VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi-590018, Karnataka

Case Study Report on

"PARSE THE GIVEN STRUCTURE"

Submitted in partial fulfillment of the requirement of 6th semester

Bachelor of Engineering
in
Computer Science & Engineering

Submitted by

Group ID: 18SSC17

Under the Guidance of
Prof. Maya B. S
Assistant Professor
Department of CSE, BIT
Bengaluru-560004

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V. Pura, Bengaluru-560 004
2020-21

Group ID: 18SSC17			
Sl.No	**USN**	**Name**	**Signature**
1	1BI19CS400	**AKSHAY KUMAR U**	AKSHAY KUMAR.U
2	1BI18CS118	**RAJ KAMAL**	[illegible]
3	1BI18CS061	**HRATHIK S**	[illegible]
4	1BI18CS089	**N.A.NIKHITHA**	N.A. Nikhitha

Maya B.S
Assistant Professor
Dept. of Computer Science & Engineering
Bangalore Institute of Technology
K. R. Road, V. V. Pura, Bengaluru - 04

Name of Project : PARSE THE GIVEN

STRUCTURE Language : LEX AND YACC

Tools : Gedit, Lex and Yacc

Description :

What is parsing?

Syntax analyzers follow production rules defined by means of context free grammar. The way the production rules are implemented.

What is Lex?

Lex is a program that generates lexical analyzer. The lexical analyzer is a program that transforms an input stream into a sequence of tokens.

What is Yacc?

YACC stands for Yet Another Compiler Compiler. · **YACC** provides a tool to produce a parser for a given grammar. YACC is a program designed to compile a LALR (1) grammar.

Problem Statement: PARSE THE GIVEN STRUCTURE

Structure can be defined as a user defined datatype. The keyword we use is “struct”.
The syntax for Structure :

```
struct structure_name {
       // Data members ;
  };
```

According to our problem statement we need to parse the given structure and check whether the given structure is a valid structure or not. To validate the given structure we have used the concept of lex and yacc. LEX is a tool which is used for building a lexer or used in lexical analysis. YACC(Yer Another Compiler-Compiler): Yacc is a tool which is used for building parser Or used in Syntax Analysis phase. So whenever the given structure is according to the above given syntax the parser validates the Structure.

LEX: Lex generates C code for a lexical analyzer, or scanner. It uses patterns that match strings in the input and converts the strings to tokens. Tokens are numerical representations of strings, and simplify processing. As lex finds identifiers in the input stream, it enters them in a symbol table. The symbol table may also contain other information such as data type (integer or real) and location of the variable in memory. All subsequent references to identifiers refer to the appropriate symbol table index.

```
SYNTAX: ...
definitions ...

       %%

          ... rules ...

           %%

          ... subroutines ...
```

YACC: Yacc is a tool which is used for building parser Or used in Syntax Analysis phase.
So whenever the given structure is according to the above given syntax the parser validates
the Structure.

Source Code :

YACC CODE: project.y

First Section is declaration section, Here we have included **stdio.h**, As we are using printf() we have included it.

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

Here we have defined the tokens that are being used by our grammer to parse the given C Structure.

%token ID DATATYPE SC NL COMMA STRUCT LEFT RIGHT SL SR NUM

Second Section is the rules section where we have provided the CFG what accepts the valid structure syntax.

Grammer starts with Non terminal symbol start, where rule says the input should start with struct keyword followed by structName left parentethis declaration of variables followed by right paranthesis and should end with semicolon, next comes declaring the structure variables that is done by struct keyword followed by structName and variable list.
Some of the accepted structure syntax according to above rule :-

```
struct name
{
        int a;
};
struct name names[100];

struct name
{
        int a;
};
struct name name1,name2,name3;
```

start also has one more alternative rule that says Grammer starts with Non terminal symbol start, where rule says the input should start with struct keyword followed by structName left parentethis declaration of variables followed by right paranthesis and should end with semicolon, Followed by comma variable list and Semicolon.

