Software Testing

Other Debugging Tools

Log Files

- Log files are simply files containing text and values written out by programs as they execute.
- information can be gathered whether the programmer is actually sitting there or not.
- can be gathered over a long period of time
- can be triggered to gather output only when specific erroneous conditions arise.
- Can gather information in a production system.
- Good for working with distributed programs

Log4c

- Our own logging system
- Has 3 levels of logging severity
  - Error – very important
  - Warning – you should read this
  - Info – information you might want to see (most debugging info)
- Can be
  - Filtered to write only messages of higher than a given severity
  - Can be enabled and disabled
  - Can auto flush to make sure output or not to increase efficiency

Log4c Functions

Function	Description
l4cOpen	Open a log file for overwriting or appending
l4cClose	Close an open log file
l4cError l4cWarning l4cInfo	Write a message to the log with a specific severity level.
l4cPrintf	Write a formatted message to the log file.
l4cEnable l4cDisable l4cIsEnabled	Enable or disable log file.
l4cFilter l4cGetFilter	Set filter level or get filter level.
l4cEnableAutoFlush l4cDisableAutoFlush	Turn autoFlush on or off.

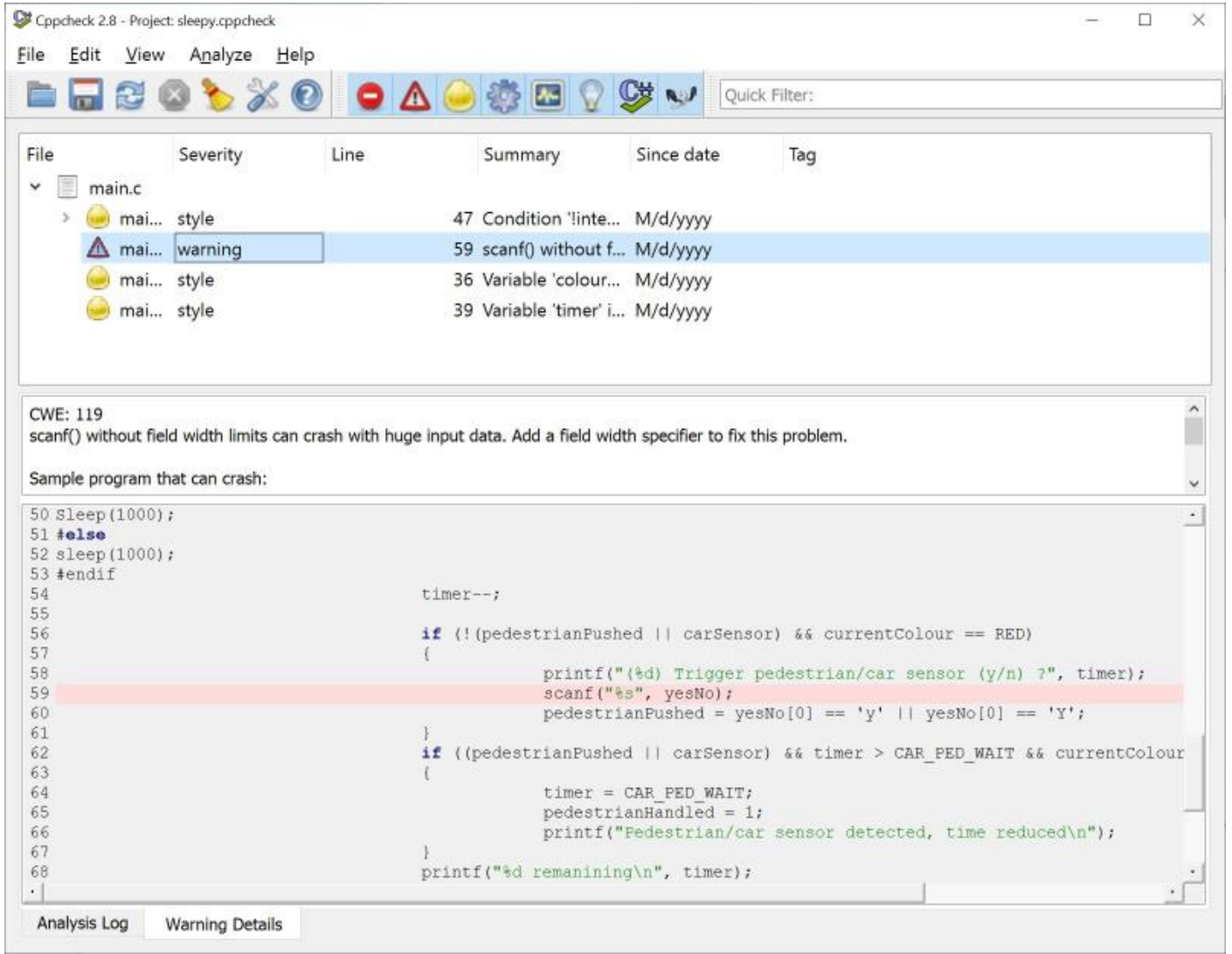
Assertions

- #include <assert.h>
- Added to production systems to stop program when something goes very wrong
- Tells the programmer the impossible happened and needs to be fixed
- assert(logical condition)
  - If the condition is triggered, the assert fires.
- E.g. assert(length > 0);

Lint

- A program which checks code for many possible errors
- Can give clues about the source of a bug
- Available for many languages
- CPPcheck for windows
  - https://github.com/danmar/cppcheck/releases/tag/2.8

CPPcheck


Cppcheck 2.8 - Project: sleepy.cppcheck
File Edit View Analyze Help
Quick Filter:
File Severity Line Summary Since date Tag
main.c
mai... style 47 Condition '!inte... M/d/yyyy
mai... warning 59 scanf() without f... M/d/yyyy
mai... style 36 Variable 'colour... M/d/yyyy
mai... style 39 Variable 'timer' i... M/d/yyyy
CWE: 119
scanf() without field width limits can crash with huge input data. Add a field width specifier to fix this problem.
Sample program that can crash:
50 Sleep(1000);
51 #else
