Debugging

- We can also have multi-dimensional arrays in C e.g.

  ```
  int matrix[2][3] = {{1,2,3},{4,5,6}};
  ```

- Now `matrix[0][1]==2`
- Can have more than 2-dimensional arrays:

  ```
  int arr3d[3][2][4] = {
      {{1, 2, 3, 4}, {5, 6, 7, 8}},
      {{9, 10, 11, 12}, {13, 14, 15, 16}},
      {{17, 18, 19, 20}, {21, 22, 23, 24}}
  };
  ```

- The elements of `arr3d` will be allocated in memory in the order `arr3d[0][0][0]`, `arr3d[0][0][1]`, `arr3d[0][0][2]`, `arr3d[0][0][3]`, `arr3d[0][1][0]`, `arr3d[0][1][1]` etc.

```
int arr3d[3][2][4] = {
    {{1, 2, 3, 4}, {5, 6, 7, 8}},
    {{9, 10, 11, 12}, {13, 14, 15, 16}},
    {{17, 18, 19, 20}, {21, 22, 23, 24}}
};
```

- What is the type of `arr3d[0][0][0]`?
- What is the type of `arr3d[0][0]`?
- What is the type of `arr3d[0]`?
- What is the type of `arr3d`?
- What does `int (*p)[2][4]=arr3d;` do?
- For further fun with pointers and arrays, read `https://www.oreilly.com/library/view/understanding-and-using/9781449344535/ch04.html`

1 Debugging C Code

- We will use the powerful command-line debugger `gdb` (The GNU Debugger)
  - Many graphical debuggers use `gdb` as a backend.
- Can be used to debug various languages e.g. C, C++, Fortran, Go, Rust
- Online manual: `http://sourceware.org/gdb/current/onlinedocs/gdb/`
- It has its own interactive shell: it can recall history with the arrow keys, aut-complete words with the TAB key etc.

2 Debugging C Code

- To add debugging data to a compiled program, we use the `-g` option e.g.

  ```
  gcc -Wall -Wextra -std=c11 -pedantic -o main main.c
  changes to
  gcc -g -Wall -Wextra -std=c11 -pedantic -o main main.c
  ```

- We can still run the resulting program as before e.g.

  ```
  ./main
  ```

- To start the debugger we run e.g.

  ```
  gdb ./main
  ```

- or we can specify the program after starting `gdb`:

  ```
  gdb
  ...
  (gdb) file main
  ```

- use the `quit` command to exit

  ```
  (gdb) quit
  ```

```
gdb ./main
GNU gdb (Debian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 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 ./main...done.
(gdb)
```

```c
#include<stdio.h>
void print_even_total(int *b, int max_index);
int main(){
        int a[5]={1,2,3,4,5};
        print_even_total(a,5);
        return 0;
}

/* Sum up every other number in the array pointed *
 * to by b, up to, but not including b[max_index] */
void print_even_total(int *b, int max_index) {
        int c;
        for(int i=0;i!=max_index;i++) {
                c+=b[i];
                i++;
        }
        printf("%d\n",c);
}
```

3 Running the program

- To run your program use the `run` command (or the shorthand `r`)

  ```
  (gdb) run
  ```

- The program should run in the same way as if you were running it outside `gdb`. If runs fine normally (e.g. without getting a segmentation fault), then it should run fine here too (if slower).
- If the program does have issues, you'll get some useful information like the line number where it crashed and the parameters to the function that caused the error:

```
Starting program: /home/homeblue01/6/kmpb26/gdb/segfault

Program received signal SIGSEGV, Segmentation fault.
0x000055555555471f in print_even_total (b=0x7fffffffe990,
        max_index=5) at segfault.c:14
14                      c+=b[i];
```

- The `backtrace` command lets us see the stack trace of what functions called the current function

```
(gdb) backtrace
#0 0x000055555555471f in print_even_total (b=0x7fffffffe990,
        max_index=5) at segfault.c:14
#1 0x00005555555546ec in main () at segfault.c:5
```

- The `list` command lets us see the lines in the source code around where we are

```
(gdb) list
9     /* Sum up every other number in the array pointed *
10     * to by b, up to, but not including b[max_index] */
11    void print_even_total(int *b, int max_index) {
12            int c;
13            for(int i=0;i!=max_index;i++) {
14                    c+=b[i];
15                    i++;
16            }
17            printf("%d\n",c);
18    }
```

- The `print` command lets us see the value of a variable

```
(gdb) print b[i]
Cannot access memory at address 0x7ffffffff000
(gdb) print i
$1 = 416
(gdb) print b
$2 = (int *) 0x7fffffffe980
```

- The `print/x` command lets us see the value of a variable in hexadecimal

```
(gdb) print/x i
$3 = 0x1a0
```

- Note: if there are multiple variables with the same name, the one that gets printed out is whichever one is currently in scope!

