CS-1319-1
Monsoon 2023
Assignment 3

Group 7
Saptarishi Dhanuka
Shivam Kedia

PLDI

Assignment 3

The Makefile

1. `parser` is our executable
2. we need:
   - all the object files `.o` that are created during compilation,
   - `lex.yy.c` produced by `flex`
   - `7_A3.tab.h` (header for parser's state machine) and `7_A3.tab.c` (parser's state machine) produced by `bison`

   to make our executable. So, these are `INCLUDES`
3. we are asked to compile using:
   - `-ll` to link the flex library
   - `-ly` to link the bison library
   - `$(LDFLAGS)` to link the bison library also (not super sure which one was to be used so we used both)
   - `-Werror` because we'd like to treat all compiler warnings as errors, as reqd.

   So, these are our `COMPILE_FLAGS`
4. we remove all default suffix build rules using `.SUFFIXES` so the targets produced by our `.l` and `.y` files aren't messed up due to execution of both rules defined by us + default rules based on file's suffix
5. `clean` just cleans up the unnecessary files and the executable to ensure a fresh start
6. `debug` was a target we used to fix our `parser`:
   - `-v` simply creates a human-readable format of the state machine and the rules and dumps it into a `.output` file
   - `-t` is the flag that helps enable the trace option
   - `main.c` has an `#ifdef` which simply checks if `YYDEBUG` has been defined. If yes, it sets a variable `yydebug` to `1` so that traces are showed. All traces are showed since we do this in `main()`

- YYDEBUG is only defined as a flag passed to the compiler $(CC) since it's supposed to be in the main() scope. It's not used anywhere in the .y file, hence it is not passed as a flag when invoking bison

7. build force rebuilds a parser executable as required
8. the intermediate files, i.e, $(INCLUDES) are removed after a successful compilation of the parser executable since they are not needed. However, I have chosen to not remove these intermediates files when using the debug target. This was arbitrary and there's always the option to make clean.
9. the flex and bison rules have been separated out of build and the Makefile rules make it obvious which files are needed for the lexer and which ones for the parser; files produced by these are dependencies for the final parser as mentioned above in (2)
10. due to this separation, make never causes 7_A3.c to be overwritten. Since I have not had to debug that problem, idk why it would happen to others. Or perhaps I did it in Assignment 2 and forgot about it. Anyway, I'm assuming either flex or bison would overwrite 7_A3.c adhering to default file naming rules. But since we've explicitly specified outfile names using the -o flag that will not happen.
11. below is our Makefile

```
CC=gcc-13
TARGET_EXEC=parser
LEX=flex
YACC=bison
COMPILE_FLAGS=-ll -ly $(LDFLAGS) -Werror
BISON_DEBUG_FLAGS=-Wall -rstates -rsolved -rall -v -t
HEADER=7_A3.tab.h
INCLUDES= *.o lex.yy.c 7_A3.tab.*
DEBUG_INCLUDES=7_A3.output

.SUFFIXES:

.PHONY = clean
clean:
  rm -f $(INCLUDES) $(TARGET_EXEC) $(DEBUG_INCLUDES)

debug: 7_A3.c 7_A3.l 7_A3.y
  $(LEX) 7_A3.l
 $(YACC) 7_A3.y --header=$(HEADER) $(BISON_DEBUG_FLAGS) -o 7_A3.
```

```
tab.c
  $(CC) lex.yy.c 7_A3.tab.c 7_A3.c $(COMPILE_FLAGS) -DYYDEBUG=1
-o $(TARGET_EXEC)

build: lex.yy.c 7_A3.tab.c 7_A3.c
  $(CC) $^ $(COMPILE_FLAGS) -o $(TARGET_EXEC)
  rm -f $(INCLUDES)

7_A3.tab.c: 7_A3.y
  $(YACC) $^ --header=$(HEADER) -o $@

lex.yy.c: 7_A3.l
  $(LEX) $^
```

Changes to our lexer

1. We included the `tab.h` file so that we know what the symbolized terminals mean since bison keeps their values in the .h file
2. Instead of just printing out the lexeme like in the previous assignment, we now use `return` statements to return the corresponding symbolised terminal token to the parser. These tokens were defined by us in the `.y` file with `%token`.
3. Changed regex of `intconst` as mentioned in the assignment
4. We added separate regexes for each punctuator so that we could return the corresponding symbolised terminal token to the parser separately.
5. We also returned the keywords like `INT` and `CHAR` separately
6. Return `integer_constant` and `char_constant` separately

