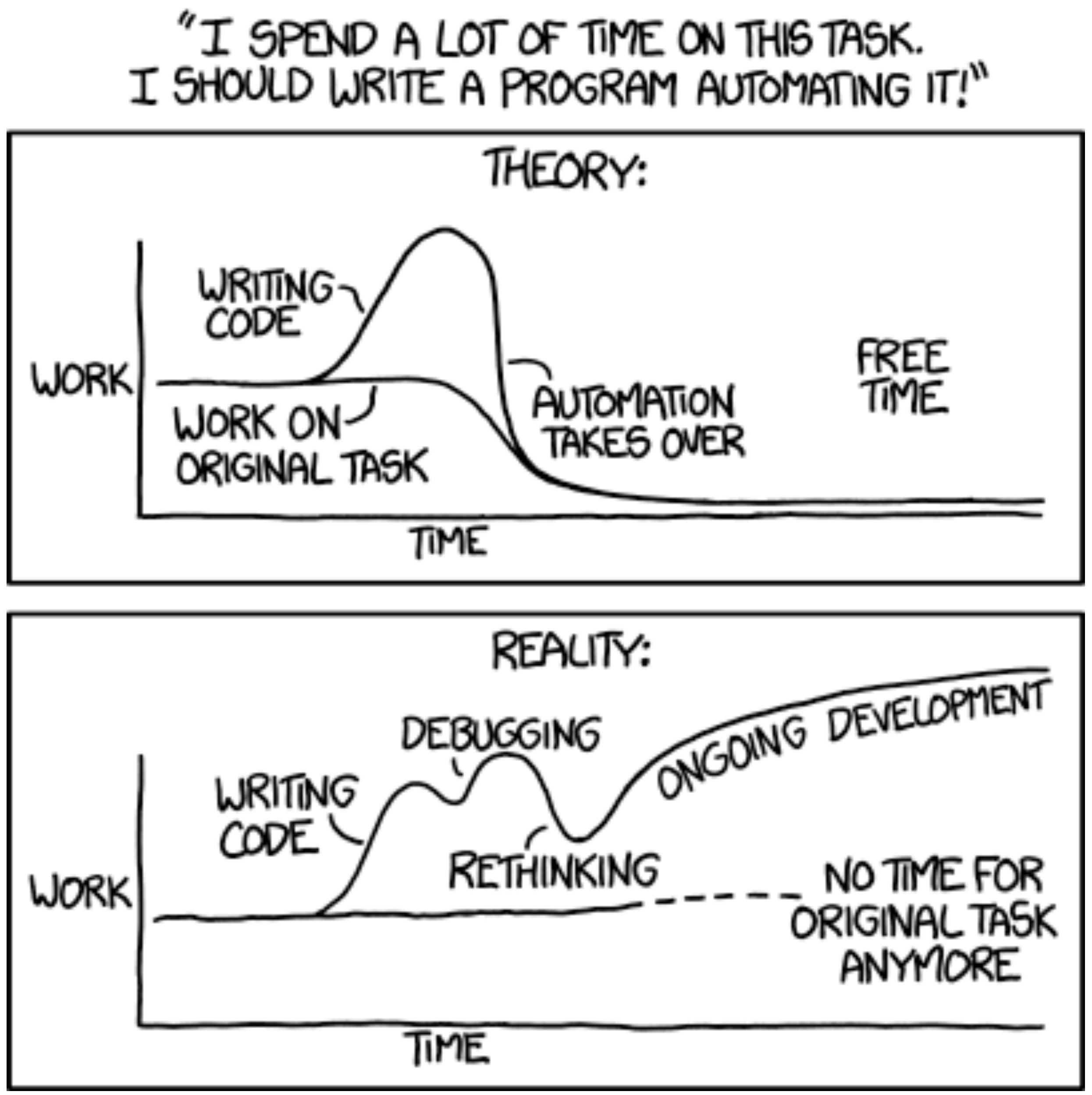


GDB & Debugging

Dr. Gareth Roy (x6439)
gareth.roy@glasgow.ac.uk

- Getting started
- Running GDB
- Interacting with our Code

- Getting started
- Running GDB
- Interacting with our Code

Simple Example

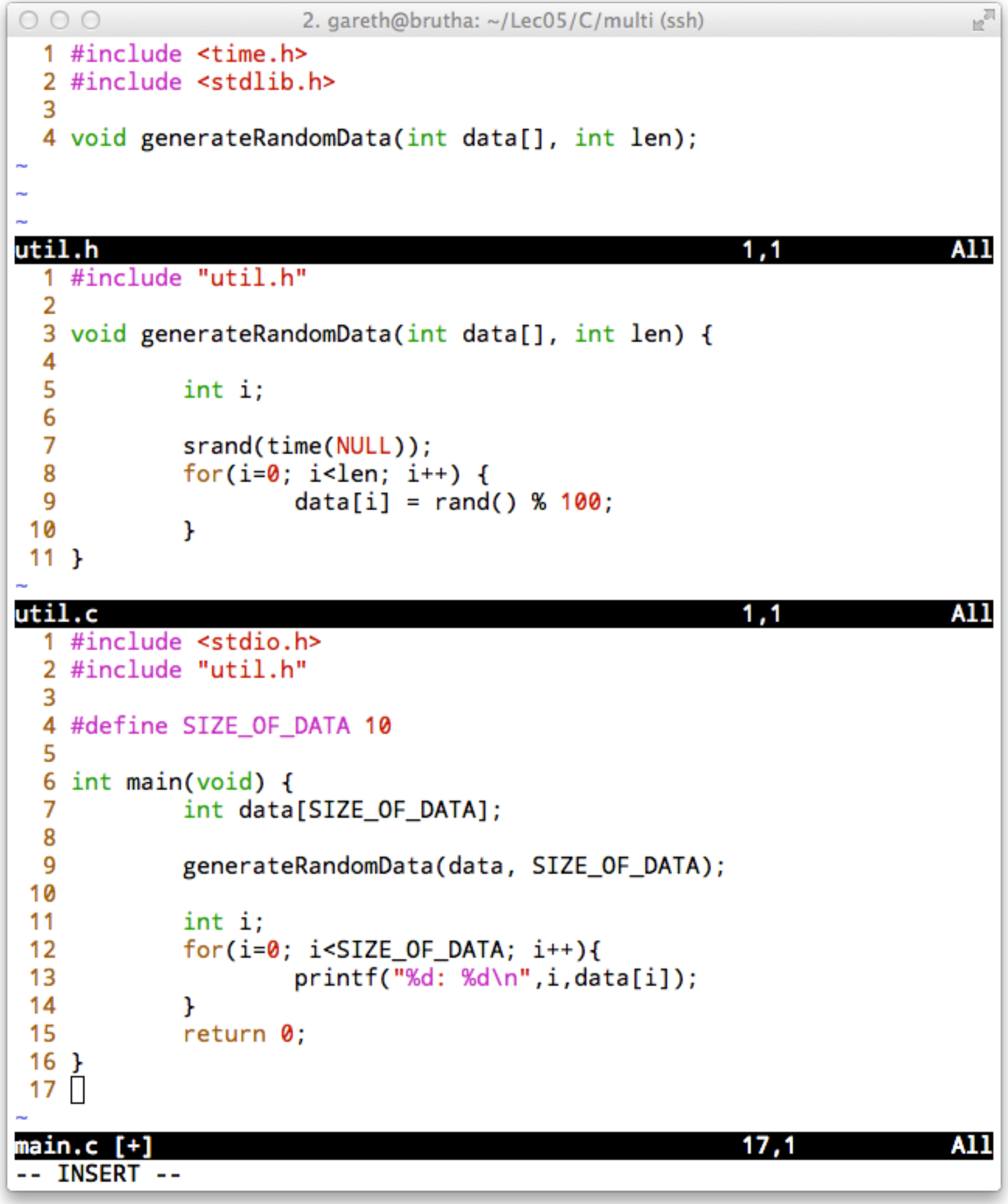
- We'll use the same code example to experiment with GDB.

