'

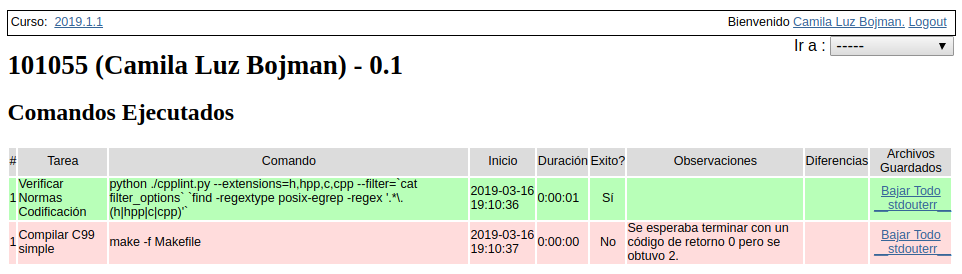
implicit declaration of function 'wordscounter\_get\_words'

En este caso los errores nos comunican que se llamó a las respectivas funciones pero nunca fueron declaradas previamente. Es decir, el compilador no sabe qué tipo de datos ni cuantos reciben cada una y que tipo de datos devuelven.

**c) ¿El sistema reportó algún WARNING? ¿Por qué?**

No. El error implicit declaration of function citado en el punto anterior en realidad se trata de un warning pero el compilador lo trata como un error porque se le pasó el flag -Werror.

**Paso 2**

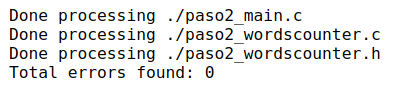
Se hace entrega del archiv “paso2.zip” a sercom

**a. Describa ​ en breves palabras​ las correcciones realizadas respecto de la versión anterior.**

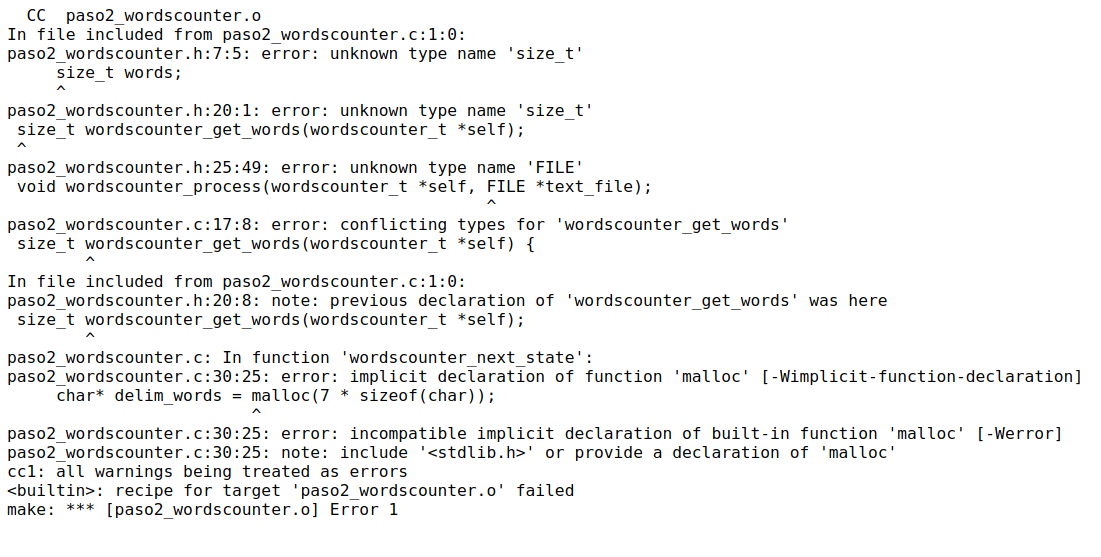
Las correcciones que se hicieron fueron mayoritariamente de estilo, especificamente las mencionadas en los errores de la entrega del paso1.zip. No obstante, se incluyó en el main.c el archivo wordscounter.h que contiene la declaración de la estructura y las funciones que el compilador “reclamaba” en el paso anterior.

