# C instead of Java

## Why? PMD etc

# Splint

# FramaC, Blast, clang

# Eclipse

# 9/27 Dondero’s target errors:

Erin, Below are some comments that I find myself writing repeatedly on COS 217 student submissions. Splint does not report the stylistic errors that motivate these comments, nor does any other tool. I hope you find them helpful. Bob Dondero

* The function is very long. It would be better to split it into multiple subordinate functions, each of which does a single well-defined job.
* The function contains many levels of nesting. It would be better to factor some of the nested code into subordinate functions.
* The function contains substantial chunks of cloned code. It would be better to factor the cloned code into a subordinate function, and call that function from multiple places.
* The function should validate its parameters.
* The function makes multiples calls into the functions of another module. It might be better to define the function in that other module, thereby reducing the level of coupling between the two modules.
* The module’s interface contains the definition of a structure type. That’s a violation of encapsulation. Instead the structure type definition should be in the module’s implementation. The interface should define an opaque pointer type.
* The same structure type is defined in multiple modules. That’s a maintenance nightmare: changing one definition and forgetting to change the other would introduce an error.
* The function’s name should be prefixed by the name of its module.
* The function should have a comment.
* The function’s comment should describe what the function does, not how the function works.
* The function’s comment should refer to the function’s parameters, by name.
* The function’s comment should explicitly state what the function returns.
* The function’s comment should explicitly state which global variables the function uses or affects.
* The function’s comment should explicitly state what the function reads from stdin (or any other input stream) and writes to stdout (or another other output stream).
* The function dynamically allocates memory, and then relies upon the caller to free that memory. That’s fine, but the function’s comment should inform the caller that the caller “owns” that memory.
* Poor indentation. Difficult to understand. Did you use Emacs?
* Please limit line lengths to 72 characters. Long lines wrap, and so make the code difficult to understand.
* Please use descriptive names for variables, functions, etc.
* The module’s interface file should have the same name as the module’s implementation file (except, of course, for the filename suffixes).
* Each module should have both an interface file and an implementation file.
* An error message should begin with the name of the program that is reporting the error. That name should be argv[0]; it should not be hardcoded.
* It would be better to use enum, not #define, to define names for integral constants.
* It would be better to use enum, not “const”, to define names for integral constants.

# 10/9

## Thesis Proposal

# 10/14

## Good progress and proposal. Next steps are finding a parser and consequently a language. Look into Splint and GCC only enough to ensure that building a new setup is the right idea. After language and parser and determined, spend some time both hacking out a “solution” without any form of plug-in/customization. This can then be refactored and referenced while looking into how to build up an architecture. This week also focus on getting documentation/recording setup.

### Aux: look into getting svn put up on penguins

### Probable parsers = JavaCC and Bison

## Email: 10/15 - The parser that I used for my dissertation is named Sable, not Sabre. No wonder I couldn't find it yesterday!  Here a reference: <http://sablecc.org/> It seems that a C99 (but not a C90) grammar is available for Sable.  See this reference:  <http://sablecc.sourceforge.net/grammars.html>.  But it also seems that the grammar works only with one particular version of Sable.  Of course that's not good.  So it might be wise to avoid Sable.

## Completed a very rough draft of the Problem Description section.

## IDIOT: Checkstyle and PMD are both customizable!

## Svn repository (off penguins) working with automatic login

# 10/21 as of 10/27

## Ok that other customizable things exist. Makes the architecture possibly a little less hard/challenging/interesting but still filling a void. Still need to focus on grammar/language (probably JavaCC or Bison/Yacc but grammar for JavaCC is a little iffy). Write follow up email at beginning of break but no meeting next week.

## David Walker: In general, it is just fine to write an analyzer for language X in a different language Y.  So you shouldn't be worried about using a Java parser generator to analyze C.  If I were you I would choose to choose JavaCC if Java is the language you are most comfortable programming in.  Also, I recommend making your choice of parser generator based on the fact that someone else had already written a grammar for C using that parser generator.  Writing a parser for C is a hard thing to do -- you don't want to do that yourself.  You want to reuse a parser someone else has written.