- For completeness it consists of three source files:

  - **main.c** - creates an array of integers, calls a function to generate random data and prints it to the screen.

  - **util.h** - includes **time.h** and **stdlib.h** and has a function prototype.

  - **util.c** - has a function that takes an array and fills it with a series of random numbers.

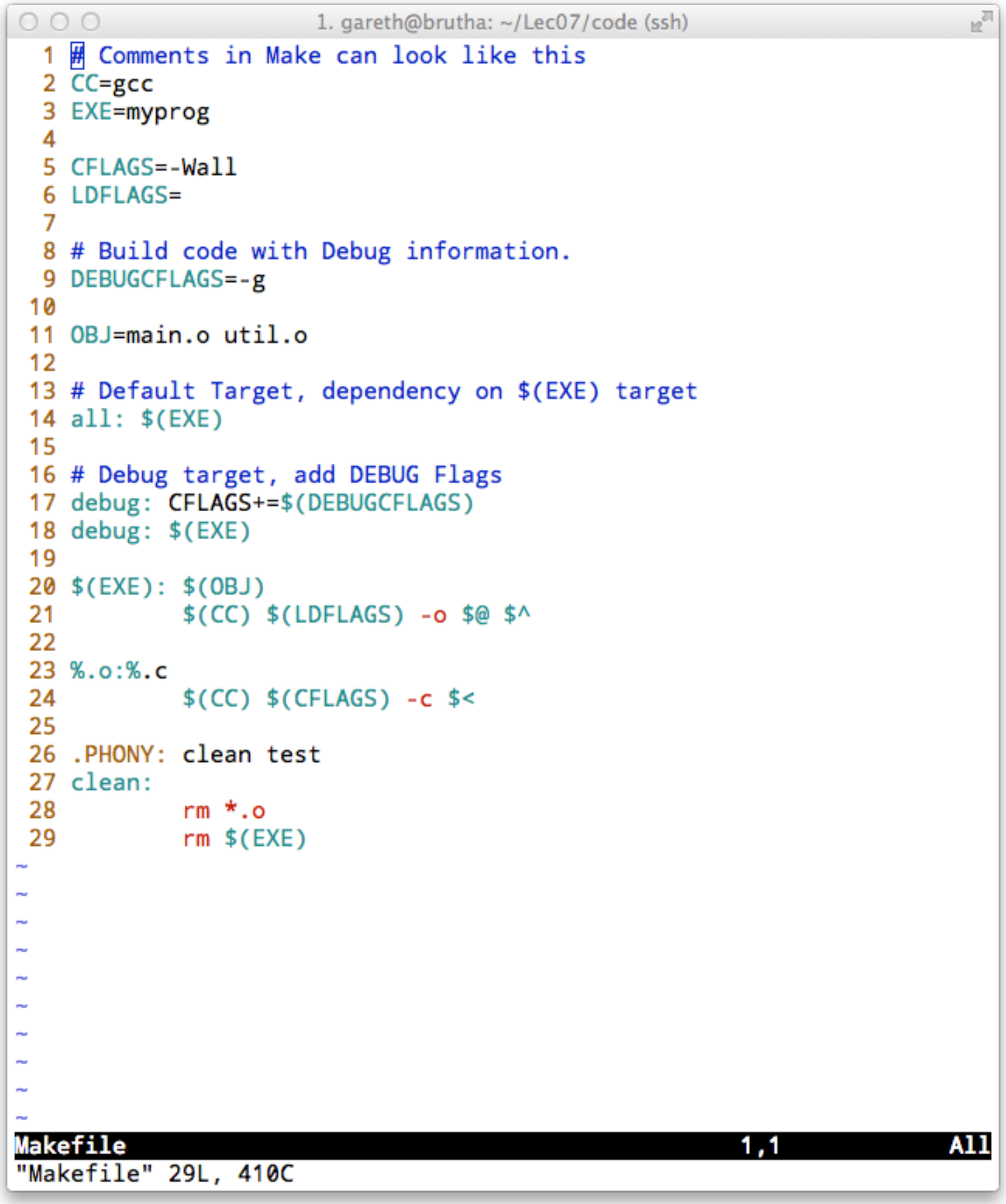


2. gareth@brutha: ~/Lec05/C/multi (ssh)

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

void generateRandomData(int data[], int len);
~
~
~
util.h                                                  1,1          All
#include "util.h"

void generateRandomData(int data[], int len) {

        int i;

        srand(time(NULL));
        for(i=0; i<len; i++) {
                data[i] = rand() % 100;
        }
}
~
util.c                                                  1,1          All
#include <stdio.h>
#include "util.h"

#define SIZE_OF_DATA 10

int main(void) {
        int data[SIZE_OF_DATA];

        generateRandomData(data, SIZE_OF_DATA);

        int i;
        for(i=0; i<SIZE_OF_DATA; i++){
                printf("%d: %d\n",i,data[i]);
        }
        return 0;
}

