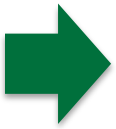
# CS 50:
# Software Design and Implementation

gdb: the GNU debugger

# Agenda



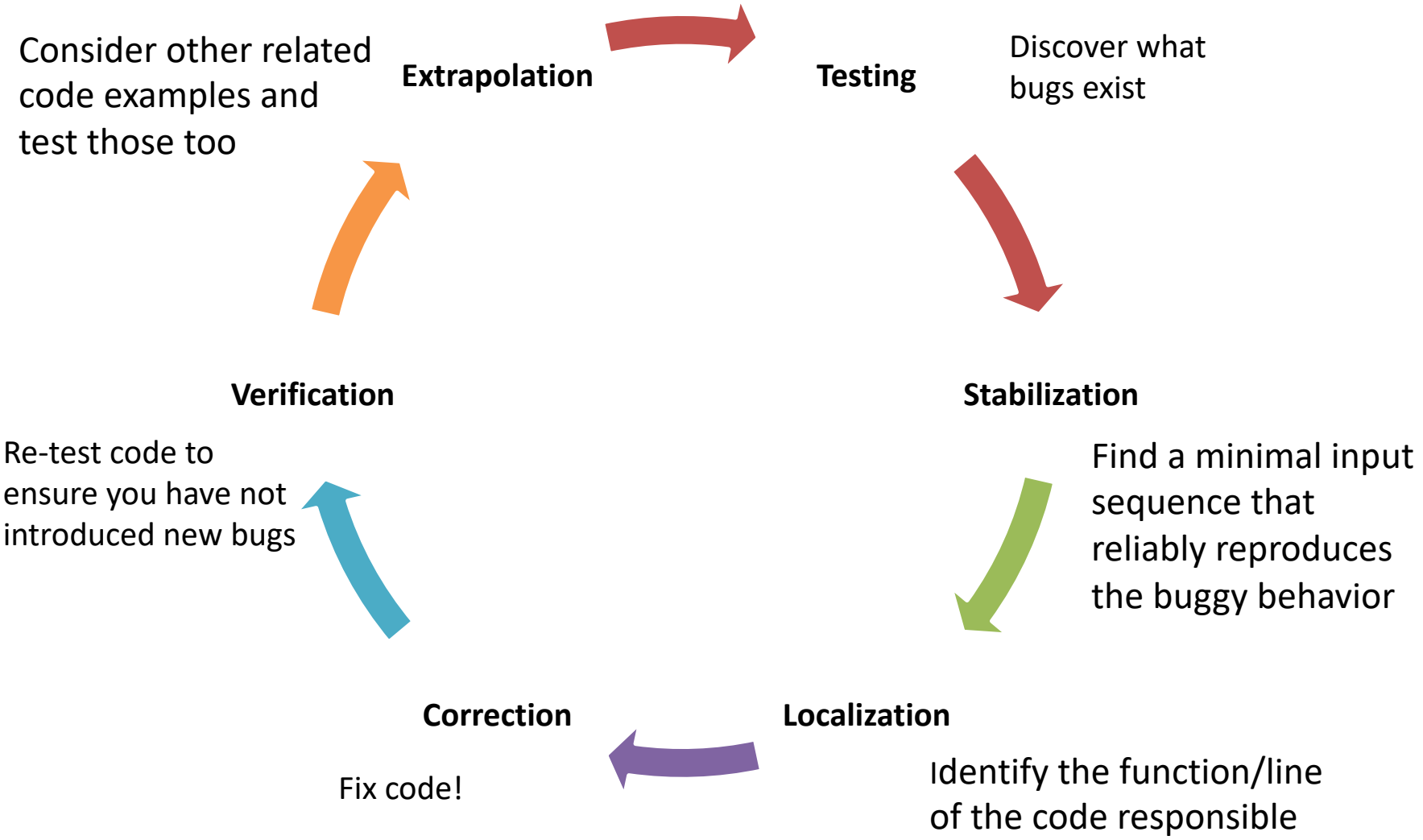
1. Debugging
2. The GNU debugger: gdb
3. Activity

# We haven't figured out to completely prevent bugs, but some things can help

**Things that can help prevent bugs**

- A good design and design methodology
- Consistent style (e.g., use C program idioms as much as possible)
- Boundary conditions/edge case tests
- Assertions and sanity testing
- Defensive programming
- Designing for testing
- Avoid files that have a large number of functions, and functions that have a large number of lines; aim for functions that do one thing, and do it well!
- Limit use of global variables whenever possible

