Department of Computer Science and Technology

7th Semester

Compiler Design Laboratory

Assignment 1:

Objective:

Familiarisation with LEX and the design of a rudimentary lexical analyser.

Key Issues/Features:

1. A LEX file had to be written, which would recognize keywords, identifiers and operators. The output would be a statement printed for each token identified in the sample program. For example, for an 'if-else' statement, the output will print 'IF keyword found' and 'ELSE keyword found'.
2. For identifiers, instead of printing anything, a function '*void install_id()*' would be invoked. This function inserted the identifier name into a Symbol Table, here represented by a hash table with chaining. The index of the hash table was based on the sum of the ASCII values of the characters in the identifier name.
3. The Symbol Table was printed after the all the print statements (which described each token) described in point no.1. Each line corresponded to a hash index, and identifier names printed according to their position in the chain.

Remarks:

1. We used the string '*yytext*' to obtain the names of the identifiers.
2. The function '*yylex()*' defined in the LEX-generated file *lex.yy.c* (a scanning subroutine used for matching tokens, calling auxiliary subroutines and macros) is called in the main subroutine of the LEX file before printing the symbol table.
3. We have written our own version of *yywrap()* function, which returns 1, signalling the end of scanning.

Assignment 2:

Objective:

Familiarisation with YACC and design of a parser that recognises declaration statements.

Key Issues/Features:

1. We had to write a YACC file where we specified the grammar for our language. In this case, the language consisted only of declaration statements.
2. We did the above according to the procedure mentioned in the tutorial provided –
   2.1. We first included the necessary files and declared the global variables needed in the Declaration Section.
   2.2. Possible types of terminals and return types of non-terminals were specified in a *union* variable. Terminals were declared with *'%token*' tag and non-terminals with *'%type*' tag and types specified for each.
   2.3. Production rules were specified in the Grammar Rules Section. In this case, the rules were –

   DECLARATION → DECLARATION DECLARATION_STATEMENT
   | DECLARATION_STATEMENT
   DECLARATION_STATEMENT → ID_TYPE ID_LIST ;
   ID_TYPE → type_token
   ID_LIST → ID_LIST , id_name
   |id_name

   (Here words in capital letters represent non-terminals, words in small letters, comma and semicolon represent terminal symbols)
   2.4. The user's subroutines were coded in the Program Section. A symbol table was constructed (as in the previous assignment), and same identifier name declared twice (under same or different identifier types) generated an error.
3. The lexical analyser was used to return the token types i.e whether the token was an identifier name, or identifier type, or a value assigned to an identifier. These return values were assigned to terminal symbols by the YACC file.

Remarks:

1. The *yyparse()* function contained in *yy.tab.c* generated by YACC was called in the main subroutine.
2. A subroutine for constructing the symbol table and displaying it was called afterwards.
3. The function *yyerror()* needed by the parser was coded in the YACC file which printed the error message and aborted execution of the program.

Assignment 3:

Objective:

Incorporating action routines for assignment statement, if, if-else structures, for and while loops and generating intermediate code.

Key Issues/Features:

1. The grammar for each of the programming constructs is defined in the YACC file.
2. According to the token type returned by the LEX file, the relevant grammar is used. For each of the programming constructs mentioned above, production rules were defined and intermediate code defined.
3. For example, for WHILE-loop, the following production rule was used:
   `whl → TOKWHILE OPENPAREN cond CLOSEPAREN OPENBRACE lines CLOSEBRACE`

Remarks:

The additional grammar was incorporated in the YACC file used in the previous assignment.