**Name:Kundan Kabra  
Gr No: 21810729  
Roll No: 321023  
Div: A-1**

**3 B :-**

%{

/\*

\* Word recognizer with a symbol table.

\*/

enum{

LOOKUP = 0, /\* default-looking rather than defining. \*/

VERB,

ADJ,

ADV,

NOUN,

PREP,

PRON,

CONJ,

INT

};

int state;

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

int lookup\_word(char \*word);

%}

%%

\n {state = LOOKUP; }

/\* whenever a line starts with a reserved part of speech name \*/

/\* start defining words of that type \*/

^verb { state = VERB; }

^adj { state = ADJ; }

^adv { state = ADV; }

^noun { state = NOUN; }

^prep { state = PREP; }

^pron { state = PRON; }

^conj { state = CONJ; }

^int { state = INT; }

/\* a normal word, define it or look it up \*/

[a-zA-Z]+ {

if(state != LOOKUP) {

/\* define the current word \*/

add\_word(state, yytext);

} 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;

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

default:

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

break;

}

}

}

. /\* ignore anything else \*/;

%%

int yywrap(void)

{

}

int main()

{

yylex();

}

/\* define a linked list of words and types \*/

struct word{

char \*word\_name;

int word\_type;

struct word \*next;

};

struct word \*word\_list; /\* first element in 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;

}

/\* word not there, allocate a new entry and link it on the list \*/

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

wp->next = word\_list;

/\* have to copy the word itself as well \*/

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

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

wp->word\_type = type;

word\_list = wp;

return 1; /\* it worked \*/

}

int lookup\_word(char \*word)

{

struct word \*wp = word\_list;

/\* search down the list looking for the word \*/

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

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

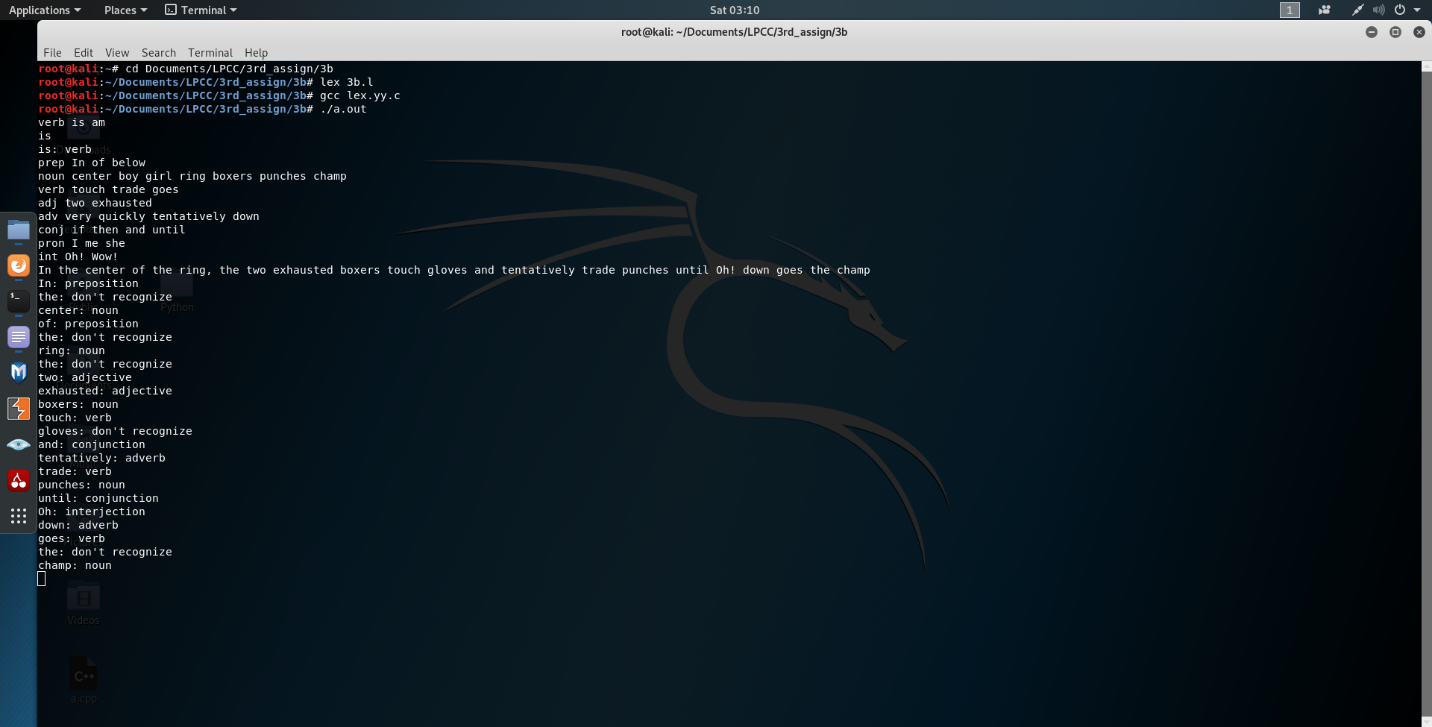
return wp->word\_type;

}

return LOOKUP; /\* not found \*/

}

**OUTPUT :-**



**3D :-**

%{

/\*

\* Word recognizer with a symbol table.

\*/

enum {

LOOKUP =0, /\* default - looking rather than defining. \*/

KEYWORD,

IDENTIFIER,

NUMBER,

LOGICAL,

ARITHMETIC,

RELATIONAL,

FORMAT,

PUNC,

ASSIGN

};

int state;

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

int lookup\_word(char \*word);

%}

%%

\n { state = LOOKUP; } /\* end of line, return to default state \*/

/\* whenever a line starts with a reserved part of speech name \*/

/\* start defining words of that type \*/

^keyword { state = KEYWORD; }

^identifier { state = IDENTIFIER; }

^number { state = NUMBER; }

^logical { state = LOGICAL; }

^arithmetic { state = ARITHMETIC; }

^relational { state = RELATIONAL; }

^format { state = FORMAT; }

^punc { state = PUNC; }

^assign { state = ASSIGN; }

[a-zA-Z0-9\.&&\|\|\+-\/%\\*\<\>==\<=\>=%d%s%f%s\{\}\,\;\':\"\?\!()]+ {

/\* a normal word, define it or look it up \*/

if(state != LOOKUP) {

/\* define the current word \*/

add\_word(state, yytext);

} else {

switch(lookup\_word(yytext)) {

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

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

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

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

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

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

case FORMAT: printf("%s: Format Specifier\n", yytext); break;

case PUNC: printf("%s: Punctuation Symbol\n", yytext); break;

case ASSIGN: printf("%s: Assignment Op\n", yytext); break;

default:

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

break;

}

}

}

. /\* ignore anything else \*/ ;

%%

int yywrap(void)

{

}

int main()

{

yylex();

}

/\* define a linked list of words and types \*/

struct word {

char \*word\_name;

int word\_type;

struct word \*next;

};

struct word \*word\_list; /\* first element in 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;

}

/\* word not there, allocate a new entry and link it on the list \*/

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

wp->next = word\_list;

/\* have to copy the word itself as well \*/

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

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

wp->word\_type = type;

word\_list = wp;

return 1; /\* it worked \*/

}

int

lookup\_word(char \*word)

{

struct word \*wp = word\_list;

/\* search down the list looking for the word \*/

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

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

return wp->word\_type;

}

return LOOKUP; /\* not found \*/

}

**OUTPUT :-**