# A bug lifecycle methodology can help identify and fix issues



# Can't I just use a bunch of `printf` statements (with conditional compilation)?

**Works, but…**

- **can lead to ugly, unreadable code**
- **If you take this approach, even a little, use discipline to enable/disable such messages with a single switch**

**printf_example.c**

```
1  #include <stdio.h>
2
3  int main() {
4     int a, b, c;
5
6     #ifdef DEBUG
7        printf("Testing output:\n");
8        printf("\ta = %d\n\tb = %d\n\tc = %d\n",a, b, c);
9        fflush(stdout);
10    #endif
11    a = 3; b = 10; c = -1;
12    #ifdef DEBUG
13       printf("After initialization:\n");
14       printf("\ta = %d\n\tb = %d\n\tc = %d\n",a, b, c);
15       fflush(stdout);
16    #endif
17
18    printf("<running some code now...>\n");
19
20    return 0;
21 }
```

**If you use `printf`, consider following with `fflush(stdout)` immediately**

**Debug not set, so testing code not sent for compilation**

**No debugging output**

```
$ mygcc printf_example.c
<unused variable warnings>
$ ./a.out
<running some code now...>
$ mygcc printf_example.c -DDEBUG
<unused variable warnings>
$ ./a.out
Testing output:
a = 32765
b = 0
c = 0
After initialization:
a = 3
b = 10
c = -1
<running some code now...>
```

**Debug *is* set, so testing code *is* sent for compilation**

**Debugging output shown**

# gdb is a better solution than printf statements

```
 1 #include <stdio.h>
 2
 3 int main() {
 4     printf("Did I fail 1\n");
 5     printf("Did I fail 2\n");
 6     printf("Did I fail 3\n");
 7     printf("Did I fail 4\n");
 8     printf("Did I fail 5\n");
 9     printf("Did I fail 6\n");
10     printf("Did I fail 7\n");
11 }
```

# Agenda

1. Debugging
2. The GNU debugger: gdb
3. Activity

# gdb can help fix (purposely) buggy code

**bugsort.c**

```
12 #include <stdio.h>
13 #include <stdlib.h>
14
15
16 /* ****************** main ***************** */
17 int
18 main()
19 {
20   const int numSlots = 10;  // number of slots in array
21   int sorted[numSlots];   // the array of items
22
23   /* fill the array with numbers */
24   for (int n = 0; n < numSlots; n++) {
25     int item;     // a new item
26     scanf("%d", &item);   // read a new item
27     for (int i = n; i > 0; i--) {
28       if (sorted[i] > item) {
29         sorted[i+1] = sorted[i]; // bump it up to make room
30       } else {
31         sorted[i] = item; // drop the new item here
32       }
33     }
34   }
35
36   /* print the numbers */
37   for (int n = 0; n < numSlots; n++) {
38     printf("%d ", sorted[n]);
39   }
40   putchar('\n');
41 }
```

```
$ vi bugsort.c
$ mygcc -o sort bugsort.c
$ ./sort
1
0
2
9
3
8
4
7
5
6
0 9 9 9 9 9 9 9 9 9
$ echo 1 2 3 9 8 7 6 15 12 24 | ./sort
0 24 24 24 24 24 24 24 24 1365200872
```

**Code does not behave as expected!**

**Code meant to read 10 integers and output them in sorted order (for now we won't worry about whether the input was an integer or that we actually got 10 integers, this is just to demonstrate gdb)**

# Start debugger on executable, set breakpoints with break or b

**bugsort.c**

```
12 #include <stdio.h>
13 #include <stdlib.h>
14
15
16 /* ****************** main ***********
17 int
18 main()
19 {
20   const int numSlots = 10;  // number o
21   int sorted[numSlots];    // the array
22
23   /* fill the array with numbers */
24   for (int n = 0; n < numSlots; n++) {
25     int item;      // a new item
26     scanf("%d", &item);    // read a new
27     for (int i = n; i > 0; i--) {
28       if (sorted[i] > item) {
29         sorted[i+1] = sorted[i]; // bump
30       } else {
31         sorted[i] = item; // drop the n
32       }
33     }
34   }
35
36   /* print the numbers */
37   for (int n = 0; n < numSlots; n++) {
38     printf("%d ", sorted[n]);
39   }
40   putchar('\n');
41 }
```

