Debugging

CS2023 Winter 2004

Outcomes: Debugging

- *Practice of Programming*, Chapter 5, on reserve in the library
- After the conclusion of this section you should be able to
  - Use simple techniques to find bugs
  - Know how to approach “hard” bugs
  - Use the gdb debugger effectively

Debugging

- Finding the error and fixing it
  - finding and understanding the error is 90% of the work
  - know the typical errors of the language
- Debuggers
  - best use is to find out the state of program when it fails: where it failed and values of variables
  - can easily waste time stepping through a program blindly

Simple Techniques

- **Look for familiar patterns**

```
int n;
scanf("%d", n);

int n;
scanf("%d", &n);
```

Simple Techniques

- **Look for familiar patterns**

```
int n=1;
double d = PI;
printf("%d %f\n", d, n);
```

```
-266631570 0.000000
```

  - Other common errors:
    - using **`%f`** instead of **`%lf`** when reading a double with **`scanf`**
    - not initializing local variables

Simple Techniques

- Many of these common errors caught by enabling all warnings of compiler (**`-Wall`**)

```
printferr.c: In function `main':
printferr.c:8: warning: int format, double arg
  (arg 2)
printferr.c:8: warning: double format,
  different type arg (arg 3)
```

- Don't ignore compiler warnings!

Simple Techniques

- Examine the most recent change
  - Bug most likely in new code or has been exposed by it
  - Use source control!

Simple Techniques

- Don't make same mistake twice

```
for (i = 1; i < argc; i++) {
  if(argv[i][0] != '-')/* options finished*/
    break;
  switch (argv[i][1]) {
  case '0':             /* output filename */
    outname = argv[i];
    break;
  case 'f':
    from = atoi(argv[i]);
    break;
  case 't':
    to = atoi(argv[i]);
    break;
  ...
```

Simple Techniques

– output file name always had **`-o`** attached to it

```
outname = &argv[i][2];
```

– argument like **`-f123`** converted to zero

```
from = atoi(&argv[i][2]);
```

– Same error occurs again!

Simple Techniques

- Debug now, not later
- Use debugger to get a stack trace and print values of variables

```
#0  0x4205c84e in _IO_vfscanf_internal () from
  /lib/i686/libc.so.6
#1  0x4206061e in scanf () from /lib/i686/libc.so.6
#2  0x08048469 in main () at primefind.c:23
#3  0x42017589 in __libc_start_main () from
  /lib/i686/libc.so.6
```

  - error is at line 23 when `scanf` called

Simple Techniques

- Read before typing
  - set a time limit on inspection with debugger
  - print listing of critical part of program on paper
  - encourages more time for reflection
- Explain your code to someone else
  - even a teddy bear!

Hard Bugs

- How to find a bug:
  1. Stabilize the error
  2. Locate the source of the error
  3. Fix the error
- Stabilize the error: make it reproducible
  - Often due to lack of initialization of local variables
  - If problem is strange and unpredictable, chance is it is due to using unitialized pointers or pointers that point to memory that has been deallocated

Example

- Program calculates and lists tax withholdings for employees in alphabetical order. When program run initially:

```
Formatting, Fred Freeform    $5,877
Goto, Gary                   $1,666
Modula, Mildred              $10,788
Many-Loop, Mavis             $8,889
Statement, Sue Switch        $4,000
Whileloop, Wendy             $7,860
```

Example

- When program run a second time:

```
Formatting, Fred Freeform     $5,877
Goto, Gary                    $1,666
Many-Loop, Mavis              $8,889
Modula, Mildred              $10,788
Statement, Sue Switch         $4,000
Whileloop, Wendy              $7,860
```

- Enter another employee (**`Fruit-Loop, Frita`**), and she shows up in an incorrect position
  - Remember that **`Modula, Mildred`** had just been entered before she showed up in wrong spot

Example

- Hypothesis: problem has something to do with entering a single new employee. Run program again:

```
Formatting, Fred Freeform     $5,877
Fruit-Loop, Frita             $5,771
Goto, Gary                    $1,666
Many-Loop, Mavis              $8,889
Modula, Mildred              $10,788
Statement, Sue Switch         $4,000
Whileloop, Wendy              $7,860
```

- Hypothesis confirmed, but need to test further

Example

- Add another single employee, Hardcase, Henry

```
Formatting, Fred Freeform    $5,877
Fruit-Loop, Frita            $5,771
Goto, Gary                   $1,666
Hardcase, Henry              $  493
Many-Loop, Mavis             $8,889
Modula, Mildred             $10,788
Statement, Sue Switch        $4,000
Whileloop, Wendy             $7,860
```

- More complicated than simply when new employee added!

Example

- Problem arises with names with hyphens
- Examine code:
  - Two different sorting routines used:
    - when employee entered (rough sort)
    - when data is saved
  - problem: data printed before it's sorted
    - rough sort didn't handle punctuation characters

Fixing a Bug

- Understand the problem before you fix it
- Understand the program, not just the problem
- Confirm error diagnosis
- Save original source code before making change
  - Use source control!
- Fix the problem, not the symptom
  - focus on fixing underlying problem, not a bandaid solution
- Check your fix and look for similar errors

Gnu Debugger: gdb

- See *Guide to Faster, Less Frustrating Debugging*, by Norman Matloff (link in course web site)
- Compile all source files with -g option

```
cc -g -o prog prog.c
```

  - produces information for debugger, such as line numbers, variable names, ...
- Works well in emacs
  - shell executing debugger in one half of split-frame, source in the other

Gnu Debugger: gdb

```
gdb prog
```

- Basic operations:

  set breakpoints

  ```
  b main
  ```

  stop at beginning of main

  ```
  b 30
  ```

  stop at line 30 (don't execute it yet)

Gnu Debugger: gdb

run program

```
r
```

print variables

```
p N
```

prints contents of variable **N**

print several variables at a time

```
printf "X = %d, Y = %d\n",X,Y
```

Gnu Debugger: gdb

step through program one line at a time, without stopping inside functions called

```
n
```

step through program one line at a time, and enter any user-defined functions called

```
s
```

display variable each time program pauses

```
disp N
```

print stack trace

```
bt
```

Logging vs. Debugging

- Debugging: tracking program flow and values of variables
- Logging: keeping a record of program activity (in a file, or to stdout or stderr)
  - Can point to problems where a debugger can be used for closer inspection
  - Doesn't require access to source code