**b. Captura de pantalla indicando la correcta ejecución de verificación de normas de**

**programación.**

****

**c. Captura de pantalla indicando los errores de generación del ejecutable. Explicar cada uno e indicar si se trata de errores del compilador o del l​inker.**

unknown type name 'size\_t'

unknown type name 'FILE'

Como se explicó en el **paso 2** respuesta **c)** este es un error de compilación. El código hace referencia a un tipo de datos que no está declarado.

paso2\_wordscounter.c:17:8: error: conflicting types for 'wordscounter\_get\_words'

size\_t wordscounter\_get\_words(wordscounter\_t \*self) {

^

In file included from paso2\_wordscounter.c:1:0:

paso2\_wordscounter.h:20:8: note: x

size\_t wordscounter\_get\_words(wordscounter\_t \*self);

^

Este error se deriva del anterior. Al no conocer el tipo de datos size\_t es probable que el compilador haya generado una declaración default de la función que devuelva un entero. Pero, al llegar a la línea donde la función está definida se encuentra con una declaración no concuerda con la anterior.

paso2\_wordscounter.c: In function 'wordscounter\_next\_state':

paso2\_wordscounter.c:30:25: error: implicit declaration of function 'malloc' [-Wimplicit-function-declaration]

char\* delim\_words = malloc(7 \* sizeof(char));

^

paso2\_wordscounter.c:30:25: error: incompatible implicit declaration of built-in function 'malloc' [-Werror]

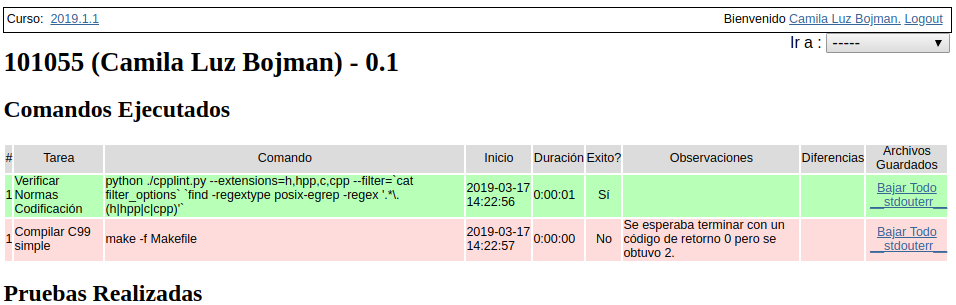
paso2\_wordscounter.c:30:25: note: include '<stdlib.h>' or provide a declaration of 'malloc'

Por último, este error se conecta levemente a los anteriores. La función malloc se encuentra definida en la misma librería estándar que la estructura size\_t. Esta librería no está incluida y el compilador no conoce la declaración de la función.

En todos los casos estamos tratando con errores de compilación donde al compilador carece de información.

**Paso 3**

Se hace entrega del archiv “paso3.zip” a sercom



**a. Describa en breves palabras las correcciones realizadas respecto de la versión anterior.**

El cambio principal fue la inclusión de librerías estándar. En el archivo paso3\_wordscounter.c se incluyó la librería mencionada en los errores del paso anterior (paso3\_wordscounter.c). Y en el código de paso3\_wordscounter.h se incluyeron las librerías string.h y stdio.h.

Además, se hicieron los cambios pertinentes en el código a causa de la alteración de los nombres de los archivos.

**b. Captura de pantalla indicando los errores de generación del ejecutable. Explicar cada uno e indicar si se trata de errores del compilador o del linker.**

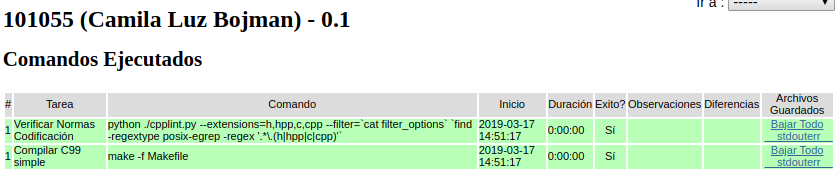
paso3\_main.o: In function `main':

/home/sercom\_backend/build/paso3\_main.c:27: undefined reference to `wordscounter\_destroy'

El error citado nos informa que la función wordscounter\_destroy no está definida. Este es un error del linker, el compilador conoce la declaración de la función y puede compilar pero el linker desconoce cómo proceder cuando se llama a esta función.