**Make sure code compiled with –ggdb flag (mygcc does)**
**Flag gives debugging info to gdb**

```
$ gdb sort
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
<snip>
Reading symbols from sort...done.
(gdb) break main
Breakpoint 1 at 0x773: file bugsort.c, line 19.
(gdb) list 26
21    int sorted[numSlots];    // the array of items
22
23    /* fill the array with numbers */
24    for (int n = 0; n < numSlots; n++) {
25      int item;      // a new item
26      scanf("%d", &item);    // read a new item
27      for (int i = n; i > 0; i--) {
28        if (sorted[i] > item) {
29          sorted[i+1] = sorted[i]; // bump it up to make room
30        } else {
(gdb) b 27
Breakpoint 2 at 0x817: file bugsort.c, line 27.
(gdb) b 37
Breakpoint 3 at 0x878: file bugsort.c, line 37.
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000000773 in main at bugsort.c:19
2       breakpoint     keep y   0x0000000000000817 in main at bugsort.c:27
3       breakpoint     keep y   0x0000000000000878 in main at bugsort.c:37
```

**Reads symbols generated with –ggdb flag**

**Set breakpoint by giving function name or line number**

**Use break or b (shortcut)**

**Info break lists all break points**
**Enb shows if enabled (all are here)**

**List shows lines before and after line number**
**List by itself (no line number) shows lines around current line**

# Use r to run program until first breakpoint

bugsort.c

```
12 #include <stdio.h>
13 #include <stdlib.h>
14
15
16 /* ****************** main ***********
17 int
18 main()
19 {
20   const int numSlots = 10;  // number o
21   int sorted[numSlots];    // the array
22
23   /* fill the array with numbers */
24   for (int n = 0; n < numSlots; n++) {
25     int item;      // a new item
26     scanf("%d", &item);    // read a new
27     for (int i = n; i > 0; i--) {
28       if (sorted[i] > item) {
29         sorted[i+1] = sorted[i]; // bump it up to make room
30       } else {
31         sorted[i] = item; // drop the new item here
32       }
33     }
34   }
35
36   /* print the numbers */
37   for (int n = 0; n < numSlots; n++) {
38     printf("%d ", sorted[n]);
39   }
40   putchar('\n');
41 }
```

Disable breakpoint 2

```
(gdb) disable 2
(gdb) i break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000000773 in main at bugsort.c:19
2       breakpoint     keep n   0x0000000000000817 in main at bugsort.c:27
3       breakpoint     keep y   0x0000000000000878 in main at bugsort.c:37
(gdb) run
Starting program: /thayerfs/home/d84607y/cs50/activities/day14/sort


Breakpoint 1, main () at bugsort.c:19
19 {
```

Start program running
Runs until breakpoint

Currently at line 19

# Step (or just s) runs a line and moves to the next line, stepping into functions

**bugsort.c**

```
12 #include <stdio.h>
13 #include <stdlib.h>
14
15
16 /* ****************** main ***********
17 int
18 main()
19 {
20   const int numSlots = 10;  // number o
21   int sorted[numSlots];    // the array
22
23   /* fill the array with numbers */
24   for (int n = 0; n < numSlots; n++) {
25     int item;      // a new item
26     scanf("%d", &item);    // read a new
27     for (int i = n; i > 0; i--) {
28       if (sorted[i] > item) {
29         sorted[i+1] = sorted[i]; // bump
30       } else {
31         sorted[i] = item; // drop the ne
32       }
33     }
34   }
35
36   /* print the numbers */
37   for (int n = 0; n < numSlots; n++) {
38     printf("%d ", sorted[n]);
39   }
40   putchar('\n');
41 }
```

```
(gdb) step
20   const int numSlots = 10;  // number of slots in array
(gdb) s
21   int sorted[numSlots];   // the array of items
(gdb) s
24   for (int n = 0; n < numSlots; n++) {
(gdb)
26     scanf("%d", &item);   // read a new item
(gdb)
__isoc99_scanf (format=0x555555554964 "%d") at
isoc99_scanf.c:27
27 isoc99_scanf.c: No such file or directory.
(gdb) finish
Run till exit from #0  __isoc99_scanf (format=0x555555554964
"%d") at isoc99_scanf.c:27
```

Step (or s) runs one line

Enter runs the last command (s here)

Oops, step goes into functions called Here we go into `scanf.c` code

Finish runs to the end of current function and returns to debugging when back at caller

Note: execution for this slide started at line 20 (arrow 1) and is currently at line 26 (arrow 2) before stepping into `scanf.c`

# `print` (or `p`) prints variable values, use `info locals` to see all local variables

bugsort.c

```
12 #include <stdio.h>
13 #include <stdlib.h>
14
15
16 /* ****************** main ***********
17 int
18 main()
19 {
20   const int numSlots = 10;  // number o
21   int sorted[numSlots];   // the array
22
23   /* fill the array with numbers */
24   for (int n = 0; n < numSlots; n++) {
25     int item;     // a new item
26     scanf("%d", &item);   // read a new
27     for (int i = n; i > 0; i--) {
28       if (sorted[i] > item) {
29         sorted[i+1] = sorted[i]; // bump
30       } else {
31         sorted[i] = item; // drop the n
32       }
33     }
34   }
35
36   /* print the numbers */
37   for (int n = 0; n < numSlots; n++) {
38     printf("%d ", sorted[n]);
39   }
40   putchar('\n');
41 }
```

