**Name : Aditi Nikam Roll No : 323045**

**GR No : 21910513 Class : TY Comp C2**

**3 A**

**Code:**

%%

is |

am |

are |

were |

was |

be |

being |

been |

do |

does |

did |

will |

would |

should |

can |

could |

has |

have |

had |

plays |

learn |

go {printf("%s: is a verb\n", yytext);}

very |

simply |

gently |

quietly |

calmly |

angrily {printf("%s: is a adverb\n", yytext);}

Alex |

viit |

football |

kind |

physics |

diwali {printf("%s: is a noun\n", yytext);}

if |

then |

and |

but |

or {printf("%s: is a conjunction\n", yytext);}

their |

my |

your |

his |

her |

its {printf("%s: is a adjective\n", yytext);}

to |

from |

behind |

above |

below |

between {printf("%s: is a preposition\n", yytext);}

I |

you |

he |

she |

we |

they {printf("%s: is a pronoun\n", yytext);}

[a-zA-Z]+ {printf("%s: Don't Recognize\n", yytext);}

.|\n {ECHO;}

%%

int yywrap(){

return 0;

}

int main(){

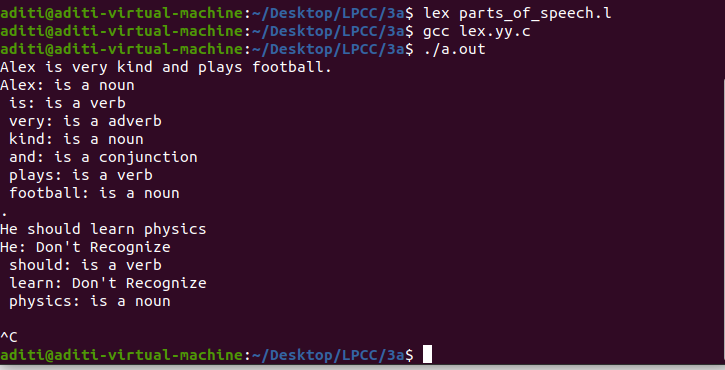
yylex();

yywrap();

return 0;

}

**Output:**



**3 B**

**Code:**

%{

enum{

set = 0,

verb ,

adjective ,

adverb ,

noun ,

preposition ,

pronoun ,

conjunction ,

};

int state=0;

int add\_word(int type, char \*word) ;

int lookup\_word (char \*word) ;

%}

%%

\n { state = set; }

^verb {state = verb;}

^adjective {state = adjective;}

^conjunction {state = conjunction;}

^pronoun {state = pronoun;}

^adverb {state = adverb;}

^noun {state = noun;}

^preposition {state = preposition;}

[a-zA-Z]+ {

if(state != set){

add\_word(state, yytext);

} else{

switch(lookup\_word(yytext)) {

case verb: printf("%s: verb\n", yytext); break;

case adjective: printf("%s: adjective\n", yytext); break;

case conjunction: printf("%s: conjunction\n", yytext); break;

case pronoun: printf("%s: pronoun\n", yytext); break;

case adverb: printf("%s: adverb\n", yytext); break;

case noun: printf("%s: noun\n", yytext); break;

case preposition: printf("%s: preposition\n", yytext); break;

default: printf("%s: Don't Recognize\n", yytext); break;

}

}

}

. ;

%%

int yywrap(){

return 0;

}

int main(){

yylex();

yywrap();

return 0;

}

struct word {

char \*word\_name;

int word\_type;

struct word \*next;

};

struct word \*word\_list;

extern void \*malloc();

int add\_word(int type, char \*word)

{

struct word \*wp;

if(lookup\_word(word) != set) {

printf("word %s already defined \n", word);

return 0;

}

wp = (struct word \*) malloc(sizeof (struct word) ) ;

wp->next = word\_list;

wp->word\_name = (char \*) malloc (strlen(word) +1);

strcpy (wp->word\_name, word);

wp->word\_type = type;

word\_list = wp;

return 1;

}

int lookup\_word (char \*word) {

struct word \*wp = word\_list;

for(; wp; wp = wp->next) {

if(strcmp(wp->word\_name, word) == 0)

