Fundamentals Project

Convert C to java

Supervised: Dr/Osama Fathy

Name: Ibrahim Ahmed ID: 2016030003

Name: Andrew Wagih ID: 2016030026

Name: Rwida Osama ID: 2016030127

Lexical Analysis: -

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.

Lexeme: -

A *lexeme* is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

Token: -

A *lexical token* or simply *token* is a string with an assigned and thus identified meaning. It is structured as a pair consisting of a *token name* and an optional *token value*. The token name is a category of lexical unit.Common token names are

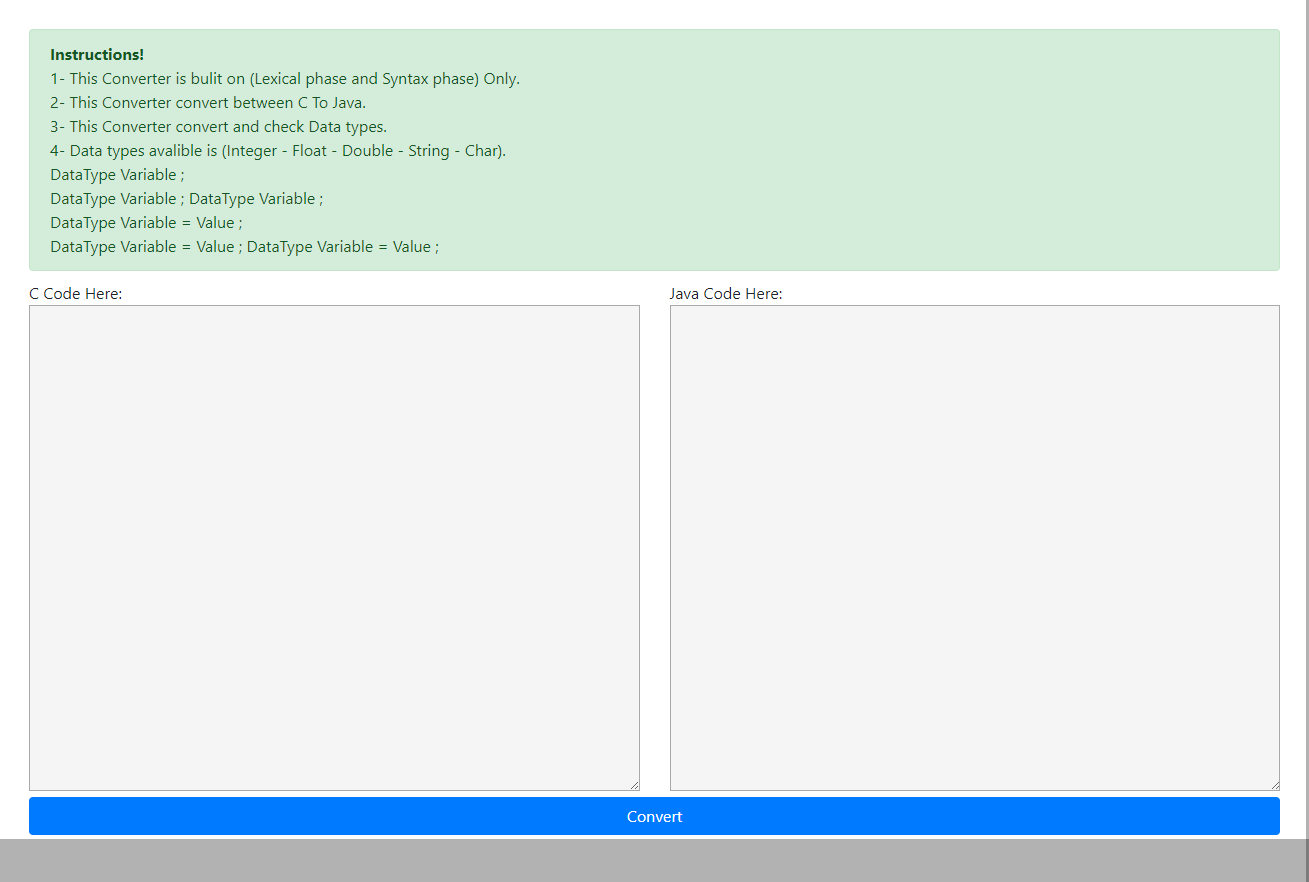
* identifier: names the programmer chooses.
* keywords: names already in the programming language.
* separator (also known as punctuators): punctuation characters and paired-delimiters.
* operator: symbols that operate on arguments and produce results.
* literal: numeric, logical, textual, reference literals.
* comment: line, block.