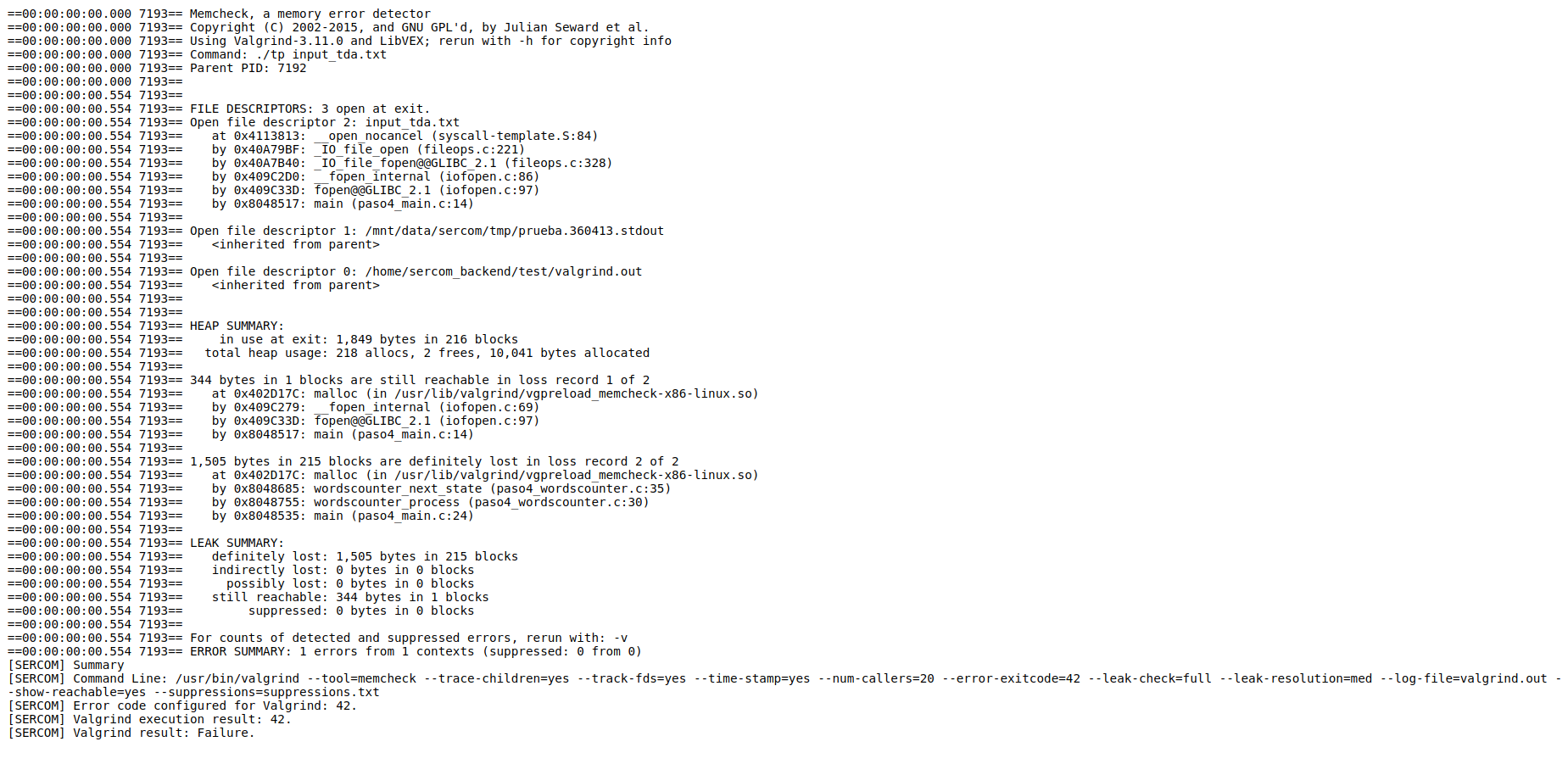
**Paso 4**

Se hace entrega del archiv “paso4.zip” a sercom

**a. Describa en breves palabras las correcciones realizadas respecto de la versión anterior.**

Por un lado, al igual que en el diff del paso 2 con el 3, se hicieron los reformas necesarias para incluir los archivos con el nombres cambiado.

