**Logo

Description automatically generated**

**FACULTY OF COMPUTING AND INFORMATICS (FCI)**

**TCP 2451 Programming Language Translation**

**Mini Project**

**TRIMESTER 1, 2021/2022**

**Prepared by**

**Student Name : Yong Wen Kai**

**Student ID : 1171101664**

**Contact No. : +6016-396 3254**

**E-mail Address : 1171101664@student.mmu.edu.my**

**Course Name : Bachelor of Computer Science (Hons) in Software**

**Engineering**

**Subject Lecturer and Tutor,**

**Mr. Nathar Shah Bin Packier Mohammad**

# Table of Contents

[Introduction 2](#_Toc88968585)

[Designing a Language 2](#_Toc88968586)

[Lexical Analysis 3](#_Toc88968587)

[DFA Diagram 3](#_Toc88968588)

[Example of lexical analysis 4](#_Toc88968589)

[Syntax Analysis 5](#_Toc88968590)

[Example of syntax analysis 6](#_Toc88968591)

# Introduction

This mini project is done by using python programming language. The whole project consists of 3 python files, which is LexicalAnalyser.py, ShifReduceParser.py and main.py. The main.py will be the main file that combines all functions in all files. The purpose of this program is to proof the concept of compiler by compiling duck language.

# Designing a Language

This duck language is designed by using context-free grammar written in Backnus-Naur form (BNF).

Start symbol:

* <Duck\_Quack>
* <Duck\_Honk>

Terminal Symbols: {quack, honk}

Production Rules:

1. <Duack\_Quack> ::= quack
2. <Duck\_Honk> ::= honk
3. <Duck\_Curious> ::= <Duck\_Quack><Duck\_Honk>
4. <Duck\_Happy> ::= <Duck\_Curious><Duck\_Quack>
5. <Duck\_Angry> ::= <Duck\_Honk><Duck\_Honk>

The duck language will only able to accept two string which is **quack** and **honk**.

# Lexical Analysis

Example input: quack / honk / quack honk / quack honk quack / honk honk

Regular Expression: (quack|honk)

|  |  |  |  |
| --- | --- | --- | --- |
| **Token** | **Sample Lexeme** | **Pattern** | **Regular Expression** |
| <Duck\_Quack> | quack | quack | <Duck\_Quack> -> quack |
| <Duck\_Honk> | honk | honk | <Duck\_Honk> -> honk |

### DFA Diagram

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **State** | **a** | **c** | **h** | **k** | **n** | **o** | **q** | **u** |
| **A** |  |  | B |  |  |  | C |  |
| **B** |  |  |  |  |  | D |  |  |
| **C** |  |  |  |  |  |  |  | E |
| **D** |  |  |  |  | F |  |  |  |
| **E** | G |  |  |  |  |  |  |  |
| **F** |  |  |  | H |  |  |  |  |
| **G** |  | I |  |  |  |  |  |  |
| **H** |  |  |  |  |  |  |  |  |
| **I** |  |  |  | J |  |  |  |  |
| **J** |  |  |  |  |  |  |  |  |

A picture containing wall, white, decorated, necklet

Description automatically generated

### Example of lexical analysis

1. User input: **quack**

Text

Description automatically generated

1. User input: **honk**

Text

Description automatically generated

1. User input: **quack honk**

Text

Description automatically generated

1. User input: **quack honk quack**

Text

Description automatically generated with medium confidence

1. User input: **honk honk**

Text

Description automatically generated

# Syntax Analysis

The syntax analysis is implemented by using stack and queue data structure which is perfect for current situation. The program is designed to take the user input and break it into tokens so that it can be analyzed and translated into duck language. For example, if the user input is **quack**, the program will first move the **quack**, into the stack. After that, the program will analyze the token in the stack. In this case, **quack** is equal to <Duck\_Quack>. This step will repeat until there is no more new input. Finally, the program will decide whether the token that being stacked together matches any duck language. If so, the program will print out the duck language. If not, the program will show the output saying that the input is not a duck language.

### Example of syntax analysis

1. User input: **quack**

Text

Description automatically generated

1. User input: **honk**

Text

Description automatically generated

1. User input: **quack honk**

Text

Description automatically generated

1. User input: **quack honk quack**

Text

Description automatically generated

1. User input: **honk honk**

Text

Description automatically generated