4 Setting Breakpoints

- Breakpoints can be used to stop the run of the program part-way through the code. We can set a breakpoint using the `break` command.

```
(gdb) break 3
Breakpoint 1 at 0x6b8: file segfault.c, line 3.
```

- This sets a breakpoint on line 3 of the source code. We can also specify the source code file:

```
(gdb) break segfault.c:5
Breakpoint 2 at 0x7b8: file segfault.c, line 5.
```

- You can also tell gdb to break whenever a function gets run

```
(gdb) break print_even_total
Breakpoint 3 at 0x555555554702: file segfault.c,
        line 13.
```

- You can set as many breakpoints as you want. The program will stop running when you reach any of them.
- You can get a list of currently-set breakpoints with `info breakpoints`

```
(gdb) info breakpoints
Num     Type           Disp Enb Address
    What
1       breakpoint     keep y   0x00005555555546b8
    in main at segfault.c:3
        breakpoint already hit 1 time
2       breakpoint     keep y   0x00005555555547b8
    in main at segfault.c:5
3       breakpoint     keep y   0x0000555555554702
    in print_even_total at segfault.c:13
```

- and delete one with `delete`

```
(gdb) delete 1
```

- After reaching a breakpoint, you can use `continue` to continue running the code until the next breakpoint (or `run` to start again from scratch).

```
(gdb) run
Starting program: /home/homeblue01/6/kmpb26/gdb/segfault

Breakpoint 1, main () at segfault.c:4
4               int a[5]={1,2,3,4,5};
(gdb) continue
Continuing.
21854
[Inferior 1 (process 23899) exited normally]
```

- The `finish` command will run until the end of the current function.

5 More complicated breakpoints

- You can also set breakpoints that will be triggered only if certain conditions occur

```
(gdb) break segfault2.c:14 if i>=3
```

- You can also set a temporary breakpoint with `tbreak`. This works like a normal breakpoint, but gets removed after the first time you hit it.

6 `step` and `next`

- The `step` command lets you step through your code one line at a time.

```
(gdb) step
```

- `next` works the same way, but treats function calls as single instructions, rather than going into each function that is called and stepping through it line-by-line.

```
(gdb) next
```

- You may end up doing this repeatedly. To save you from typing, most commands have an abbreviated form e.g. `s` for `step`. Also, if you press ENTER without a command, gdb will just repeat your last command again.

7 Watchpoints

- If a variable is in scope, you can use the `watch` command to be told whenever its value changes (the watchpoint gets deleted when the variable goes out of scope).

```
(gdb) watch i
Hardware watchpoint 3: i
(gdb) cont
Continuing.

Hardware watchpoint 3: i

Old value = 4
New value = 5

(gdb) cont
Continuing.

Watchpoint 3 deleted because the program has left the
block in which its expression is valid.
```

8 Calling functions and setting variables

- You can also call functions in your program from inside `gdb`

```
(gdb) call print_even_total(a,4)
-1431630641
```

- And you can modify variables too!

```
(gdb) print i
$2 = 0
(gdb) set variable i=20
(gdb) print i
$3 = 20
```

9 Attaching to a running process

```
#include<stdio.h>
int main(){
  int b=0;
  for(int a=0;;a++){
    b+=a;
    printf("%d\n",b);
  }
}
```

- You can also attach to an already-running program:

```
pgrep infinite_loop
2718
gdb attach 2718
```

10 Command-line options

- We can supply command-line options to a program e.g.

```
./main -some-option
```

- To do this when running under `gdb` we can run e.g.

```
gdb --args ./main -some-option
```

- In a running `gdb` session, we can list the current command-line arguments with

  ```
  show args
  ```

- and we can change them with

  ```
  set args -some-other-option
  ```