Regular expression: -

A regular expression (shortened as regex or regexp. also referred to as rational expression) is a sequence of [characters](https://en.wikipedia.org/wiki/Character_(computing)) that define a *search*[*pattern*](https://en.wikipedia.org/wiki/Pattern_matching). Usually such patterns are used by [string searching algorithms](https://en.wikipedia.org/wiki/String_searching_algorithm) for "find" or "find and replace" operations on [strings](https://en.wikipedia.org/wiki/String_(computer_science)), or for input validation. It is a technique developed in [theoretical computer science](https://en.wikipedia.org/wiki/Theoretical_computer_science) and [formal language](https://en.wikipedia.org/wiki/Formal_language) theory.

Tools:

* HTML
* JavaScript
* Bootstrap
* Vue (js)

Our Project:

Code:

<!DOCTYPE html>

<html>

<head>

    <title>C To Java</title>

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"

        integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">

        <link rel="shortcut icon" type="image/png" src="icon.png" />

</head>

<body>

    <div class="container-fluid" id="app" style="padding: 30px;">

        <div class="row conyainer-text">

            <div class="col-12">

                <div class="alert alert-success" style="margin-bottom: 10px">

                    <strong>Instructions!</strong><br>

                    <ul class="nav" style="display: block">

                        <li class="nav-item">

                            1- This Converter is bulit on (Lexical phase and Syntax phase) Only.

                        </li>

                        <li class="nav-item">

                            2- This Converter convert between C To Java.

                        </li>

                        <li class="nav-item">

                            3- This Converter convert and check Data types.

                        </li>

                        <li class="nav-item">

                            4- Data types avalible is (Integer - Float - Double - String - Char).

                        </li>

                        <li class="nav-item">

                            <ul class="nav" style="display: block">

                                <li class="nav-item">

                                    DataType Variable ;

                                </li>

                                <li class="nav-item">

                                    DataType Variable ; DataType Variable ;

                                </li>

                                <li class="nav-item">

                                    DataType Variable = Value ;

                                </li>

                                <li class="nav-item">

                                    DataType Variable = Value ; DataType Variable = Value ;

                                </li>

                            </ul>

                        </li>

                    </ul>

                </div>

            </div>

            <div class="col-6">

                C Code Here:

                <textarea v-model="textOne" id="" cols="30" rows="20" style="width:100%; background-color: #f5f5f5;"></textarea>

            </div>

            <div class="col-6">

                Java Code Here:

                <textarea v-model="textTwo" id="" cols="30" rows="20" style="width:100%; background-color: #f5f5f5;"></textarea>

            </div>

            <div class="col-12">

                <button class="btn btn-primary btn-block"  v-on:click="convert" style="margin-bottom: 20px">Convert</button>

            </div>

        </div>

    </div>

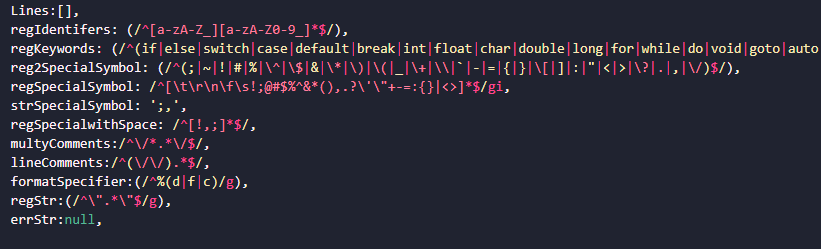
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