Además, se agregó una definición de la función que impedía el linkeo del programa. Si bien la función carece de sentencias y no tiene ninguna utilidad permite al linker referenciarla y así solucionar el error que se generaba en el paso 3.

**b. Captura de pantalla del resultado de ejecución con Valgrind​ de la prueba ‘TDA’. Describir los errores reportados por Valgrind.**

==00:00:00:00.554 7193== Open file descriptor 2: input\_tda.txt

==00:00:00:00.554 7193== 344 bytes in 1 blocks are still reachable in loss record 1 of 2

==00:00:00:00.554 7193== at 0x402D17C: malloc (in /usr/lib/valgrind/vgpreload\_memcheck-x86-linux.so)

==00:00:00:00.554 7193== by 0x409C279: \_\_fopen\_internal (iofopen.c:69)

==00:00:00:00.554 7193== by 0x409C33D: fopen@@GLIBC\_2.1 (iofopen.c:97)

==00:00:00:00.554 7193== by 0x8048517: main (paso4\_main.c:14)

Estas líneas de error nos muestran que el main en la línea 14 hace un llamado a fopen pero nunca cierra el archivo. Nos explica que el archivo input\_tda.txt aún sigue abierto al terminar la ejecución. Y que esta “leak memory” es “still reachable” lo cual significa que no se perdió el puntero a la misma y aún puede liberarse el espacio en memoria allocado.

==00:00:00:00.554 7193== 1,505 bytes in 215 blocks are definitely lost in loss record 2 of 2

==00:00:00:00.554 7193== at 0x402D17C: malloc (in /usr/lib/valgrind/vgpreload\_memcheck-x86-linux.so)

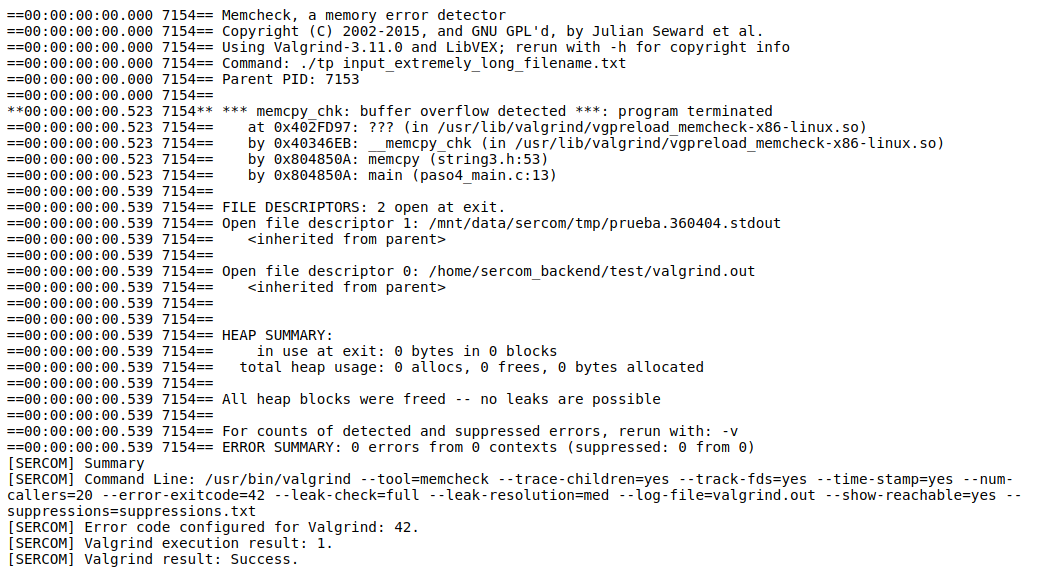
==00:00:00:00.554 7193== by 0x8048685: wordscounter\_next\_state (paso4\_wordscounter.c:35)

==00:00:00:00.554 7193== by 0x8048755: wordscounter\_process (paso4\_wordscounter.c:30)

==00:00:00:00.554 7193== by 0x8048535: main (paso4\_main.c:24)

Podemos notar por la descripción del error que se deriva del de un llamado del main a wordscounter\_proces y de este último a wordscounter\_next\_state. Más especificamente en la liena 35 del archivo paso4\_wordscounter.c. Si vamos a la línea previamente mencionada notamos que se pide memoria para un sting. Esta porción de memoria no es liberada al terminar la ejecución.

