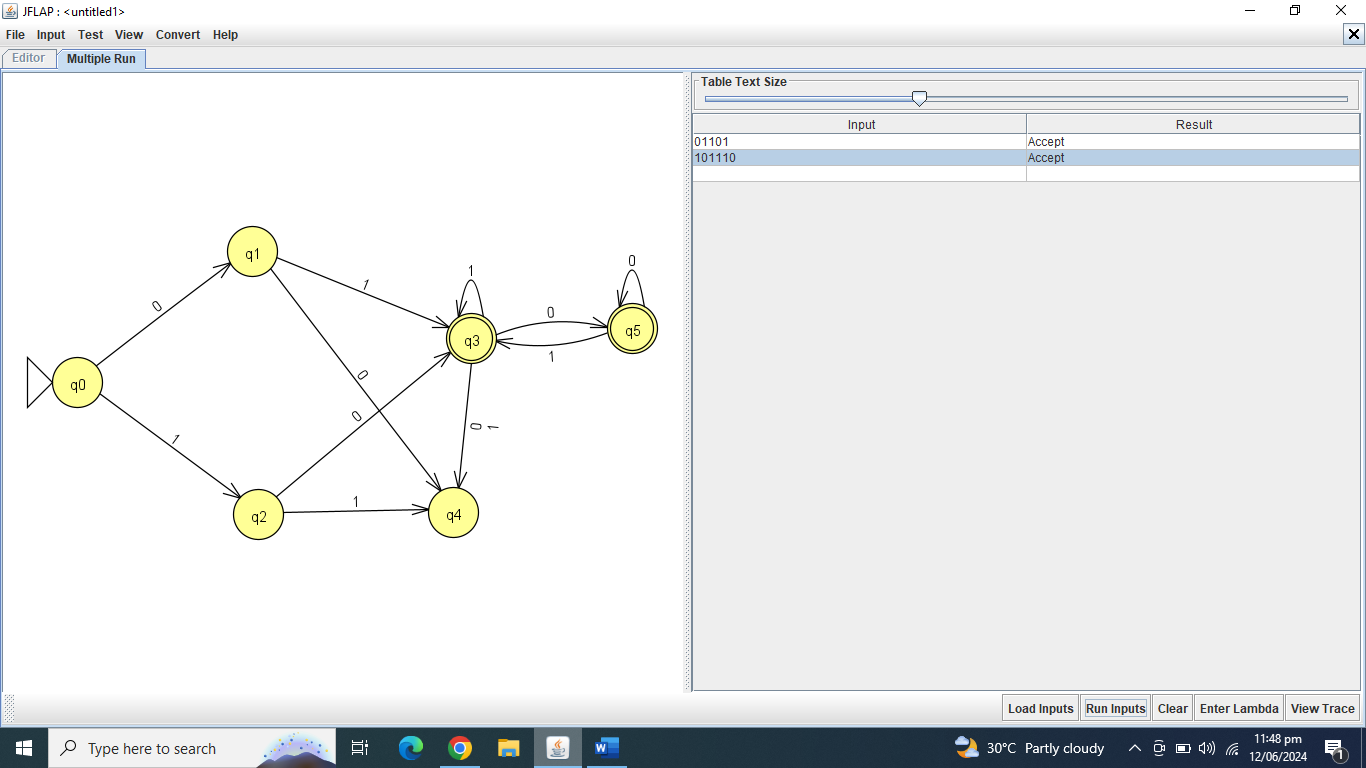
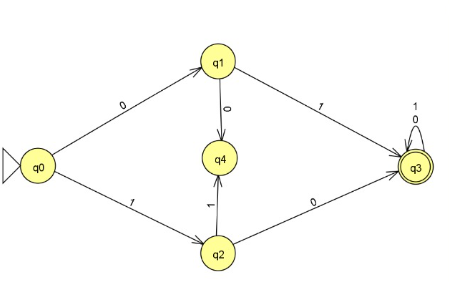
Draw a DFA using JFLAP which accepts strings start with 01 and 10 only.





Write a C++ program using regex library, which will prompt the user for input If user input any Variable identifier(starts will any alphabet) then output will be “you have entered a Variable Identifier”, If user input any number then output will be “you have entered a Number”, if user input any Keyword(if, for, else, while, include, Goto) then output will be “you have entered a keyword” and if the user input other than these then output will be “Invalid Input”.

#include <iostream>

#include <string>

#include <regex>

int main() {

std::string input;

std::regex identifier("[a-zA-Z][a-zA-Z0-9\_]\*");

std::regex number("[0-9]+");

std::regex keywords("(if|for|else|while|include|goto)");

std::cout << "Enter input: ";

std::getline(std::cin, input);

if (std::regex\_match(input, identifier)) {

std::cout << "You have entered a Variable Identifier.\n";

} else if (std::regex\_match(input, number)) {

std::cout << "You have entered a Number.\n";

} else if (std::regex\_match(input, keywords)) {

std::cout << "You have entered a Keyword.\n";

} else {

std::cout << "Invalid Input.\n";

}

return 0;

}

Write a find and replace C++ program using regex library. Program will get input any sentence, then ask any word within sentence which you want to replace and then a new word to replace.

