# GDB

**Jeroen Engelberts - jeroene@sara.nl**
**Consultant Supercomputing**

## Layout Presentation

- What is GDB?
- Preparations at compilation
- Running GDB
- Analyzing Your Program
- Two Examples
- Questions & Answers

## What is GDB?

- GNU Debugger
- A debugger for languages like FORTRAN, C and C++ (ao)
- Allows you to inspect what your program is doing at a certain point in time during execution
- Errors like dreaded **segmentation faults** can be traced with gdb
- Further documentation: http://sourceware.org/gdb/current/onlinedocs/gdb_toc.html

## Compilation for use with GDB

- Normally, a  program would be compiled with:
  - **gcc [flags] <source files> -o <binary>**
- For example:
  - **gcc -O3 -Wall -Werror hello.c -o hello.x**
- Debugging information, like line numbers, are included with “**-g**”:
  - **gcc -g -O3 -Wall -Werror hello.c -o hello.x**
- Often higher optimization levels are specified, like “**-O3**”. These will make line number actions, like setting break points, less accurate.
- Therefore, use “**-O0**” if time permits:
  - **gcc -g -O0 -Wall -Werror hello.c -o hello.x**

# Running GDB

- You can start GDB on the command line with "**gdb**":

  **(gdb)**

  – Then you can load your binary with:

  **(gdb) file hello.x**
- It is also possible to load your program directly:

  **gdb hello.x**

- GDB has an extensive help function, accessible with:

  **(gdb) help [command]**

  in which [command] should be replaced by the desired command

# Analyzing your program (1)

- So, you end up with “segmentation fault”. What now?
  - You can set a break point on a certain line number:
    **(gdb) break my_proc.c:12**
  - You can set a break point on a function:
    **(gdb) break my_func**
  - After reaching a break point, you can continue with:
    **(gdb) continue**
  - You can step into a function:
    **(gdb) step**
  - You can run until the next line:
    **(gdb) next**
  - You can run until the end of the function:
    **(gdb) finish**

## Analyzing your program (2)

- So, you end up with “segmentation fault”. What now?
  - You can print variables:

    **(gdb) print a**
  - You can print pointers (handy for seg faults):

    **(gdb) print *a**
  - You can print the hexademical value:

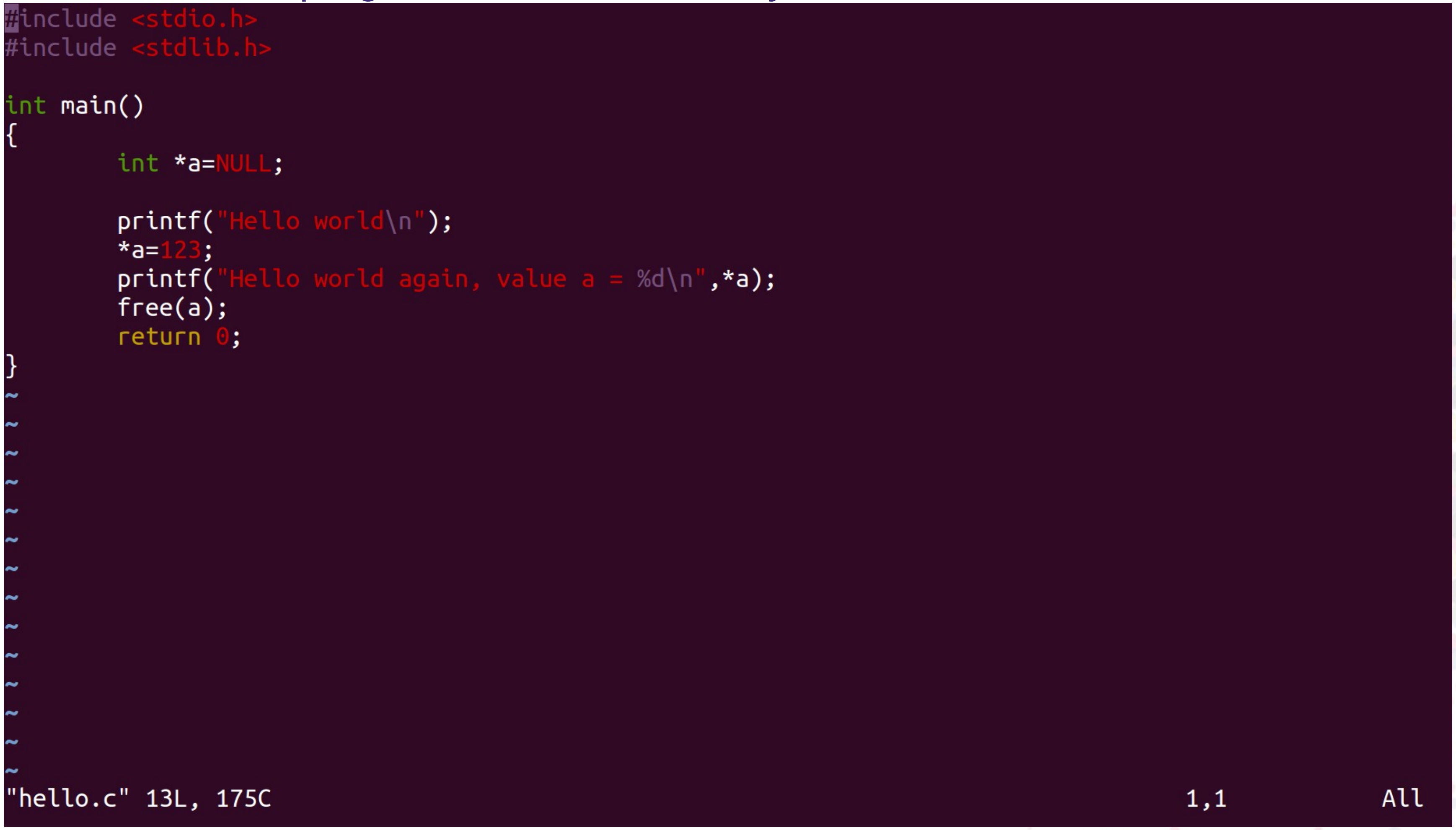
    **(gdb) print/x a**
  - You can watch a variable:

    **(gdb) watch a**

    The program will pause whenever the value **a**, or actually the memory location where **a** is stored, is modified

# Example 1

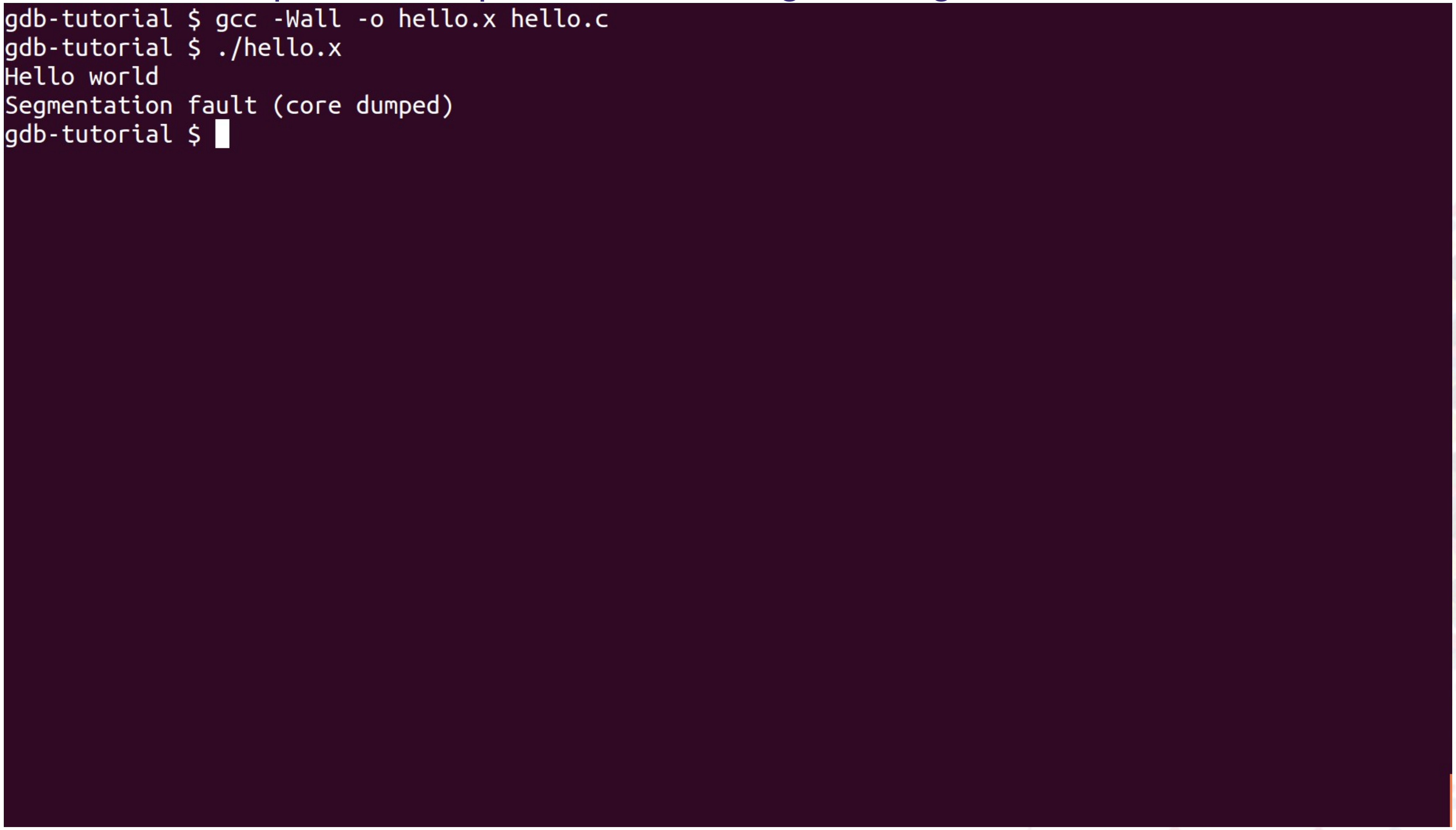
- Small program with a small, nasty, error:



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

int main()
{
        int *a=NULL;

        printf("Hello world\n");
        *a=123;
        printf("Hello world again, value a = %d\n",*a);
        free(a);
        return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"hello.c" 13L, 175C                                                          1,1           All
```

## Example 1

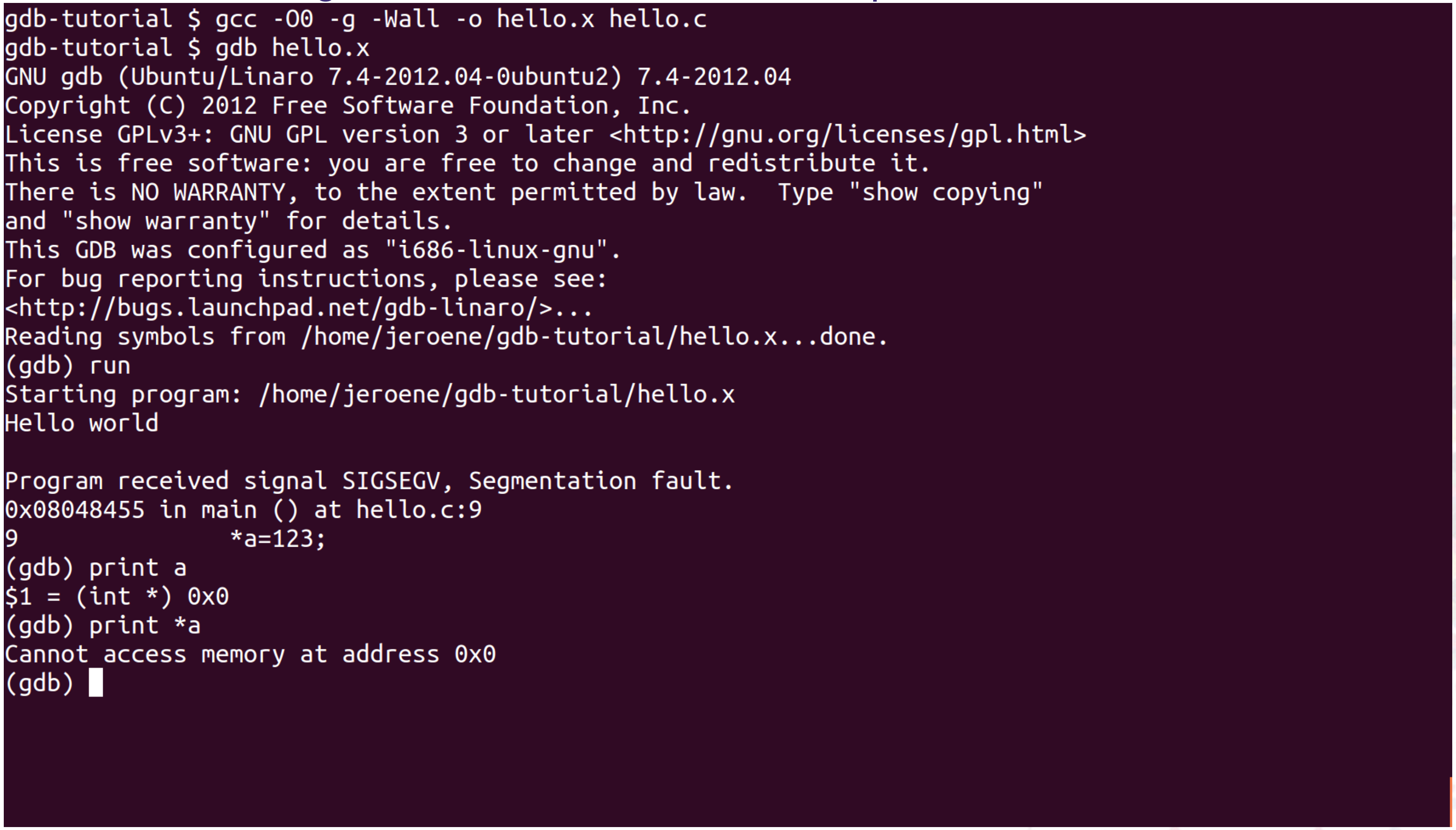
- Compilation, no problem – running into segmentation fault:



```
gdb-tutorial $ gcc -Wall -o hello.x hello.c
gdb-tutorial $ ./hello.x
Hello world
Segmentation fault (core dumped)
gdb-tutorial $ 
```

# Example 1

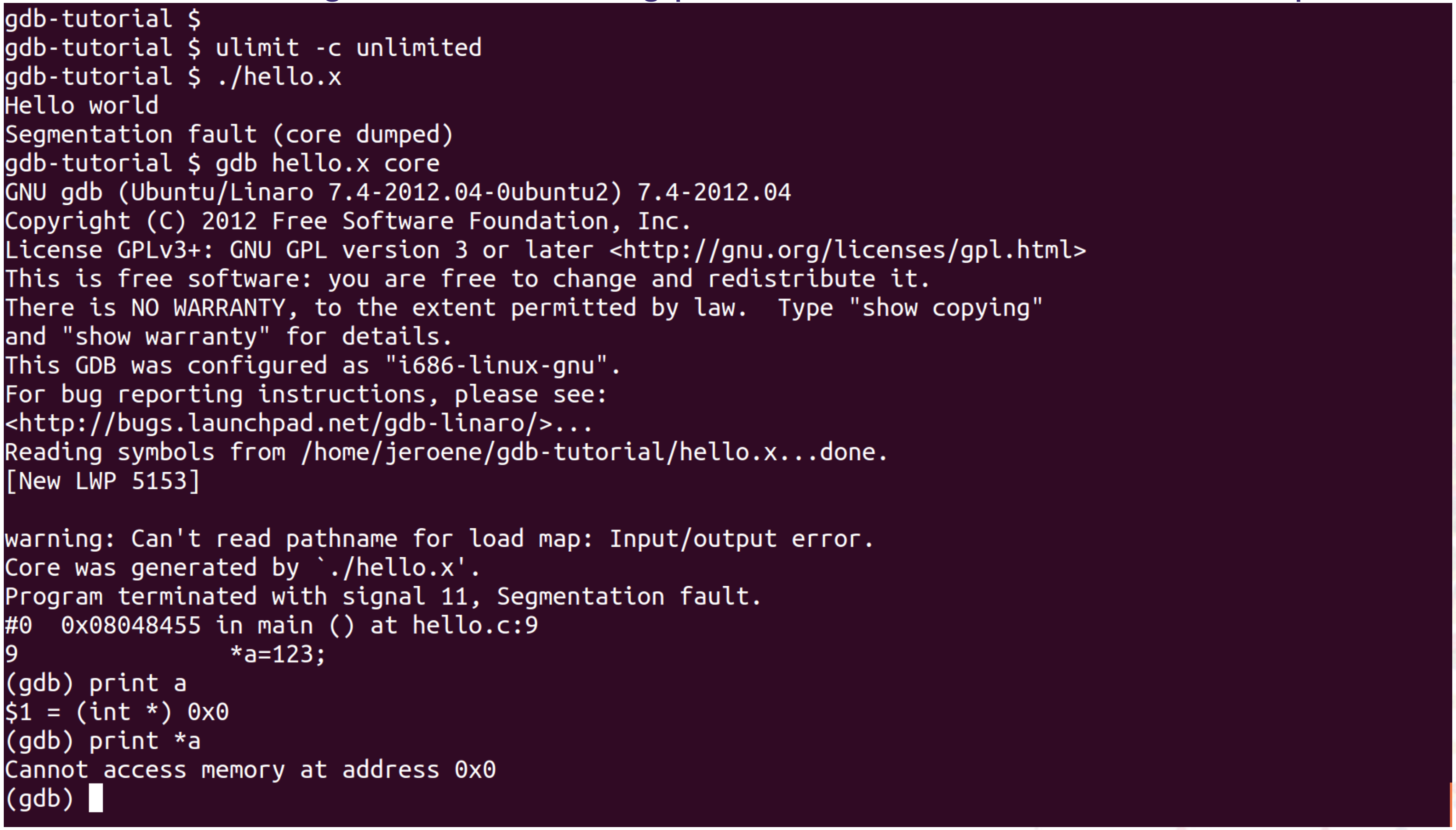
- Running from within GDB – locate the problem:



```
gdb-tutorial $ gcc -O0 -g -Wall -o hello.x hello.c
gdb-tutorial $ gdb hello.x
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/jeroene/gdb-tutorial/hello.x...done.
(gdb) run
Starting program: /home/jeroene/gdb-tutorial/hello.x
Hello world

Program received signal SIGSEGV, Segmentation fault.
0x08048455 in main () at hello.c:9
9               *a=123;
(gdb) print a
$1 = (int *) 0x0
(gdb) print *a
Cannot access memory at address 0x0
(gdb) 
```

# Example 1

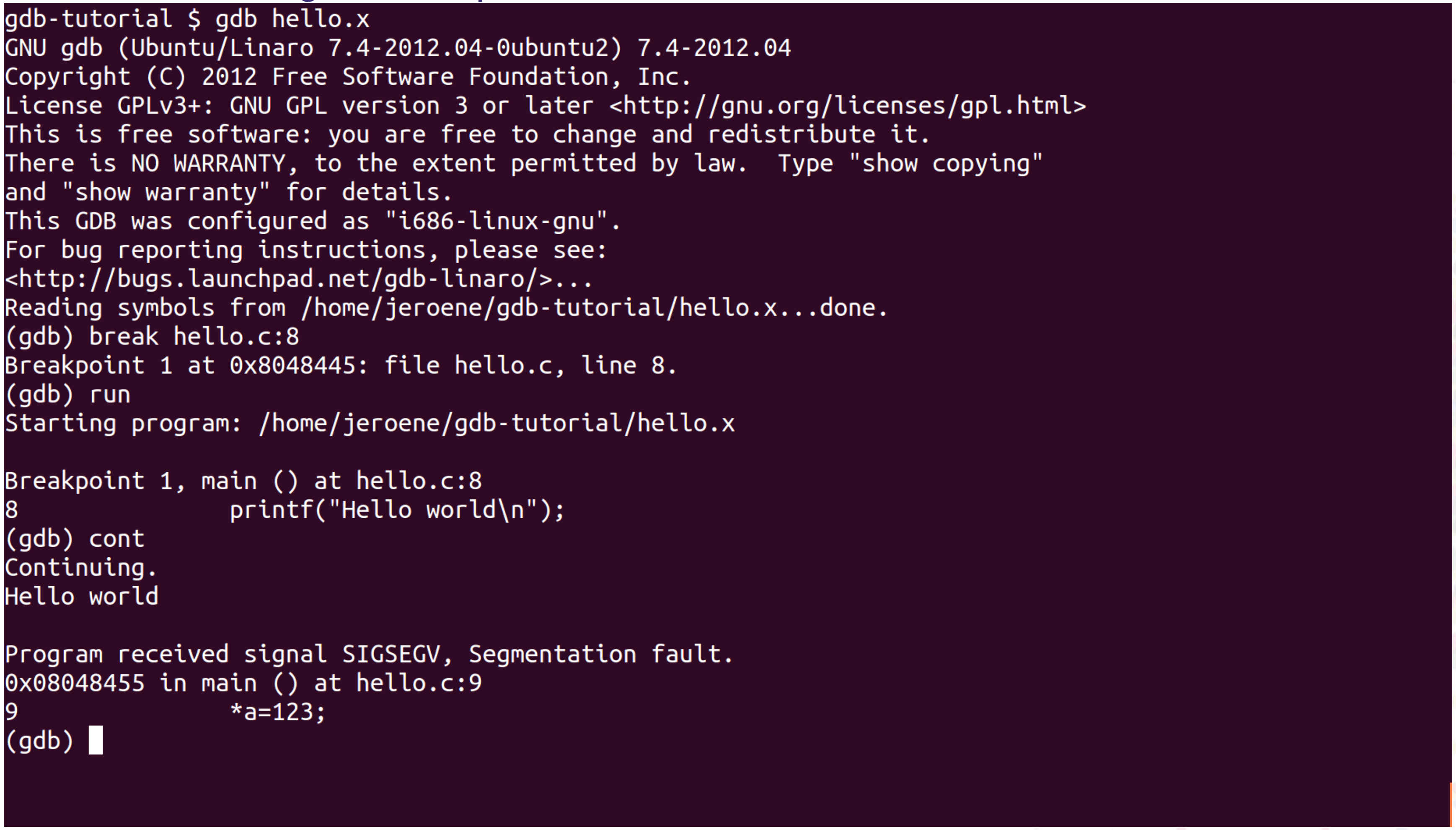
- Running in shell – locating problem with GDB with a core dump:



```
gdb-tutorial $
gdb-tutorial $ ulimit -c unlimited
gdb-tutorial $ ./hello.x
Hello world
Segmentation fault (core dumped)
gdb-tutorial $ gdb hello.x core
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/jeroene/gdb-tutorial/hello.x...done.
[New LWP 5153]

warning: Can't read pathname for load map: Input/output error.
Core was generated by `./hello.x'.
Program terminated with signal 11, Segmentation fault.
#0  0x08048455 in main () at hello.c:9
9               *a=123;
(gdb) print a
$1 = (int *) 0x0
(gdb) print *a
Cannot access memory at address 0x0
(gdb) 
```

## Example 1

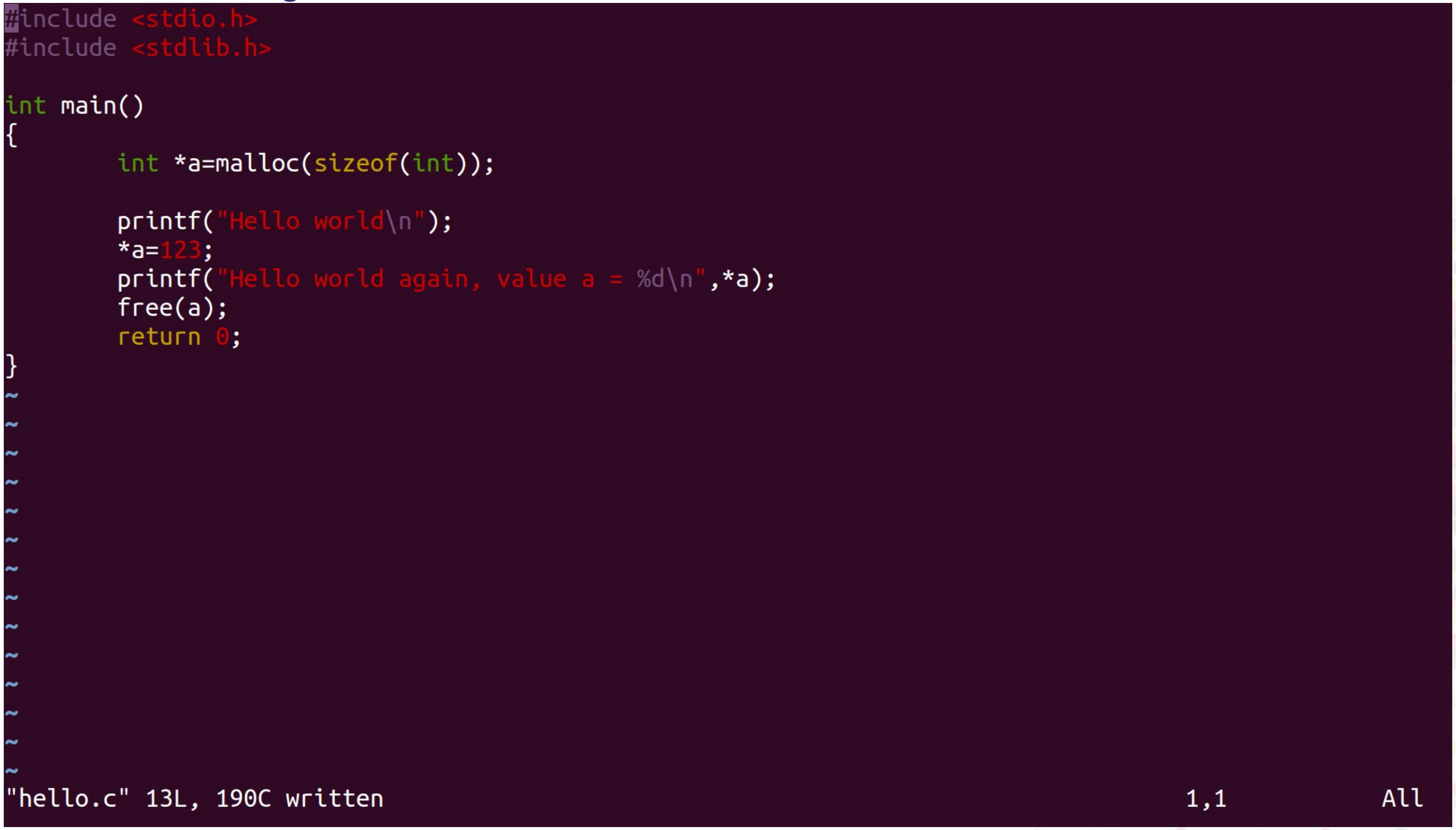
- Setting a break point before the error, continue into it:



```
gdb-tutorial $ gdb hello.x
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/jeroene/gdb-tutorial/hello.x...done.
(gdb) break hello.c:8
Breakpoint 1 at 0x8048445: file hello.c, line 8.
(gdb) run
Starting program: /home/jeroene/gdb-tutorial/hello.x

Breakpoint 1, main () at hello.c:8
8               printf("Hello world\n");
(gdb) cont
Continuing.
Hello world

Program received signal SIGSEGV, Segmentation fault.
0x08048455 in main () at hello.c:9
9               *a=123;
(gdb) █
```

# Example 1

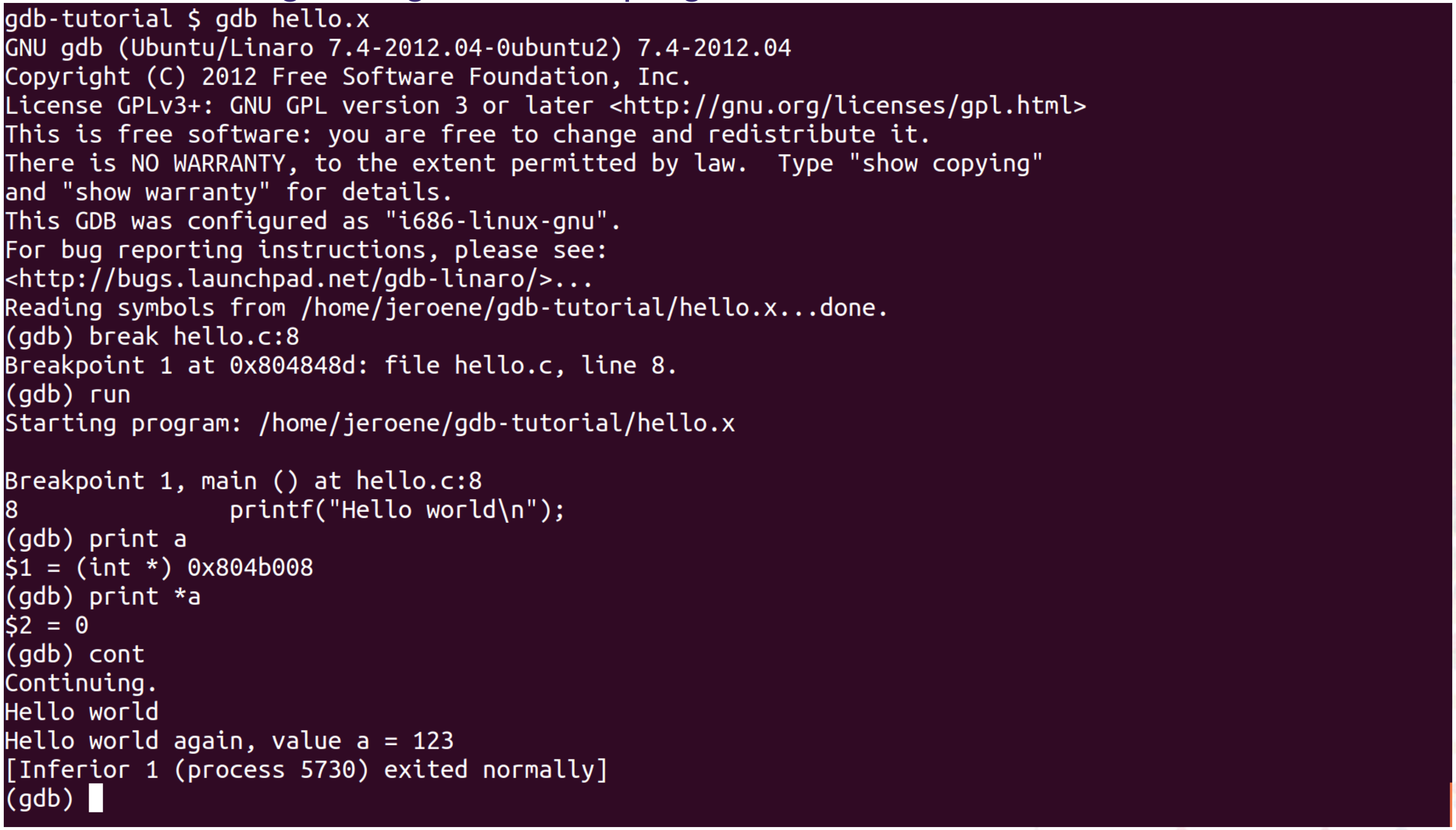
- Fixing the error in the source code:



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

int main()
{
        int *a=malloc(sizeof(int));

        printf("Hello world\n");
        *a=123;
        printf("Hello world again, value a = %d\n",*a);
        free(a);
        return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"hello.c" 13L, 190C written                                              1,1           All
```

