1. Write a Lex program to check the given number is prime number or not

Lex Program to Check for Prime Number:

%{

#include <stdio.h>

#include <stdlib.h>

int is\_prime(int num);

%}

%%

// Match a number

[0-9]+ {

int num = atoi(yytext);

if (is\_prime(num)) {

printf("%d is a prime number.\n", num);

} else {

printf("%d is not a prime number.\n", num);

}

}

.|\n { /\* Ignore other characters and new lines \*/ }

%%

// Function to determine if a number is prime

int is\_prime(int num) {

if (num <= 1) return 0; // 0 and 1 are not prime

for (int i = 2; i\*i <= num; i++) {

if (num % i == 0) return 0; // Not prime

}

return 1; // Prime

}

int main() {

yylex(); // Start the lexical analysis

return 0;

}

1. Write a Lex program to count number of words start with “a”.

Lex Program to Count Words Starting with "a"

%{

#include <stdio.h>

int count = 0; // Variable to hold the count of words starting with 'a'

%}

%%

// Match words starting with 'a' or 'A'

[aA][a-zA-Z]\* { count++; }

// Match any other words (not starting with 'a' or 'A')

[a-zA-Z]+ { /\* Ignore these words \*/ }

// Ignore whitespace and punctuation

[ \t\n]+ { /\* Ignore whitespace \*/ }

[.,!?;] { /\* Ignore punctuation \*/ }

%%

// Main function

int main() {

yylex(); // Start lexical analysis

printf("Number of words starting with 'a': %d\n", count);

return 0;

}

1. Write a Lex program to check no.of websites ending with “.com”

Lex Program to Count Websites Ending with ".com"

%{

#include <stdio.h>

int count = 0; // Variable to hold the count of websites ending with '.com'

%}

%%

// Match URLs ending with '.com'

[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.com\b { count++; }

// Match any other text (not ending with '.com')

[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,} { /\* Ignore these \*/ }

// Ignore any other characters

.|\n { /\* Ignore all other characters \*/ }

%%

// Main function

int main() {

yylex(); // Start lexical analysis

printf("Number of websites ending with '.com': %d\n", count);

return 0;

}

1. Write a Lex program to check the given Mobile Number is valid or not.

Lex Program to Check whether the given Mobile Number is valid or not.

%{

#include <stdio.h>

#include <stdlib.h>

int valid\_count = 0; // Variable to hold the count of valid mobile numbers

%}

%%

// Match mobile numbers with optional country code and exactly 10 digits

(\+?\d{1,3}[- ]?)?\d{10} { valid\_count++; printf("%s is a valid mobile number.\n", yytext); }

// Match any invalid number format

[^0-9\n]+ { /\* Ignore non-digit characters \*/ }

// Ignore newlines

\n { /\* Ignore newlines \*/ }

%%

// Main function

int main() {

printf("Enter mobile numbers (one per line). Press Ctrl+D (EOF) to finish:\n");

yylex(); // Start lexical analysis

printf("Total valid mobile numbers: %d\n", valid\_count);

return 0;

}

1. Write a Lex program to check whether the given number is even or odd.

Lex Program to Check Even or Odd Numbers:

%{

#include <stdio.h>

#include <stdlib.h>

void check\_even\_odd(int num);

%}

%%

// Match any integer number

[0-9]+ {

int num = atoi(yytext);

check\_even\_odd(num);

}

// Match any non-digit characters (ignore)

.|\n { /\* Ignore all other characters \*/ }

%%

// Function to check if a number is even or odd

void check\_even\_odd(int num) {

if (num % 2 == 0) {

printf("%d is even.\n", num);

} else {

printf("%d is odd.\n", num);

}

}

int main() {

printf("Enter numbers (one per line). Press Ctrl+D (EOF) to finish:\n");

yylex(); // Start lexical analysis

return 0;

}

1. Write a Lex program to convert decimal to binary number.

Lex Program to Convert Decimal to Binary:

%{

#include <stdio.h>

#include <stdlib.h>

void decimal\_to\_binary(int num);

%}

%%

// Match any integer number

[0-9]+ {

int num = atoi(yytext);

decimal\_to\_binary(num);

}

// Match any non-digit characters (ignore)

.|\n { /\* Ignore all other characters \*/ }

%%

// Function to convert decimal to binary

void decimal\_to\_binary(int num) {

if (num == 0) {

printf("Binary: 0\n");

return;

}

int binary[32]; // Array to store binary digits

int index = 0;

while (num > 0) {

binary[index] = num % 2; // Store remainder (0 or 1)

num = num / 2; // Divide the number by 2

index++;

}

// Print binary digits in reverse order

printf("Binary: ");

for (int i = index - 1; i >= 0; i--) {

printf("%d", binary[i]);

}

printf("\n");

}

int main() {

printf("Enter decimal numbers (one per line). Press Ctrl+D (EOF) to finish:\n");

yylex(); // Start lexical analysis

return 0;

}

1. Write a Lex program to do arithmetic operations “+,-,\*,^,/”

Lex Program for Basic Arithmetic Operations:

%{

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

void perform\_operation(double num1, char operator, double num2);

%}

%%

// Match numbers (integers or decimals)

[0-9]+(\.[0-9]+)? {

yylval = atof(yytext);

return NUMBER;

}

// Match arithmetic operators

[\+\-\\*/\^] {

return \*yytext;

}

// Ignore whitespace

[ \t\n]+ { /\* Ignore whitespace \*/ }

// Handle invalid input

. {

printf("Invalid character: %s\n", yytext);

}

%%

// Define tokens

enum { NUMBER = 256 };

int main() {

double num1, num2;

char operator;

printf("Enter expressions (e.g., 2 + 3, 4 - 5, etc.). Press Ctrl+D (EOF) to finish:\n");

while (1) {

if (scanf("%lf %c %lf", &num1, &operator, &num2) != 3) {

printf("Invalid input format. Please enter in the format 'number operator number'.\n");

while(getchar() != '\n'); // Clear the input buffer

continue;

}

perform\_operation(num1, operator, num2);

}

return 0;

}

// Function to perform the arithmetic operation

void perform\_operation(double num1, char operator, double num2) {

double result;

switch (operator) {

case '+':

result = num1 + num2;

printf("%.2lf + %.2lf = %.2lf\n", num1, num2, result);

break;

case '-':

result = num1 - num2;

printf("%.2lf - %.2lf = %.2lf\n", num1, num2, result);

break;

case '\*':

result = num1 \* num2;

printf("%.2lf \* %.2lf = %.2lf\n", num1, num2, result);

break;

case '/':

if (num2 != 0) {

result = num1 / num2;

printf("%.2lf / %.2lf = %.2lf\n", num1, num2, result);

} else {

printf("Error: Division by zero.\n");

}

break;

case '^':

result = pow(num1, num2);

printf("%.2lf ^ %.2lf = %.2lf\n", num1, num2, result);

break;

default:

printf("Invalid operator: %c\n", operator);

break;

}

}