~
main.c [+]                                              17,1         All
-- INSERT --
```

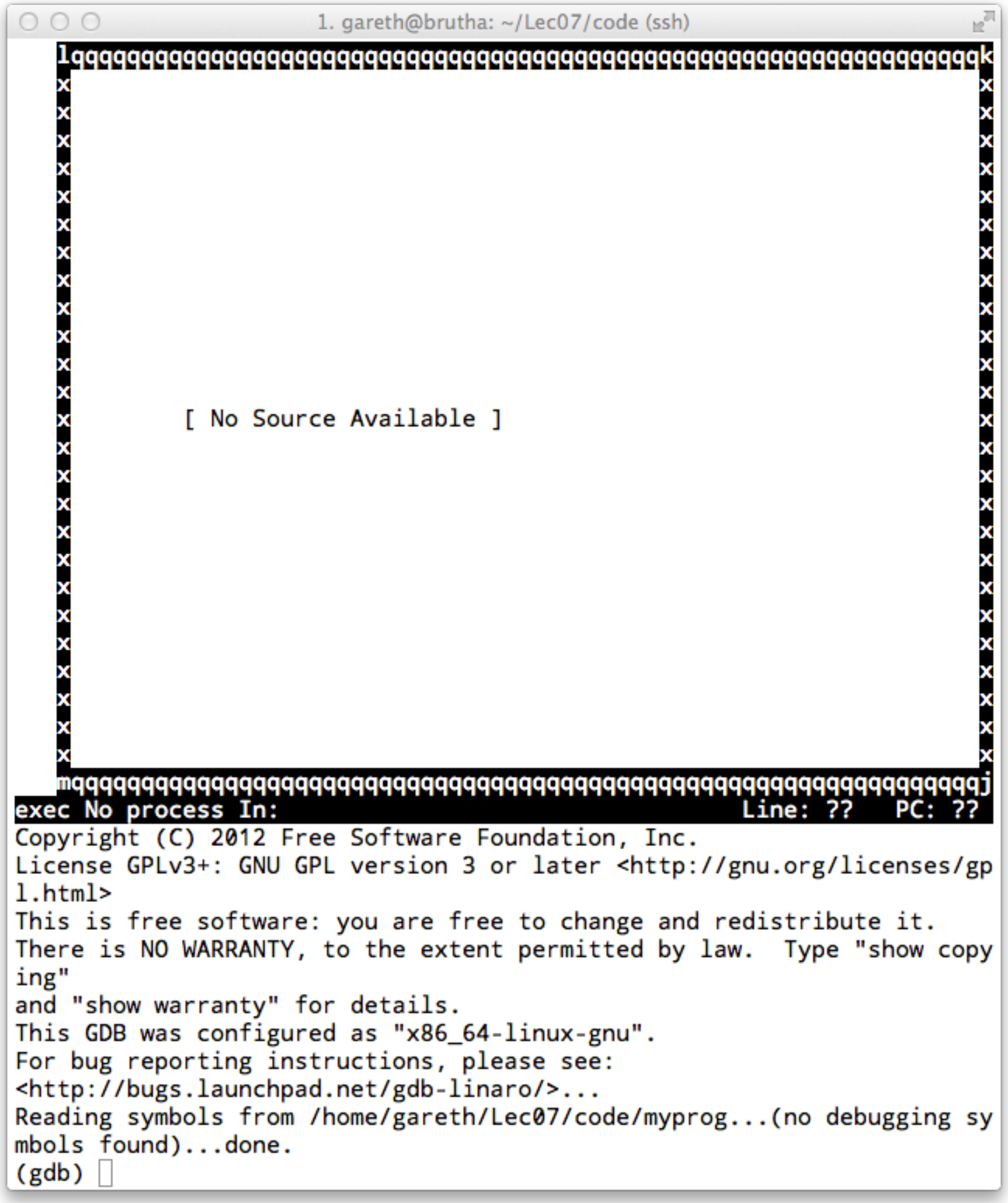
Adding Debug Information

- To our small project we've added a makefile.
- As part of our makefile we've added a target called **debug**.
- This target allows us create a version of our code with debug information.
- In order to use a debugger we have to insert special information into the object files so it can trace the execution of the program.
- In **gcc** to compile code which adds support for debugging we must specify the **-g** flag.
- A one-line compilation would look like:
  - **gcc -g -o myprog *.c**


1. gareth@brutha: ~/Lec07/code (ssh)

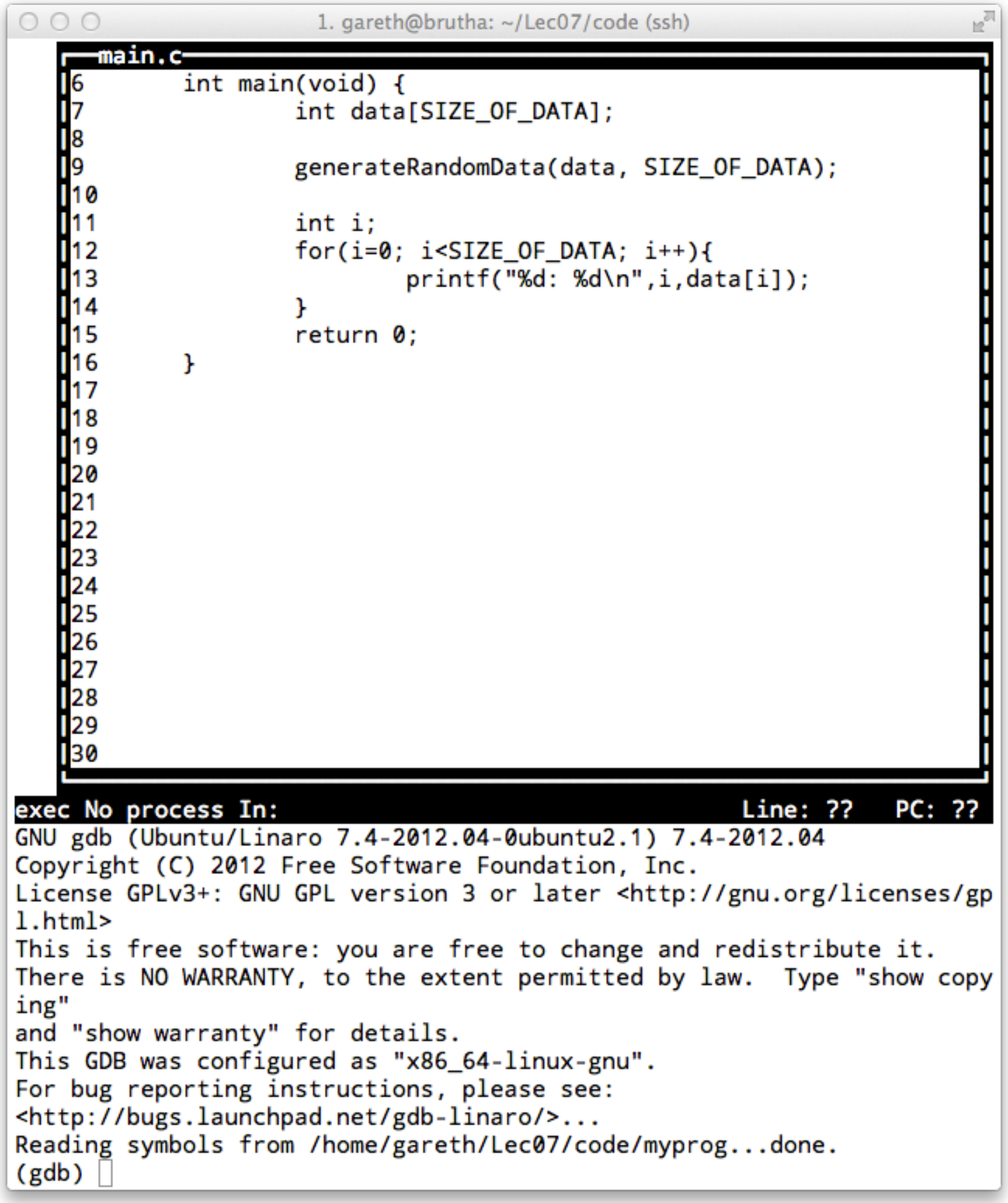
```
# Comments in Make can look like this
CC=gcc
EXE=myprog

