C STYLE CHECKER

Flex and Bison Implementation.

November 28, 2013

Bruce Dearing

Acadia University

OVERVIEW

INTRODUCTION

PROBLEM DESCRIPTION

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

SOLUTION DESCRIPTION

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

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 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);
}
```

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);
}
```

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

```
/*
 * File:     A2P1.c
 * Author:   My Name 100123456
 * Date:     2011/09/12
 * Version:  1.0
 *
 * Purpose:
 * ...
 */
```

FUNCTION COMMENT

```
/*
 * Name:         my_func
 * Purpose:      ...
 * Arguments:    ...
 * Output:       ...
 * Modifies:     ...
 * Returns:      ...
 * Assumptions:  ...
 * Bugs:         ...
 * Notes:        ...
 */
```

COMMENT CHECK DESIGN

The check_comments program is a combined scanner parser.

```
%x COMMENT /* Exclusive start state.  */
%%
"/*"          { BEGIN(COMMENT); }
<COMMENT>"*/" { BEGIN(INITIAL); return(END_COMMENT);}
```

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
```

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

POSSIBLE EXTENSIONS

POSSIBLE EXTENSIONS

C Preprocessor

Construct the C style checker to first run the files through C preprocessor.

SUMMARY AND CONCLUSIONS