**VNRVJIET**

**NAME OF LAB:** COMPILER DESIGN

**WEEK:**

**DATE: / /2024**

**EXPERIMENT NAME:** Lex,Yacc Lab Programs

## Write a program in C to implement Lexical analyzer

#include <stdio.h> #include <ctype.h> #include <string.h> #include <stdlib.h>

const char\* keywords[] = {"int", "float", "if", "else", "while", "return"}; const char operators[] = "+-\*/=<>";

int isKeyword(const char\* word) {

for (int i = 0; i < sizeof(keywords) / sizeof(keywords[0]); i++) { if (strcmp(word, keywords[i]) == 0) {

return 1;

}

}

return 0;

}

int isOperator(char ch) {

for (int i = 0; i < strlen(operators); i++) { if (ch == operators[i]) {

return 1;

}

}

return 0;

}

void lexicalAnalyzer(const char\* input) { int i = 0;

int length = strlen(input); while (i < length) {

if (isspace(input[i])) { i++;

continue;

}

if (isalpha(input[i])) { char buffer[20]; int j = 0;

while (isalnum(input[i])) { buffer[j++] = input[i++];

}

buffer[j] = '\0';

if (isKeyword(buffer)) { printf("Keyword: %s\n", buffer);

} else {

printf("Identifier: %s\n", buffer);

}

} else if (isdigit(input[i])) { char buffer[20];

int j = 0;

while (isdigit(input[i])) {

buffer[j++] = input[i++];

}

buffer[j] = '\0'; printf("Number: %s\n", buffer);

} else if (isOperator(input[i])) { printf("Operator: %c\n", input[i]); i++;

} else {

printf("Unknown token: %c\n", input[i]); i++;

}

}

}

int main() {

char input[100];

printf("Enter input for lexical analysis: "); fgets(input, sizeof(input), stdin); printf("\nLexical Analysis:\n"); lexicalAnalyzer(input);

return 0;

}

# Output:



## Write a program in ‘lex’ to implement lexical analyser

%{

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

%}

%option noyywrap

%%

int|float|if|else|while|return|void { printf("KEYWORD: %s\n", yytext); } [0-9]+ { printf("NUMBER: %s\n", yytext); }

[a-zA-Z\_][a-zA-Z0-9\_]\* { printf("IDENTIFIER: %s\n", yytext); } "+"|"-"|"\*"|"/"|"="|"<"|">"|"=="|"<="|">=" { printf("OPERATOR: %s\n", yytext); }

";"|","|"\("|"\)"|"\{"|"\}" { printf("PUNCTUATION: %s\n", yytext); } [ \t\n]+

. { printf("ERROR: Unrecognized character:

%s\n", yytext); }

%%

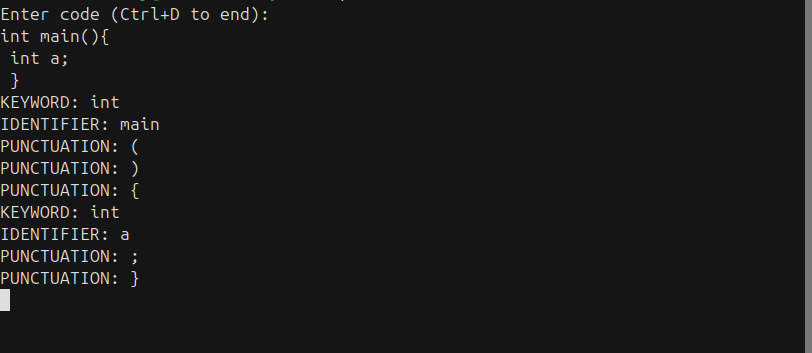
int main() {

printf("Enter code (Ctrl+D to end):\n"); yylex();

return 0;

}

# Output:



## Write a program in ‘yacc’ to implement desk calculator

### cal.l

%{

#include "y.tab.h"

%}

%%

[0-9]+ { yylval = atoi(yytext); return NUMBER; } [\t ]

\n { return '\n'; }

. { return yytext[0]; }

%%

int yywrap() { return 1;