CFLAGS=-Wall
LDFLAGS=

# Build code with Debug information.
DEBUGCFLAGS=-g

OBJ=main.o util.o

# Default Target, dependency on $(EXE) target
all: $(EXE)

# Debug target, add DEBUG Flags
debug: CFLAGS+=$(DEBUGCFLAGS)
debug: $(EXE)

$(EXE): $(OBJ)
	$(CC) $(LDFLAGS) -o $@ $^

%.o:%.c
	$(CC) $(CFLAGS) -c $<

.PHONY: clean test
clean:
	rm *.o
	rm $(EXE)
~
~
~
~
~
~
~
~
~
~
Makefile                                              1,1            All
"Makefile" 29L, 410C
```

Adding Debug Information



```
1. gareth@brutha: ~/Lec07/code (ssh)

[ No Source Available ]

exec No process In:                                    Line: ??    PC: ??
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gp
l.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 copy
ing"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/gareth/Lec07/code/myprog...(no debugging sy
mbols found)...done.
(gdb)
```

Without **-g**



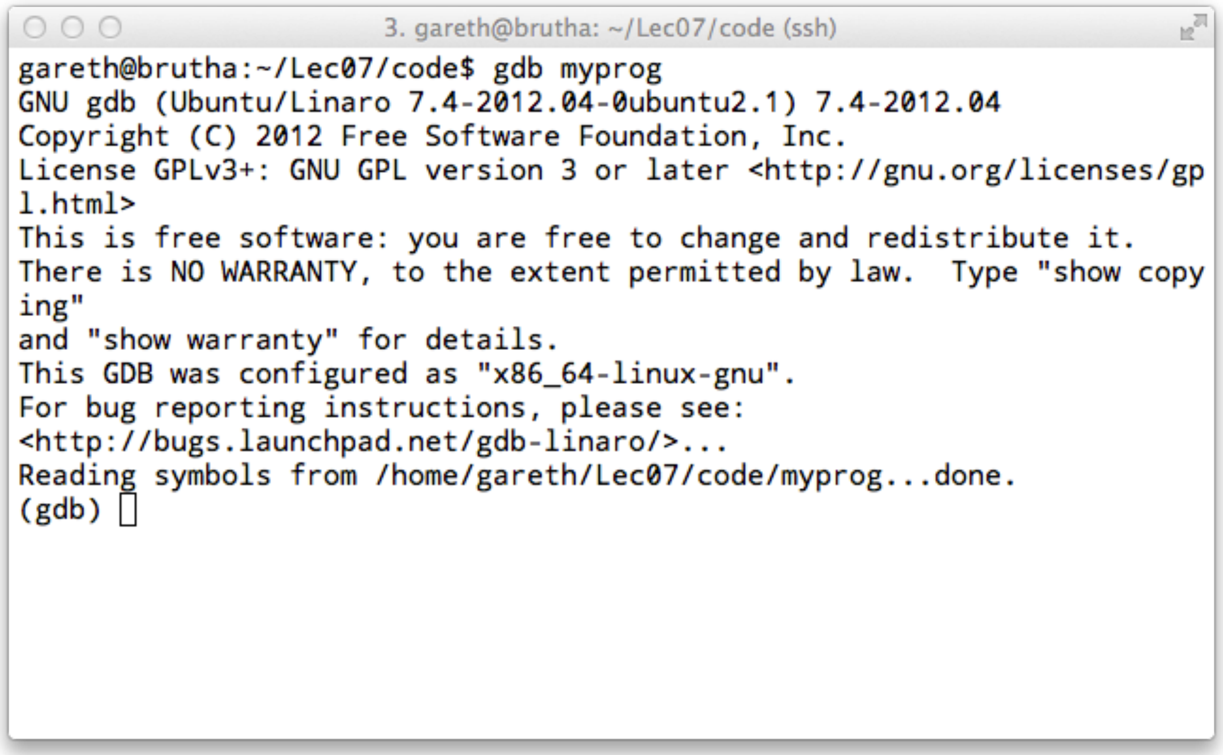
```
1. gareth@brutha: ~/Lec07/code (ssh)

main.c
        int main(void) {
                int data[SIZE_OF_DATA];

                generateRandomData(data, SIZE_OF_DATA);

                int i;
                for(i=0; i<SIZE_OF_DATA; i++){
                        printf("%d: %d\n",i,data[i]);
                }
                return 0;
        }

