DECAF Project

1 LEXICAL PHASE

The flex tool is used to generate the lexical analyzer for the project. It reads the decaf.l file as input and generates the lex.yy.c as the output file. The tokens are defined in the decaf.l file, which returns the token when the lexeme matches a particular pattern. The regex for handling comments is also declared in the same file.

The tokens recognized are: '+', '-', '*', '/', '=', '(', ')', ',', ';', '.', ':=', '<', '<=', '<>', '>', '>=';
numbers: 0-9 {0-9}; identifiers: a-zA-Z {a-zA-Z0-9} and keywords.

These programs perform character parsing and tokenizing via the use of a deterministic finite automaton (DFA). A DFA is a theoretical machine accepting regular languages. These machines are a subset of the collection of Turing machines. DFAs are equivalent to read-only right moving Turing machines. The syntax is based on the use of regular expressions.

Examples:

The following flex input specifies a scanner which, when it encounters the string ‘username’ will replace it with the user's login name:

1.
```
%%
username    printf( "%s", getlogin() );
```

2.
```
int num_lines = 0, num_chars = 0;

%%
\n      ++num_lines; ++num_chars;
.       ++num_chars;

%%

int main()
        {
        yylex();
        printf( "# of lines = %d, # of chars = %d\n",
```

```
        num_lines, num_chars );
}
```

This scanner counts the number of characters and the number of lines in its input. It produces no output other than the final report on the character and line counts. The first line declares two globals, num_lines and num_chars, which are accessible both inside yylex() and in the main() routine declared after the second '%%'. There are two rules, one which matches a newline ('\n') and increments both the line count and the character count, and one which matches any character other than a newline (indicated by the '.' regular expression).

2 SYNTAX PHASE

The bison tool is used to create the decaf.tab.c and the decaf.h file from the decaf.y file. It contains the yy.parse() function which in turn calls the yylex() function for fetching tokens, then matches the tokens to the grammar to assure the syntactic correctness of the program. The tokens needed by the Bison parser will be generated using flex. Since the tokens are provided by flex we must provide the means to communicate between the parser and the lexer. The data type used for communication, *YYSTYPE*, is set using Bison's *%union* declaration.

Since in this sample we use the reentrant version of both flex and yacc we are forced to provide parameters for the *yylex* function, when called from *yyparse*. This is done through Bison's *%lex-param* and *%parse-param* declarations.

Important data structure and functions: yylval, YYSTYPE, yyerror(), yyparse()