2013-11-29

C Style Checker

C STYLE CHECKER

Flex and Bison Implementation.

November 29, 2013

Bruce Dearing

Acadia University

OVERVIEW

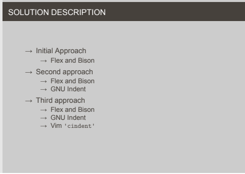
1. Introduction
2. Implementation Description
3. Possible Extensions
4. Summary and Conclusions
5. Questions?

INTRODUCTION

PROBLEM DESCRIPTION

COMP 2103 students are required to format their programs according to a set of style guidelines.

Currently no style checker for C has been in use.
The solution to this problem would require a program which would maintain as much flexibility as possible while producing a detailed analysis of a students code.
Ideally, the program would be used as the back end for a web page, so that someone could submit their program and the web server would return either confirmation that the style meets the guidelines. or a list of non-conforming constructs.



SOLUTION DESCRIPTION

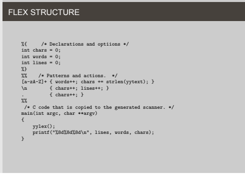
→ Initial Approach
  → Flex and Bison
→ Second approach
  → Flex and Bison
  → GNU Indent
→ Third approach
  → Flex and Bison
  → GNU Indent
  → Vim `'cindent'`

- In order to provide a reasonable amount of functionality while still remaining flexible, I initially chose to use a combination of LALR compiler tools.
- Look-Ahead LR parser reading context free BNF grammar.
- Backus–Naur Form BNF
- Unfortunately constructing a grammar that could also parse all the possible combinations of white space proved problematic.
- Therefore my second approach was to use a combination of compiler tools and a gnu open source project called indent.
- The indent program changes the appearance of a C program by inserting or deleting white space based on a set of supplied flags.
- indent can be used to format the code to the desired specifications.
- As the name suggests gnu indent can handle indentation, however, it is not very flexible in implementing different levels of indentation.
- For indentation I use Vim's cindent, which is considerably more configurable using cinoptions.

WHAT ARE FLEX AND BISON?

Flex and Bison are a set of tools originally designed for constructing compilers. They have proven to be very useful in building programs which handle structured input.

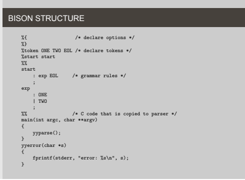
- Flex is a fast lexical analyser generator.
- It is a tool for generating programs that perform pattern-matching on text.



FLEX STRUCTURE

```
%{     /* Declarations and optiions */
int chars = 0;
int words = 0;
int lines = 0;
%}
%%    /* Patterns and actions.  */
[a-zA-Z]+ { words++; chars += strlen(yytext); }
\n        { chars++; lines++; }
.         { chars++; }
%%
 /* C code that is copied to the generated scanner. */
main(int argc, char **argv)
{
    yylex();
    printf("%8d%8d%8d\n", lines, words, chars);
}
```

- The first section contains declarations and option settings.
- The second section is a list of patterns and actions, and
- the third section is C code that is copied to the generated scanner.



BISON STRUCTURE

```
%{                  /* declare options */
%}
%token ONE TWO EOL /* declare tokens */
%start start
%%
start
    : exp EOL     /* grammar rules */
    ;
exp
    : ONE
    | TWO
    ;
%%                 /* C code that is copied to parser */
main(int argc, char **argv)
{
    yyparse();
}
yyerror(char *s)
{
    fprintf(stderr, "error: %s\n", s);
}
```

- Bison is a general-purpose parser generator that converts an annotated context-free grammar into a deterministic LR or generalized LR (GLR) parser.
- The input file for the Bison utility is a Bison grammar file The general form of a Bison grammar file is as follows:

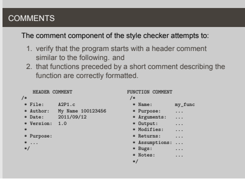
IMPLEMENTATION DESCRIPTION

STYLE CHECKER DESIGN

The C style checker is composed of four main components which perform the following checks:

→ Comments;
→ Indentation;
→ Common errors, and Style;
→ Format, and White space.

