Robust Coding and Debugging

CSE 403
Lecture 26

Summer reading

Don’t do this

- Don’t use assertions to check unusual conditions
- You need explicit error code for this
- Only use them to ensure that illegal conditions are avoided

Memory

- The memcpy examples are from Writing Solid Code: Microsoft’s Techniques for Developing Bug-Free C Programs
- Although the book is general, lots of the guidelines focus on memory issues
  - Marking freed memory
  - Not accessing freed memory
  - Dealing with details of realloc
- These are real issues, but appear less frequently in other languages

Writing solid code

- Shred your garbage
  ```c
  void FreeMemory(void *pv) {
    Assert(pv != NULL);
    memset(pv, 0xA3, sizeofBlock(pv);
    free(pv);
  }
  ```
  - Force early failure, increase determinism
  - Why 0xA3?
Should debug code be left in shipped version

Pro:
- Debug code useful for maintenance
- Removing debug code change behavior
  - Bugs in release but not debug versions

Con:
- Efficiency issues
- Different behavior for debug vs. release
  - Early fail vs. recover

Step through your code

Maguire
- Step through new code in the debugger the first time it is used
  - Add code, set break points, run debugger
  - Add code, run tests, if failure, run debugger

Knuth
- Developed tool to print out first two executions of every line of code

Candy machine interfaces

Error prone return values or arguments
  ```c
  char c;
  c = getchar();
  if (c == EOF) ... 
  ```
- Classic bad example, getchar() returns an int!
- Alternate approach
  - bool fGetChar(char pch);
- Many bugs with malloc returning NULL

Another coding quiz

```c
char tolower(char ch){
  
}
```

Handling out of range inputs

- Ignore
- Return error code
- Assert
- Redefine the function to do something reasonable
- Write functions that, given valid inputs, cannot fail

Debugging

- What are the key steps in debugging a program?
Kernigan and Pike's debugging wisdom

- Look for common patterns
  - Common bugs have distinct signatures
    - `int n; scanf("%d", &n);`
- Examine most recent change
- Don't make the same mistake twice
- Debug it now, not later
- Get a stack trace
- Read before typing

K & P, II

- Explain your code to someone else
- Make the bug reproducible
- Divide and conquer
  - Find simplest failing case
- Display output to localize your search
  - Debugging with `printf()`
- Write self checking code
- Keep records

My favorite bugs (and stupidities)

- BI280 Business Basic Interpreter written in C for CP/M
- Sporadic failure of parsing
- Only happened when Basic program was changed (after being loaded)
- Parsing done by interpreter, each time a line was executed
- Adding `printf`s to code also changed behavior

The Bug

- Uninitialized variable
- Variable used by the parser to hold a character that would be either a binary operator or end of line
- Parsing algorithm looked at last character and tested if it was a binary operator to continue parsing

Don't do this

```java
try {
    doSomething();
} catch (Exception e){
}
```

- Can cover up very bad things
- Violates K&P: Debug it now, not later

Apocryphal (but still a good story)

- A program which fails only in the month of September
Apocryphal (but still a good story)

- A program which fails only in the month of September

```c
char monthName[9];
strcpy(monthName, "September");
```

ConferenceXP

- Video conferencing system would run (after initial install) for about an hour and then fail
- System would not work at all at this point
- In a week it would start working again (for an hour)
- Repeated recovery every week

Solution

- Install process (erroneously) set event log to "overwrite events older than 7 days"
- Application was very verbose in logging
- Failure when log was full