La especificación definitely lost nos indica que el programa además perdió el puntero a esa porción de memoria.

**c. Captura de pantalla del resultado de ejecución con Valgrind​ de la prueba ‘Long Filename’. Describir los errores reportados por Valgrind.**

Las salida de valgrind nos reporta un único error, el acceso de memoria que no está reservada para el buffer que se trata de escribir.

Si vamos a la línea que genera el overflow podemos notar que se usa la función memcpy para copiar al buffer filepath el contenido del buffer argv[1]. El problema es que se le indica que copie todos los caracteres del segundo buffer sin tener en cuenta que el primero no cuenta con tanto espacio.

**d. ¿Podría solucionarse este error utilizando la función strncpy​ ? ¿Qué hubiera ocurrido con la ejecución de la prueba?**

strncpy recibe un parametro entero n un buffer de entrada y uno de salida. La función copia hasta n caracteres del buffer de entrada al de salida. Se puede pensar que podría solucionar el error que se nombra en el punto **c)** ya que, en el caso de que n sea mayor a la longitud del buffer de salida, transcribe tantos bytes como entran en el este buffer. Así como utiliza todo el buffer de salida “al máximo” no guarda un byte para para guardar un carácter nulo. Por esto último a la hora de operar con el string de salida se accedería a memoria inválida y se generaría un segmentation fault.

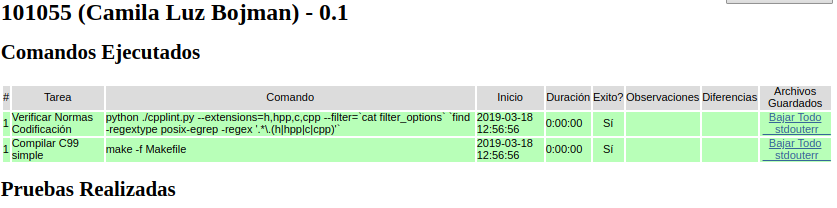
**e. Explicar de qué se trata un ​ segmentation fault​ y un ​ buffer overflow​ .**

El segmentation fault se genera cuando un programa intenta ingresar a una porción de memoria a la que no tiene autorización. O intenta acceder a memoria de la manera en que no tiene permitido, por ejemplo tratando de escribir segmentos de "sólo lectura".

Los programas habitualmente reservan una porción de memoria (buffer) para ser escrita. El buffer overflow se da cuando un programa se excede de la capacidad del buffer al punto de sobrescribir datos en la memoria adyacente.

**Paso 5**

Se hace entrega del archivo “paso5.zip” al sercom.

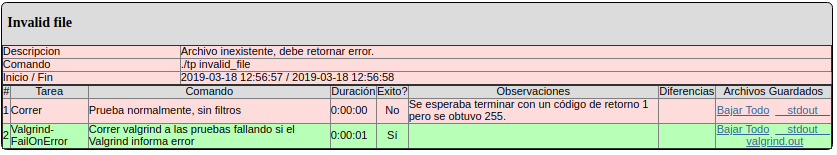


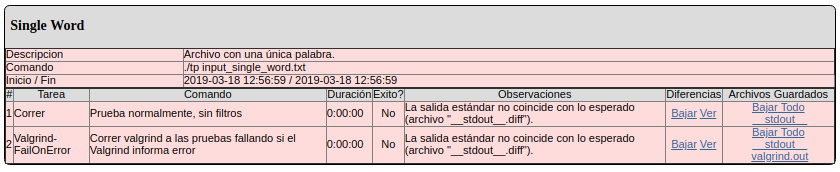
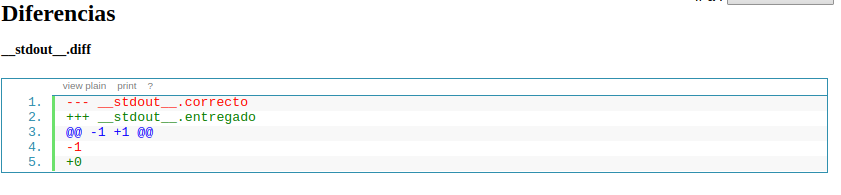
**a. Describa en breves palabras las correcciones realizadas respecto de la versión anterior.**

El programa ya no reserva memoria en el main para guardar los parámetros pasados por línea de comando. Tampoco sigue reservando memoria para la cadena de símbolos delimitadores de palabras.