}

### cal.y

%{

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

void yyerror(const char \*s); int yylex(void);

%}

%token NUMBER

%left '+' '-'

%left '\*' '/'

%%

calclist:

| calclist expression '\n' { printf("Result: %d\n", $2); }

;

expression:

NUMBER { $$ = $1; }

| expression '+' expression { $$ = $1 + $3; }

| expression '-' expression { $$ = $1 - $3; }

| expression '\*' expression { $$ = $1 \* $3; }

| expression '/' expression { if ($3 == 0) { yyerror("Division by zero");

exit(1);

} else {

$$ = $1 / $3;

}

};

%%

int main() {

printf("Enter expressions (Ctrl+D to exit):\n"); yyparse();

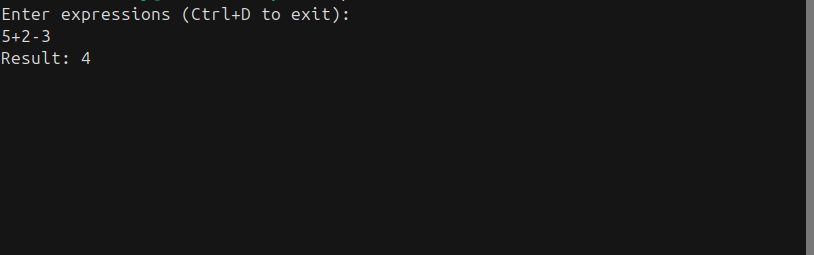
return 0;

}

void yyerror(const char \*s) { fprintf(stderr, "Error: %s\n", s);

}

# Output:



## Write a program in ’yacc’ to validate an expression

|  |  |  |  |
| --- | --- | --- | --- |
| **prg.l**  %{ |  | | |
| #include "y.tab.h"  %} |
| %%  [\_a-zA-Z][\_a-zA-Z0-9]\* | { | return | ID; } |
| [0-9]+ | { | yylval | = atoi(yytext); return NUM; } |
| [+\-\*/] | { | return | yytext[0]; } |
| \( | { | return | '('; } |
| \)  [ \t\n]  .  %% | {  {  { | return  }  return | ')'; }  yytext[0]; } |

### prg.y

%{

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

void yyerror(const char \*s); int yylex(void);

%}

%token NUM ID

%left '+' '-'

%left '\*' '/'

%%

exp:

exp '+' exp

| exp '-' exp

| exp '\*' exp

| exp '/' exp

| '(' exp ')'

| NUM

| ID

;

%%

int main() {

printf("Enter an expression: "); if (yyparse() == 0) {

printf("Valid expression\n");

} else {

printf("Invalid expression\n");

}

return 0;

}

void yyerror(const char \*s) { printf("Invalid expression\n"); exit(1);

}

int yywrap() { return 1;

}

# Output:



## Write a C program to implement first and follow functions

#include <stdio.h> #include <ctype.h> #include <string.h> #define MAX 10

char productions[MAX][MAX]; char firsts[MAX][MAX]; char follows[MAX][MAX];

int num\_productions;

void addToSet(char set[], char value) { int len = strlen(set);

for (int i = 0; i < len; i++) { if (set[i] == value) return;

}

set[len] = value; set[len + 1] = '\0';

}

void findFirst(char symbol, char result[]) { if (!isupper(symbol)) {

addToSet(result, symbol); return;

}

for (int i = 0; i < num\_productions; i++) { if (productions[i][0] == symbol) {

if (productions[i][2] == '#') { addToSet(result, '#');

} else {

for (int j = 2; productions[i][j] != '\0'; j++) { char next\_symbol = productions[i][j];

char temp[MAX] = ""; findFirst(next\_symbol, temp);

for (int k = 0; temp[k] != '\0'; k++) {

if (temp[k] != '#') addToSet(result, temp[k]);

}

if (strchr(temp, '#') == NULL) break;

if (productions[i][j + 1] == '\0') addToSet(result, '#');

