

Compiler design lab

Mini PROJECT

SUBMMITTED BY:

**Abhishek M**

**(Reg. No:16030141CSE006)**

SYNOPSIS:

IMPLEMENTATION OF A LEXICAL ANALYZER USING LEX

Lexical analyzer supplies services next\_token(), back\_token()

- Lexical analyzer reads text from the input file and identifies tokens. This

happens when function next\_token() is called.

- When a token is identified in the input text, it should be stored in a data

structure. For each token, the following attributes are saved:

\* token type

\* token lexeme

\* number of the line in the input text in which this token was found.

- Blanks, tabs, new lines – are not tokens, and should be ignored

- For each token, print (on a separate line) its type (e.g. rel\_op , number , etc.)

and lexeme

- Each operation, keyword, separation sign and each type of number should be

implemented as a token of a different kind

- Kinds of tokens are coded with integer numbers, for example:

# define ID\_tok 1

# define COMMA\_tok 2

As you can see I already defined all the types I was suppost to, I do not understand how to implement next and back, some guideness whould be great,also I saved the line number

SOFTWARE USED:

Ubuntu

Ubuntu is an open-source operating system (OS) based on the Debian GNU/Linux distribution.

Ubuntu incorporates all the features of a Unix OS with an added customizable GUI, which makes it popular in universities and research organizations. Ubuntu is primarily designed to be used on personal computers, although a server editions does also exist.

Ubuntu was first released in 2004. The project is sponored by Canonical Ltd., a U.K.-based company that generates revenue by selling support and services to complement Ubuntu. Canonical releases a new version of Ubuntu every six months and provides support in the form of patches and security releases for 18 months thereafter.

Ubuntu consists of many software packages, which are licensed under GNU General Public License. This allows users to copy, change, develop and redistribute their own version of the program.

Ubuntu comes with a wide range of software programs, including FireFox and LibreOffice. There is also proprietary software that can be run on Ubuntu.

IMPLEMENTATION OF A LEXICAL ANALYZER USING LEX

AIM:

To write a program for implementing a Lexical analyser using LEX tool in Linux platform.

ALGORITHM:

Step1: Lex program contains three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, %%. The format is as follows: definitions %% rules %% user\_subroutines

Step2: In definition section, the variables make up the left column, and their definitions make up the right column. Any C statements should be enclosed in %{..}%. Identifier is defined such that the first letter of an identifier is alphabet and remaining letters are alphanumeric.

Step3: In rules section, the left column contains the pattern to be recognized in an input file to yylex(). The right column contains the C program fragment executed when that pattern is recognized. The various patterns are keywords, operators, new line character, number, string, identifier, beginning and end of block, comment statements, preprocessor directive statements etc.

Step4: Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.

Step5: When yylex() matches a string in the input stream, it copies the matched text to an external character array, yytext, before it executes any actions in the rules section.

Step6: In user subroutine section, main routine calls yylex(). yywrap() is used to get more input.

Step7: The lex command uses the rules and actions contained in file to generate a program, lex.yy.c, which can be compiled with the cc command. That program can then receive input, break the input into the logical pieces defined by the rules in file, and run program fragments contained in the actions in file.

PROGRAM CODE:

%{

int COMMENT=0;

%}

identifier [a-zA-Z][a-zA-Z0-9]\*

%%

#.\* {printf("\n%s is a preprocessor directive",yytext);}

int |

float |

char |

double |

while |

for |

struct |

typedef |

do |

if |

break |

continue |

void |

switch |

return |

else |

goto {printf("\n\t%s is a keyword",yytext);}

"/\*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}

{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}

\{ {if(!COMMENT)printf("\n BLOCK BEGINS");}

\} {if(!COMMENT)printf("BLOCK ENDS ");}

{identifier}(\[[0-9]\*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}

\".\*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}

[0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);}

\)(\:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}

\( ECHO;

= {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);}

\<= |

\>= |

\< |

== |

\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}

%%

int main(int argc, char \*\*argv)

{

FILE \*file;

file=fopen("var.c","r");

if(!file)

{

printf("could not open the file");

exit(0);

}

yyin=file;

yylex();

printf("\n");

return(0);

}

int yywrap()

{

return(1);

}

INPUT:

//var.c

#include<stdio.h>

#include<conio.h>

void main()

{

int a,b,c;

a=1;

b=2;

c=a+b;

printf("Sum:%d",c);

}

RESULT:

Thus the program for implementation of Lexical Analyzer using Lex tool has been executed successfully.

OUTPUT:

