BHARATIYA VIDYA BHAVAN'S

SARDAR PATEL INSTITUTE OF TECHNOLOGY

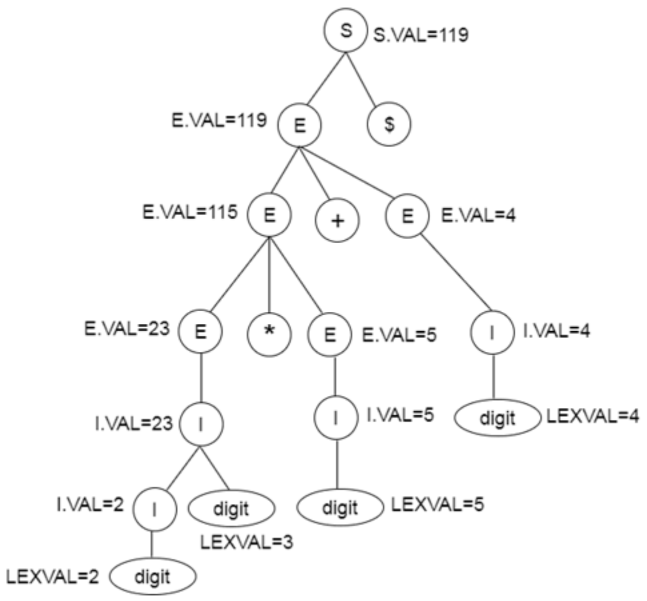
(Empowered Autonomous Institute Affiliated to University of Mumbai)

[Knowledge is Nectar]

Department of Computer Engineering

e

Course - System Programming and Compiler Construction (SPCC)

UID	2021300108
Name	Hatim Sawai
Class and Batch	TE Computer Engineering - Batch C
Date	26-02-2024
Lab #	4
Aim	Write a program to implement SDT using Lex/Yacc
Objective	The objective of this experiment is to implement a parser using Yacc and Lex to process a sequence of directional moves (N, S, E, W) within a grid. The parser aims to recognize valid input strings, derive a final position on the grid, and handle error cases, demonstrating proficiency in syntax-directed translation.
Theory	Syntax-directed translation  S S.VAL=119 E.VAL=119 E $ E.VAL=115 E + E E.VAL=4 E.VAL=23 E * E E.VAL=5 I I.VAL=4 I.VAL=23 I I I.VAL=5 digit LEXVAL=4 I.VAL=2 I digit LEXVAL=3 digit LEXVAL=5 LEXVAL=2 digit Syntax-directed translation refers to a method of compiler implementation where the source language translation is completely driven by the parser. A common method of syntax-directed translation is translating a string into a sequence of actions by attaching one such action to each rule of a grammar. Thus, parsing a string of the grammar produces a sequence of rule applications. SDT provides a simple way to attach semantics to any such syntax.

SDT fundamentally works by adding actions to the productions in a context-free grammar, resulting in a Syntax-Directed Definition (SDD). Actions are steps or procedures that will be carried out when that production is used in a derivation. A grammar specification embedded with actions to be performed is called a syntax-directed translation scheme (sometimes simply called a 'translation scheme').

Each symbol in the grammar can have an attribute, which is a value that is to be associated with the symbol. Common attributes could include a variable type, the value of an expression, etc. Given a symbol X, with an attribute t, that attribute is refer to as X.t.

Thus, given actions and attributes, the grammar can used for translating strings from its language by applying the actions and carrying information through each symbol's attribute.

What is a lex file?

A LEX file is a lexicon data file create by Linguistic Library, an Adobe development kit use add linguistic services, such as spelling and grammar checkers, to Adobe products. LEX files are use to store language-specific data for document validation in Adobe products.

What is output of Lex tool?

Lex is a program that generates lexical analyzer. It is use with YACC parser generator. The lexical analyzer is a program that transforms an input stream into a sequence of tokens. It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.

What is the use of lex tool?

Lex can perform simple transformations by itself but its main purpose is to facilitate lexical analysis, the processing of character sequences such as source code to produce symbol sequences called tokens for use as input to other programs such as parsers.

Is lex a compiler?

Lex is a tool in lexical analysis phase to recognize tokens using regular expression. Lex tool itself is a lex compiler.

What is YACC file?

Yacc, which stands for "Yet Another Compiler Compiler," is a tool used in the construction of compilers and interpreters for programming languages. It takes a formal grammar description as input and generates a parser, which is a component responsible for analyzing the syntax of a language and structuring it according to the specified grammar rules. Yacc is often used in combination with Lex, a lexical analyzer generator, to create complete language processors.

What is the output of YACC file?

The output of a Yacc file typically includes a parser program written in C or another target language. This parser is responsible for analyzing the input based on the specified grammar rules and generating a parse tree or other data structure that represents the syntactic structure of the input language.

What is the use of YACC tools?