# Example 1

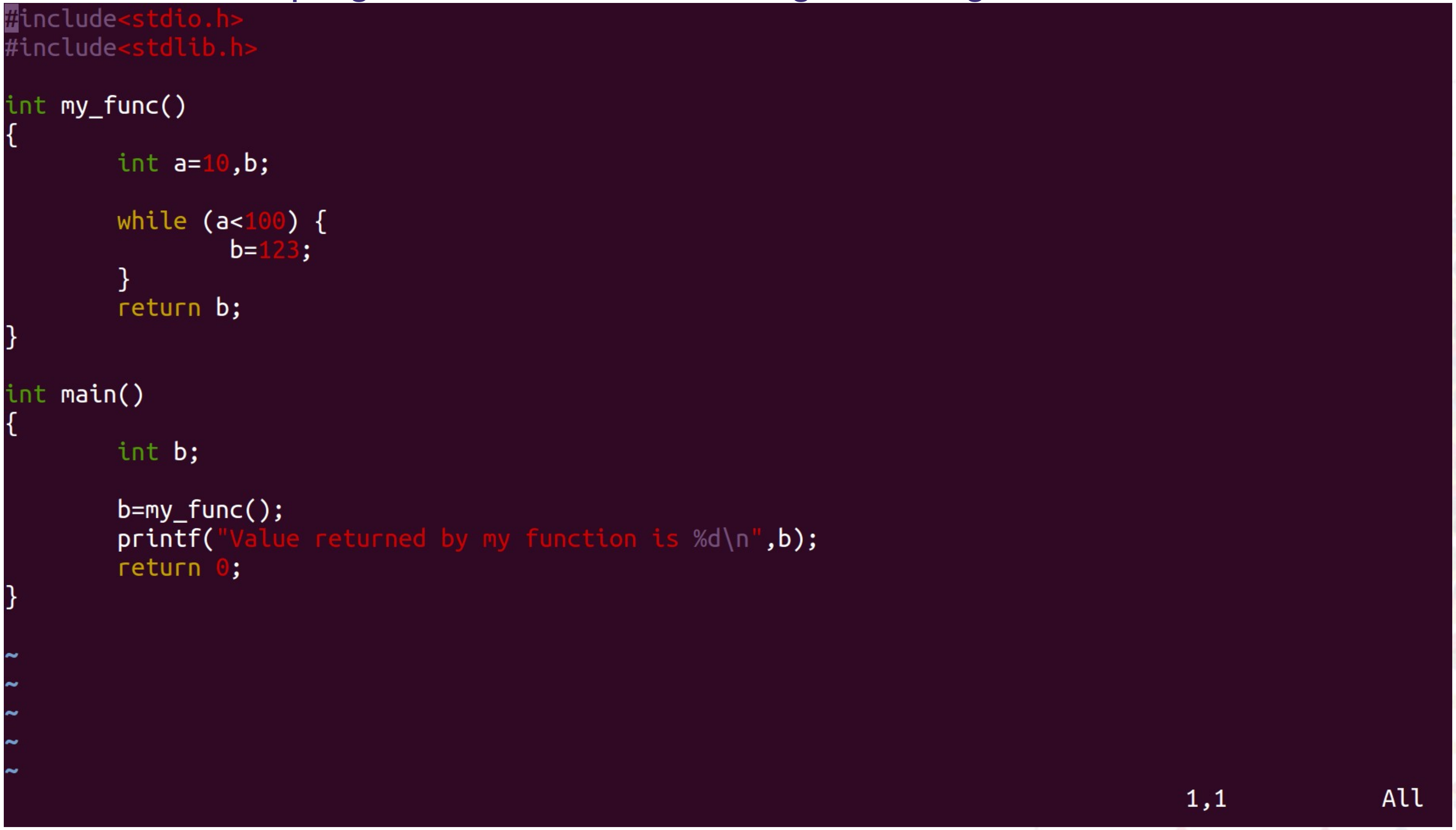
- Going through the fixed program within GDB:



```
gdb-tutorial $ gdb hello.x
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/jeroene/gdb-tutorial/hello.x...done.
(gdb) break hello.c:8
Breakpoint 1 at 0x804848d: file hello.c, line 8.
(gdb) run
Starting program: /home/jeroene/gdb-tutorial/hello.x

Breakpoint 1, main () at hello.c:8
8               printf("Hello world\n");
(gdb) print a
$1 = (int *) 0x804b008
(gdb) print *a
$2 = 0
(gdb) cont
Continuing.
Hello world
Hello world again, value a = 123
[Inferior 1 (process 5730) exited normally]
(gdb) 
```

