|  |
| --- |
| **Name:**Liya Elizabeth Joe  **Roll No:**48 |
| **Exp. No:** 8 **Date:** 20/10/2020 |
| **Study of Lex Program** |
| **Aim:** To familiarize with the lex program. |
| Lex is a program generator designed for lexical processing of character input streams.It reads an inputstream specifying the lexical analyzer and outputssource code implementing the lexer in theC programming language.  **Structure of Lex**  The general structure of a lex program is:  {definitions}  %%  {rules}  %%  {user subroutines}  **Sections of Lex Program**  Lex program is classified into 3 sections.They include-Definition,Transition rules,Auxiliary procedures/subroutines.   * **Definition** : This definesmacros and imports header files written inC. It is also possible to write any C code here, which will be copied verbatim into the generated source file. * **Rules :** Thissection associatesregular expression patterns with Cstatements. When the lexer sees text in the input matching a given pattern, it will execute the associated C code. * **User Subroutines/Auxiliary procedure**s : This part contains C statements andfunctions that are copied verbatim to the generated source file. These statements presumably contain code called by the rules in the rules section.   **Built-in Functions in Lex**  Some of the built in functions used in lex are:   1. **yylex()**-This function implies the main entry point for lex, reads the input stream, generates tokens, returns zero at the end of the input stream . It is called to invoke the lexer (or scanner) and each time yylex() is called, the scanner continues processing the input from where it last left off. 2. **yywrap()** -It is called by lex when input is exhausted (or at EOF). Default yywrap always returns 1. 3. **yymore()**-It returns the next token. 4. **yyless(k)**-It returns the first k characters in yytext . 5. **yyparse()**-It parses (i.e builds the parse tree) of lexeme . 6. **yytext** - where text matched the most recently is used. 7. **yyin** - points current file parsed by the lexer. 8. **yyout** - points file that output of lexer will be written   **Compilation and Execution**  There are two stages for compilation. In the first stage, the lex program is passed into a lex compiler that turns the user’s expressions and actions (called source in this memo) into the host general-purpose language; the generated program is named yylex. In the second stage the yylex program is passed into the C compiler that recognizes expressions in a stream (called input in this memo) and performs the specified actions for each expression as it is detected.  Execution of the lex program is the same as for the C program.  The steps for compilation and execution are:   1. lex programname.l 2. gcc lex.yy.c 3. ./a.out.   **Sample Lex Program**  Lex program to count number of words  %{  #include<stdio.h>  #include<string.h>  int i = 0;  %}    /\* Rules Section\*/  %%  ([a-zA-Z0-9])\* {i++;} /\* Rule for counting  number of words\*/    "\n" {printf("%d\n", i); i = 0;}  %%    int yywrap(void){}    int main()  {  printf("Enter the string");  yylex();    return 0;  } |
| **Result:** Familiarized with the structure , sections and functions of the lex program. |
| **Remarks:**(To be filled by faculty) |
| **Output of Sample Program** |