**Name: Shlok Zanwar**

**Roll no: 323071**

**Gr no: 21910163**

**Batch : C3**

**Assignment 3**

**A**

%{

#include<stdio.h>

%}

%%

[\t]+ /\* ignore whitespace \*/ ;

is |

am |

play |

run |

eat |

sing |

dance |

jump |

cook |

sleep |

swim |

walk |

are |

were |

was |

be |

being |

been |

do |

does |

did |

will |

would |

should |

can |

could |

has |

have |

had |

come |

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

very |

rapidly |

slowly |

gently |

calmly |

proudly |

badly |

sadly |

sweetly |

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

to |

from |

upon |

beside |

behind |

near |

against |

above |

below |

over |

beyond |

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

if |

since |

although |

because |

then |

unless |

and |

so |

but |

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

gentle |

perfect |

sharp |

rough |

cruel |

young { printf("%s:It is an adjective\n", yytext); }

I |

me |

we |

you |

he |

she |

it |

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

[a-zA-Z]+ {

printf("%s: don't recognize, might be a noun\n", yytext);

}

.|\n { ECHO; /\* normal default anyway \*/ }

%%

int yywrap(void){}

int main ( )

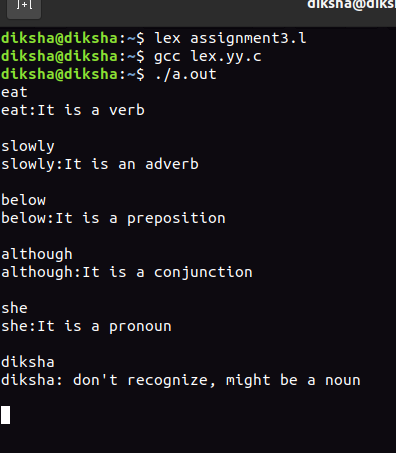
{

yylex();

return 0;

}

**Output**

****

**B**

%{

/\*

\* Word recognizer with a symbol table.

\*/

#include<stdio.h>

enum {

lookup = 0,

VERB,

ADJ,

ADV,

NOUN,

PREP,

PRON,

CONJ

};

int state;

int lookup\_word(char \*word) ;

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

%}

%%

\n { state = lookup; }

^VERB { state = VERB; }

^ADJ { state = ADJ;}

^ADV { state = ADV; }

^NOUN { state = NOUN; }

^PREP { state = PREP; }

^PRON { state = PRON; }

^CONJ { state = CONJ; }

[a-zA-Z]+ {

if (state!=lookup) {

add\_word(state, yytext); /\* define the current word \*/

}

else

{

switch ( lookup\_word(yytext) )

{

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

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

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

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

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

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

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

default: printf ( "%s: don't recognize\n" , yytext) ;

break;

}

}

}

.

%%

int yywrap(void){}

int main( )

{

yylex();

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) != lookup) {

printf("!!! warning: 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)

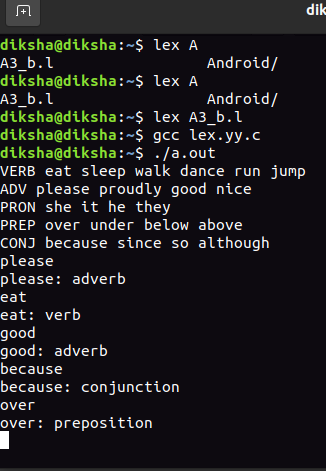
return wp->word\_type;

}

return lookup;

}

**Output:**

****

**C:**

%{

#include<stdio.h>

%}

DIGIT[0-9]

NUMBER {DIGIT}\*

REAL {DIGIT}\*[.]{NUMBER}

TEXT [A-Za-z]

KEYWORDS "auto"|"break"|"enum"|"struct"|"typedef"

DATATYPE "int"|"float"|"double"|"long"|"void"|"unsigned char"

CONDITIONAL "if"|"else"|"else if"|"switch"|"case"

ITERATIVE "for"|"while"|"do"

SEMICOLAN ";"

IDENTIFIER {TEXT}({DIGIT}|{TEXT}|"\_")\*

NONIDENTIFIER {DIGIT}({TEXT}|{DIGIT}|"\_")\*

ARITHMETIC\_OP "+"|"-"|"/"|"%"|"\*";