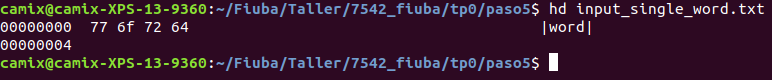
También se agrega un bloque en el main para cerrar archivo en caso de que se haya pasado el parámetro con el nombre del mismo.

**b. Describa el motivo por el que fallan las prueba ‘Invalid File’ y ‘Single Word’. ¿Qué información entrega SERCOM para identificar el error? Realice una captura de pantalla.**

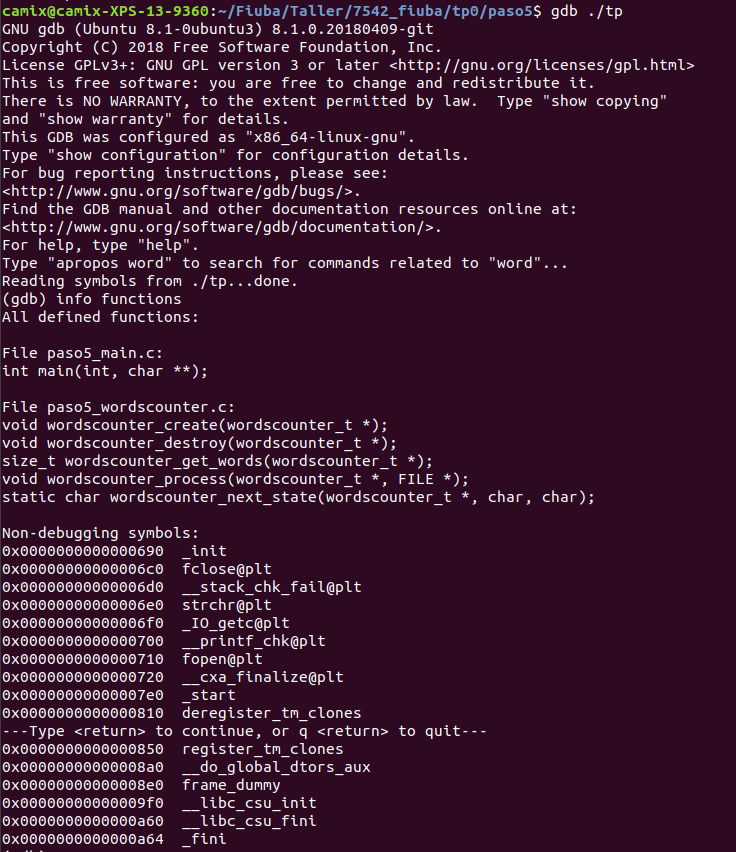
Sercom nos describe que la respuesta esperada por el programa era un 1. Es decir, que el archivo especificado no exista o no se poseen permisos de lectura. En este caso específico, conociendo la prueba, se le pasa al programa el nombre de un archivo inexistente.

El error que nos muestra Sercom al procesar el archivo input\_single\_word.txt es que se esperaba que imprima un 1 (porque el archivo contiene una única palabra) pero se imprimió un 0. 

**c. Captura de pantalla de la ejecución del comando hexdump​ . ¿Cuál es el último carácter del archivo input\_single\_word.txt?**

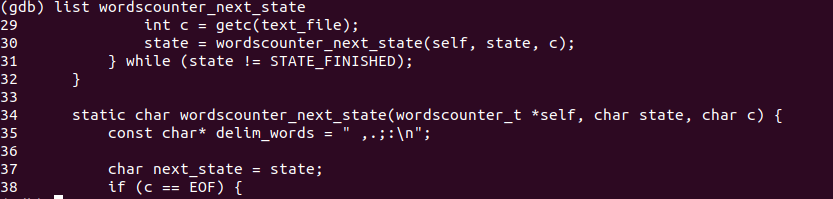
EL último caracter del archivo es una d.