Yacc (Yet Another Compiler Compiler) is a tool used for generating parsers. It takes a formal grammar description as input and produces a parser in a programming

	language, commonly C. This generated parser is employed in the development of compilers and interpreters, enabling the processing and analysis of input based on the specified language grammar. Yacc helps automate the creation of parsers, saving developers significant effort in implementing language syntax parsing. Below is a simplified algorithm for the Yacc code provided earlier, which parses a sequence of directional moves and calculates the final position on a grid: 1. **Lexical Analysis (Lexer - lex.yy.c):** • Define tokens for "B", "N", "S", "E", "W". • Recognize these tokens and pass them to the Yacc parser. 2. **Yacc Parser (y.tab.c):** • Define the grammar rules for the language in BNF form. • Specify the token types and their attributes using **%token** and **%union**. • Write semantic actions to perform calculations and error handling. • Implement the **move** function to update the current position. • Handle error cases and print appropriate messages using **yyerror**. • Define the **main** function to call **yyparse** and start the parsing process. 3. **Semantic Actions:** • When encountering a directional move token, update the position accordingly. • Perform error checks to handle invalid input strings. • Print the final position or error message based on the parsing result. 4. **Compilation and Execution:** • Compile the Lex file (**lex.yy.c**) using **lex**. • Compile the Yacc file (**y.tab.c**) using **yacc** or **bison**. • Compile the main program (e.g., **main.c**) including the Yacc-generated and Lex-generated files. • Execute the resulting program, input the desired string, and observe the output. This algorithm outlines the key steps involved in creating a parser using Yacc and Lex for processing directional moves in a grid.

Implementation/ Code

test.l

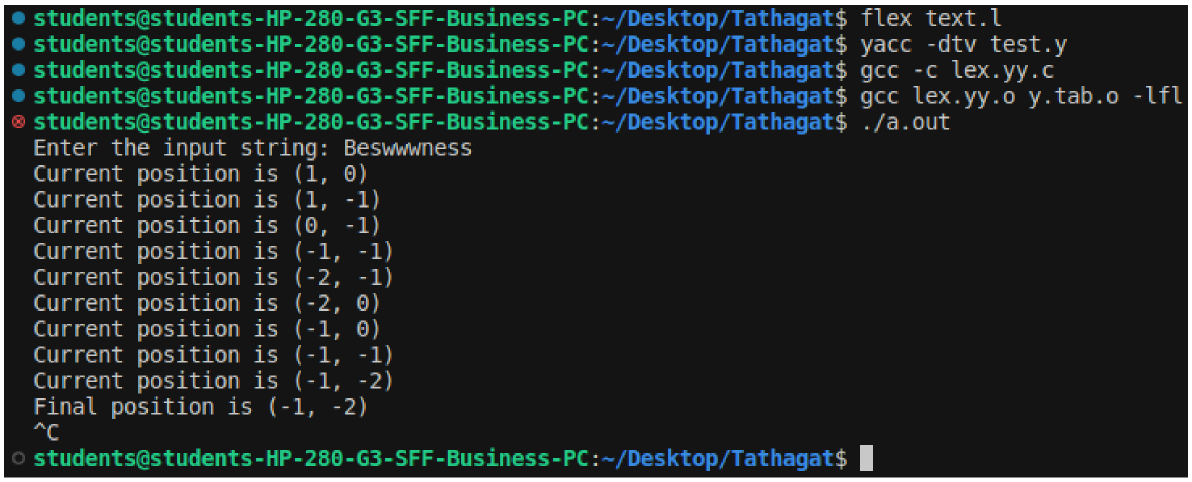
```
%{
#include "y.tab.h"
%}

%%
n { return N; }
e { return E; }
s { return S; }
w { return W; }
B { return B; }
\n { return '\n'; }
. { return yytext[0]; }
%%

int yywrap(void) {
    return 1;
}
```

test.y

```
%{
#include <stdio.h>

int yylex(void);
void yyerror(const char *s);

int posX = 0;
int posY = 0;

%}

%union {
    int num;
}

%token N E S W B

%%
stmt: B direction_list '\n' { printf("Final position is (%d, %d)\n", posX, posY); }
    | B direction_list { printf("Final position is (%d, %d)\n", posX, posY); }
    ;

direction_list: direction { }
              | direction_list direction { }
              ;

direction: N { posY++; printf("Current position is (%d, %d)\n", posX, posY); }
         | E { posX++; printf("Current position is (%d, %d)\n", posX, posY); }
         | S { posY--; printf("Current position is (%d, %d)\n", posX, posY); }
         | W { posX--; printf("Current position is (%d, %d)\n", posX, posY); }
         ;
```

```
%%
void yyerror(const char *s) {
    printf("Invalid string\n");
}

int main() {
    printf("Enter the input string: ");
    yyparse();
    return 0;
}
```

Output



```
students@students-HP-280-G3-SFF-Business-PC:~/Desktop/Tathagat$ flex text.l
students@students-HP-280-G3-SFF-Business-PC:~/Desktop/Tathagat$ yacc -dtv test.y
students@students-HP-280-G3-SFF-Business-PC:~/Desktop/Tathagat$ gcc -c lex.yy.c
students@students-HP-280-G3-SFF-Business-PC:~/Desktop/Tathagat$ gcc lex.yy.o y.tab.o -lfl
students@students-HP-280-G3-SFF-Business-PC:~/Desktop/Tathagat$ ./a.out
Enter the input string: Beswwwness
Current position is (1, 0)
Current position is (1, -1)
Current position is (0, -1)
Current position is (-1, -1)
Current position is (-2, -1)
Current position is (-2, 0)
Current position is (-1, 0)
Current position is (-1, -1)
Current position is (-1, -2)
Final position is (-1, -2)
^C
students@students-HP-280-G3-SFF-Business-PC:~/Desktop/Tathagat$
```

Conclusion

In conclusion, this experiment provided valuable hands-on experience with Yacc and Lex for parser generation. I successfully implemented a parser that processes directional moves in a grid. Through this project, I gained insights into syntax-directed translation and enhanced my understanding of grammar, tokens, and error handling.

References

[1] mechomotive. (October 24, 2021). Syntax-directed translation. https://mechomotive.com/syntax-directed-translation/

[2] ChatGpt. (2023). Yacc Grid Parser https://chat.openai.com/share/8b3a2dcf-6958-435b-a5a8-665d631d32f6