I entered a number of values for `scanf`

```
9 8 7 6 5 4 3 2 1 0
main () at bugsort.c:27
27      for (int i = n; i > 0; i--) {
Value returned is $1 = 1
(gdb) print item
$2 = 9
(gdb) p &item
$3 = (int *) 0x7fffffffe4a4
(gdb) info locals
i = 21845
item = 9
n = 0
numSlots = 10
sorted = {0, 0, 0, 0, 9, 0, -136489376, 32767, -6920, 32767}
```

After I hit enter, gdb moves on to next line of code

`print` (or `p`) prints a variable's value

`p &item` prints `item`'s address

`info locals` prints the values of all local variables (handy!)

# next (or n) steps over function calls

**bugsort.c**

```
12 #include <stdio.h>
13 #include <stdlib.h>
14
15
16 /* ****************** main ***********
17 int
18 main()
19 {
20   const int numSlots = 10;  // number o
21   int sorted[numSlots];   // the array
22
23   /* fill the array with numbers */
24   for (int n = 0; n < numSlots; n++) {
25     int item;     // a new item
26     scanf("%d", &item);   // read a new
27     for (int i = n; i > 0; i--) {
28       if (sorted[i] > item) {
29         sorted[i+1] = sorted[i]; // bump it up to make room
30       } else {
31         sorted[i] = item; // drop the new item here
32       }
33     }
34   }
35
36   /* print the numbers */
37   for (int n = 0; n < numSlots; n++) {
38     printf("%d ", sorted[n]);
39   }
40   putchar('\n');
41 }
```

3 1 2

Next (or n) "steps over" functions (does not "step into" them like step)

```
(gdb) next
24   for (int n = 0; n < numSlots; n++) {
(gdb) n
26     scanf("%d", &item);   // read a new item
(gdb) n
27     for (int i = n; i > 0; i--) {
(gdb) p item
$4 = 8
(gdb) p n
$5 = 1
```

Next runs `scanf` code but picks up debugging after `scanf` returns

`scanf` reads the next integer from the string we previously typed

In this case `scanf` reads 8 (we previously typed 9…0)

Note: `for` loop is never entered (`i=n=0`) so after hitting end of the `for` from line 27 at line 33 (indicated by arrow 2), execution loops back to line 24 (arrow 3)

# gdb shows the line to be run, before it runs

**bugsort.c**

```
12 #include <stdio.h>
13 #include <stdlib.h>
14
15
16 /* ****************** main ***********
17 int
18 main()
19 {
20   const int numSlots = 10;  // number o
21   int sorted[numSlots];   // the array
22
23   /* fill the array with numbers */
24   for (int n = 0; n < numSlots; n++) {
25     int item;     // a new item
26     scanf("%d", &item);   // read a new
27     for (int i = n; i > 0; i--) {
28       if (sorted[i] > item) {
29         sorted[i+1] = sorted[i]; // bump
30       } else {
31         sorted[i] = item; // drop the ne
32       }
33     }
34   }
35
36   /* print the numbers */
37   for (int n = 0; n < numSlots; n++) {
38     printf("%d ", sorted[n]);
39   }
40   putchar('\n');
41 }
```

3 1 2

**Just pressing Enter runs the previous command (n here)**

```
(gdb) n
28        if (sorted[i] > item) {
(gdb)
31          sorted[i] = item; // drop the new item here
(gdb)
27      for (int i = n; i > 0; i--) {
(gdb)
24    for (int n = 0; n < numSlots; n++) {
(gdb)
26      scanf("%d", &item);   // read a new item
(gdb) p item
$6 = 8
(gdb) n
27      for (int i = n; i > 0; i--) {
(gdb) p item
$7 = 7
(gdb) p sorted[0]
$8 = 0
(gdb) p sorted[1]
$9 = 8
(gdb) i locals
i = 0
item = 7
n = 2
numSlots = 10
sorted = {0, 8, 0, 0, 9, 0, -136489376, 32767, -6920, 32767}
```

**gdb shows the command *before* it runs (e.g., `item` doesn't change from 7 to 8 until after `scanf` line is run with `next`)**