    <script src="function.js"></script>

</body>

</html>

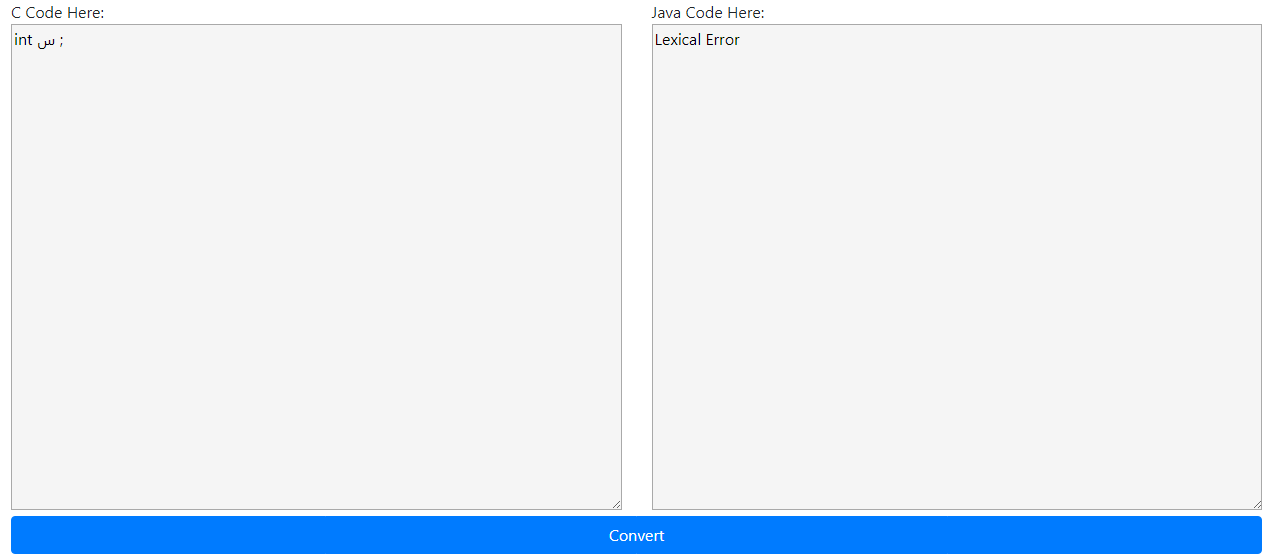
Regular Expression:

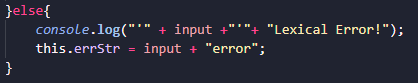


Lexical Check:



If we input any char not exit in our regular expression:

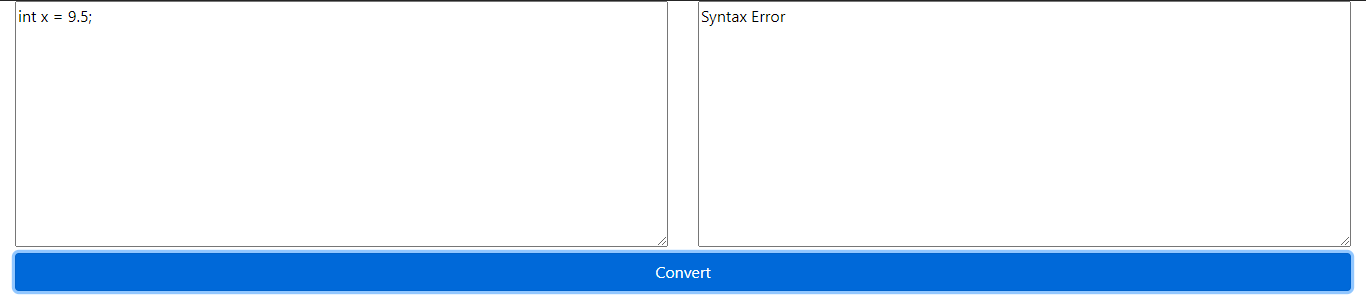




Syntax Analysis:

syntax error is an error in the [syntax](https://en.wikipedia.org/wiki/Syntax_(programming_languages)) of a sequence of characters or [tokens](https://en.wikipedia.org/wiki/Token_(parser)) that is intended to be written in [compile-time](https://en.wikipedia.org/wiki/Compiler). A program will not compile until all syntax errors are corrected. For [interpreted languages](https://en.wikipedia.org/wiki/Interpreted_language), however, a syntax error may be detected during [program execution](https://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)), and an interpreter's error messages might not differentiate syntax errors from errors of other kinds.

Syntax Analysis:

if happen lexical error his stop , if not his check sentence.

SyntaxPhase(*input*) {

            var res = *this*.regWithSpace(input);

            var inputArray = res.split(/[ \t\r\n\f]/);

*this*.errStr = null;

            for (var i = 0; i < inputArray.length; i++) {

*this*.LexicalPhase(inputArray[i]);

                if(*this*.LexicalPhase(inputArray[i])){

*this*.errStr = "error";

                }

            }

            if (*this*.errStr != null) {

*this*.textTwo = "Lexical Error";

            } else {

*this*.textTwo = input;

                var res2 = input.replace(/[;]/gi, '$&\n');

*this*.Lines = *this*.LinebyLine(res2);

*this*.Lines = *this*.Lines.filter(*item* => item);

                console.log(*this*.Lines);

                for(var i=0; i<*this*.Lines.length; i++)

                {

                    console.log(*this*.Lines);

                    var lineToken = *this*.regWithSpace(*this*.Lines[i]).split(" ");

                    lineToken = lineToken.filter(*item* => item);

                    console.log(*this*.Lines[i][*this*.Lines[i].length-1]);

                    console.log(lineToken);

                    if(*this*.Lines[i][*this*.Lines[i].length-1].match(/[;]/g)){

                        if (*this*.CheckDataInit(lineToken[0])) {

                            console.log(lineToken[0] + " keyword Done");

                            if (*this*.CheckIdentiferInit(lineToken[1])) {

                                console.log(lineToken[1] + " identifer Done");

                                if(lineToken.length == 5){

                                if (*this*.CheckOperationInit(lineToken[2])) {

                                    console.log(lineToken[2] + " operator Done");

                                    if (*this*.CheckNumberInit(lineToken[0],lineToken[3])) {

                                        console.log(lineToken[3] + " naumber!");

                                    } else {

                                        console.log( "syntax error num");

*this*.textTwo = "Syntax Error";

                                    }

                                }else{

                                    console.log( "syntax error op");

*this*.textTwo = "Syntax Error";

                                }

                            }

                            else{

                                if(lineToken.length == 3){

                                    if (*this*.CheckDataInit(lineToken[0])) {

                                        console.log(lineToken[0] + " keyword Done");

                                        if (*this*.CheckIdentiferInit(lineToken[1])) {

                                            console.log(lineToken[1] + " identifer Done");

                                        }else{

                                            console.log( "syntax error id");

*this*.textTwo = "Syntax Error";

                                        }

                                    }else{

                                        console.log( "syntax error key");

*this*.textTwo = "Syntax Error";

                                    }

                                }

                                else{

                                    console.log( "syntax error op");

*this*.textTwo = "Syntax Error";

                                }

                            }

                            }else{

                                console.log( "syntax error id");

*this*.textTwo = "Syntax Error";

                            }

                            }else{

                                console.log( "syntax error key");

*this*.textTwo = "Syntax Error";

                            }

                    }

                    else{

                        console.log("Syntax Error! ;");

*this*.textTwo = "Syntax Error";

                    }

                }