**d. Captura de pantalla con el resultado de la ejecución con gdb​ . Explique brevemente los comandos utilizados en gdb​ . ¿Por qué motivo el debugger no se detuvo en el breakpoint de la línea 45: self->words++; ​ ?**



El comando gdb ./pro arranca a gdb. Es decir, comienza la ejecución del debugger, no del programa.

Luego se ejecutó info functions cuya función es enumerar todas las funciones y tipos de datos del programa.

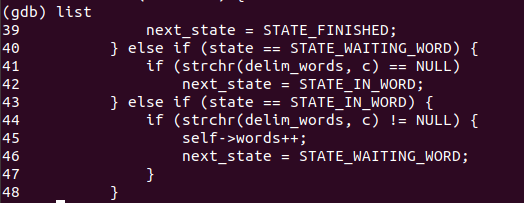


Luego se usó el comando list wordscounter\_next\_state.

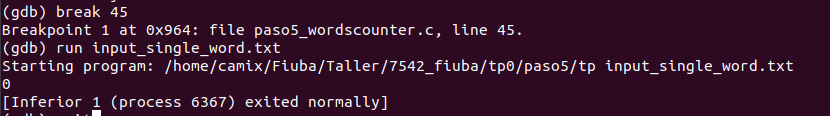
“*Print lines centered around the beginning of function function.*”

documentación de list function que figura en ftp://ftp.gnu.org/old-gnu/Manuals/gdb/html\_node/gdb\_46.html

Es decir, que el comando imprime una cierta cantidad de líneas previas a la definición de la función que se pase como parámetro y la misma cantidad de líneas desde el comienzo de esta función.



A continuación se ejecutó el comando list. En este caso específico, como las últimas líneas impresas fueron por medio del mismo comando, se imprimen las líneas que le siguen.

El comando break 45 ejecutado posteriormente setea un *punto de quiebre* en la línea 45 del *archivo actual*. Siendo el *archivo actual* el último que se imprimió. Y siendo un el  *punto de quiebre* una sentencia donde la ejecución se va a detener en caso de pasar por ahí.

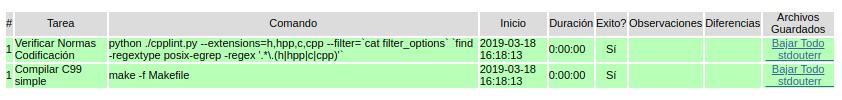
Luego se corrió el archivo con el comando run y con el nombre del archivo a procesar pasado por parametro input\_single\_word.txt.

Fue evidente que el programa no pasa nunca por la línea 45 porque nunca se detuvo en ella. Es decir, nunca se encontró un delimitador de palabras entonces nunca se sumó uno a la cantidad de palabras procesadas. Esto explica perfectamente el error que documenta sercom, se esperaba que el programa imprima 1 pero imprimió 0.

Finalmente, terminada la ejecución del programa y el debuggueo se cerro gdb con el comando quit.

**Paso 6**

Se hace entrega del archivo “paso6.zip” al sercom.

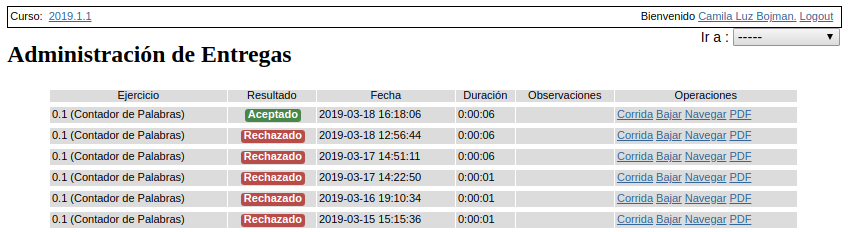
**a. Describa en breves palabras las correcciones realizadas respecto de la versión anterior.**

Se corrigió la salida en caso de error, deja de ser un -1 y pasa a ser un 1 como la consigna lo indica.

También se comenzó a contemplar en el contador de palabras tenga en cuenta a la última palabra del archivo si está no ternía un delimitador al final.

O sea que se corrigieron los errores que se generaba en las pruebas del **paso 5**

**b. Captura de pantalla mostrando t​ odas las entregas realizadas​ , tanto exitosas como fallidas.**



**c. Captura de pantalla mostrando la ejecución de la prueba ‘Single Word’ de forma local​ con las distintas variantes indicadas.**