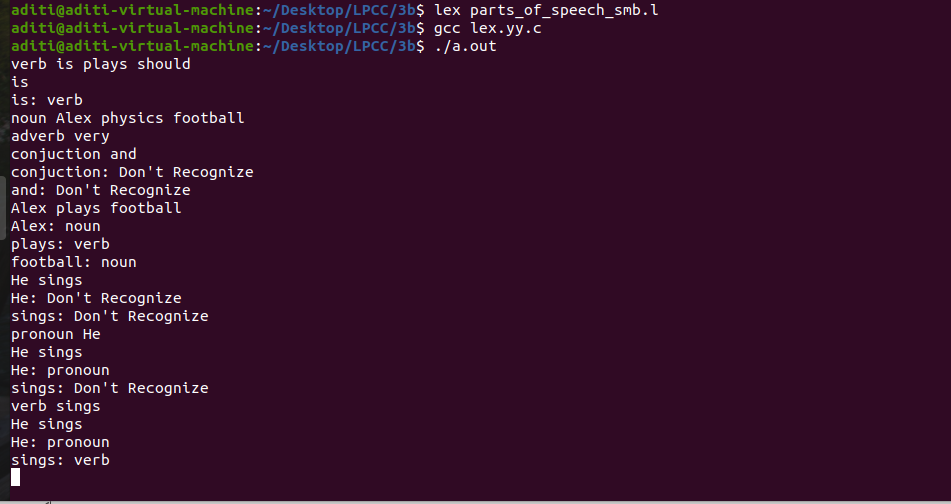
return wp->word\_type;

}

return set;

}

**Output:**



**3 C**

**Code:**

%%

[\t ]+ ;

int |

double |

long |

unsigned |

float |

char |

short {printf("%s: is a data type\n", yytext);}

"-" |

"+" |

"\*" |

"/" |

"++" |

"--" {printf("%s: is a arithmetic operator\n", yytext);}

"==" |

"!=" |

"<" |

">" |

"<=" |

">=" {printf("%s: is a relational operator\n", yytext);}

"&&" |

"||" |

"!" {printf("%s: is a logical operator\n", yytext);}

"&" |

"^" |

"|" |

"~" |

"<<" |

">>" {printf("%s: is a bitwise operator\n", yytext);}

"=" {printf("%s: is a assignment operator\n", yytext);}

"{" |

"}" {printf("%s: is a parentheses\n", yytext);}

"//\*" {printf("%s: is a comment\n", yytext);}

"(" |

")" {printf("%s: is a bracket\n", yytext);}

";" {printf("%s: is a semi colon\n", yytext);}

"," {printf("%s: is comma\n", yytext);}

include |

main |

return |

for |

while |

do |

scanf |

printf {printf("%s: is a reserved word\n", yytext);}

if |

else {printf("%s: is a conditional statement\n", yytext);}

"%s" |

"%d" |

"%u" |

"%i" |

"%c" {printf("%s: is a format specifier\n", yytext);}

[0-9]+ |

[0-9]+\.[0-9]+ |

\.[0-9]+ {printf("%s: is a number\n", yytext);}

(\_|[a-zA-Z])([a-zA-Z]|[0-9])\* {printf("%s: is a variable\n", yytext);}

.|\n {ECHO;}

%%

int yywrap(){

return 0;

}

int main(){

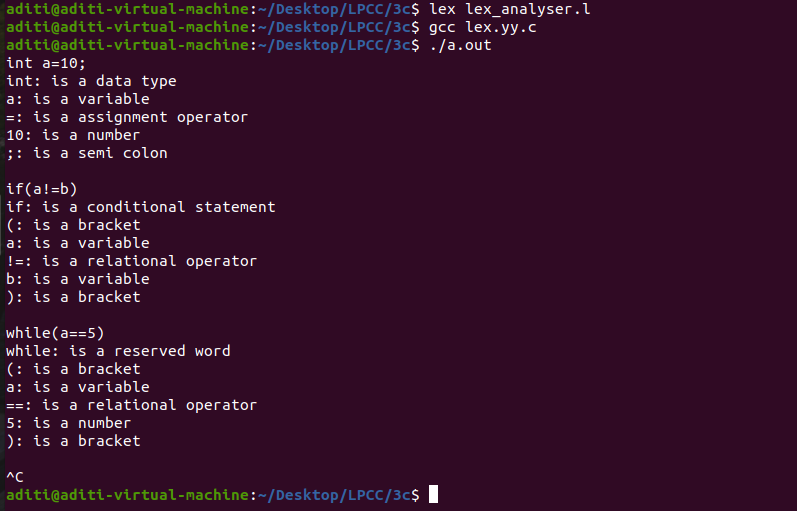
yylex();

