University of Central Punjab

Compiler Construction

Project Phase 1

VigoLang

A Satirical Programming Language

Student Name:	Muhammad Ahmad
Registration:	L1F22BSCS0634
Section:	G1
Teacher:	Sir Asif Farooq

Contents

1 Language Overview **2**
1.1 Introduction 2
1.2 Design Philosophy 2
1.3 Disclaimer 2

2 Keywords **3**

3 Operators and Punctuation **4**
3.1 Operators (7 Unique Operators) 4
3.2 Punctuation (7 Unique Punctuation Marks) 4

4 Regular Expressions **5**

5 Finite Automata Diagrams **6**
5.1 Identifier Finite Automaton 6
5.2 Integer Finite Automaton 7
5.3 Float Finite Automaton 8

6 Design Rationale **9**
6.1 Why These Symbol Choices? 9
6.1.1 Colon-Suffixed Arithmetic Operators 9
6.1.2 Pascal-Style Assignment 9
6.1.3 Question Mark Comparisons 9
6.1.4 Double Braces and Brackets 9
6.2 Why Caret Prefix for Identifiers? 9
6.3 Political Satire as Programming 10

7 Sample Test Program **11**

8 Conclusion **13**
8.1 Project Summary 13
8.2 Originality Statement 13

9 References **13**

1 Language Overview

1.1 Introduction

VigoLang is a satirical, Urdu-inspired programming language that uses political terminology from Pakistani discourse. The language employs keywords drawn from contemporary political scenarios to create a unique coding experience while maintaining the structural integrity of a proper programming language.

1.2 Design Philosophy

The language is designed with the following principles:

- **Satirical Expression:** Uses political metaphors as programming constructs
- **Unique Syntax:** Employs custom operators and punctuation for distinctiveness
- **Educational Value:** Teaches compiler construction principles effectively
- **Cultural Context:** Reflects contemporary Pakistani political discourse

1.3 Disclaimer

This language is purely a work of satire and academic exercise. The creator has no political affiliation, sympathy, or association with any political party, organization, or establishment. All terminology is used strictly for creative and educational purposes.

2 Keywords

VigoLang contains 20 unique keywords, each representing a common programming construct with satirical Pakistani political terminology.

Keyword	**Meaning**	**Example Usage**
ghq	main function	`ghq[[ ]] {{ ... }}`
safehouse	class definition	`safehouse Data {{ ... }}@`
nro	return statement	`nro 0@`
order_hai	if condition	`order_hai [[ x >? 5 ]]`
doosra_order	else if	`doosra_order [[ x >? 3 ]]`
warna_vigo	else statement	`warna_vigo {{ ... }}`
long_march	for loop	`long_march [[ i := 0@ ... ]]`
jab_tak_missing	while loop	`jab_tak_missing [[ x >? 0 ]]`
deal_ho_gai	break statement	`deal_ho_gai@`
chaltay_raho	continue statement	`chaltay_raho@`
farmaan	print/output	`farmaan << "text"@`
taftish	input statement	`taftish >> var@`
qaidi_no	integer type	`qaidi_no x := 5@`
bayania	string type	`bayania s := "text"@`
float_sarkar	float type	`float_sarkar f := 3.14@`
ishara	character type	`ishara c := 'A'@`
namaloom	void type	`namaloom func[[ ]] {{ }}`
ain	constant modifier	`ain qaidi_no MAX := 100@`
neutral	boolean true	`qaidi_no flag := neutral@`
janwar	boolean false	`qaidi_no flag := janwar@`

Table 1: VigoLang Keywords

3 Operators and Punctuation

3.1 Operators (7 Unique Operators)

Category	**Symbol**	**Description**	**Example**
Arithmetic	`+:`	Addition	`a +: b`
Arithmetic	`-:`	Subtraction	`x -: 5`
Arithmetic	`*:`	Multiplication	`p *: q`
Assignment	`:=`	Assignment	`x := 10`
Comparison	`=?`	Equality	`x =? y`
Comparison	`>?`	Greater than	`a >? b`
Comparison	`<?`	Less than	`x <? 10`

Table 2: VigoLang Operators

3.2 Punctuation (7 Unique Punctuation Marks)

Symbol	**Description**	**Usage**
`@`	Statement terminator	`qaidi_no x := 5@`
{{	Block start	`order_hai [[ x ]]` {{ `code` }}
}}	Block end	`order_hai [[ x ]]` {{ `code` }}
`[[`	Expression start	`order_hai [[ x >? 5 ]]`
`]]`	Expression end	`order_hai [[ x >? 5 ]]`
,	Separator	`func[[ a, b ]]`
`#* *#`	Multi-line comment	`#* comment *#`
`###`	Single-line comment	`### comment... eol`

