The Dynamic Debugger *gdb*

This handout introduces the basics of using *gdb*(1), a very powerful dynamic debugging tool. No-one always writes programs that execute perfectly every time, and while reading the program source can help find bugs, some can only be discovered by running the program and seeing what happens. That's where a dynamic debugger comes in; it lets you stop execution during the run and look at variables.

Setting It Up

To use *gdb*, you must compile your program using *gcc* and give the `-g` flag:

```
gcc -ansi -g program.c -o program
```

The `-g` flag tells the compiler to add information for the debugger. To debug your program, simply say:

```
gdb program
```

The rest of this handout contains several sample programs and how to use *gdb* to find the bugs. I recommend you use *gdb* to get used to it; in particular, make extensive use of its help command! Just type

```
help
```

to its prompt, and it will tell you what to do. Some of the features that I did not show which you will find particularly useful are the backtrace facility, which shows you what routines have been called and the values of the parameters.

Example 1

Here's a program that is supposed to add 2 to a variable `j` every time through the **`for`** loop:

```
#include <stdio.h>
int main(void)
{
        int i, j = 0;

        for(i = 0; i < 100; i++);
                j += 2;

        printf("The value of j is: %d\n", j);
        return(0);
}
```

I compile and run it, and it does not work:

```
$ gcc -ansi -g -o sample1 sample1.c
$ sample1
The value of j is: 2
```

Oops! Let's get out *gdb* and see what happens. What I type is in red, what the computer prints is in plain face, and my comments are in *italics*.

```
$ gdb sample1
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
```

```
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sample1...done.
(gdb) l                                          list 10 lines
1 #include <stdio.h>
2
3 int main(void)
4 {
5 int i, j = 0;
6
7 for(i = 0; i < 100; i++);
8 j += 2;
9
10 printf("The value of j is: %d\n", j);
(gdb) b main                                     put in a breakpoint
                                                 the program will stop at main
                                                 when it is executed
Breakpoint 1 at 0x652: file sample1.c, line 5.
(gdb) run                                        run the program
Starting program: /home/bishop/ecs36a/20/sample1

Breakpoint 1, main () at sample1.c:5
5 int i, j = 0;
(gdb) n                                          do the next statement
7 for(i = 0; i < 100; i++);
(gdb) p i                                        print the value of i
$1 = 0
(gdb) p j                                        print the value of j
$2 = 0
(gdb) n                                          do the next statement
8 j += 2;
(gdb) p i                                        print the value of i
$3 = 100
(gdb) p j                                        print the value of j
$4 = 0                                           aha! why is it not 200? It's not
                                                 getting incremented right, so let's
                                                 check the for loop ... and sure
                                                 enough, that's where the problem is!
(gdb) q
A debugging session is active.

Inferior 1 [process 15273] will be killed.

Quit anyway? (y or n) y
```

Example 2

Now for a more complex example. Here's a program that's supposed to multiply `s` by 2 until `s` is greater than 100:

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

```
int main(void)
{
    int i = 1, s;

    s = 3;
    while(i = 1){
        s += s;
        if (s > 100)
            i = 0;
    }

    return(0);
}
```

I compile and run it, and it hangs; I need to kill it with control-C:

```
$ gcc -ansi -g -o sample2 sample2.c
$ sample1
```

Again, let's use *gdb* to figure out what happened:

```
$ gdb sample2
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sample2...done.
(gdb) l                                          list 10 lines
1 #include <stdio.h>
2
3 int main(void)
4 {
5 int i = 1, s;
6
7 s = 3;
8 while(i = 1){
9 s += s;
10 if (s > 100)
(gdb) b 9                                       stop the program at line 9
Breakpoint 1 at 0x61c: file sample2.c, line 9.
(gdb) r                                         start the program

Starting program: /home/bishop/ecs36a/20/sample2

Breakpoint 1, main () at sample2.c:9
```

```
9 s += s;
(gdb) watch i                          watch the value of i
Hardware watchpoint 2: i
(gdb) watch s                          watch the value of s
Hardware watchpoint 3: s
(gdb) c                                continue execution
Continuing.

Hardware watchpoint 3: s               program stops here because
                                       the value of s changed
Old value = 3
New value = 6
main () at sample2.c:10
10 if (s > 100)
(gdb) c                                keep going until the next iteration
Continuing.                            will make s > 100

Breakpoint 1, main () at sample2.c:9
9 s += s;
(gdb) c
Continuing.

Hardware watchpoint 3: s

Old value = 6
New value = 12
main () at sample2.c:10
10 if (s > 100)
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9
9 s += s;
(gdb) c
Continuing.

Hardware watchpoint 3: s

Old value = 12
New value = 24
main () at sample2.c:10
10 if (s > 100)
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9
9 s += s;
(gdb) c
Continuing.

Hardware watchpoint 3: s

Old value = 24
New value = 48
```

```
main () at sample2.c:10
10 if (s > 100)
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9
9 s += s;
(gdb) c
Continuing.

Hardware watchpoint 3: s

Old value = 48
New value = 96
main () at sample2.c:10
10 if (s > 100)
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9
9 s += s;
(gdb) c
Continuing.

Hardware watchpoint 3: s                    program stops here because
                                            the value of s changed; note the new
Old value = 96                              value is greater than 100
New value = 192
main () at sample2.c:10
10 if (s > 100)
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9
9 s += s;
(gdb) c
Continuing.

Hardware watchpoint 3: s                    program stops here because
                                            the value of s changed; and now we
Old value = 192                             see the bug --- the program fails to
New value = 384                             stop when s > 100, so we check the
                                            condition in the loop; that depends on i
Breakpoint 1, main () at sample2.c:9
9 s += s;
$8 = 384
(gdb) watch i                               so now let's look at i's value;
Hardware watchpoint 2: i                    when i = 1, the program loops
(gdb) c
Continuing.

Hardware watchpoint 2: i

Old value = 1                               here i changes from 1 to 0
```

```
New value = 0                                   so the loop should stop
main () at sample2.c:8
8 while(i = 1){
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9            but it didn't --- why?
9 s += s;
$9 = 768
(gdb) c                                         let's go a bit further ...
Continuing.

Hardware watchpoint 2: i

Old value = 1                                   here i changes from 1 to 0
New value = 0                                   so how did it become 1?
main () at sample2.c:8
8 while(i = 1){                                   that's how!
```

Notice there seems to be a bug in *gdb*. The watchpoint for `i` worked when `i` was changed at line 11, but not at line 8. I'm not sure why it didn't; a similar, more complex, debugger, *lldb*, does catch this.

Example 3

This one is a character counter (with apologies to Kernighan and Ritchie, from whose book it was mangled):

```
1   #include <stdio.h>
2
3   main() /* counts digits, white space, others */
4   {
5       int c, i, num_digits[10], num_white, num_other;
6       num_white = num_other = 0;
7       for(i = 0; i <= 10; i++) /* initialize */
8           num_digits[i] = 0;
9       while(c = getchar() != EOF){
10          switch(c){
11          case '0': case '1': case '2': case '3':
12          case '4': case '5': case '6': case '7':
13          case '8': case '9':
14              num_digits[c - '0']++;
15              break;
16          case ' ': case '\t': case '\n':
17              num_white++;
18              break;
19          default:
20             num_other++;
21              break;
22          }
23      }
24      printf("digits =");
25      for(i = 0; i <= 10; i++)
26          printf(" %d", num_digits[i]);
27      printf(", white space = %d, other = %d\n",
28                                  num_white, num_other);
29
```

```
30      return(0);
31  }
```

When I give it the following input:

```
7 4 88 6 6  3 4 7 87 8 2 34
  5 7  3 21 123q52
```

I get the following output:

```
digits = 0 0 0 0 0 0 0 0 0 0 1090144256, white space = 0, other = 47
```

There are two clear errors here. First, notice the huge number, 1090144256; then, when you count the number of numbers after "`digits =` ", it's 11 when it should be 10 (one number for each of the digits 0, ... 9). Second, notice all the other numbers are 0 except for the "other", which is 47.

So we will run *gdb*, and look first at the output. We'll put a breakpoint at line 26 and look at the array each time through. That will not only let us see how `i` changes, and how the values in the array change. But we will deal with the first before the second.

```
$ gdb sample3
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sample3...done.
(gdb) l 1,31                                    list the program
1 #include <stdio.h>
2
3 int main(void) /* counts digits, white space, others */
4
5 int c, i, num_digits[10], num_white, num_other;
6 num_white = num_other = 0;
7 for(i = 0; i < 10; i++) /* initialize */
8 num_digits[i] = 0;
9 while(c = getchar() != EOF)
10 switch(c)
11 case '0': case '1': case '2': case '3':
12 case '4': case '5': case '6': case '7':
13 case '8': case '9':
14 num_digits[c - '0']++;
15 break;
16 case ' ': case '\t': case '\n':
17 num_white++;
18 break;
19 default:
20 num_other++;
```

```
21 break;
22
23
24 printf("digits =");
25 for(i = 0; i <= 10; i++)
26 printf(" %d", num_digits[i]);
27 printf(", white space = %d, other = %d\n",
28 num_white, num_other);
29
30 return(0);
31
(gdb) b 26                                          put the breakpoint at line 26
Breakpoint 1 at 0x7c2: file sample3.c, line 26.
(gdb) cond 1 i>=9                                   stop when i is 9
(gdb) comm 1                                        every time you stop ...
Type commands for breakpoint(s) 1, one per line.    the array num_digits
End with a line saying just "end".
>p i                                               print the values of i and the
>p num_digits                                      the elements of the array num_digits
>end
(gdb) run < sample.data
Starting program: /home/bishop/ecs36a/20/sample3 < sample.data

Breakpoint 1, main () at sample3.c:26
26 printf(" %d", num_digits[i]);
$1 = 9                                              this is right
$2 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0                   but this isn't
(gdb) n                                             execute the next statement
25 for(i = 0; i <= 10; i++)
(gdb) n                                             execute the next statement

Breakpoint 1, main () at sample3.c:26
26 printf(" %d", num_digits[i]);
$3 = 10                                            and this is wrong, as the loop should
$4 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0                  have ended; check the condition
(gdb)                                              in the for loop, and that's one bug
```

Going beyond the end of the array `num_digits` would explain the big number, as that part of memory probably contains miscellaneous or old data that prints as a large integer.