**`i` (short for info) `locals` shows values of all local variables**

# gdb shows the line to be run, before it runs

```
12 #include <stdio.h>
13 #include <stdlib.h>
14
15
16 /* ****************** main ***********
17 int
18 main()
19 {
20   const int numSlots = 10;  // number o
21   int sorted[numSlots];   // the array
22
23   /* fill the array with numbers */
24   for (int n = 0; n < numSlots; n++) {
25     int item;     // a new item
26     scanf("%d", &item);   // read a new
27     for (int i = n; i > 0; i--) {
28       if (sorted[i] > item) {
29         sorted[i+1] = sorted[i]; // bump
30       } else {
31         sorted[i] = item; // drop the n
32       }
33     }
34   }
35
36   /* print the numbers */
37   for (int n = 0; n < numSlots; n++) {
38     printf("%d ", sorted[n]);
39   }
40   putchar('\n');
41 }
```

Use s (step) to illustrate stepping into a function

bugsort.c

```
(gdb) s
28        if (sorted[i] > item) {
(gdb) s
31          sorted[i] = item; // drop the new item here
(gdb) s
27      for (int i = n; i > 0; i--) {
(gdb) s
28        if (sorted[i] > item) {
(gdb) s
29          sorted[i+1] = sorted[i]; // bump it up to make
room
(gdb) s
27      for (int i = n; i > 0; i--) {
(gdb) s
24    for (int n = 0; n < numSlots; n++) {
(gdb) s
26      scanf("%d", &item);   // read a new item
(gdb) s
__isoc99_scanf (format=0x555555554964 "%d") at
isoc99_scanf.c:27
27 isoc99_scanf.c: No such file or directory.
(gdb) backtrace
#0  __isoc99_scanf (format=0x555555554964 "%d") at
isoc99_scanf.c:27
#1  0x0000555555554817 in main () at bugsort.c:26
(gdb) finish
Run till exit from #0  __isoc99_scanf (format=0x555555554964
"%d") at isoc99_scanf.c:27
main () at bugsort.c:27
27      for (int i = n; i > 0; i--) {
Value returned is $10 = 1
```

Back trace (bt) shows the call stack

scanf.c is called by main()

Finish runs current function and returns to debugging at caller

# Continue (or c) restarts program running to next breakpoint

```
12 #include <stdio.h>
13 #include <stdlib.h>
14
15
16 /* ****************** main ***********
17 int
18 main()
19 {
20   const int numSlots = 10;  // number o
21   int sorted[numSlots];   // the array
22
23   /* fill the array with numbers */
24   for (int n = 0; n < numSlots; n++) {
25     int item;     // a new item
26     scanf("%d", &item);   // read a new
27     for (int i = n; i > 0; i--) {
28       if (sorted[i] > item) {
29         sorted[i+1] = sorted[i]; // bump
30       } else {
31         sorted[i] = item; // drop the n
32       }
33     }
34   }
35
36   /* print the numbers */
37   for (int n = 0; n < numSlots; n++) {
38     printf("%d ", sorted[n]);
39   }
40   putchar('\n');
41 }
```

**Run to next breakpoint**

**bugsort.c**

```
(gdb) continue
Continuing.
10


Breakpoint 3, main () at bugsort.c:37
37   for (int n = 0; n < numSlots; n++) {
(gdb) continue
Continuing.
0 10 10 10 10 10 10 10 10 10
[Inferior 1 (process 4317) exited normally]
```

# gdb cheat sheet

| command | shortcut | purpose |
|---|---|---|
| run [arglist] | r | Start your program running (with arglist, if specified) |
| break [file:]function | b | Set a breakpoint at function (in file) |
| commands NN | | A list of commands to run every time breakpoint #NN is reached |
| list [file:]function | l | Type the text of the program in the vicinity of where it is presently stopped |
| backtrace | bt | Display the program call stack |
| frame [args] | f | The frame command allows you to move from one stack frame to another, and to print the stack frame you select. args may be either the address of the frame or the stack frame number. Without an argument, frame prints the current stack frame |
| print expr | p | Display the value of an expression |
| continue | c | Continue running your program (after stopping, e.g. at a breakpoint) |
| next | n | Execute next program line (after stopping); step over any function calls in the line |
| step | s | Execute next program line (after stopping); step into any function calls in the line |
| info break | i b | List breakpoints |
| disable N | | Disable breakpoint N where N shown in info break |
| enable N | | Enable breakpoint N where N shown in info break |
| delete N | | Remove breakpoint N where N shown in info break |
| info locals | i loc | Print the values of all local variables |
| help [name] | h | Show information about GDB command name, or general information about using GDB |
| quit | q | Exit from GDB |

# Agenda

1. Debugging
2. The GNU debugger: gdb
3. Activity