}

}

}

}

}

void findFollow(char symbol, char result[]) {

if (symbol == productions[0][0]) addToSet(result, '$');

for (int i = 0; i < num\_productions; i++) {

for (int j = 2; productions[i][j] != '\0'; j++) { if (productions[i][j] == symbol) {

if (productions[i][j + 1] != '\0') { char temp[MAX] = "";

findFirst(productions[i][j + 1], temp); for (int k = 0; temp[k] != '\0'; k++) {

if (temp[k] != '#') addToSet(result, temp[k]);

}

if (strchr(temp, '#') != NULL) { char temp\_follow[MAX] = "";

findFollow(productions[i][0], temp\_follow); for (int k = 0; temp\_follow[k] != '\0'; k++) {

addToSet(result, temp\_follow[k]);

}

}

} else {

if (productions[i][0] != symbol) { char temp\_follow[MAX] = "";

findFollow(productions[i][0], temp\_follow); for (int k = 0; temp\_follow[k] != '\0'; k++) {

addToSet(result, temp\_follow[k]);

}

}

}

}

}

}

}

int main() {

printf("Enter the number of productions: "); scanf("%d", &num\_productions);

printf("Enter the productions (e.g., S=AB or A=a):\n"); for (int i = 0; i < num\_productions; i++) {

scanf("%s", productions[i]);

}

for (int i = 0; i < num\_productions; i++) { char symbol = productions[i][0];

char result[MAX] = ""; findFirst(symbol, result); strcpy(firsts[i], result);

}

for (int i = 0; i < num\_productions; i++) { char symbol = productions[i][0];

char result[MAX] = ""; findFollow(symbol, result); strcpy(follows[i], result);

}

printf("\nFirst and Follow sets:\n");

for (int i = 0; i < num\_productions; i++) {

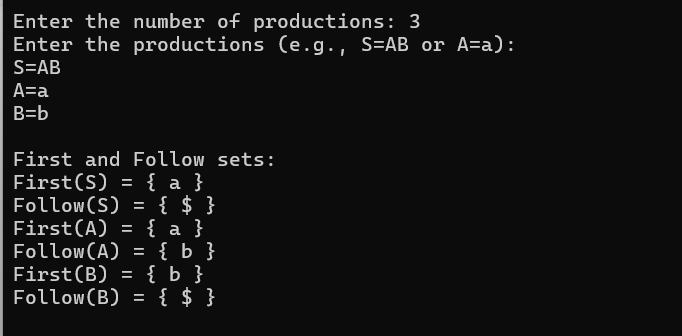
printf("First(%c) = { %s }\n", productions[i][0], firsts[i]); printf("Follow(%c) = { %s }\n", productions[i][0], follows[i]);

}

return 0;

}

# Output:



## Write a C program to implement LL(1) parser

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <ctype.h>

void add\_symbol(char \*, char); void FIND\_FIRST(char \*, char); void FIND\_FOLLOW(char \*, char); void FIRST\_SHOW();

void FOLLOW\_SHOW();

int CREATE\_LL1\_TABLE();

void PARSING\_TABLE\_SHOW(int); void LL1\_PARSER(char \*);

int top = 0;

int t = 0, nt = 0, ch, cr = 0, count = 0; char FIRST[100][100], FOLLOW[100][100];

char T[100], NT[100], G[100][100], STACK[100]; int LL1[100][100];

void main() {

int i, j, flag, fl, ch1; char STR[100];

printf("Enter production rules of grammar in the form A->B\n\n"); flag = 1;

fl = 1;

while (flag == 1) {

printf("\n1) Insert Production Rules\n2) Show Grammar\n3) Exit"); printf("\nEnter your choice: ");

scanf("%d", &ch1);

switch (ch1) { case 1:

printf("Enter production rule %d: ", cr + 1); scanf("%s", G[cr++]);

for (i = 0; i < nt && fl == 1; i++) { if (NT[i] == G[cr - 1][0]) fl = 0;

}

if (fl == 1) NT[nt++] = G[cr - 1][0];

fl = 1;

for (i = 3; G[cr - 1][i] != '\0'; i++) {

if (!isupper(G[cr - 1][i]) && G[cr - 1][i] != '!') {

for (j = 0; j < t && fl == 1; j++) { if (T[j] == G[cr - 1][i]) fl = 0;

}

if (fl == 1) T[t++] = G[cr - 1][i];

fl = 1;

}

}

break; case 2:

if (cr > 0) {

printf("\nGrammar");

printf("\nStarting symbol: %c", G[0][0]); printf("\nNon-terminal symbols: ");

for (i = 0; i < nt; i++) printf("%c ", NT[i]); printf("\nTerminal symbols: ");

for (i = 0; i < t; i++) printf("%c ", T[i]); printf("\nProduction rules: ");

for (i = 0; i < cr; i++) printf("%s ", G[i]); printf("\n");

} else {

printf("!Enter at least one production rule.");

}

break;

case 3:

flag = 0; break;

}

}

FIRST\_SHOW(); FOLLOW\_SHOW(); T[t++] = '$';

T[t] = '\0';

flag = CREATE\_LL1\_TABLE(); PARSING\_TABLE\_SHOW(flag);

if (flag == 0) {

printf("Enter string for parsing: "); scanf("%s", STR);

LL1\_PARSER(STR);

}

}

void FIRST\_SHOW() {

int i, j;

char arr[100];

for (i = 0; i < nt; i++) { arr[0] = '\0'; FIND\_FIRST(arr, NT[i]);

for (j = 0; arr[j] != '\0'; j++) { FIRST[i][j] = arr[j];

}

FIRST[i][j] = '\0';

count = 0;

}

printf("\nFIRST:\n\n"); for (i = 0; i < nt; i++) {

printf("FIRST(%c): { ", NT[i]);

for (j = 0; FIRST[i][j + 1] != '\0'; j++) {

printf("%c, ", FIRST[i][j]);

}

printf("%c }\n", FIRST[i][j]);

}

}

void FOLLOW\_SHOW() {

int i, j;

char arr[100];

for (i = 0; i < nt; i++) { count = 0;

arr[0] = '\0'; FIND\_FOLLOW(arr, NT[i]);

for (j = 0; arr[j] != '\0'; j++) { FOLLOW[i][j] = arr[j];

}

FOLLOW[i][j] = '\0';

}

printf("\nFOLLOW:\n\n"); for (i = 0; i < nt; i++) {

printf("FOLLOW(%c): { ", NT[i]);

for (j = 0; FOLLOW[i][j + 1] != '\0'; j++) {

printf("%c, ", FOLLOW[i][j]);

}

printf("%c }\n", FOLLOW[i][j]);

}

}

void PARSING\_TABLE\_SHOW(int flag) { int i, j;

if (flag == 0) {

printf("\n\nPredictive Parsing Table:\n\n\t"); for (j = 0; j < t; j++) {

printf("\t%c\t", T[j]);

}

printf("\n \n\n"); for (i = 0; i < nt; i++) {

printf("%c\t|\t", NT[i]); for (j = 0; j < t; j++) { if (LL1[i][j] != 0) {

printf("%s\t\t", G[LL1[i][j] - 1]);

} else {

printf("\_\t\t");

}

}

printf("\n\n");

}

}

}

void FIND\_FIRST(char \*arr, char ch) { int i;

if (!isupper(ch)) { add\_symbol(arr, ch);

} else {

for (i = 0; i < cr; i++) { if (ch == G[i][0]) {

if (G[i][3] == '!') {

add\_symbol(arr, G[i][3]);

} else {

FIND\_FIRST(arr, G[i][3]);

}

}

}

}

}

void FIND\_FOLLOW(char arr[], char ch) { int i, j, k, l, fl = 1, flag = 1;

if (ch == G[0][0]) add\_symbol(arr, '$');

for (i = 0; i < cr; i++) {

for (j = 3; G[i][j] != '\0' && flag == 1; j++) { if (ch == G[i][j]) {

flag = 0;

if (G[i][j + 1] != '\0' && isupper(G[i][j + 1])) { for (k = 0; k < nt; k++) {

if (NT[k] == G[i][j + 1]) {

for (l = 0; FIRST[k][l] != '\0'; l++) {

if (FIRST[k][l] != '!') add\_symbol(arr,

FIRST[k][l]);

if (FIRST[k][l] == '!') fl = 0;