So now we fix that bug. We run the program again and get:

```
digits = 0 0 0 0 0 0 0 0 0 0, white space = 0, other = 47
```

Back to *gdb*! This time, we focus on like 14, where we should increment the appropriate element of the array `num_digits`:

```
$ gdb sample3
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
```

```
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sample3...done.
(gdb) b 14                                          add a breakpoint at line 14
Breakpoint 1 at 0x76e: file sample3.c, line 14.
(gdb) run < sample.data                             run it!
Starting program: /home/bishop/ecs36a/20/sample3 < sample.data
digits = 0 0 0 0 0 0 0 0 0 0, white space = 0, other = 47
[Inferior 1 (process 5667) exited normally]
           the program ran to completion, so it never reached line 14!
           so we need to check what is being read --- this happens at line 10
(gdb) b 10                                          add a breakpoint at line 10
Breakpoint 2 at 0x555555554740: file sample3.c, line 10.
(gdb) run < sample.data                             run it again
Starting program: /home/bishop/ecs36a/20/sample3 < sample.data

Breakpoint 2, main () at sample3.c:10
10 switch(c){
(gdb) p c                                           print the value in c
$1 = 1                                              odd --- 1 is control-A, which
                                                    isn't in the file
(gdb) c                                             let's keep going ...
Continuing.

Breakpoint 2, main () at sample3.c:10
10 switch(c){
(gdb) p c                                           print the value in c
$2 = 1                                              it's fixed at 1, so check where
                                                    where it is read
(gdb) n                                             execute the next statement
20 num_other++;                                     as expected
(gdb) n                                             leave the switch statement
21 break;
(gdb) n                                             and here is where c is read
9 while(c = getchar() != EOF){
```

We check our table of precedence and associativity of C operators. In it, we see the "!=" operator has higher precedence than the assignment operator. So this reads a character; if it is not EOF, `c` is set to 1; otherwise, it's set to 0. And that's our bug.

When we fix it and run the program, the output is

```
digits = 0 2 4 4 3 2 2 4 4 0, white space = 21, other = 1
```

which is right.

Example 4

This program is supposed to compute and print $n!$:

```
1   #include <stdio.h>
2
3   int nfact(int n)
4   {
```

```
5       int x;
6
7       x = nfact(n-1);
8
9       return(n * x);
10  }
11
12  int main(int argc, char *argv[])
13  {
14    int n;
15
16    if (argc != 2){
17        fprintf(stderr, "Usage: %s [ number ]\n", argv[0]);
18        return(1);
19    }
20
21    printf("%d! = %d\n", n, nfact(n));
22  }
```

but when it runs, we get this:

```
Segmentation fault (core dumped)
```

If you look in the directory, there is a file called *core*. That's a memory dump of the memory of *nfact*. So let's being it into *gdb*:

```
$ gdb nfact core
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from nfact...done.
[New LWP 7884]
Core was generated by `./nfact 4'.                          here gdb says what caused
Program terminated with signal SIGSEGV, Segmentation fault.  the crash; it tried to
#0  0x000055a96b7906dd in nfact (n=-174675) at nfact.c:7     access memory outside its
7 x = nfact(n-1);                                           address space; it happened
                                                            at line 7
(gdb) bt 10                                                 this is "backtrace" and
                                                            prints the last 10 function
                                                            calls and function parameters
#0  0x000055a96b7906dd in nfact (n=-174675) at nfact.c:7
#1  0x000055a96b7906e2 in nfact (n=-174674) at nfact.c:7
#2  0x000055a96b7906e2 in nfact (n=-174673) at nfact.c:7
#3  0x000055a96b7906e2 in nfact (n=-174672) at nfact.c:7
#4  0x000055a96b7906e2 in nfact (n=-174671) at nfact.c:7
```

```
#5  0x000055a96b7906e2 in nfact (n=-174670) at nfact.c:7
#6  0x000055a96b7906e2 in nfact (n=-174669) at nfact.c:7
#7  0x000055a96b7906e2 in nfact (n=-174668) at nfact.c:7
#8  0x000055a96b7906e2 in nfact (n=-174667) at nfact.c:7
#9  0x000055a96b7906e2 in nfact (n=-174666) at nfact.c:7
(More stack frames follow...)
```

Clearly, the recursion is going on too deeply — look at the values of the parameters, which are increasing negative numbers. So we look for a missing base case, which if present would have stopped the execution. So we add it, and get:

```
4! = 24
```