

Laboratory 7

Title of the Laboratory Exercise: Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.

1. Introduction and Purpose of Experiment

Students learn to

2. Aim and Objectives

Aim

- To write a program

Objectives

At the end of this lab, the student will be able to

- Define

3. Experimental Procedure

Students are required to carry out the following steps:

- Algorithm
- Write the Lex program
- Write yacc program
- Compile and execute the program (steps)
- Complete the documentation for the given problem

4. Presentation of Results

tokens.l

```
%{  
#include "parser.tab.h"  
%}  
  
%option noyywrap  
  
%%  
  
[0-9]+      { return DIGIT; }  
[a-zA-Z]+  { return LETTER; }  
[ \t]      { ; }
```

```
\n      { return 0; }
.      { return yytext[0]; }

%%
```

parser.y

```
%{

%}

%token DIGIT LETTER

%%

start  : LETTER s

s      : LETTER s
      | DIGIT s
      | /* empty */
      ;

%%
```

main.c

```
#include <stdio.h>
#include <stdlib.h>

#include "parser.tab.h"

int main(int argc, char* argv[]) {
    printf("Enter String : ");
    yyparse();
    printf("valid identifier\n");
    return 0;
}

void yyerror(char* s) {
    printf("invalid: %s\n", s);
    exit(0);
}
```

Makefile

```
all: ident

parser.tab.c parser.tab.h: parser.y
    bison -d parser.y

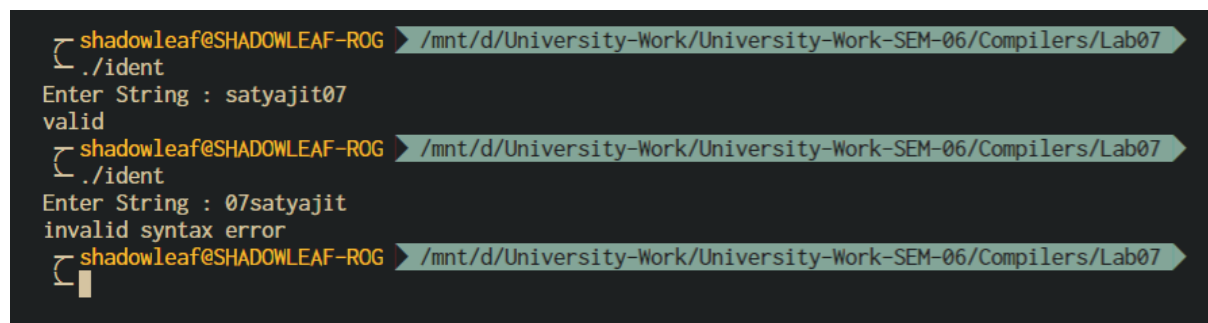
lex.yy.c: tokens.l parser.tab.h
```

```
flex tokens.l

ident: main.c lex.yy.c parser.tab.c parser.tab.h
      gcc -o ident main.c parser.tab.c lex.yy.c

clean:
      rm ident parser.tab.c parser.tab.h lex.yy.c
```

5. Analysis and Discussions



```
shadowleaf@SHADOWLEAF-ROG > /mnt/d/University-Work/University-Work-SEM-06/Compilers/Lab07
└─ ./ident
Enter String : satyajit07
valid
shadowleaf@SHADOWLEAF-ROG > /mnt/d/University-Work/University-Work-SEM-06/Compilers/Lab07
└─ ./ident
Enter String : 07satyajit
invalid syntax error
shadowleaf@SHADOWLEAF-ROG > /mnt/d/University-Work/University-Work-SEM-06/Compilers/Lab07
└─
```

Figure 0-1 OUTPUT

6. Conclusions

As Bison reads tokens, it pushes them onto a stack along with their semantic values. The stack is called the parser stack. Pushing a token is traditionally called shifting.

The function `yyparse` is implemented using a finite-state machine. The values pushed on the parser stack are not simply token type codes; they represent the entire sequence of terminal and nonterminal symbols at or near the top of the stack. The current state collects all the information about previous input which is relevant to deciding what to do next.

Each time a lookahead token is read, the current parser state together with the type of lookahead token are looked up in a table. This table entry can say, "Shift the lookahead token." In this case, it also specifies the new parser state, which is pushed onto the top of the parser stack. Or it can say, "Reduce using rule number n." This means that a certain number of tokens or groupings are taken off the top of the stack, and replaced by one grouping. In other words, that number of states are popped from the stack, and one new state is pushed.

The basic way to declare a token type name (terminal symbol) is as follows:

```
%token name
```

Bison will convert this into a definition in the parser, so that the function yylex (if it is in this file) can use the name name to stand for this token type's code.

7. Comments

a. Limitations of Experiments

Yacc is inflexible in some ways:

- good error handling is hard (basically, its algorithm is only defined to parse a correct string correctly, otherwise, all bets are off; this is one of the reasons that GCC moved to a hand-written parser)
- context-dependency is hard to express, whereas with a hand-written recursive descent parser you can simply add a parameter to the functions

Furthermore, lex/yacc object code is often bigger than a hand-written recursive descent parser (source code tends to be the other way round).

b. Limitations of Results

The solution program written here throws a syntax error when the grammar is not matched, the function of a lexer should be to point the line number and column number of the syntax error, which this program does not, a major limitation of the result obtained.

c. Learning happened

We learnt the basics of YACC and LEX to convert an FSM grammar rules to program that can validate strings which start with letter followed by any number of letters or numbers.

d. Recommendations

Add the functionality to display the error along with line and column number for better debugging.

Component	Max Marks	Marks Obtained
Viva	6	
Results	7	

NAME: SATYAJIT GHANA

REG NO: 17ETCS002159

Documentation	7	
Total	20	