}

break;

}

}

} else if (G[i][j + 1] != '\0' && !isupper(G[i][j + 1])) { add\_symbol(arr, G[i][j + 1]);

}

if ((G[i][j + 1] == '\0' || fl == 0) && G[i][0] != ch) { fl = 1;

FIND\_FOLLOW(arr, G[i][0]);

}

}

}

}

}

void add\_symbol(char \*arr, char ch) { int i, flag = 0;

for (i = 0; arr[i] != '\0'; i++) { if (ch == arr[i]) {

flag = 1; break;

}

}

if (flag == 0) { arr[count++] = ch; arr[count] = '\0';

}

}

int CREATE\_LL1\_TABLE() {

int i, j, k, fl, pos, flag = 0; char arr[100];

for (i = 0; i < cr; i++) { arr[0] = '\0';

count = 0; FIND\_FIRST(arr, G[i][3]);

for (j = 0; j < count; j++) { if (arr[j] == '!') {

FIND\_FOLLOW(arr, G[i][0]);

break;

}

}

for (k = 0; k < nt; k++) { if (NT[k] == G[i][0]) {

pos = k; break;

}

}

for (j = 0; j < count; j++) { if (arr[j] != '!') {

for (k = 0; k < t; k++) { if (arr[j] == T[k]) {

if (LL1[pos][k] > 0) {

printf("\nConflict between %s and %s!", G[LL1[pos][k] -

1], G[i]);

printf("\nGrammar is not LL(1).\n"); return 1;

} else {

LL1[pos][k] = i + 1;

}

break;

}

}

}

}

}

return flag;

}

void LL1\_PARSER(char \*STR) { int i = 0, pos, pos1, n; STR[strlen(STR)] = '$'; STACK[top++] = '$'; STACK[top] = G[0][0];

printf("\nParsing sequence and actions\n\n"); printf("STACK\t\t\tINPUT\t\t\tACTION\n");

printf(" \n"); while (STACK[top] != '$') {

for (int j = 0; STACK[j] != '\0'; j++) printf("%c ", STACK[j]); printf("\t\t");

for (int j = i; STR[j] != '\0'; j++) printf("%c ", STR[j]); if (STR[i] == STACK[top]) {

printf("\t\tMatch: %c", STACK[top]); STACK[top--] = '\0';

i++;

} else {

for (pos = 0; pos < nt; pos++) if (STACK[top] == NT[pos]) break; for (pos1 = 0; pos1 < t; pos1++) if (STR[i] == T[pos1]) break; n = LL1[pos][pos1];

if (n == 0) {

printf("\nError: Invalid input string.\n"); return;

}

printf("\t\tApply: %s", G[n - 1]); STACK[top--] = '\0';

for (int k = strlen(G[n - 1]) - 1; k >= 3; k--) {

if (G[n - 1][k] != '!') STACK[++top] = G[n - 1][k];

}

}

printf("\n");

}

if (STACK[top] == '$' && STR[i] == '$') { printf("\nParsing completed successfully.\n");