Some of the accepted structure syntax according to above rule :-

```
struct name
{
        int a;
}, names[100];

struct name
{
        int a;
}, name1,name2,name3;
```

```
%%
start:STRUCT ID LEFT declare RIGHT SC
                initialize SC{ printf("Valid
                Structure Syntax\n");
                };
|STRUCT ID LEFT declare RIGHT COMMA
                initialize SC{ printf("Valid Structure
                Syntax\n");
                };

declare:DATATYPE varlist SC
|DATATYPE varlist SC declare
|STRUCT ID LEFT declare RIGHT SC;
varlist:varlist COMMA ID
|ID SL NUM SR | ID SL NUM SR SL NUM SR |
|ID;
initialize:STRUCT ID varlist | varlist
%%


void yyerror(const char *str)
{
      printf("Invalid %s\n",str);
}
int yywrap()
{
      return 0;
}
main(int argc, char **argv)
{
       printf("Enter Structure Syntax:\n");
       yyparse();
}
```

LEX CODE: project.l

In declaration section we are including y.tab.h header file.

%{

#include<stdio.h> #include "y.tab.h"

%}

In rules section we are matching the tokens and returning it to YACC code, we are matching tokens using Regular Expressions.

%%

int|float|char return DATATYPE; //if the input is int / float / char we are returning it as DATATYPE

"[" return SL; //If input is [then it is square left paranthesis return it as SL;

"]" return SR; //If input is **]** then it is square Right paranthesis return it as SR;

[0-9]* return NUM; // Matching Digit and returning it as NUM for array number indexing;

struct return STRUCT; // Matching struct for keyword and we return it as STRUCT;

"{" return LEFT; //Matching Left paranthesis and return it as LEFT;

"}" return RIGHT; //Matching Right paranthesis and return it as RIGHT;

"," return COMMA; // Matching Comma and return it as COMMA;

";" return SC; //Matching Semi colon and return it as SC;

[_a-zA-Z]+[a-zA-Z0-9]* return ID; //A valid variable name where it should start with alphabet and followed by it can have alphabets , numbers and return it as ID;

"\n" ;

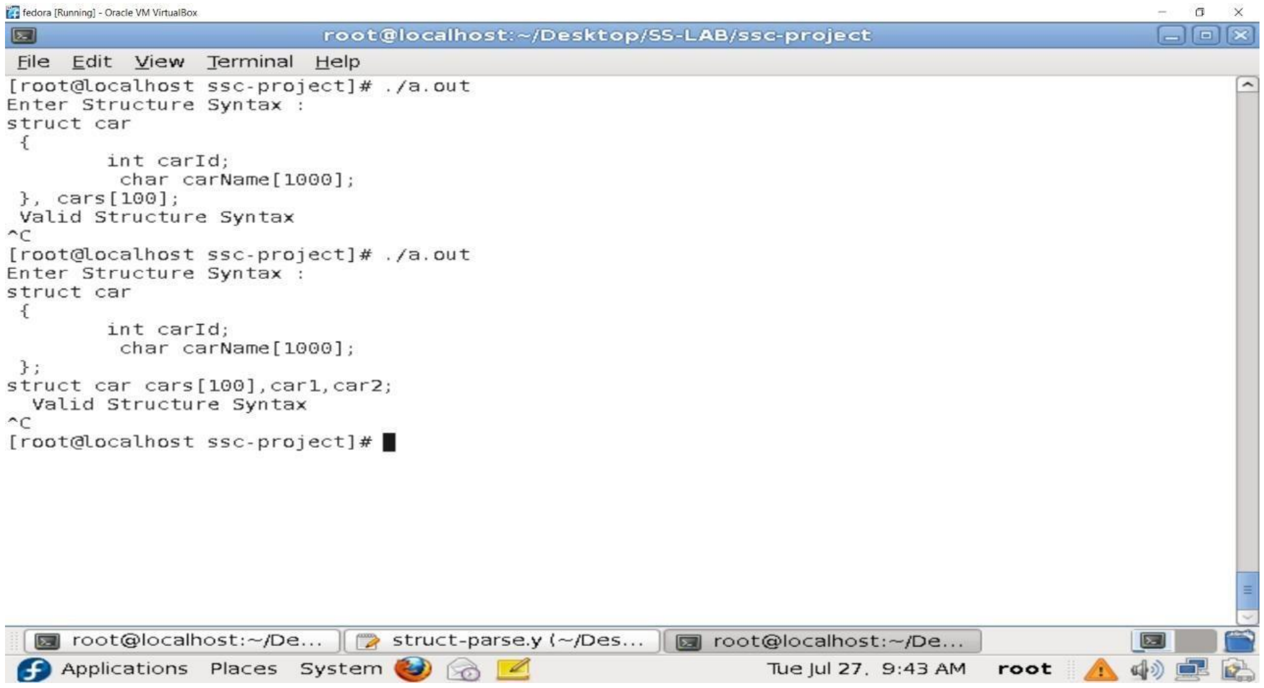
%%

Execution Commands :