yywrap();

return 0;

}

**Output:**



**3 D**

**Code:**

%{

enum{

set = 0,

data\_type ,

arithmetic\_operator ,

operator,

logical\_operator ,

relational\_operator ,

bitwise\_operator ,

assignment\_operator ,

parentheses ,

comment,

bracket,

semi\_colon,

comma,

reserved\_word,

conditional\_statement,

format\_specifier,

number,

variable,

};

int state=0;

int add\_word(int type, char \*word) ;

int lookup\_word (char \*word) ;

%}

%%

\n { state = set; }

^data\_type {state = data\_type;}

^arithmetic\_op {state = arithmetic\_operator;}

^operator {state = operator;}

^logical\_op {state = logical\_operator;}

^relational\_op {state = relational\_operator;}

^bitwise\_op {state = bitwise\_operator;}

^assignment\_op {state = assignment\_operator;}

^parentheses {state = parentheses;}

^comment {state = comment;}

^bracket {state = bracket;}

^semi\_colon {state = semi\_colon;}

^comma {state = comma;}

^reserved {state = reserved\_word;}

^conditional {state = conditional\_statement;}

^format\_specifier {state = format\_specifier;}

^number {state = number;}

^variable {state = variable;}

[a-zA-Z]+ {

if(state != set){

add\_word(state, yytext);

} else{

switch(lookup\_word(yytext)) {

case data\_type: printf("%s: data type\n", yytext); break;

case arithmetic\_operator: printf("%s: arithmetic operator\n", yytext); break;

case operator: printf("%s: operator\n", yytext); break;

case logical\_operator: printf("%s: logical operator\n", yytext); break;

case relational\_operator: printf("%s: relational operator\n", yytext); break;

case bitwise\_operator: printf("%s: bitwise operator\n", yytext); break;

case assignment\_operator: printf("%s: assignment operator\n", yytext); break;

case parentheses: printf("%s: parentheses\n", yytext); break;

case comment: printf("%s: comment\n", yytext); break;

case bracket: printf("%s: bracket\n", yytext); break;

case semi\_colon: printf("%s: semi colon\n", yytext); break;

case comma: printf("%s: comma\n", yytext); break;

case reserved\_word: printf("%s: reserved word\n", yytext); break;

case conditional\_statement: printf("%s: conditional statement\n", yytext); break;

case format\_specifier: printf("%s: format specifier\n", yytext); break;

case number: printf("%s: number\n", yytext); break;

case variable: printf("%s: variable\n", yytext); break;

default: printf("%s: Don't Recognize\n", yytext); break;

}

}

}

. ;

%%

int yywrap(){

return 0;

}

int main(){

yylex();

yywrap();

return 0;

}

struct word {

char \*word\_name;

int word\_type;

struct word \*next;

};

struct word \*word\_list;

extern void \*malloc();

int add\_word(int type, char \*word)

{

struct word \*wp;

if(lookup\_word(word) != set) {

printf("word %s already defined \n", word);

return 0;

}

wp = (struct word \*) malloc(sizeof (struct word) ) ;

wp->next = word\_list;

wp->word\_name = (char \*) malloc (strlen(word) +1);

strcpy (wp->word\_name, word);

wp->word\_type = type;

word\_list = wp;

return 1;

}

int lookup\_word (char \*word) {

struct word \*wp = word\_list;

for(; wp; wp = wp->next) {

if(strcmp(wp->word\_name, word) == 0)

return wp->word\_type;

}

return set;

}

**Output:**