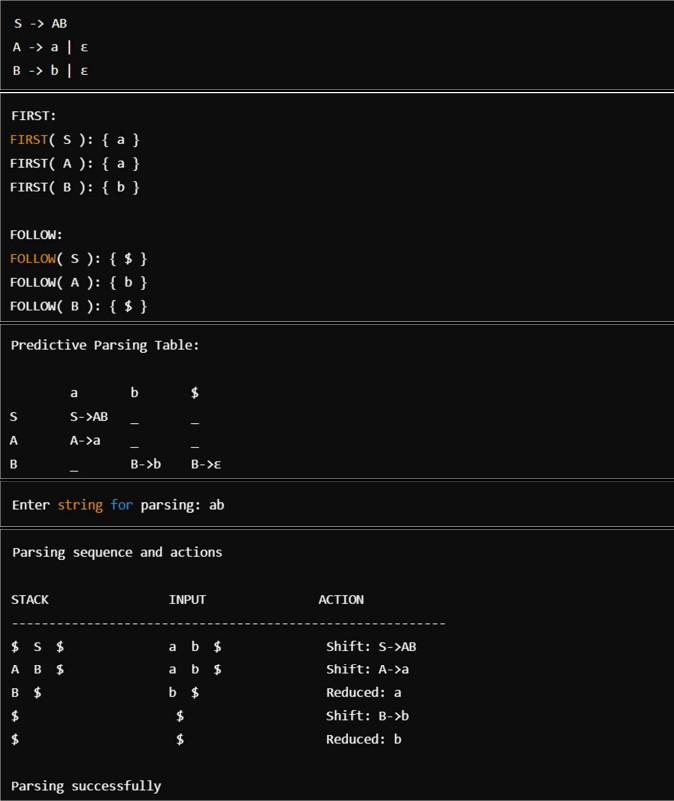
} else {

printf("\nError: Parsing unsuccessful.\n");

}

}

# Output:



## Write a program in ‘lex’ to count no of keywords

%{

#include <stdio.h> #include <string.h>

int keyword\_count = 0; char \*keywords[] = {

"auto", "break", "case", "char", "const", "continue", "default", "do", "double",

"else", "enum", "extern",

"float", "for", "goto", "if", "inline", "int", "long", "register", "restrict", "return", "short", "signed",

"sizeof", "static", "struct", "switch", "typedef", "union", "unsigned", "void", "volatile", "while", NULL

};

%}

%%

[a-zA-Z\_][a-zA-Z0-9\_]\* {

for (int i = 0; keywords[i] != NULL; i++) { if (strcmp(yytext, keywords[i]) == 0) {

keyword\_count++; break;

}

}

}

[ \t\n]+

.|\n

%%

int main() {

printf("Enter text (Ctrl+D to end):\n"); yylex();

printf("Number of keywords: %d\n", keyword\_count); return 0;

}

int yywrap(){ return 1;

}

# Output:



## Write a program in ‘lex’ to check numbers divisible by 7

%{

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

int isDivisibleBy7(int num) { return (num % 7 == 0);

}

%}

%%

[0-9]+ {

int num = atoi(yytext);

if (isDivisibleBy7(num)) {

printf("%d is divisible by 7.\n", num);

} else {

printf("%d is not divisible by 7.\n", num);

}

}

\n ;

. ;

%%

int main() {

printf("Enter the number to check if they are divisible by 7 :\n"); yylex();

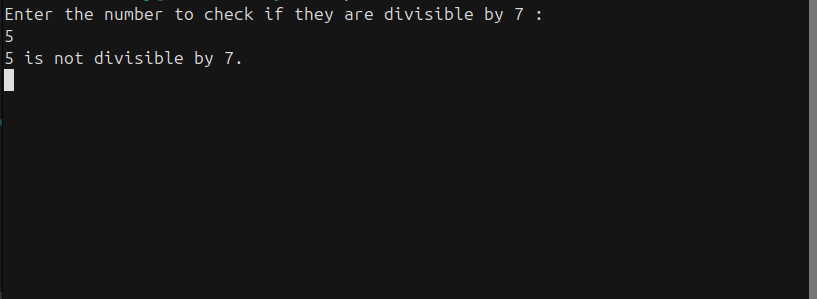
return 0;

}

int yywrap() { return 1;

}

# Output:



## Write a program in ‘lex’ to count no of vowels and consonants

%{

#include <stdio.h>

int vowel\_count = 0;

int consonant\_count = 0;

%}

%%

[aAeEiIoOuU] { vowel\_count++; } [bBcCdDfFgGhHjJkKlLmMnNpPqQrRsStTvVwWxXyYzZ] { consonant\_count++; }

[ \t\n]+

.|\n

%%

int main() {

printf("Enter text (Ctrl+D to end):\n"); yylex();

printf("Number of vowels: %d\n", vowel\_count); printf("Number of consonants: %d\n", consonant\_count); return 0;

}

int yywrap(){ return 1;

}

# Output:



## Write a ‘lex’ program to count no of keywords, identifiers, numbers

%{

#include <stdio.h> #include <string.h>

int identifier\_count = 0; int number\_count = 0;

int keyword\_count = 0;