Table 3: VigoLang Punctuation

4 Regular Expressions

The following table defines the regular expressions used for tokenizing VigoLang:

Token Type	**Regular Expression**	**Example**
Identifier	`\^[a-zA-Z]([a-zA-Z]|[0-9]|_)*`	`^count1`
Integer	`[+|-]?[0-9]+`	`42`, `-10`
Float	`[+|-]?[0-9]+\.[0-9]+`	`3.14`, `-0.75`
Exponential	`[0-9]+(\.[0-9]+)?[eE][+|-]?[0-9]+`	`2.5e8`, `1E-3`
String Literal	`"[^"]*"`	`"Hello World"`
Char Literal	`'[^']'`	`'A'`
Addition	`\+:`	`+:`
Subtraction	`-:`	`-:`
Multiplication	`\*:`	`*:`
Assignment	`:=`	`counter := 10`
Equality	`=\?`	`a =? b`
Greater Than	`>\?`	`x >? y`
Less Than	`<\?`	`x <? y`
Output Stream	`<<`	`<< message`
Input Stream	`>>`	`>> value`
Statement Term	`@`	`print@`
Block Start	`\{\{`	`{{`
Block End	`\}\}`	`}}`
Expression Start	`\[\[`	`[[`
Expression End	`\]\]`	`]]`
Separator	`,`	`arg1, arg2`
Increment	`\+\+`	`++i`
Decrement	`--`	`count--`
Multi-line Comment	`#\*([^*]|\*+[^*#])*\*#`	`#* sample comment *#`
Single-line Comment	`###.*`	`### this is a note`
Keywords	Exact match	`ghq`, `safehouse`, `return`

Table 4: Regular Expressions for VigoLang Tokens

5 Finite Automata Diagrams

5.1 Identifier Finite Automaton

Regular Expression: `^[a-zA-Z]([a-zA-Z]|[0-9]|_)*`

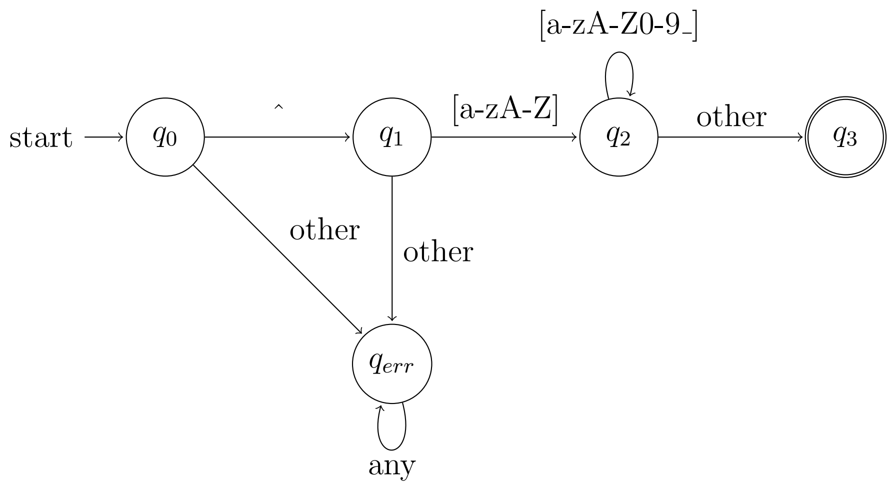

Figure 1: Finite Automaton for Identifiers

States:

- q_0: Start state
- q_1: After reading caret (^)
- q_2: Intermediate state (looping on valid characters)
- q_3: Final accept state (reached on 'other')
- q_{err}: Dead/error state