LOGICAL\_OP "&&"|"||"|"!"|"!="

RELATIONAL\_OP "<"|">"|"<="|">="|"=="

UNARY "++"|"--"

FUNCTION {ACCESS}{DATATYPE}{IDENTIFER}"("({DATATYPE}{IDENTIFIER})\*")"

%%

[ \n\t]+ ;

{CONDITIONAL} {printf("%s\t==> CONDITIONAL \n",yytext);}

{ITERATIVE} {printf("%s\t==> ITERATIVE \n",yytext);}

{DATATYPE} {printf("%s\t==> DATATYPE\n",yytext);}

{KEYWORDS} {printf("%s\t==> KEYWORDS \n",yytext);}

{IDENTIFIER} {printf("%s\t==> IDENTIFIER \n",yytext);}

{REAL} {printf("%s\t==> REAL \n",yytext);}

{NUMBER} {printf("%s\t==> NUMBER \n",yytext);}

{NONIDENTIFIER} {printf("%s\t==> NONIDENTIFIER \n",yytext);}

{SEMICOLAN} {printf("%s\t==> SEMICOLAN \n",yytext);}

{UNARY} {printf("%s\t==> UNARY \n",yytext);}

{ARITHMETIC\_OP} {printf("%s\t==> ARITHMETIC\_OP \n",yytext);}

{LOGICAL\_OP} {printf("%s\t==> LOGICAL OP \n",yytext);}

{RELATIONAL\_OP} {printf("%s\t==> RELATIONAL OP \n",yytext);}

"=" {printf("%s\t==> ASSIGNMENT OP \n",yytext);}

"{" {printf("%s\t==> BLOCK BEGIN \n",yytext);}

"}" {printf("%s\t==> BLOCK END \n",yytext);}

"(" {printf("%s\t==> PARATHRSIS BEGIN \n",yytext);}

")" {printf("%s\t==> PARATHRSIS END \n",yytext);}

. ;

%%

int yywrap(void){}

int main ( )

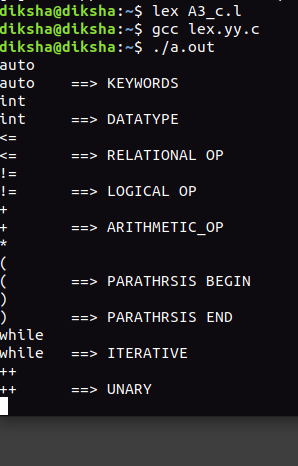
{

yylex();

return 0;

}

**Output:**

****

**D:**

%{

/\*

\* Word recognizer with a symbol table.

\*/

#include<stdio.h>

enum {

LOOKUP = 0,

TEXT,

KEYWORD,

DATATYPE,

CONDITIONAL,

ITERATIVE,

ARITHMETIC\_OP

};

int state;

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

int lookup\_word(char \*word) ;

%}

%%

\n { state = LOOKUP; }

^text { state =TEXT; }

^keyword { state = KEYWORD; }

^datatype { state = DATATYPE; }

^conditional { state = CONDITIONAL; }

^iterative { state = ITERATIVE; }

^arithmetic\_op { state = ARITHMETIC\_OP; }

[a-zA-Z]+ {

if (state!=LOOKUP) {

add\_word(state, yytext);

}

else

{

switch ( lookup\_word(yytext) )

{

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

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

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

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

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

case ARITHMETIC\_OP: printf("%s: arithmetic\_op \n", yytext); break;

default:

printf ( "%s: Don't recognize\n" , yytext) ;

break;

}

}

}

.

%%

int yywrap(void){}

int main( )

{

yylex();

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) != LOOKUP) {

printf("!!! warning: 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) {

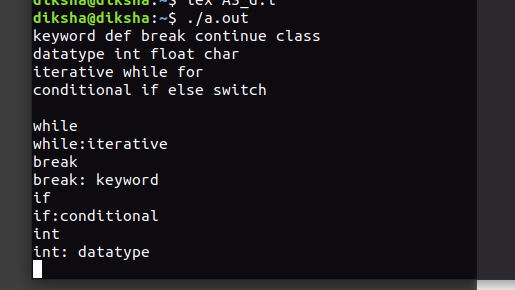
if(strcmp(wp->word\_name, word) == 0)

return wp->word\_type;

}

return LOOKUP;

}

**Output:**