Taller 4 – Uso de gdb

Reporte

Bug 1 – Error al compilar con la bandera –Wall

```
estudiante@GGIECUT1WRK040:~/Documentos/taller4/src$ make
gcc -g -Wall bug.c -o bug
bug.c: In function ‘main’:
bug.c:40:2: warning: this ‘if’ clause does not guard... [-Wmisleading-indentation]
  if(tam > 1000)
  ^~
bug.c:42:3: note: ...this statement, but the latter is misleadingly indented as if it were guarded by the ‘if’
   return -1;
   ^~~~~~
estudiante@GGIECUT1WRK040:~/Documentos/taller4/src$
```

Nada más compilamos el archivo bug.c nos fijamos que hay un warning, el cual nos avisa de que un if no tiene sus respectivas llaves, por lo tanto, lo corregimos encerrando la sentencia dentro del if como corresponde y su return -1, también agregamos un return 0 al final del main para que el programa funcione correctamente al terminar.

```
        if(tam > 1000){
                printf("Arreglo demasiado grande\n");
                return -1;
        }
        int arreglo[tam];       //Solo para propositos del taller
        inicializar_arreglo(arreglo, 10000);
        mostrar_arreglo(arreglo,10000);

}
```

Una vez corregido al volver a compilar, podemos observar que ya no vuelve a salir el warning.

```
estudiante@GGIECUT1WRK040:~/Documentos/taller4/src$ make
gcc -g -Wall bug.c -o bug
estudiante@GGIECUT1WRK040:~/Documentos/taller4/src$
```

Bug 2 – SIGSEGV, Segmentation fault.

Este bug lo encontramos cuando utilizando el debugger, el programa se cae dándonos un SIGSEV, Segmentation fault, buscando en internet cuales pueden ser posibles causas y comparando con el código en cuestión determinamos que esto se debe a que en el main del archivo estamos creando un arreglo de tamaño <tam> y luego al utilizar la función que recibe como argumento el tamaño, enviamos un tamaño diferente estándar de 10000, el cual aparte de ser fijo no se adecúa con el propósito del programa como tal, es por esto que procedemos a cambiarlo por el tamaño que nosotros estaríamos mandando como argumento al utilizar el programa.

```
(gdb) b inicializar_arreglo
Punto de interrupción 2 at 0x555555554a0b: file bug.c, line 53.
(gdb) r -n 10
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/oscar/Documentos/PS20192S/taller4/taller4/bug -n 10

Breakpoint 2, inicializar_arreglo (arr=0x7fffffffded0, tam=10000) at bug.c:53
53              for(int i = tam; i >= 0; i++){
(gdb) n
54                      arr[i] = 2*i;
(gdb) n

Program received signal SIGSEGV, Segmentation fault.
0x0000555555554a2c in inicializar_arreglo (arr=0x7fffffffded0, tam=10000) at bug.c:54
54                      arr[i] = 2*i;
(gdb)
```

Este es el error que encontramos.

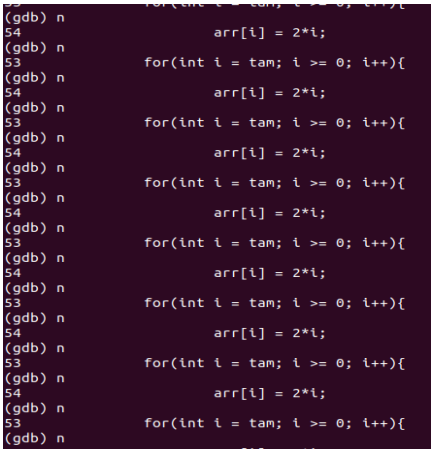
```
        if(tam > 1000){
                printf("Arreglo demasiado grande\n");
                return -1;
        }
        int arreglo[tam];       //Solo para propositos del talle
        inicializar_arreglo(arreglo, 10000);
        mostrar_arreglo(arreglo,10000);
        return 0;
}
```

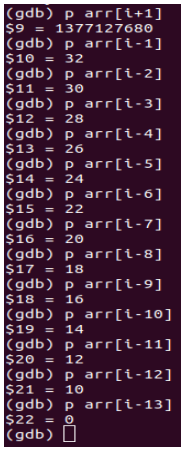
Así lo solucionamos, cambiando el argumento para tamaño.

```
        if(tam > 1000){
                printf("Arreglo demasiado grande\n");
                return -1;
        }
        int arreglo[tam];       //Solo para propositos del taller
        inicializar_arreglo(arreglo, tam);
        mostrar_arreglo(arreglo, tam);
        return 0;
}
```