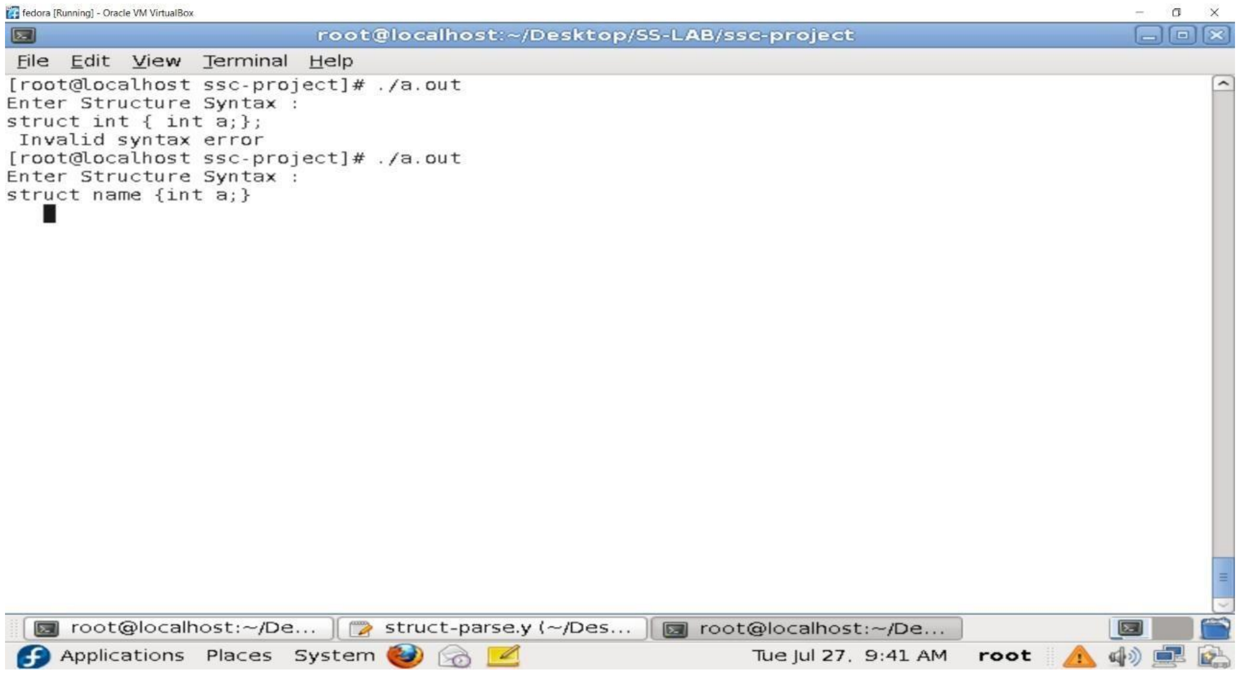
```
lex project.l
yacc –d project.y
cc lex.yy.c y.tab.c –ll
./a.out
```

Output:

VALID INPUTS**:**


```
[root@localhost ssc-project]# ./a.out
Enter Structure Syntax :
struct car
 {
        int carId;
         char carName[1000];
 }, cars[100];
 Valid Structure Syntax
^C
[root@localhost ssc-project]# ./a.out
Enter Structure Syntax :
struct car
 {
        int carId;
         char carName[1000];
 };
struct car cars[100],car1,car2;
  Valid Structure Syntax
^C
[root@localhost ssc-project]# 
```

INVALID INPUTS:


```
[root@localhost ssc-project]# ./a.out
Enter Structure Syntax :
struct int { int a;};
 Invalid syntax error
[root@localhost ssc-project]# ./a.out
Enter Structure Syntax :
struct name {int a;}
```