exec No process In:                                    Line: ??    PC: ??
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gp
l.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 copy
ing"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/gareth/Lec07/code/myprog...done.
(gdb)
```

With **-g**

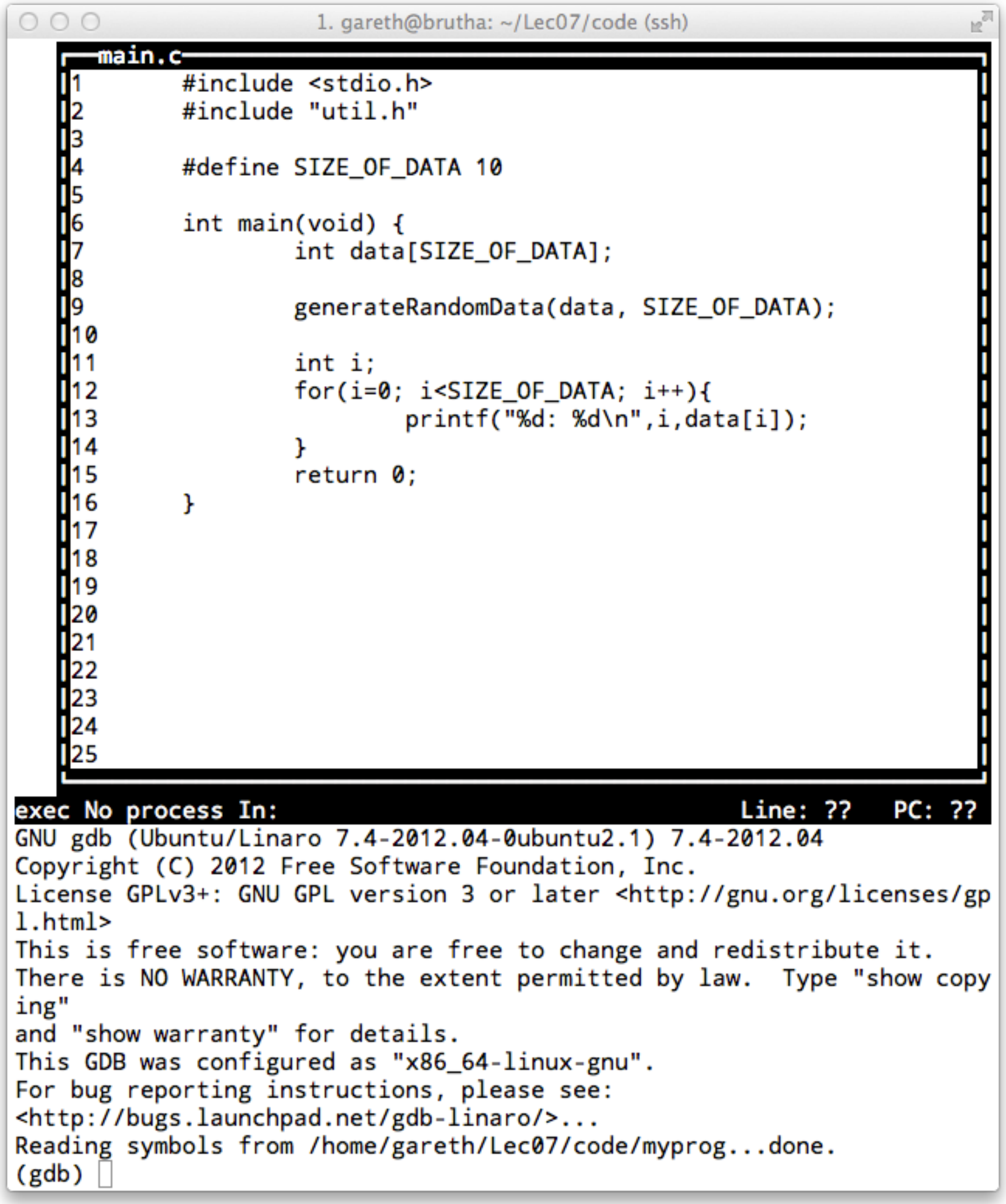
- Getting started
- Running GDB
- Interacting with our Code

Running GDB

- You can start the gdb and load a program for debugging by:
  - **gdb <executable>**
- gdb comes with a simpler terminal UI, which can be used via:
  - **gdb --tui <executable>**
  - **gdbtui <executable>**



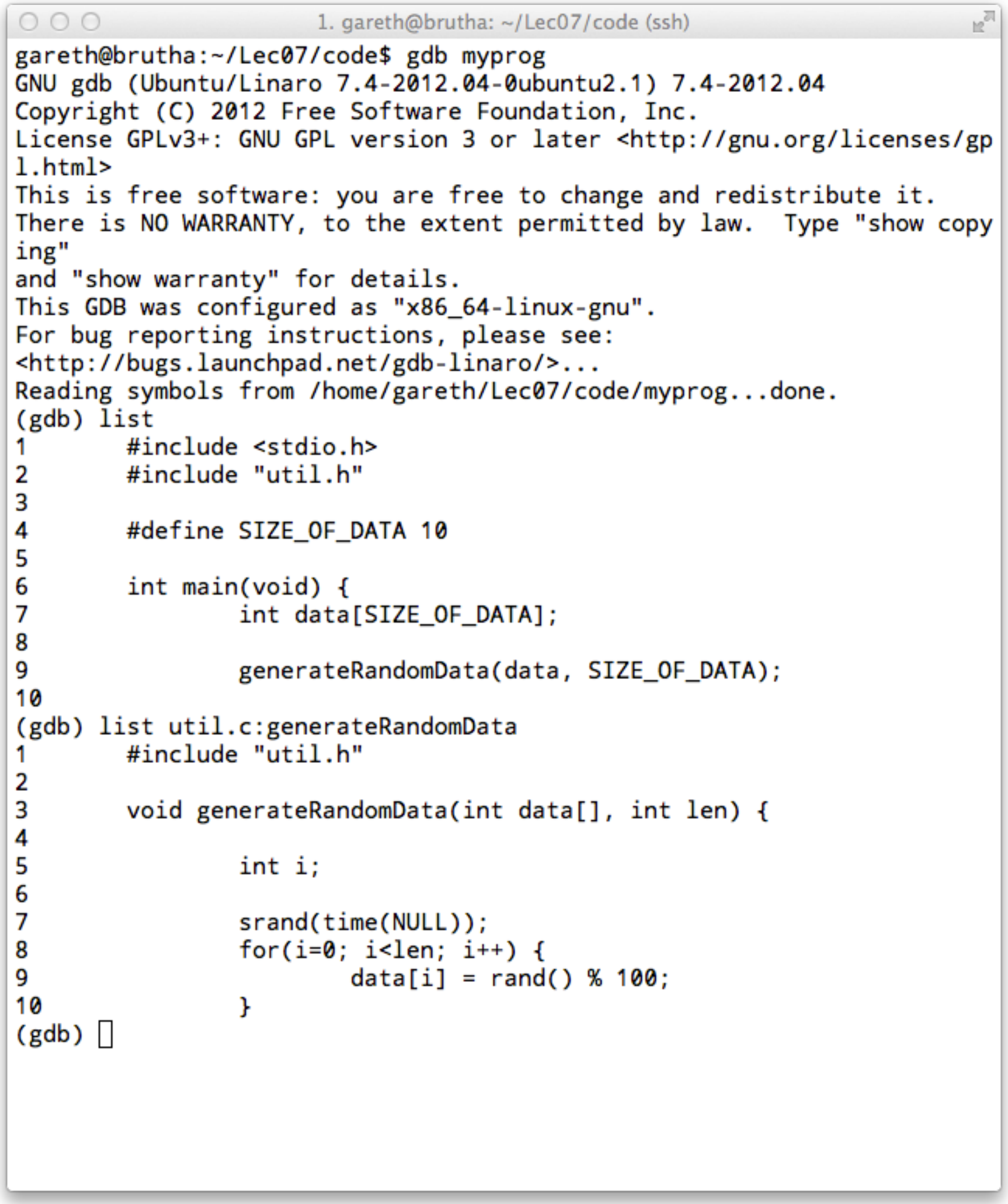
```
gareth@brutha:~/Lec07/code$ gdb myprog
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gp
l.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copy
ing"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/gareth/Lec07/code/myprog...done.
(gdb)
```



```
main.c
        #include <stdio.h>
        #include "util.h"

        #define SIZE_OF_DATA 10

        int main(void) {
                int data[SIZE_OF_DATA];

                generateRandomData(data, SIZE_OF_DATA);

                int i;
                for(i=0; i<SIZE_OF_DATA; i++){
                        printf("%d: %d\n",i,data[i]);
                }
                return 0;
        }