52 sleep(1000);
53 #endif
54 timer--;
55
56 if (!(pedestrianPushed || carSensor) && currentColour == RED)
57 {
58 printf("(%d) Trigger pedestrian/car sensor (y/n) ?", timer);
59 scanf("%s", yesNo);
60 pedestrianPushed = yesNo[0] == 'y' || yesNo[0] == 'Y';
61 }
62 if ((pedestrianPushed || carSensor) && timer > CAR_PED_WAIT && currentColour
63 {
64 timer = CAR_PED_WAIT;
65 pedestrianHandled = 1;
66 printf("Pedestrian/car sensor detected, time reduced\n");
67 }
68 printf("%d remanining\n", timer);
Analysis Log
Warning Details

Core Dumps

- a much older debugging technique which are rarely used nowadays.
- A copy of the memory allocated for program written it to a file.
- in binary and is very laborious to go through by hand.
- automatic dumped readers which can allow you to explore these files in a more human readable way.

Conditional Compilation

- We can turn debugging on an off with

```
#define DEBUG 1

...

if(DEBUG) printf("%s (%d): z=%d\n", __FILE__, __LINE__, z);
```

- This does a check of the condition during runtime, slowing program
- Conditional compilation is more efficient

```
#define DEBUG

...
#ifdef DEBUG
printf("%s (%d): z=%d\n", __FILE__, __LINE__, z);
#endif
```

Debug and Release Builds

- Debug builds do more checking of error conditions
- Slow your program
- Release builds are faster but do not checking

Debugging Other Languages

- Languages differ
  - Complied vs interpreted
  - Strongly typed vs weakly typed

- These differences affect how to approach testing and debugging

Compiled vs. Interpreted

- Compiled
  - A compiler goes through the whole program at once before it runs
  - Detects problems before the program ever runs

- Interpreted
  - Program is run line by line
  - Program is parsed only just before line is executed
  - Problems are not detected until line is executed
  - Rarely executed lines can make it to production with bugs in them because they were never executed

Strongly vs. Weakly Typed

- Strongly typed
  - All assignments and function calls are check to make sure the right types are used
  - Errors detected at compilation time

- Weakly typed
  - Variables and parameters can accept any type
  - Sending the wrong variable by mistake is not detected until the line is executed
  - Rarely executed lines can have errors in them that are not detected until months later.