5.2 Integer Finite Automaton

Regular Expression: `[+|-]?[0-9]+`

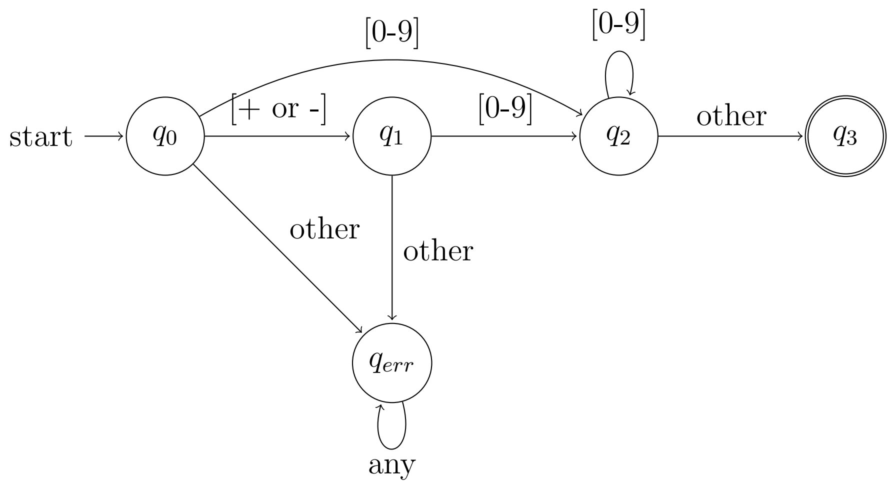

Figure 2: Finite Automaton for Integers

States:

- q_0: Start state
- q_1: After optional sign (+/-)
- q_2: Intermediate state (after at least one digit)
- q_3: Final accept state
- q_{err}: Dead/error state

5.3 Float Finite Automaton

Regular Expression: `[+ or -]?[0-9]+\.[0-9]+`

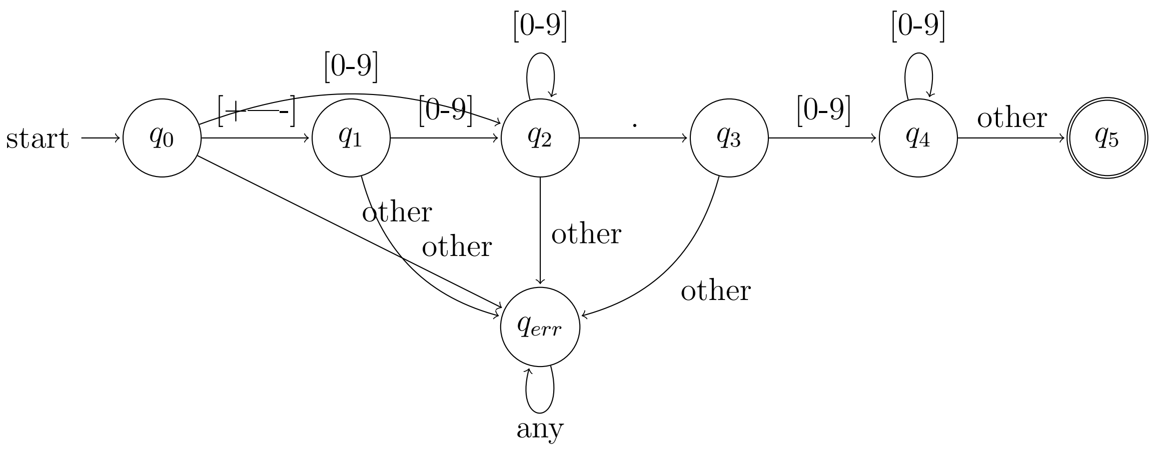

Figure 3: Finite Automaton for Floats

States:

- q_0: Start state
- q_1: After optional sign
- q_2: Reading integer part
- q_3: After decimal point
- q_4: Intermediate state (reading fractional part)
- q_5: Final accept state
- q_{err}: Dead/error state

6 Design Rationale

6.1 Why These Symbol Choices?

6.1.1 Colon-Suffixed Arithmetic Operators

The operators `+:`, `-:`, and `*:` were chosen for the following reasons:

- **Uniqueness:** No other programming language uses this pattern
- **Visual Symmetry:** Creates consistency with question mark comparisons
- **Colon Philosophy:** The colon represents "action" or "operation"
- **Easy Parsing:** Simple 2-state finite automata for recognition

6.1.2 Pascal-Style Assignment

The `:=` operator was selected because:

- **Proven Design:** Successfully used in Pascal, Ada, and Go
- **Clear Distinction:** Visually different from equality operator `=?`
- **Direction Hint:** Colon-first suggests "target receives value"

6.1.3 Question Mark Comparisons

Operators like `=?`, `>?`, and `<?` have philosophical meaning:

- Represents inquiry: "Is this condition true?"
- Fits the satirical theme of political uncertainty
- Creates unique language identity

6.1.4 Double Braces and Brackets

The punctuation {{ }} for blocks and `[[ ]]` for expressions provides:

- **Strong Visual Presence:** Blocks and expressions stand out clearly
- **No Ambiguity:** Two characters required, preventing single-character confusion
- **Modern Aesthetic:** Used in modern template languages

6.2 Why Caret Prefix for Identifiers?

The requirement that all identifiers start with `^`:

- **Zero Keyword Conflicts:** Identifiers can never clash with keywords
- **Instant Recognition:** Immediately distinguishes variables from keywords
- **Unique Identity:** Creates distinctive visual style for VigoLang

6.3 Political Satire as Programming

The keywords were carefully chosen to map common political discourse in Pakistan to programming concepts:

- **ghq** (main): References command headquarters, the central authority
- **safehouse** (class): Hidden organizational structures
- **nro** (return): Historical political deal that "returns" power
- **order_hai** (if): Hierarchical decision-making
- **warna_vigo** (else): Severe consequences for non-compliance
- **long_march** (for): Political protest representing iteration
- **deal_ho_gai** (break): Settlement ending negotiations
- **neutral/janwar** (true/false): Claims of neutrality vs. accusations

This makes VigoLang not just a programming language but also a form of social commentary.

7 Sample Test Program

The following VigoLang program demonstrates all 20 keywords and solves a simple problem: calculating factorial and checking number properties.

```
#* VigoLang Test Program
   Author: Muhammad Ahmad
   Purpose: Calculate factorial and check number properties *#

ghq[[ ]] {{
    qaidi_no ^number := 5@
    qaidi_no ^result := 1@
    qaidi_no ^counter := 1@
    qaidi_no ^temp := 0@

    farmaan << "Calculating factorial"@

    order_hai [[ ^number <? 0 ]] {{
        farmaan << "Error: negative number"@
        ^number := 0@
    }}
    doosra_order [[ ^number =? 0 ]] {{
        ^result := 1@
        farmaan << "Factorial is 1"@
    }}
    warna_vigo {{
        long_march [[ ^counter := 1@ ^counter <? ^number +: 1@ ^counter := ^counter +: 1 ]] {{
            ^result := ^result *: ^counter@

            order_hai [[ ^counter =? 10 ]] {{
                farmaan << "Limit reached"@
                deal_ho_gai@
            }}

            order_hai [[ ^counter =? 3 ]] {{
                chaltay_raho@
            }}
        }}
    }}

    farmaan << ^result@

    qaidi_no ^check := 2@
    jab_tak_missing [[ ^check >? 0 ]] {{
        ^check := ^check -: 1@
    }}

    ^calculate_sum[[ ^number, ^result ]]@

    nro 0@
}}

namaloom ^calculate_sum[[ qaidi_no ^a, qaidi_no ^b ]] {{
    qaidi_no ^sum := ^a +: ^b@
    farmaan << ^sum@
}}
```

```
safehouse ^calculator {{
    qaidi_no ^value@
    float_sarkar ^ratio := 2.5@
    bayania ^name := "calc"@
    ishara ^grade := 'A'@
}}@

ain qaidi_no ^max_limit := 100@
ain qaidi_no ^min_limit := 1@

qaidi_no ^active := neutral@
qaidi_no ^inactive := janwar@

bayania ^message := "complete"@

farmaan << "Program finished"@
```

Listing 1: VigoLang Test Program - Factorial Calculator

8 Conclusion

8.1 Project Summary

VigoLang successfully demonstrates the implementation of a complete lexical analyzer for a custom programming language. The project achieves all specified requirements:

- 20 unique, self-designed keywords
- 7 custom operators with unique syntax
- 7 distinctive punctuation marks
- Complete regular expressions for all token types
- Finite automata diagrams for key patterns
- Comprehensive test program (50+ lines)
- Functional Flex-based scanner
- Proper error detection and reporting
- Complete documentation

8.2 Originality Statement

This project is entirely original work:

- All 20 keywords are self-created with satirical Pakistani political themes
- Operator syntax (`+:`, `-:`, `:=`, `=?`, etc.) is unique to VigoLang
- Punctuation choices (`@`, {{, `[[`, etc.) are distinctive
- Test program is original, demonstrating practical problem-solving

9 References

1. Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools* (2nd ed.). Pearson Education.
2. GNU Flex Manual. Retrieved from `https://www.gnu.org/software/flex/manual/`
3. Regular Expressions Tutorial. Retrieved from `https://www.regular-expressions.info/`