exec No process In:                                   Line: ??   PC: ??
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gp
l.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 copy
ing"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/gareth/Lec07/code/myprog...done.
(gdb)
```

Seeing code in plain GDB

- If we use gdb without the terminal UI, we need to be able to see the code.
- To do this we can use the list command:
  - **list** - by default prints ten lines.
  - **list <line>** - prints the code at the line number specified.
  - **list <function>** - prints the code of the given function
  - **list <filename:function>** - prints the **function** in **filename**.
  - **list 1,4** - prints lines, 1 to 4
- It is often difficult to see what is going on by just using **gdb** so in the following examples we'll be using **gdbtui**.

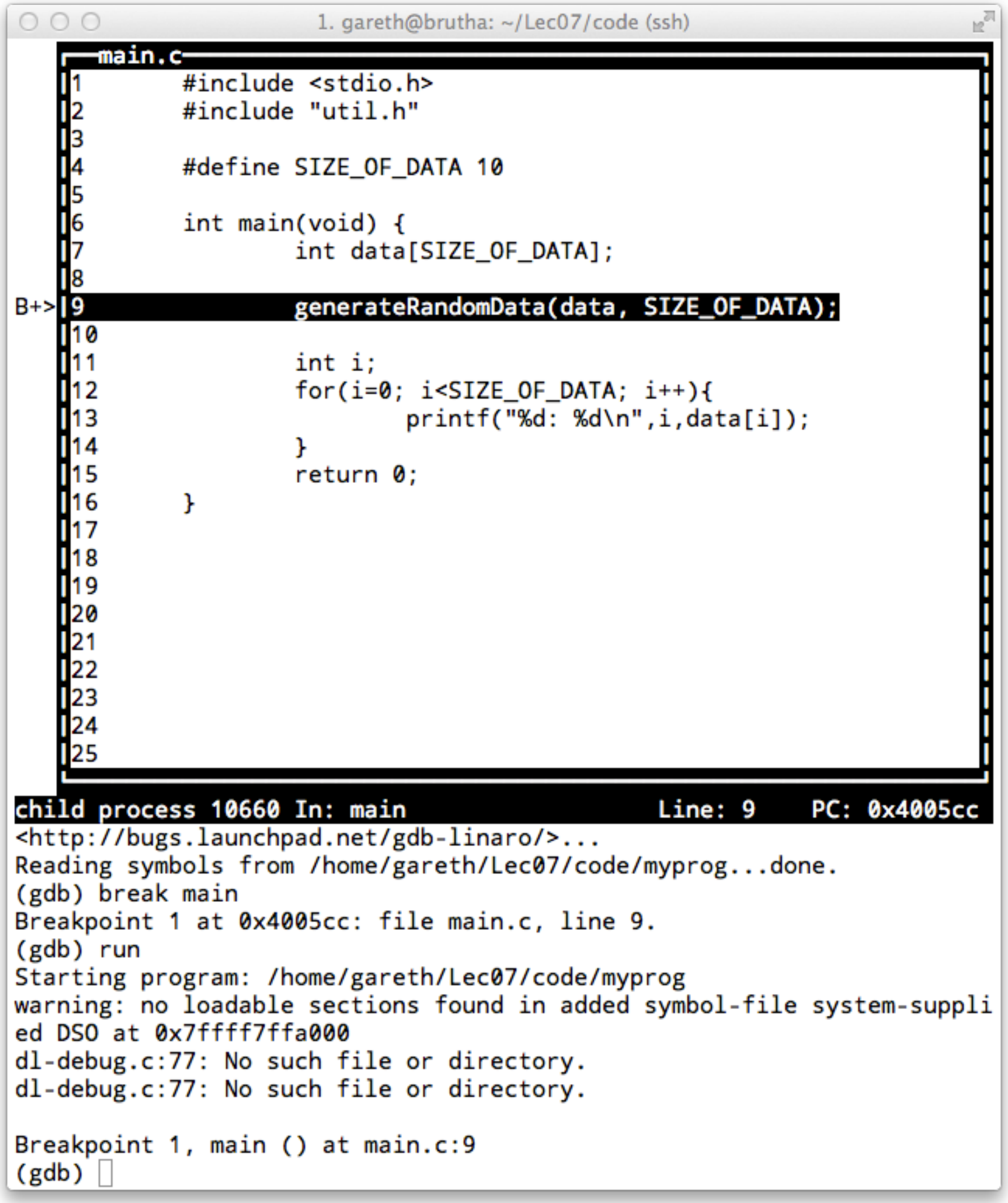


```
gareth@brutha:~/Lec07/code$ gdb myprog
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gp
l.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copy
ing"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/gareth/Lec07/code/myprog...done.
(gdb) list
1       #include <stdio.h>
2       #include "util.h"
3
4       #define SIZE_OF_DATA 10
5
6       int main(void) {
7               int data[SIZE_OF_DATA];
8
9               generateRandomData(data, SIZE_OF_DATA);
10
(gdb) list util.c:generateRandomData
1       #include "util.h"
2
3       void generateRandomData(int data[], int len) {
4
5               int i;
6
7               srand(time(NULL));
8               for(i=0; i<len; i++) {
9                       data[i] = rand() % 100;
10              }
(gdb) 
```

- Getting started
- Running GDB
- Interacting with our Code

Breakpoints

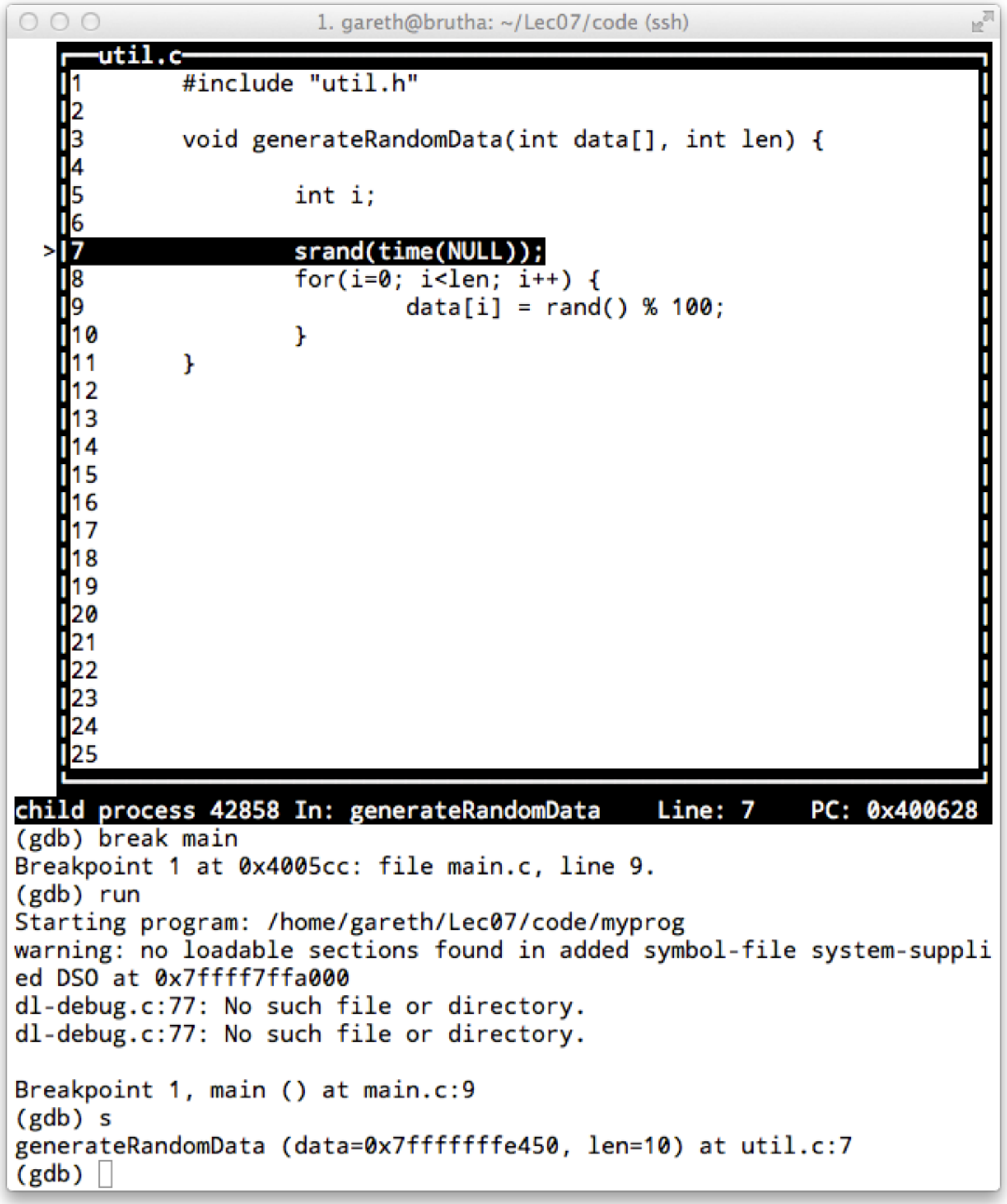
- Breakpoints are how we tell gdb to pause execution.
- We can set breakpoints at any statement in our code, or at function definitions.
- We set a breakpoint using the **break** or **b** keywords.
  - **break <line number>**
  - **break <function>**
  - **break <filename:function>**
- Once we've set our breakpoints we can start our code running with the **run** command (if we hadn't set any breakpoints the code would just run to completion).
- You can see a list of all the breakpoints in you current session by doing
  - **info breakpoints**
- You can use delete to remove set breakpoints:
  - **delete** - deletes all breakpoints
  - **delete <number>** - deletes breakpoint at number