The components are tied together with a shell script which is accessed through a web based interface.

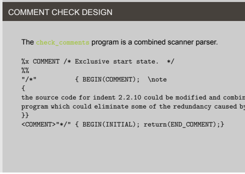


COMMENTS

The comment component of the style checker attempts to:

1. verify that the program starts with a header comment similar to the following. and
2. that functions preceded by a short comment describing the function are correctly formatted.

```
   HEADER COMMENT                 FUNCTION COMMENT
/*                                 /*
 * File:     A2P1.c                 * Name:        my_func
 * Author:   My Name 100123456      * Purpose:     ...
 * Date:     2011/09/12             * Arguments:   ...
 * Version:  1.0                    * Output:      ...
 *                                  * Modifies:    ...
 * Purpose:                         * Returns:     ...
 * ...                              * Assumptions: ...
 */                                 * Bugs:        ...
                                    * Notes:       ...
                                    */
```

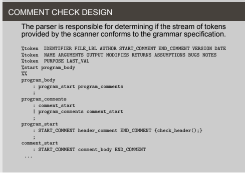

COMMENT CHECK DESIGN

The check_comments program is a combined scanner parser.

```
%x COMMENT /* Exclusive start state.  */
%%
"/*"          { BEGIN(COMMENT);  \note
{
the source code for indent 2.2.10 could be modified and combin
program which could eliminate some of the redundancy caused by
}}
<COMMENT>"*/" { BEGIN(INITIAL); return(END_COMMENT);}
```

The starting comment character '/*' triggers the scanner to enter an 'exclusive' start state `<COMMENT>` to capture comment tokens and pass those tokens off to the parser.

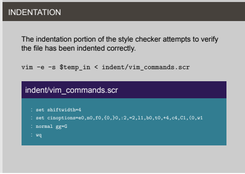
The parser then handles the tokens and verifies the comment is complete.



COMMENT CHECK DESIGN

The parser is responsible for determining if the stream of tokens provided by the scanner conforms to the grammar specification.

```
%token  IDENTIFIER FILE_LBL AUTHOR START_COMMENT END_COMMENT VERSION DATE
%token  NAME ARGUMENTS OUTPUT MODIFIES RETURNS ASSUMPTIONS BUGS NOTES
%token  PURPOSE LAST_VAL
%start program_body
%%
program_body
    : program_start program_comments
    ;
program_comments
    : comment_start
    | program_comments comment_start
    ;
program_start
    : START_COMMENT header_comment END_COMMENT {check_header();}
    ;
comment_start
    : START_COMMENT comment_body END_COMMENT
 ...
```



INDENTATION

The indentation portion of the style checker attempts to verify the file has been indented correctly.

```
vim -e -s $temp_in < indent/vim_commands.scr
```

indent/vim_commands.scr

```
: set shiftwidth=4
: set cinoptions=e0,n0,f0,{0,}0,:2,=2,l1,b0,t0,+4,c4,C1,(0,w1
: normal gg=G
: wq
```

The program begins by correctly indenting a temporary copy of the supplied file by passing the file into vim in execute mode and supplying a set of script commands.

The temporary file is then compared against the original and the diff output is run through a lexical analyzer to report error messages when differences in indentation exist.

COMMON ERRORS AND STYLE

The Common errors and style portion of the style checker consists of two components.

1. `common_errors` program which initiates checks for:
   → Common white space errors. and
   → Code block bracket location.
2. `composite_check` program which combines the ANSI C grammar specification with a combination of additional grammar productions to produce style error checks.

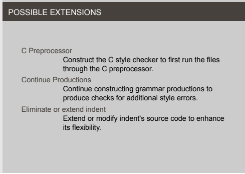
FORMAT AND WHITE SPACE

The format portion of the style checker utilizes the GNU indent program to verify the file has been correctly formatted.