%}

%option noyywrap

%{

char \*keywords[] = {"if", "else", "while", "for", "int", "float", "return", "break", "continue", "void"};

#define NUM\_KEYWORDS (sizeof(keywords)/sizeof(keywords[0]))

%}

%%

[a-zA-Z\_][a-zA-Z0-9\_]\* {

int is\_keyword = 0;

for (int i = 0; i < NUM\_KEYWORDS; i++) {

if (strcmp(yytext, keywords[i]) == 0) { is\_keyword = 1;

keyword\_count++; break;

}

}

if (!is\_keyword) { identifier\_count++;

}

}

[0-9]+ { number\_count++; } [ \t\n\r]+

.

%%

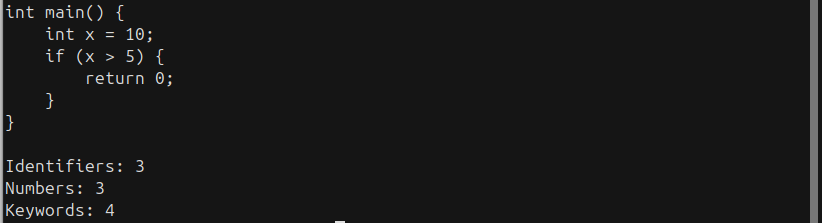
int main() {

yylex();

printf("Identifiers: %d\n", identifier\_count); printf("Numbers: %d\n", number\_count); printf("Keywords: %d\n", keyword\_count); return 0;

}

# Output:



## Write a ‘lex’ program to count no of articles

%{

#include <stdio.h> #include <string.h> int article\_count = 0;

%}

%%

[aA]([ \t]+) { article\_count++; }

[aA]([ \t]+)?[nN] { article\_count++; } [tT]he { article\_count++; }

\n ;

. ;

%%

int main() {

printf("Enter text (Ctrl+D to end):\n"); yylex();

printf("Number of articles (a, an, the): %d\n", article\_count); return 0;

}

int yywrap(){ return 1;

}

# Output:



## Write a C program to generate 3 address code for assignment states

#include <stdio.h> #include <string.h> #include <ctype.h>

int temp\_count = 1;

void new\_temp(char \*temp) { sprintf(temp, "t%d", temp\_count++);

}

void generate\_three\_address\_code(char \*lhs, char \*rhs) { char temp1[3], temp2[3];

char operand1[10], operand2[10]; char operator1, operator2;

if (sscanf(rhs, "%s %c %s %c %s", operand1, &operator1, operand2, &operator2, rhs) == 5) {

if (operator1 == '\*' || operator2 == '\*') { if (operator1 == '\*') {

new\_temp(temp1);

printf("%s = %s %c %s\n", temp1, operand1, operator1, operand2); printf("%s = %s %c %s\n", lhs, temp1, operator2, rhs);

} else {

new\_temp(temp1);

printf("%s = %s %c %s\n", temp1, operand2, operator2, rhs); printf("%s = %s %c %s\n", lhs, operand1, operator1, temp1);

}

} else {

new\_temp(temp1);

printf("%s = %s %c %s\n", temp1, operand1, operator1, operand2); printf("%s = %s %c %s\n", lhs, temp1, operator2, rhs);

}

} else if (sscanf(rhs, "%s %c %s", operand1, &operator1, operand2) == 3) { new\_temp(temp1);

printf("%s = %s %c %s\n", temp1, operand1, operator1, operand2); printf("%s = %s\n", lhs, temp1);

} else {

printf("Invalid expression format!\n");

}

}

int main() {

char expr[50];

char lhs[10], rhs[30];

printf("Enter an assignment statement (e.g., a = b + c \* d): "); fgets(expr, sizeof(expr), stdin);

expr[strcspn(expr, "\n")] = 0;

if (sscanf(expr, "%s = %[^\n]", lhs, rhs) == 2) { printf("Three Address Code:\n"); generate\_three\_address\_code(lhs, rhs);

} else {

printf("Invalid input format!\n");

}

return 0;

}

# Output:

