**Left recursion -**

#include <stdio.h>

#include <string.h>

#define MAX\_PRODUCTIONS 10

#define MAX\_LENGTH 10

void eliminateLeftRecursion(char nonTerminal, char productions[][MAX\_LENGTH], int n) {

char alpha[MAX\_PRODUCTIONS][MAX\_LENGTH], beta[MAX\_PRODUCTIONS][MAX\_LENGTH];

int alphaCount = 0, betaCount = 0;

// Splitting into α and β productions

for (int i = 0; i < n; i++) {

if (productions[i][0] == nonTerminal) { // Left-recursive case

strcpy(alpha[alphaCount++], productions[i] + 1);

} else { // Non-recursive case

strcpy(beta[betaCount++], productions[i]);

}

}

// If no left recursion

if (alphaCount == 0) {

printf("No Left Recursion Detected!\n");

return;

}

// Define new non-terminal (A')

char newNonTerminal = nonTerminal + 1; // Example: 'A' -> 'B'

printf("\nGrammar after eliminating left recursion:\n");

// Printing A → β A'

for (int i = 0; i < betaCount; i++) {

printf("%c -> %s %c'\n", nonTerminal, beta[i], newNonTerminal);

}

// Printing A' → α A' | ε

for (int i = 0; i < alphaCount; i++) {

printf("%c' -> %s %c'\n", newNonTerminal, alpha[i], newNonTerminal);

}

printf("%c' -> ε\n", newNonTerminal);

}

int main() {

char nonTerminal;

int n;

char productions[MAX\_PRODUCTIONS][MAX\_LENGTH];

// User Input

printf("Enter the non-terminal: ");

scanf(" %c", &nonTerminal);

printf("Enter the number of productions: ");

scanf("%d", &n);

printf("Enter the productions (e.g., Aα or β):\n");

for (int i = 0; i < n; i++) {

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

}

// Process the input

eliminateLeftRecursion(nonTerminal, productions, n);

return 0;

}

**First and Follow -**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX 10

char prod[MAX][MAX], first[MAX][MAX], follow[MAX][MAX];

int n;

// Function to find FIRST set of a given non-terminal

void findFirst(char c, int idx) {

for (int i = 0; i < n; i++) {

if (prod[i][0] == c) {

if (!isupper(prod[i][2])) { // If terminal, add directly

int len = strlen(first[idx]);

first[idx][len] = prod[i][2];

first[idx][len + 1] = '\0'; // Null-terminate

} else { // If non-terminal, recurse

findFirst(prod[i][2], idx);

}

}

}

}

void findFollow(char c, int idx) {

if (idx == 0) { // Start symbol gets '$'

int len = strlen(follow[idx]);

follow[idx][len] = '$';

follow[idx][len + 1] = '\0';

}

for (int i = 0; i < n; i++) {

for (int j = 2; j < strlen(prod[i]); j++) {

if (prod[i][j] == c) {

// If there is a symbol after the current symbol

if (prod[i][j + 1] != '\0') {

findFirst(prod[i][j + 1], idx);

strcat(follow[idx], first[i]); // Add FIRST of next symbol

} else {

// If at the end, inherit FOLLOW of LHS

for (int k = 0; k < n; k++) {

if (prod[i][0] == prod[k][0]) {

strcat(follow[idx], follow[k]);

}

}

}

}

}

}

}

int main() {

printf("Enter number of productions: ");

scanf("%d", &n);

printf("Enter productions (Format: A=α):\n");

for (int i = 0; i < n; i++) {

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

first[i][0] = '\0'; // Initialize first and follow sets

follow[i][0] = '\0';

}

// Compute FIRST and FOLLOW sets

for (int i = 0; i < n; i++) findFirst(prod[i][0], i);

for (int i = 0; i < n; i++) findFollow(prod[i][0], i);

// Print FIRST sets

printf("\nFIRST sets:\n");

for (int i = 0; i < n; i++)

printf("FIRST(%c) = { %s }\n", prod[i][0], first[i]);

// Print FOLLOW sets

printf("\nFOLLOW sets:\n");

for (int i = 0; i < n; i++)

printf("FOLLOW(%c) = { %s }\n", prod[i][0], follow[i]);

return 0;

}

Simple calculator -

1A. Simple Calculator in C

This program performs basic arithme􀆟c opera􀆟ons (+, -, \*, /) on two numbers.

Code:

#include <stdio.h>

int main() {

char op;

double num1, num2, result;

prin􀆞("Enter an operator (+, -, \*, /): ");

scanf(" %c", &op);

prin􀆞("Enter two numbers: ");

scanf("%lf %lf", &num1, &num2);

switch (op) {

case '+': result = num1 + num2; break;

case '-': result = num1 - num2; break;

case '\*': result = num1 \* num2; break;

case '/': result = (num2 != 0) ? num1 / num2 : 0; break;

default: prin􀆞("Invalid operator\n"); return 1;

}

prin􀆞("Result: %.2lf\n", result);

return 0;

}

**Program to Check for Capital Letters**

%{

int count = 0;

%}

%%

[A-Z] { printf("%s capital letter\n", yytext);

count++; }

. { printf("%s not a capital letter\n", yytext); }

\n { return 0; }

%%

int yywrap() { return 1; }

int main() {

yylex();

printf("\nNumber of Capital letters in the given input - %d\n", count);

return 0;

}

**2) Program to Count the Number of Lines and Characters**

%{

int no\_of\_lines = 0;

int no\_of\_chars = 0;

%}

%%

\n { ++no\_of\_lines; }

. { ++no\_of\_chars; }

%%

int yywrap() { return 1; }

int main() {

yylex();

printf("Number of lines = %d, Number of characters = %d\n", no\_of\_lines, no\_of\_chars);

return 0;

}

**Lex Program to Count the Total Number of Characters**

%{

int char\_count = 0;

%}

%%

. { char\_count++; } // Increment count for every character

\n { char\_count++; } // Count newline characters too

%%

int yywrap() { return 1; }

int main() {

yylex();

printf("Total number of characters: %d\n", char\_count);

return 0;

}

**Lex Program to Count the Number of Words**

%{

int word\_count = 0;

%}

%%

[^\t\n ]+ { word\_count++; } // Match any non-space character sequence as a word

\n { } // Ignore newlines

%%

int yywrap() { return 1; }

int main() {

yylex();

printf("Total number of words: %d\n", word\_count);

return 0;

}

**A simple lexer that recognizes iden􀆟fiers, numbers, and operators.**

Code (lexer.l):

%{

#include <stdio.h>

%}

%%

[a-zA-Z\_][a-zA-Z0-9\_]\* { prin􀆞("IDENTIFIER: %s\n", yytext); }

[0-9]+ { prin􀆞("NUMBER: %s\n", yytext); }

[+\-\*/=] { prin􀆞("OPERATOR: %s\n", yytext); }

. { prin􀆞("OTHER: %s\n", yytext); }

%%

int main() {

yylex();

return 0;

}

int yywrap() { return 1; }

**Lex Program to Count the Number of Lines, Spaces, and Tabs**

%{

int line\_count = 0, space\_count = 0, tab\_count = 0;

%}

%%

\n { line\_count++; } // Count new lines

" " { space\_count++; } // Count spaces

"\t" { tab\_count++; } // Count tabs

%%

int yywrap() { return 1; }

int main() {

yylex();

printf("Total lines: %d\n", line\_count);

printf("Total spaces: %d\n", space\_count);

printf("Total tabs: %d\n", tab\_count);

return 0;

}

**Simple calculator program**

%{

#include <stdio.h>

#include <stdlib.h>

int yylval;

%}

%%

[0-9]+ { yylval = atoi(yytext); return 'n'; } // Recognize numbers

[+\-\*/()] { return yytext[0]; } // Operators and parentheses

\n { return 0; } // End of input

[ \t] { } // Ignore spaces and tabs

. { printf("Invalid character: %s\n", yytext); } // Handle invalid characters

%%

int yywrap() { return 1; }

**Calculator Program for Lex and Bison**

Step 1: Create the Lexer (calc.l)

%{

#include "y.tab.h"

%}

%%

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

[+\-\*/] { return yytext[0]; }

\n { return 0; }

. { return yytext[0]; }

%%

int yywrap() { return 1; }

Step 2: Create the Parser (calc.y)

%{

#include <stdio.h>

#include <stdlib.h>

%}

%token NUMBER

%%

expression: expression '+' expression { prin􀆞("%d\n", $1 + $3); }

| expression '-' expression { prin􀆞("%d\n", $1 - $3); }

| expression '\*' expression { prin􀆞("%d\n", $1 \* $3); }

| expression '/' expression { prin􀆞("%d\n", $1 / $3); }

| NUMBER { $$ = $1;

} ;

%%

int main() {

yyparse();

return 0;

}

void yyerror(const char \*s) {

prin􀆞("Error: %s\n", s);

}

How to Compile and Run:

bison -d calc.y

lex calc.l

gcc lex.yy.c calc.tab.c -o calc -ll

./calc

Enter expressions like 5+3, and it will evaluate them.