```
1. gareth@brutha: ~/Lec07/code (ssh)
main.c
        #include <stdio.h>
        #include "util.h"

        #define SIZE_OF_DATA 10

        int main(void) {
                int data[SIZE_OF_DATA];

B+>             generateRandomData(data, SIZE_OF_DATA);

                int i;
                for(i=0; i<SIZE_OF_DATA; i++){
                        printf("%d: %d\n",i,data[i]);
                }
                return 0;
        }

child process 10660 In: main                       Line: 9    PC: 0x4005cc
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/gareth/Lec07/code/myprog...done.
(gdb) break main
Breakpoint 1 at 0x4005cc: file main.c, line 9.
(gdb) run
Starting program: /home/gareth/Lec07/code/myprog
warning: no loadable sections found in added symbol-file system-suppli
ed DSO at 0x7ffff7ffa000
dl-debug.c:77: No such file or directory.
dl-debug.c:77: No such file or directory.

Breakpoint 1, main () at main.c:9
(gdb) 
```

Running the code

- As mentioned in the previous slide we run our program in the debugger by using the **run** command.
- Once we’ve stopped at a breakpoint we can start execution again by using **continue** or **c**.
- This will run the code until the next breakpoint.
- We can also execute that code line by line.
- To execute line by line, and enter into functions we use **step** or **s** (this will also enter into system functions which can lead to issues as they normally have no debug info).
- To execute line by line, but not enter into a function we use **next** or **n**.
- We can tell each of these commands the number of lines to execute:
    - **step 3**
    - **next 3**
- We can stop the execution of our program by using the **kill** command.



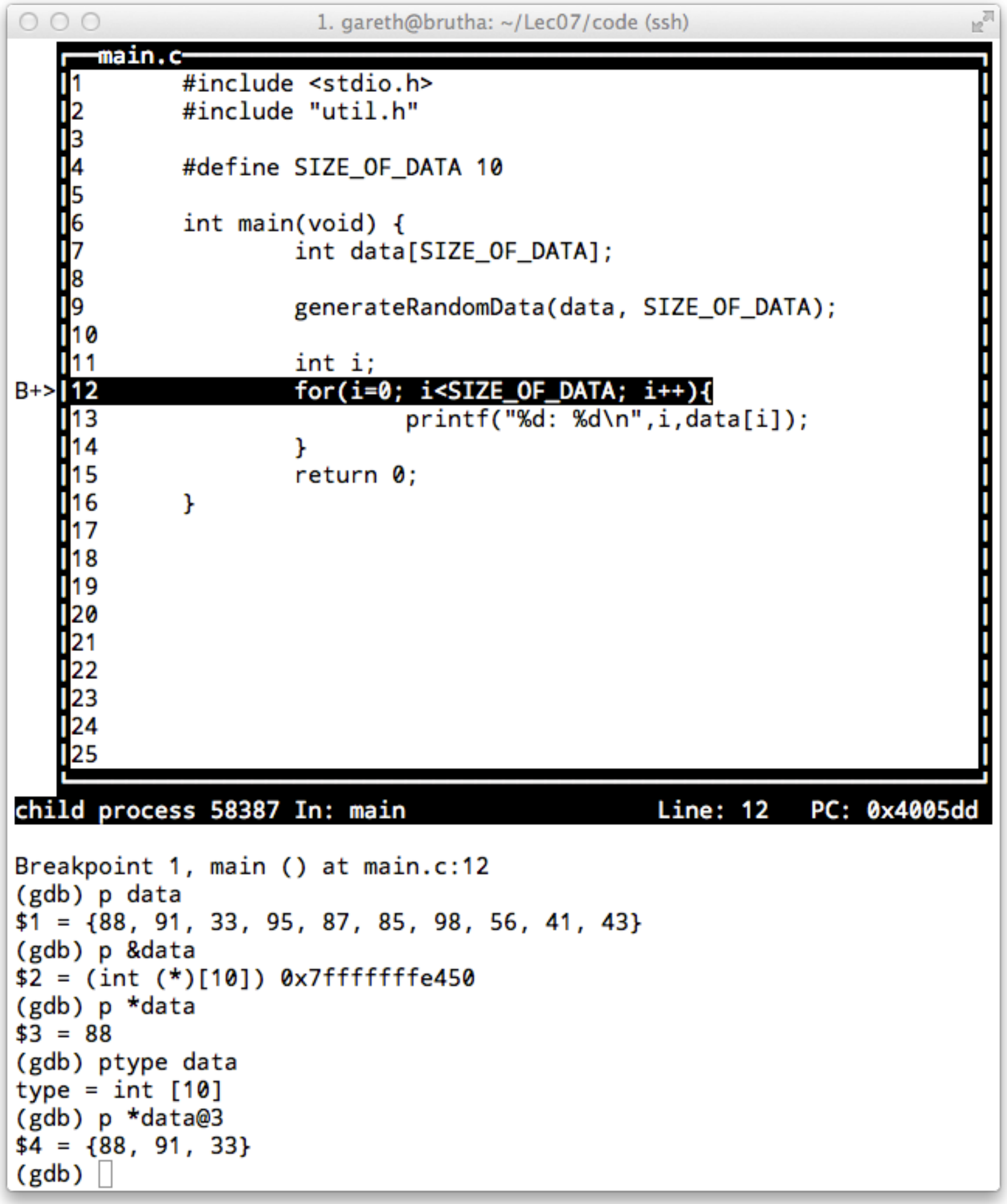
1. gareth@brutha: ~/Lec07/code (ssh)

```
util.c
        #include "util.h"

        void generateRandomData(int data[], int len) {

                int i;

>               srand(time(NULL));
                for(i=0; i<len; i++) {
                        data[i] = rand() % 100;
                }
        }
```

```
child process 42858 In: generateRandomData     Line: 7     PC: 0x400628
(gdb) break main
Breakpoint 1 at 0x4005cc: file main.c, line 9.
(gdb) run
Starting program: /home/gareth/Lec07/code/myprog
warning: no loadable sections found in added symbol-file system-suppli
ed DSO at 0x7ffff7ffa000
dl-debug.c:77: No such file or directory.
dl-debug.c:77: No such file or directory.

