1. Design a LEX Code to count the number of lines, space, tab-meta character and rest of characters in a given Input pattern.

%{

#include <stdio.h>

int lines = 0;

int spaces = 0;

int tabs = 0;

int other\_chars = 0;

%}

%%

\n { lines++; }

[ ] { spaces++; }

\t { tabs++; }

. { other\_chars++; }

%%

int main() {

yylex();

printf("Lines: %d\n", lines);

printf("Spaces: %d\n", spaces);

printf("Tabs: %d\n", tabs);

printf("Other characters: %d\n", other\_chars);

return 0;

}

int yywrap() {

return 1;

}

2.Design a LEX Code to identify and print valid Identifier of C/C++ in given Input pattern.

%{

#include <stdio.h>

// Define a buffer to store and print identifiers

char identifier[256];

%}

%%

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

// Match a valid C/C++ identifier

strncpy(identifier, yytext, yyleng);

identifier[yyleng] = '\0';

printf("Valid identifier: %s\n", identifier);

}

. ; // Ignore other characters

%%

int main() {

yylex();

return 0;

}

int yywrap() {

return 1;

}

3.Design a LEX Code to identify and print integer and float value in given Input pattern.

%{

#include <stdio.h>

%}

%%

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

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

}

[0-9]+ {

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

}

. ; // Ignore other characters

%%

int main() {

yylex();

return 0;

}

int yywrap() {

return 1;

}

1. Design a LEX Code for Tokenizing (Identify and print OPERATORS, SEPERATORS, KEYWORDS, IDENTIFERS) the following C-fragment:

int p=1,d=0,r=4;

float m=0.0, n=200.0;

while (p <= 3)

{ if(d==0)

{ m= m+n\*r+4.5; d++; }

else

{ r++; m=m+r+1000.0; }

p++; }

%{

#include <stdio.h>

#include <string.h>

// Function prototypes

void print\_token(const char\* token\_type, const char\* token);

%}

%%

// Keywords

"int"|"float"|"while"|"if"|"else" { print\_token("KEYWORD", yytext); }

// Operators

"=="|"="|"<="|"<"|">"|"\+"|"\-|\\*|\/" { print\_token("OPERATOR", yytext); }

// Separators

"("|")"|"{"|"}"|";"|"," { print\_token("SEPARATOR", yytext); }

// Identifiers

[a-zA-Z\_][a-zA-Z0-9\_]\* { print\_token("IDENTIFIER", yytext); }

// Ignore whitespaces

[ \t\n]+ { /\* ignore whitespaces \*/ }

// Anything else (should not occur in well-formed C code)

. { printf("Unrecognized character: %s\n", yytext); }

%%

void print\_token(const char\* token\_type, const char\* token) {

printf("%s: %s\n", token\_type, token);

}

int main() {

yylex();

return 0;

}

int yywrap() {

return 1;

}

1. Design a LEX Code to count and print the number of total characters, words, white spaces in given ‘Input.txt’ file.

%{

#include <stdio.h>

int total\_chars = 0;

int total\_words = 0;

int total\_whitespaces = 0;

int in\_word = 0;

%}

%%

[ \t\n]+ {

total\_whitespaces += yyleng;

in\_word = 0;

}

[a-zA-Z0-9]+ {

total\_words++;

in\_word = 1;

}

. {

total\_chars++;

if (!in\_word) {

total\_chars++;

}

}

%%

int main(int argc, char \*\*argv) {

if (argc > 1) {

FILE \*file = fopen(argv[1], "r");

if (!file) {

perror("Error opening file");

return 1;

}

yyin = file;

yylex();

fclose(file);

} else {

printf("Usage: %s <input\_file>\n", argv[0]);

return 1;

}

printf("Total characters: %d\n", total\_chars);

printf("Total words: %d\n", total\_words);

printf("Total white spaces: %d\n", total\_whitespaces);

return 0;

}

int yywrap() {

return 1;

}

1. Design a LEX Code to replace white spaces of ‘Input.txt’ file by a single blank character into ‘Output.txt’ file.

%{

#include <stdio.h>

FILE \*output\_file;

%}

%%

[ \t\n]+ {

fprintf(output\_file, " ");

}

. {

fputc(yytext[0], output\_file);

}

%%

int main(int argc, char \*\*argv) {

if (argc > 1) {

FILE \*input\_file = fopen(argv[1], "r");

if (!input\_file) {

perror("Error opening input file");

return 1;

}

output\_file = fopen("Output.txt", "w");

if (!output\_file) {

perror("Error opening output file");

fclose(input\_file);

return 1;

}

yyin = input\_file;

yylex();

fclose(input\_file);

fclose(output\_file);

} else {

printf("Usage: %s <input\_file>\n", argv[0]);

return 1;

}

return 0;

}

int yywrap() {

return 1;

}

1. design a lex code to design a dfa which accepts string having aba as a substring and ending with bb over input {a,b}.

%{

#include <stdio.h>

int not\_accepted = 0;

int accepted = 0;

%}

%s A B C D E DEAD

%%

<INITIAL>a BEGIN A;

<INITIAL>\n BEGIN INITIAL; {not\_accepted++;}

<INITIAL>[^a|] BEGIN DEAD;

<A>b BEGIN B;

<A>\n BEGIN INITIAL; {not\_accepted++;}

<A>[^b] BEGIN DEAD;

<B>a BEGIN C;

<B>\n BEGIN INITIAL; {not\_accepted++;}

<B>[^a] BEGIN DEAD;

<C>a BEGIN C;

<C>b BEGIN D;

<C>\n BEGIN INITIAL; {not\_accepted++;}

<D>b BEGIN E;

<D>a BEGIN C;

<D>\n BEGIN INITIAL; {not\_accepted++;}

<E>b BEGIN E;

<E>a BEGIN C;

<E>\n BEGIN INITIAL; {accepted++;}

<DEAD>\n BEGIN INITIAL; {not\_accepted++;}

<DEAD>. BEGIN DEAD;

%%

int yywrap(){

return 1;

}

int main(){

yylex();

printf("\nAccepted: %d\nNot Accepted: %d\n",accepted, not\_accepted);

return 0;

}

1. design a lex code to design a dfa which contain substring 00.

%{

#include <stdio.h>

%}

%s B C

%%

<INITIAL>0 BEGIN B;

<INITIAL>1 BEGIN INITIAL;

<INITIAL>\n BEGIN INITIAL; {printf(“Not Accepted”);}

<INITIAL>[^01] BEGIN INITIAL; {printf(“Not Accepted”);}

<B>0 BEGIN C;

<B>1 BEGIN INITIAL;

<B>\n BEGIN B; {printf(“Not Accepted”);}

<B>[^01] BEGIN B; {printf(“Not Accepted”);}

<C>0 BEGIN C;

<C>1 BEGIN C;

<C>\n BEGIN C; {printf(“Accepted”);}

<C>[^01] BEGIN C; {printf(“Not Accepted”);}

%%

int yywrap(){

return 1;

}

int main(){

printf(“enter the string\n”);

yylex();

return 0;

}

1. Design a DFA in LEX Code which accepts string containing even number of ‘a’ and even number of ‘b’ over input alphabet {a, b}.

%{

#include <stdio.h>

// DFA states

typedef enum { S0, S1, S2, S3 } State;

State current\_state = S0;

%}

%%

a {

switch (current\_state) {

case S0: current\_state = S2; break;

case S1: current\_state = S3; break;

case S2: current\_state = S0; break;

case S3: current\_state = S1; break;

}

}

b {

switch (current\_state) {

case S0: current\_state = S1; break;

case S1: current\_state = S0; break;

case S2: current\_state = S3; break;

case S3: current\_state = S2; break;

}

}

. { /\* Ignore other characters \*/ }

%%

int main() {

yylex();

if (current\_state == S0) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

int yywrap() {

return 1;

}

1. Design a DFA in LEX Code which accepts string containing third last element ‘a’ over input alphabet {a, b}.

%{

#include <stdio.h>

#include <string.h>

#define BUFFER\_SIZE 4 // We need to track the last 3 characters plus a null terminator

char buffer[BUFFER\_SIZE] = { '\0', '\0', '\0', '\0' };

int length = 0;

void shift\_and\_append(char c) {

// Shift the buffer to the left and append the new character

for (int i = 0; i < BUFFER\_SIZE - 2; i++) {

buffer[i] = buffer[i + 1];

}

buffer[BUFFER\_SIZE - 2] = c;

buffer[BUFFER\_SIZE - 1] = '\0';

if (length < BUFFER\_SIZE - 1) {

length++;

}

}

%}

%%

[a-b] {

shift\_and\_append(yytext[0]);

}

. { /\* Ignore other characters \*/ }

%%

int main() {

yylex();

if (length >= 3 && buffer[0] == 'a') {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

int yywrap() {

return 1;

}