#include <iostream>

#include <string>

#include <regex>

int main() {

std::string sentence;

std::string word\_to\_replace;

std::string replacement\_word;

// Get input sentence

std::cout << "Enter a sentence: ";

std::getline(std::cin, sentence);

// Get word to replace

std::cout << "Enter the word you want to replace: ";

std::cin >> word\_to\_replace;

// Get replacement word

std::cout << "Enter the replacement word: ";

std::cin >> replacement\_word;

// Define regular expression pattern for the word to replace

std::regex word\_regex("\\b" + word\_to\_replace + "\\b");

// Replace the word in the sentence

std::string new\_sentence = std::regex\_replace(sentence, word\_regex, replacement\_word);

// Output the modified sentence

std::cout << "Modified sentence: " << new\_sentence << std::endl;

return 0;

}

Write a Flex Program than can scan a valid UIT student Id

%{

#include <stdio.h>

#include <stdlib.h>

%}

%%

[0-9]{2}[A-Z]-[0-9]{3}-[A-Z]{2} { printf("Valid UIT student roll number: %s\n", yytext); }

.|\n { printf("Invalid input: %s\n", yytext); }

%%

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

yylex();

return 0;

}

int yywrap() {

return 1;

}

Write a Flex Program that can a valid Car No.

%{

#include <stdio.h>

#include <stdlib.h>

%}

%%

[A-Z]{3}-[0-9]{3} { printf("Valid registration number: %s\n", yytext); }

.|\n { printf("Invalid input: %s\n", yytext); }

%%

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

yylex();

return 0;

}

int yywrap() {

return 1;

}

Write a Yacc Program to evaluate an arithmetic expression involving operating +, -, \* and /, also recognize Divide by zero error.

Lex:

%{

#include "y.tab.h"

%}

%%

[ \t] /\* ignore whitespace \*/

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

. { return yytext[0]; } /\* return other single characters as tokens \*/

%%

int yywrap() {

return 1;

}

Yacc:

%{

#include <stdio.h>

#include <stdlib.h>

%}

%token NUMBER

%%

input: /\* empty \*/

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

;

expr: term

| expr '+' term { $$ = $1 + $3; }

| expr '-' term { $$ = $1 - $3; }

;

term: factor

| term '\*' factor { $$ = $1 \* $3; }

| term '/' factor {

if ($3 == 0) {

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

exit(EXIT\_FAILURE);

} else {

$$ = $1 / $3;

}

}

;

factor:

NUMBER { $$ = $1; }

| '(' expr ')' { $$ = $2; }

;

%%

int main() {

yyparse();

return 0;

}

void yyerror(char \*s) {

printf("Error: %s\n", s);

}

Write a Yacc program to recognize whether a given input is an HTML tag or not.

Lex:

%{

#include "y.tab.h"

%}

%%

"<"[a-zA-Z]+[0-9a-zA-Z]\*">" { yylval = strdup(yytext); return HTML\_TAG; }

.|\n /\* ignore other characters \*/

%%

int yywrap() {

return 1;

}

Yacc:

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

%}

%token HTML\_TAG

%%

input: /\* empty \*/

| input tag '\n' { printf("Tag: %s\n", $2); }

;

tag:

HTML\_TAG { $$ = $1; }

;

%%

int main() {

yyparse();

return 0;

}

void yyerror(char \*s) {

printf("Error: %s\n", s);

}

Write a C++ program that constructs the SYMBOL TABLE for the identifiers and numbers. For identifiers and digits store identifiers and its type. Use following structure to construct STM. Write only Classes Code and Driver Code, Do not need to Member Functions Code here. Variable Name Address Type Dimension Line Declared Line referenced

#include <iostream>

#include <unordered\_map>

#include <string>

using namespace std;

// Structure to store information about identifiers and numbers

struct SymbolInfo {

string address;

string type;

string dimension;

int lineDeclared;

int lineReferenced;

};

// Class to construct the symbol table

class SymbolTable {

private:

unordered\_map<string, SymbolInfo> symbolTable;

public:

// Function to insert an identifier or number into the symbol table

void insertSymbol(string name, string address, string type, string dimension, int lineDeclared, int lineReferenced);

// Function to display the symbol table

void displaySymbolTable();

};

// Driver code

int main() {

SymbolTable symTable;

// Insert symbols into the table

symTable.insertSymbol("x", "0x1234", "int", "", 5, 10);

symTable.insertSymbol("y", "0x5678", "float", "", 7, 12);

symTable.insertSymbol("z", "0x9abc", "char", "", 9, 15);

symTable.insertSymbol("a", "0xdef0", "int", "10x20", 10, 20);