# Fall Break

## Email Exchange with Dondero:

### I just wanted to write you an email checking in. I'm planning on choosing a grammar/parser generator within the next day or so. Right now I'm leaning towards Bison because I'm more confident of the grammar but I want to do another round of checking into Javacc to make sure that its really not a good choice. I'm also going to write up 'Chapter 2' of my thesis (namely discussing the other options that exist - Splint, FramaC, Checkstyle, PMD) before school starts up next week. This will obviously be rewritten/updated as I get more insight into some of the other architectures but I want to get something down on 'paper'. I also plan on having the Checkpoint Document ready for you by the beginning of next week.

### I think your plans are fine.  In particular, I agree with your strategy of writing throughout the year rather than waiting until the end. I also agree that Bison probably is the safer choice.  It's easy to believe that few people have used Javacc with a C grammar.  Before committing to the "Javacc + C grammar" combination, I think I'd need to see strong evidence that others have used that combination successfully. As soon as I can I'll create a tar file of old ish submissions for you.  I'll let you know when I've done that.  I think it's important that you start cranking out some code soon.

## 11/6 – When I’m on campus in January, everything written prior to January needs to be in a mostly finalized form.

## QUESTION: Footnotes/endnotes/what general citation format is preferable?

### Endnotes if choice but no real preference

## 11/6 – Finished chapter on related products (aka Splint, PMD and Checkstyle)

# 11/11

## Meeting:

### Good progress. Do not switch off of Bison unless you encounter huge roadblocks.

### Plan:

#### Get a check working with Bison C (like counting parameters)

#### Get a check with Bison Java (if possible)

#### Customize with Bison C

#### Customize with Bison Java

#### Choose between Bison C/Java

### Versioning at each step of architecture reconstruction (aka save the version so I can mention it as a discreet step)

## Got grammar + lex working off ansi standard by eating anything starting with # to a \n. Need to get # include working to stop syntax errors on defined structs etc.

## Got ish.tar

# 12/2/10

## Plan of attack: Try to implement enough checks with tree\_code enum version to count it as its own version (don’t worry about finding extra levels of depth if not already built into system – that will be the next version).

## Typedef issue is limited to just typedefs (not structs) and look into pseudo code in checkType in c.l.

## Worst case on includes just use c preprocessor (gcc –E and look at .i files) but then won’t be able to see comments/have to get line numbers back to source code.

## Can I check: long functions/too much nesting/etc with this version?

### Find the length of a construct by location.last\_line – location.first\_line

# 12/6/10: Cleaned names from ish dataset

# 12/9/10:

## Progress to point – fixed typedefs, fixed includes stack to start with initial file, put comments in their own start states but need to fix line numbers

## Meeting:

### Includes:

#### All std stuff is in /usr/include

#### If file no found in local folders search/try in /usr/include (which matches the definition for inclusion on “” files, <> is just a suppression of the local folder search)

#### Try to run on stdlib (io, strings) and if epic fails then see the cost of fixing it. If not worth fixing keep hack of special size\_t in lexer

### atexit(funcptr) calls function at program exit

### linenumbers and filenames not critical

# 12/28/10

## fixed linenumbers and filenames

## handle multiline macros

## have capacity to do includes like gcc BUT in order to do so I need to actually process macros otherwise I can’t get through some of the included std stuff (specifically I don’t recognize \_\_signed because it was defined in a macro)

### Question: keep hack and don’t actually include std files OR process macros? 🡪 version 1 keep hack

# 1/17/11

## Keep a well defined list of limitations (and maybe causes?)

### Like the hack for includes

### Also design choices

#### Ifndef solution of keeping track of filenames

## Are there categories of checks?

### Seems like what has been implemented already are really unrelated

## flush out more tests (as v1.x or v2.0)

## check comments only on global level

### that is assuming I can tell what level of nesting I’m in

## Possible way to move users away from the grammar file is to create a glob callback function

### Any function, if or whatever other structure

### Could be version x which is rejected (say why)

## Not necessary to run against huge test data at this stage but maybe switch to also using my ish code?

# 2/1/11

## check for comments after function but precedence for comments before if there’s a comment before and after

### 3 locations of comments: 1 before function declaration, 2 between declaration and body, 3 inside body. 1 and 2 seem like valid locations

### (After meeting: accomplished finding 1 and 2)

## count files maybe?

## Default for switch statements

## SAX XML parsing

### Call backs at beginning and end of everything (as well as on parameter, statement etc)

### User responsibility to keep track of context/sequence

### Burden off grammar but on registering callbacks

## Limitation about no subfolders

### It would be nice to follow paths but really low priority

## Structure of Thesis: keep versions to be included outright or as notes for background/history