########TASK (INPUT VARIABLE WALA)################

#include <iostream>

#include <regex>

#include <string>

#include <unordered\_set>

int main() {

std::string input;

std::unordered\_set<std::string> keywords = {"if", "for", "else", "while", "include", "goto"};

std::cout << "Please Input any Variable Identifier, Number or Keyword: ";

std::cin >> input;

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

std::regex numberRegex("^[0-9]+$");

if (keywords.find(input) != keywords.end()) {

std::cout << "You have entered a keyword" << std::endl;

}

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

std::cout << "You have entered a Variable Identifier" << std::endl;

}

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

std::cout << "You have entered a Number" << std::endl;

}

else {

std::cout << "Invalid Input" << std::endl;

}

return 0;

}

##########TASK 2 WORD REPLACE#########33

#include <iostream>

#include <regex>

#include <string>

int main() {

std::string sentence, wordToReplace, replacementWord;

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

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

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

std::cin >> wordToReplace;

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

std::cin >> replacementWord;

std::regex wordRegex("\\b" + wordToReplace + "\\b");

std::string result = std::regex\_replace(sentence, wordRegex, replacementWord);

std::cout << "Modified sentence: " << result << std::endl;

return 0;

}

#########TASK 3#################

#include <iostream>

#include <vector>

#include <string>

// Enumeration for token types

enum TokenType { KEYWORD, OPERATOR, IDENTIFIER, DIGIT };

// Structure to represent a symbol table entry

struct SymbolTableEntry {

std::string lexeme;

TokenType tokenType;

};

// Class to represent the Symbol Table

class SymbolTable {

public:

std::vector<SymbolTableEntry> table;

// Method to add an entry to the symbol table

void addEntry(const std::string& lexeme, TokenType type) {

table.push\_back({lexeme, type});

}

// Method to print the symbol table

void printTable() const {

for (const auto& entry : table) {

std::cout << "Lexeme: " << entry.lexeme << ", Type: " << entry.tokenType << std::endl;

}

}

};

// Driver code

int main() {

// Create an instance of the SymbolTable class

SymbolTable symbolTable;

// Example entries to be added to the symbol table

symbolTable.addEntry("if", KEYWORD);

symbolTable.addEntry("+", OPERATOR);

symbolTable.addEntry("variableName", IDENTIFIER);

symbolTable.addEntry("1234", DIGIT);

// Print the contents of the symbol table

symbolTable.printTable();

return 0;

}

###############TASK 4#######

#include <iostream>

#include <string>

#include <vector>

struct Symbol {

std::string identifier;

std::string type;

};

class SymbolTable {

public:

void insert(const std::string& identifier, const std::string& type);

bool exists(const std::string& identifier);

private:

std::vector<Symbol> symbols;

};

void SymbolTable::insert(const std::string& identifier, const std::string& type) {

symbols.push\_back({identifier, type});

}

bool SymbolTable::exists(const std::string& identifier) {

for (const Symbol& symbol : symbols) {

if (symbol.identifier == identifier) {

return true;

}

}

return false;

}

int main() {

SymbolTable symbolTable;

// Insert some symbols

symbolTable.insert("x", "int");

symbolTable.insert("y", "double");

symbolTable.insert("z", "string");

// Check if a symbol exists

if (symbolTable.exists("x")) {

std::cout << "x exists in the symbol table." << std::endl;

}

return 0;

}

##########TASK 5 FLEX VALID CARD NUMBER###########

%{

#include <stdio.h>

%}

%%

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

. { printf("Invalid Input\n"); }

%%

int main() {

yylex();

return 0;

}

########TASK 6 FLEX UIT CARD###########

%{

#include <stdio.h>

%}

%%

UIT-[0-9]{4}-[A-Z]{2} { printf("Valid UIT Student ID: %s\n", yytext); }

. { printf("Invalid Input\n"); }

%%

int main() {

yylex();

return 0;

}

#########TASK 7 ARITHMETIC##############

%{

#include <stdio.h>

#include <stdlib.h>

%}

%token NUMBER

%left '+' '-'

%left '\*' '/'

%right UMINUS

%%

expression:

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

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

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

| expression '/' expression { if ($3 == 0) printf("Divide by zero error\n"); else printf("%d\n", $1 / $3); }

| '-' expression %prec UMINUS { $$ = -$2; }

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

| NUMBER { $$ = $1; }

;

%%

int main() {

yyparse();

return 0;

}

int yyerror(char \*s) {

fprintf(stderr, "Error: %s\n", s);

return 0;

}

int yylex() {

int c;

while ((c = getchar()) == ' ' || c == '\t'); /\* skip whitespace \*/

if (c == '.' || isdigit(c)) {

ungetc(c, stdin);

scanf("%d", &yylval);

return NUMBER;

}

return c;

}

##########TASK 8 HTML#########

%{

#include <stdio.h>

%}

%token TAG

%start html

%%

html:

'<' TAG '>' { printf("Valid HTML Tag\n"); }

| '<' '/' TAG '>' { printf("Valid HTML Closing Tag\n"); }

| . { printf("Invalid Input\n"); }

;

%%

int main() {

yyparse();

return 0;

}

int yyerror(char \*s) {

fprintf(stderr, "Error: %s\n", s);

return 0;

}

int yylex() {

int c;

while ((c = getchar()) == ' ' || c == '\t'); /\* skip whitespace \*/

if (isalpha(c)) {

return TAG;

}

return c;

}