// Display the symbol table

symTable.displaySymbolTable();

return 0;

}

Commands to run flex file

Flex filename.l

Gcc lex.yy.c

a

Command for yacc

Bison/yacc -d filename.y

Gcc filename.tab.c

A

Correct ltask 8

**Task 2** “**you have entered a Variable Identifier”, If user input any number then output will be “you have entered a Number”, if user input any Keyword(if, for, else, while, include, Goto) then output will be “you have entered a keyword” and if the user input other than these then output will be “Invalid Input”..**

#include <iostream>

#include <regex>

#include <string>

using namespace std;

bool is\_keyword(const string& input){

static const vector<string> keywords={"if","else","while","for"};

for(const auto& keyword:keywords){

if(input==keyword){

return true;

}

}

return false;

}

bool is\_number(const string& input){

regex pattern("[0-9]+");

return regex\_match(input,pattern);

}

int main() {

string input;

// Regular expression pattern to validate input (allowing only numbers)

//regex pattern("[0-9]+");

// Prompting the user for input

cout << "Enter a number: ";

cin >> input;

// Checking if the input matches the pattern

if (is\_keyword(input)){

cout <<"keyword";

}

else if(is\_number(input)){

cout<<"number";

}

else{

cout<<"no keyword";

}

return 0;

}

**Task 3 a find and replace C++**

#include <iostream>

#include <regex>

#include <string>

using namespace std;

int main(){

string sentance,oldword,newword;

getline(cin,sentance);

cout<<"Enter the word you want to replace :";

cin>>oldword;

cout<<"Enter the new word";

cin>>newword;

string replace="\\b"+oldword+"\\b";

string result=regex\_replace(sentance,regex(replace),newword);

cout<<result;

return 0;

}

**roll-number wala**

%{

#include <stdio.h>

#include <stdlib.h>

%}

%%

[A-Z]{3}-[0-9]{3} { printf("Valid registration number: %s\n", yytext); }

.|\n { printf("Invalid registration number: %s\n", yytext); }

%%

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

yylex();

return 0;

}

int yywrap() {

return 1;

}

**UIT REGISTRATION ROLL NO**

%{

#include <stdio.h>

#include <stdlib.h>

%}

%%

[0-9]{2}[A-Z]-[0-9]{3}-[A-Z]{2} { printf("Valid UIT registration number: %s\n", yytext); }

.|\n { printf("Invalid registration number: %s\n", yytext); }

%%

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

yylex();

return 0;

}

int yywrap() {

return 1;

}

**Task 4 valid uit roll no :**

%{

#include <stdio.h>

#include <stdlib.h>

%}

%%

[0-9]{2}[A-Z]-[0-9]{3}-[A-Z]{2} { printf("Valid UIT registration number: %s\n", yytext); }

.|\n { printf("Invalid registration number: %s\n", yytext); }

%%

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

yylex();

return 0;

}

int yywrap() {

return 1;

}

**Write a Flex Program that can a valid Car No task 5:**

%{

#include <stdio.h>

#include <stdlib.h>

%}

%%

[A-Z]{3}-[0-9]{3} { printf("Valid registration number: %s\n", yytext); }

.|\n { printf("Invalid registration number: %s\n", yytext); }

%%

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

yylex();

return 0;

}

int yywrap() {

return 1;

}

**Task 8:**

#include <iostream>

#include <vector>

#include <string>

// Class to represent an entry in the symbol table

class SymbolTableEntry {

public:

std::string variableName;

std::string address;

std::string type;

std::string dimension;

int lineDeclared;

int lineReferenced;

};

// Class to manage the symbol table

class SymbolTable {

public:

void insertEntry(const SymbolTableEntry& entry) {

entries.push\_back(entry);

}

void displayTable() const {

std::cout << "Symbol Table:\n";

std::cout << "--------------------------------------------------\n";

std::cout << "Variable Name Address Type Dimension Line Declared Line Referenced\n";

std::cout << "--------------------------------------------------\n";

for (const auto& entry : entries) {

std::cout << entry.variableName << " " << entry.address << " " << entry.type << " "

<< entry.dimension << " " << entry.lineDeclared << " " << entry.lineReferenced << "\n";

}

std::cout << "--------------------------------------------------\n";

}

private:

std::vector<SymbolTableEntry> entries;

};

int main() {

SymbolTable symbolTable;

// Example entries

SymbolTableEntry entry1 = {"var1", "0x1000", "int", "", 10, 20};

SymbolTableEntry entry2 = {"var2", "0x2000", "float", "2x3", 15, 25};

// Insert entries into symbol table

symbolTable.insertEntry(entry1);

symbolTable.insertEntry(entry2);

// Display symbol table

symbolTable.displayTable();

return 0;

}