One problem that I ran into while using indent, is that it is not very flexible, there are a number of flags that can either be turned on or off, there is no middle ground if either way is acceptable. There are also some modifications that indent makes to code which cannot be turned off. No flag exists to change its implementation. Due to this lack of flexibility, there are some style errors that it picks up that are acceptable, and a certain amount of redundancy in reporting. the source code for indent 2.2.10 could be modified and combined into a future version of this program which could eliminate some of the redundancy caused by indents lack of flexibility.

POSSIBLE EXTENSIONS

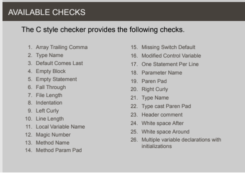
I made an initial attempt to implement Preprocessing the files before checking for errors. Running through the C preprocessor proved very useful in eliminating portions of the file which are difficult to parse without it. It can be implemented to replaces macro definitions and join broken strings and variable together. and line position can be easily recalculated with the linemarkers the preprocessor inserts into the outfile. where I ran into problems with implementation is when the C file included local header files. In order to implement the preprocessor portion a multiple file upload portion would be required. This would also eliminate the need for a GLR parser in the Common Errors and Style portion.


POSSIBLE EXTENSIONS

C Preprocessor
Construct the C style checker to first run the files through the C preprocessor.
Continue Productions
Continue constructing grammar productions to produce checks for additional style errors.
Eliminate or extend indent
Extend or modify indent's source code to enhance its flexibility.

I initially made a list of all the checks that checkstyle for java was able to perform.
After removing the java specific items and any which conflicted with the style guidelines
I was left with a baseline list of 39 checks.
I was to able code checks for 28 of them.
the source code for indent 2.2.10 could be modified and combined into a future version of this program which could eliminate some of the redundancy caused by indents lack of flexibility.

SUMMARY AND CONCLUSIONS



AVAILABLE CHECKS

The C style checker provides the following checks.

1. Array Trailing Comma
2. Type Name
3. Default Comes Last
4. Empty Block
5. Empty Statement
6. Fall Through
7. File Length
8. Indentation
9. Left Curly
10. Line Length
11. Local Variable Name
12. Magic Number
13. Method Name
14. Method Param Pad
15. Missing Switch Default
16. Modified Control Variable
17. One Statement Per Line
18. Parameter Name
19. Paren Pad
20. Right Curly
21. Type Name
22. Type cast Paren Pad
23. Header comment
24. White space After
25. White space Around
26. Multiple variable declarations with initializations

- Array Trailing Comma - Checks if array initialization contains optional trailing comma.
- Type Name - Checks that type names conform to a format specified by the style guide.
- Default Comes Last - Check that the `default` case is after all the cases in a switch statement.
- Empty Block - Checks for empty code blocks.
- Empty Statement - Detects empty statements (standalone ';').
- Fall Through - Checks for fall through in `switch` statements Finds locations where a case contains code but lacks a `break`, `return`, or `continue` statement.
- File Length - Checks for long source files.
- Indentation - Checks correct indentation of C Code.
- Left Curly - Checks the placement of left curly braces on types, functions and other blocks.
- Line Length - Checks for long lines.
- Local Variable Name - Checks that local, variable names conform to a format specified by the style guide.
- Magic Number - Checks for magic numbers.
- Method Name - Checks that method names conform to a format specified by the style guide.
- Method Param Pad - Checks the padding between the identifier of a method definition and the left parenthesis of the parameter list.
- Missing Switch Default - Checks that switch statement has `default` case.

Required Software:

- → VIM - Vi IMproved - version 7.3.547
- → flex 2.5.35
- → bison (GNU Bison) 2.7.12-4996 (any version above 2.5).
- → GNU indent 2.2.11
- → gcc 4.7.2

DEMONSTRATION

Demonstration

QUESTIONS?