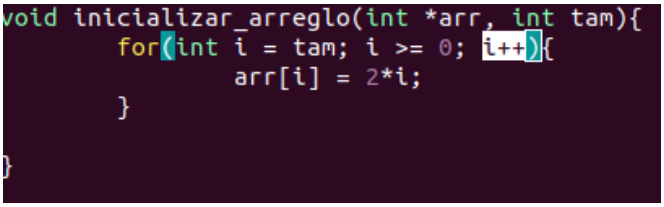
Bug 3 – El for infinito

Después de pasar un tiempo probando el comportamiento del programa en el debugger, nos damos cuenta que al entrar en el for de la función <inicializar_arreglo> nos quedamos en un loop infinito, es por esto que en las 2 primeras imágenes se ve que hacemos n (salto de línea) infinitos, y todo esto se debe a que en el for inicializamos nuestro iterador <i> en <tam> y ponemos como límite que i sea mayor o igual a 0 lo cual, al tener como regla que i se sume 1 cada vez que termine un ciclo, hace que este for no tenga fin.

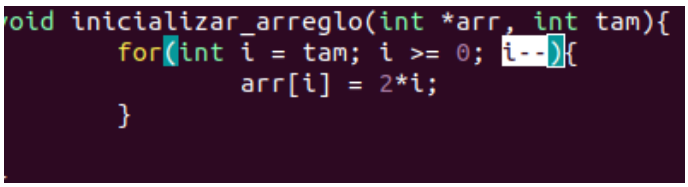




Este es el error que encontramos: la regla estaba en i++.

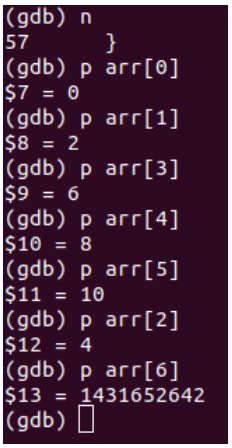


Lo cambiamos a i--.



Bug 4 – Un elemento de más

En la misma función notamos que al probar con argumento –n 5, nos estaba asignando 6 números en total ya que estaba contando también donde debería ir el carácter nulo, por esto es que podemos apreciar que se imprimen 6 elementos del arreglo (que debería solo tener 5, ya que tam=5) hasta antes de empezar a imprimir números random que es cuando ya no tiene nada que ver con nuestro arreglo, total solucionamos este error cambiando el valor inicial de nuestro iterador <i>.



Aquí es donde cambiamos el i=tam, por i=tam-1. De esta forma ya no iniciaríamos donde se supone que automáticamente va el carácter nulo, sino en el respectivo elemento.

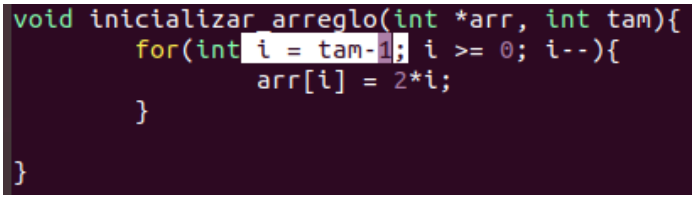
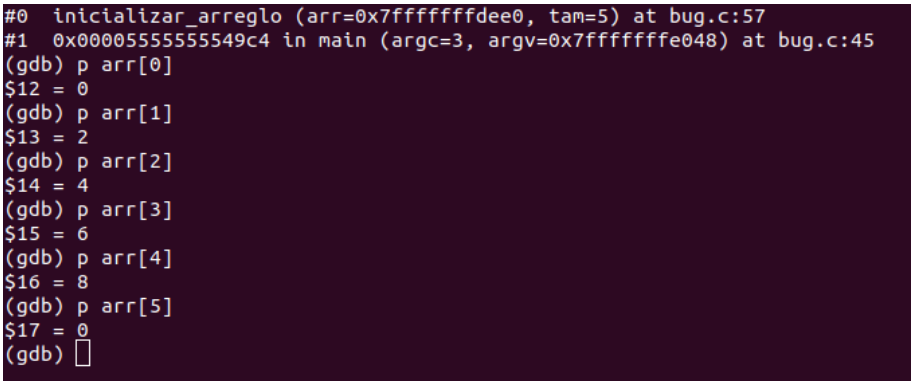


Imagen donde podemos apreciar ya la función completamente corregida, donde llamando al programa con argumento de tam=5, creamos un arreglo de solo 5 elementos en vez de 6 como antes, podemos apreciarlo al imprimir nuestros 5 elementos inicializados y el carácter nulo que se guarda en la posición final del arreglo automáticamente.



Bug 5 – El más macizo

Este bug lo encontramos en la función mostrar_arreglo, y ese título que le hemos puesto se debe a que en solo una función existen varios errores los cuales solucionamos todos de una sola vez.

```
Starting program: /home/oscar/Documentos/PS20192S/taller4/taller4/bug -n 5
Item 0 del arreglo es
Item 2 del arreglo es
Item 4 del arreglo es
Item 6 del arreglo es
Item 8 del arreglo es
[Inferior 1 (process 5841) exited normally]
(gdb) b mostrar_arreglo
Punto de interrupción 1 at 0x555555554863: file bug.c, line 21.
(gdb) r -n 5
Starting program: /home/oscar/Documentos/PS20192S/taller4/taller4/bug -n 5

Breakpoint 1, mostrar_arreglo (arr=0x7fffffffdee0, tam=5) at bug.c:21
aviso: El archivo fuente es más reciente que el ejecutable.
```

Al dirigirnos a la función en cuestión el primer error que encontramos es el orden de las variables que mandamos a imprimir como argumentos estando arr[i] donde debería ir i y viceversa.

```
void mostrar_arreglo(int *arr, int tam){
        for(int i = 0; i < tam; i++){
                printf("Item %d del arreglo es %c\n", arr[i], i);
        }
}
```

Ya corregido.

```
void mostrar_arreglo(int *arr, int tam){
        for(int i = 0; i < tam; i++){
                printf("Item %d del arreglo es %c\n", i, arr[i]);
        }
}
```

El otro error que encontramos era que daba formato de char %c al querer imprimir un int que debería ser %i, lo cual también cambiamos para que se pueda ver bien el printf en el programa.

```
void mostrar_arreglo(int *arr, int tam){
        for(int i = 0; i < tam; i++){
                printf("Item %d del arreglo es %c\n", i, arr[i]);
        }
}
```

Ya cambiado, podríamos haber usado también %d pero optamos por %i.

```
void mostrar_arreglo(int *arr, int tam){
        for(int i = 0; i < tam; i++){
                printf("Item %d del arreglo es %i\n", i, arr[i]);
        }
}
```

El último error que encontramos en la función que no era tanto un error sino más como una falla visual al momento de entender el programa era que se imprimía muy rústicamente como empezando desde elemento 0 ya que en la computadora se guardan desde el índice 0, pero los humanos entendemos que el primer elemento es el elemento 1, es por esto que decidimos solucionarlo cambiando el printf de i por i+1 sabiendo que empezábamos con el índice 0, que es el primer elemento.

```c
void mostrar_arreglo(int *arr, int tam){
        for(int i = 0; i < tam; i++){
                printf("Item %d del arreglo es %i\n", i+1, arr[i]);
        }
}
```

Imagen donde podemos apreciar mediante el debugger que el programa ya funciona sin errores o bugs (exited normaly) y con diferentes valores en el argumento tamaño.

```
(gdb) r -n 5
Starting program: /home/oscar/Documentos/PS20192S/taller4/taller4/bug -n 5
Item 1 del arreglo es 0
Item 2 del arreglo es 2
Item 3 del arreglo es 4
Item 4 del arreglo es 6
Item 5 del arreglo es 8
[Inferior 1 (process 5936) exited normally]
(gdb) r -n 10
Starting program: /home/oscar/Documentos/PS20192S/taller4/taller4/bug -n 10
Item 1 del arreglo es 0
Item 2 del arreglo es 2
Item 3 del arreglo es 4
Item 4 del arreglo es 6
Item 5 del arreglo es 8
Item 6 del arreglo es 10
Item 7 del arreglo es 12
Item 8 del arreglo es 14
Item 9 del arreglo es 16
Item 10 del arreglo es 18
[Inferior 1 (process 5940) exited normally]
```

También probamos con el argumento máximo 1000 en tamaño, e inmediatamente con el superior no permitido 1001, y comprobamos como está validado sin fallos.

```
Item 975 del arreglo es 1948
Item 976 del arreglo es 1950
Item 977 del arreglo es 1952
Item 978 del arreglo es 1954
Item 979 del arreglo es 1956
Item 980 del arreglo es 1958
Item 981 del arreglo es 1960
Item 982 del arreglo es 1962
Item 983 del arreglo es 1964
Item 984 del arreglo es 1966
Item 985 del arreglo es 1968
Item 986 del arreglo es 1970
Item 987 del arreglo es 1972
Item 988 del arreglo es 1974
Item 989 del arreglo es 1976
Item 990 del arreglo es 1978
Item 991 del arreglo es 1980
Item 992 del arreglo es 1982
Item 993 del arreglo es 1984
Item 994 del arreglo es 1986
Item 995 del arreglo es 1988
Item 996 del arreglo es 1990
Item 997 del arreglo es 1992
Item 998 del arreglo es 1994
Item 999 del arreglo es 1996
Item 1000 del arreglo es 1998
[Inferior 1 (process 5947) exited normally]
(gdb) r -n 1001
Starting program: /home/oscar/Documentos/PS20192S/taller4/taller4/bug -n 1001
Arreglo demasiado grande
```

¡Hurra! Hemos depurado nuestro programa.