Breakpoint 1, main () at main.c:9
(gdb) s
generateRandomData (data=0x7fffffffe450, len=10) at util.c:7
(gdb) 
```

Examining Variables

- Now that we can run our program, we want to be able to examine the contents of the variables.
- We can display the contents of each variable using the **print** or **p** command.
  - **print <variable>** - print variables contents
  - **print &<variable>** - print the address of variable
  - **print *<variable>** - print to data pointed to by variable
  - **ptype <variable>** - print type of variable
  - **print *<variable>@<number>** - print at number of elements pointed to by variable
- We can also specify the output format by appending a type:
  - **p/x <variable>** - hexadecimal
  - **p/u <variable>** - unsigned integer
  - **p/d <variable>** - signed integer
  - **p/o <variable>** - octal
  - **p/f <variable>** - floating point

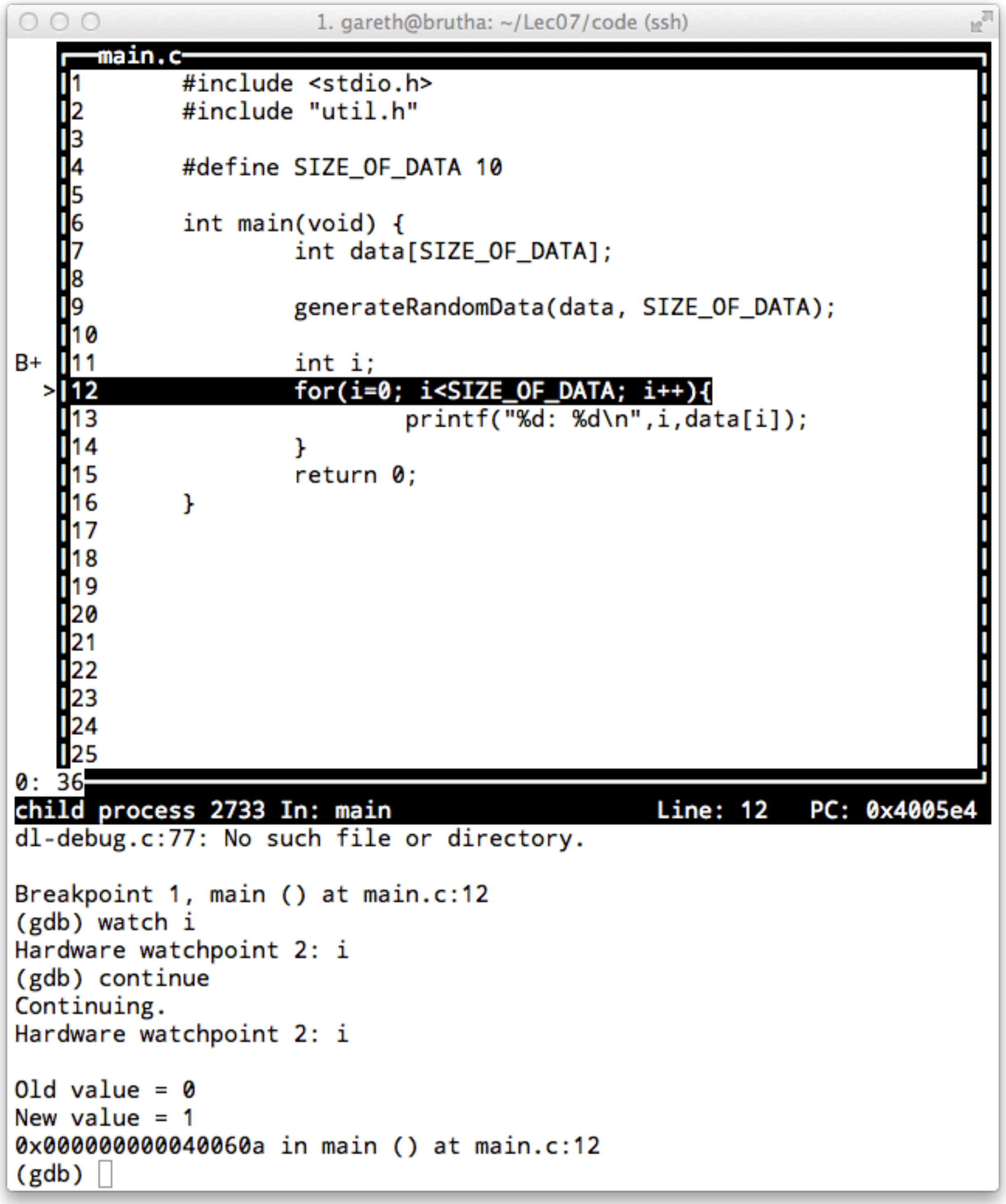


1. gareth@brutha: ~/Lec07/code (ssh)

```
   ┌─main.c─────────────────────────────────────────────────────┐
   │1       #include <stdio.h>
   │2       #include "util.h"
   │3
   │4       #define SIZE_OF_DATA 10
   │5
   │6       int main(void) {
   │7               int data[SIZE_OF_DATA];
   │8
   │9               generateRandomData(data, SIZE_OF_DATA);
   │10
   │11              int i;
B+>│12              for(i=0; i<SIZE_OF_DATA; i++){
   │13                      printf("%d: %d\n",i,data[i]);
   │14              }
   │15              return 0;
   │16      }
   │17
   │18
   │19
   │20
   │21
   │22
   │23
   │24
   │25
   └────────────────────────────────────────────────────────────┘
child process 58387 In: main                   Line: 12   PC: 0x4005dd

Breakpoint 1, main () at main.c:12
(gdb) p data
$1 = {88, 91, 33, 95, 87, 85, 98, 56, 41, 43}
(gdb) p &data
$2 = (int (*)[10]) 0x7fffffffe450
(gdb) p *data
$3 = 88
(gdb) ptype data
type = int [10]
(gdb) p *data@3
$4 = {88, 91, 33}
(gdb)
```

Watch points

- Often we have problems with variables changing during the execution of a program.
- For instance if we have written beyond the end of an array we could be overwriting locations in memory.
- Or we may have multiple pointers to an area of memory which is being accessed.
- We'd like to be able to observe an variable and stop execution when it is changed.
- To do this we can set a watch point using the watch command.
  - **watch <variable>**
    - i.e **watch i**
  - **watch <condition>**
    - i.e **watch i if i > 7**



```
main.c
        #include <stdio.h>
        #include "util.h"

        #define SIZE_OF_DATA 10

        int main(void) {
                int data[SIZE_OF_DATA];

                generateRandomData(data, SIZE_OF_DATA);

B+              int i;
>               for(i=0; i<SIZE_OF_DATA; i++){
                        printf("%d: %d\n",i,data[i]);
                }
                return 0;
        }
0: 36
child process 2733 In: main                          Line: 12    PC: 0x4005e4
dl-debug.c:77: No such file or directory.

Breakpoint 1, main () at main.c:12
(gdb) watch i
Hardware watchpoint 2: i
(gdb) continue
Continuing.
Hardware watchpoint 2: i

Old value = 0
New value = 1
0x000000000040060a in main () at main.c:12
(gdb) 
```

- Getting started
- Running GDB
- Interacting with our Code