## Example 2

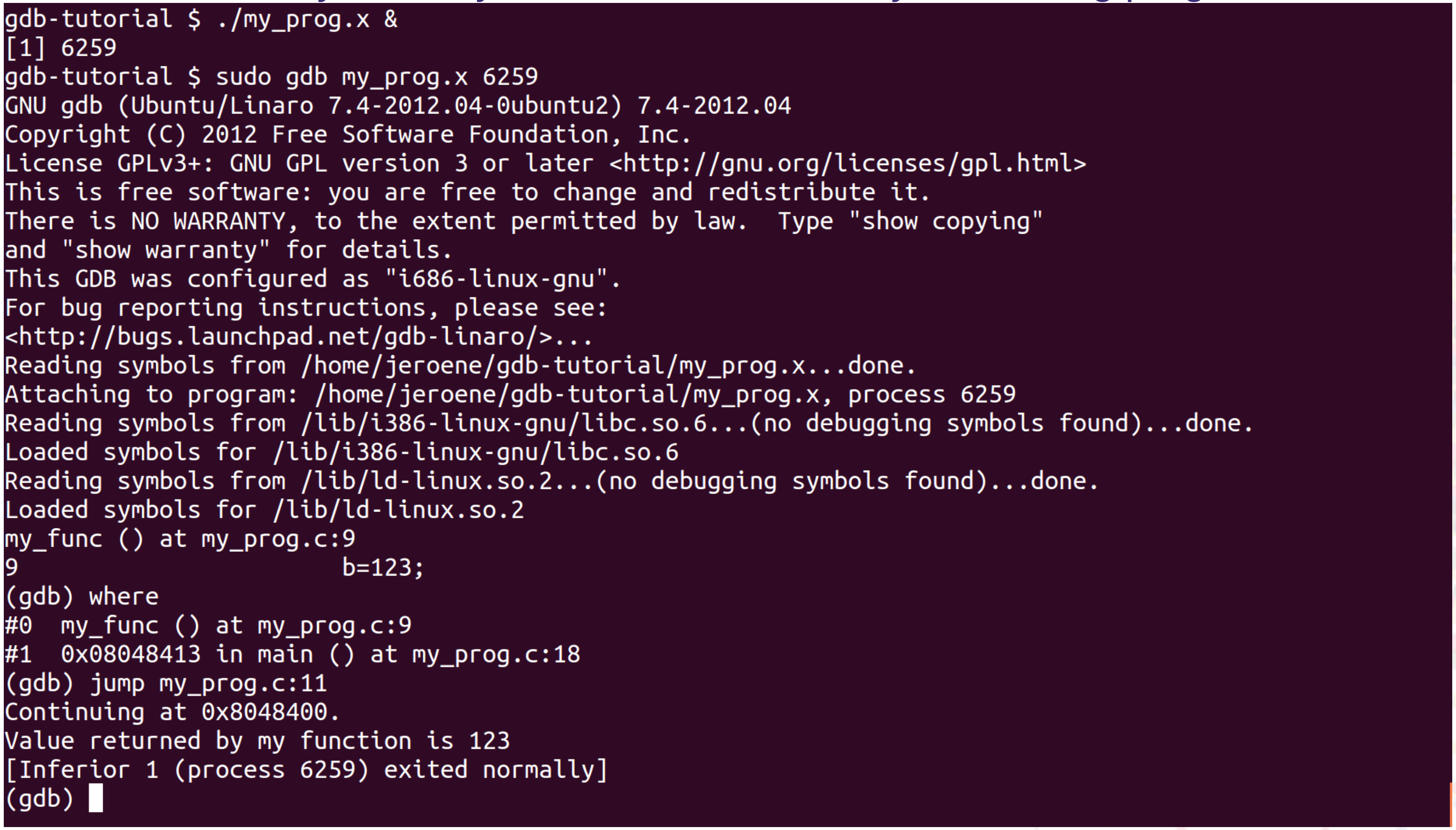
- Your program runs... is it still doing someting?



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

int my_func()
{
        int a=10,b;

        while (a<100) {
                b=123;
        }
        return b;
}

int main()
{
        int b;

        b=my_func();
        printf("Value returned by my function is %d\n",b);
        return 0;
}

~
~
~
~
~
                                                                        1,1           All
```

## Example 2

- In many cases, you can attach GDB to your running program:



```
gdb-tutorial $ ./my_prog.x &
[1] 6259
gdb-tutorial $ sudo gdb my_prog.x 6259
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/jeroene/gdb-tutorial/my_prog.x...done.
Attaching to program: /home/jeroene/gdb-tutorial/my_prog.x, process 6259
Reading symbols from /lib/i386-linux-gnu/libc.so.6...(no debugging symbols found)...done.
Loaded symbols for /lib/i386-linux-gnu/libc.so.6
Reading symbols from /lib/ld-linux.so.2...(no debugging symbols found)...done.
Loaded symbols for /lib/ld-linux.so.2
my_func () at my_prog.c:9
9                       b=123;
(gdb) where
#0  my_func () at my_prog.c:9
#1  0x08048413 in main () at my_prog.c:18
(gdb) jump my_prog.c:11
Continuing at 0x8048400.
Value returned by my function is 123
[Inferior 1 (process 6259) exited normally]
(gdb) 
```

# Questions & Answers