The .y file

1. We copied over the grammar from the assignment pdf
2. Defined the tokens which are the symbols for the terminals
3. Defined translation unit to be the grammar start symbol
4. Added separate rules for the optionals with epsilon productions (%empty) as explained in the assignment document
5. Added separate rule for `constants` - for int and char constants
6. C preamble of the `.y` file:
   - `extern char * yytext;`: This variable holds the text of the token read.
   - `extern int yylex();`: This is the function that's used to ask the lexer to start/resume scanning. This returns the actual token and helps flex and

  bison communicate. We don't directly have to use it but we mention it in the preamble to make it available to `bison` which uses these
  - `void yyerror(char *s);`: This used for error handling and reporting.

7. The yyerror body is defined as per the assignment pdf which flags the syntax error on the content which is in yytext at that time

The .c file

1. The C file is just a `main()` with a debugging facility and a call to yyparse() so that the the parsing can actually occur.
2. We just define some functions that are used for calling flex's lexer and bison and parser:
   - `int yylex()` - to call lexer (which the parser calls actually)
   - `int yyparse()` - to call the parser
   - `#if YYDEBUG...` - to provide debugging functionality (see Makefile section)

Shift-Reduce Conflict

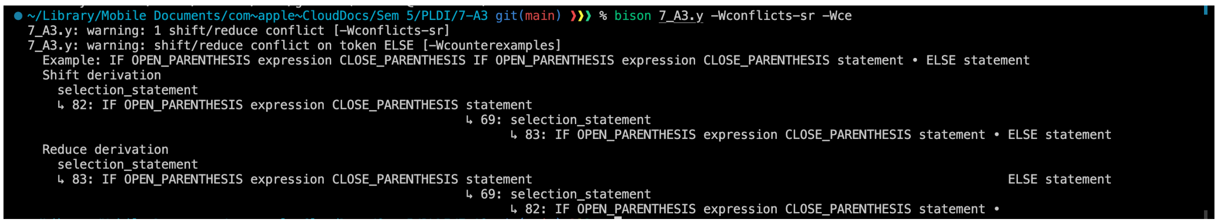
The shift-reduce conflict happens because of the following production rule:

```
  selection_statement:
      IF OPEN_PARENTHESIS expression CLOSE_PARENTHESIS statement
{printf("selection-statement\n");}
    | IF OPEN_PARENTHESIS expression CLOSE_PARENTHESIS statement
ELSE statement  {printf("selection-statement\n");}
    ;
```

We had discussed this in a DS. This is the dangling if-else problem. Once when discussing with Gautam, he mentioned that we don't really have to resolve this because bison is equipped to deal with such situations.

I'd read in this blog: https://begriffs.com/posts/2021-11-28-practical-parsing.html that `yacc` prefers a shift over a reduce for an SR conflict when the order of precedence to be shifted is higher. I'm assuming `bison` does that same.

When running the `.y` file through `bison` after using the `-Wcex` it tells us what the two parse trees can look like to illustrate the conflict:



```
● ~/Library/Mobile Documents/com~apple~CloudDocs/Sem 5/PLDI/7-A3 git(main) ❯❯❯ % bison 7_A3.y -Wconflicts-sr -Wce
  7_A3.y: warning: 1 shift/reduce conflict [-Wconflicts-sr]
  7_A3.y: warning: shift/reduce conflict on token ELSE [-Wcounterexamples]
    Example: IF OPEN_PARENTHESIS expression CLOSE_PARENTHESIS IF OPEN_PARENTHESIS expression CLOSE_PARENTHESIS statement • ELSE statement
    Shift derivation
      selection_statement
      ↳ 82: IF OPEN_PARENTHESIS expression CLOSE_PARENTHESIS statement
                                                             ↳ 69: selection_statement
                                                                   ↳ 83: IF OPEN_PARENTHESIS expression CLOSE_PARENTHESIS statement • ELSE statement
    Reduce derivation
      selection_statement
      ↳ 83: IF OPEN_PARENTHESIS expression CLOSE_PARENTHESIS statement                                                               ELSE statement
                                                             ↳ 69: selection_statement
                                                                   ↳ 82: IF OPEN_PARENTHESIS expression CLOSE_PARENTHESIS statement •
```