SER502 Team 6 Project Lyric Language

Arizona State University

Team Members:

1. Abhijna Venkatesh Maiya
2. Darshan Phaldesai
3. Hitha Shamasundar
4. Shreyas Baburayanakoppal Sunil

Contents

- Language Overview
- Research
- Prototyping
- Language Grammar
- Lexical Analyzer and Parser
- Evaluation and Runtime
- Sample Programs

Language Overview

- **Language Name: Lyric**
  a programming language inspired by music, using keywords like repeat, loop, release and so on. Its syntax resonates with tracks and rhythms, making programming a lyrical experience.

- **Technical Stack:**
  Antlr4 Version: 4.13.2
  Python Version: 3.12.7

Research

LEX (a lexical analyzer generator): breaking the text into meaningful components called tokens

YACC (Yet Another Compiler-Compiler): takes these tokens and applies grammar rules to construct a syntax tree, enabling the parsing of structured input based on context-free grammar.

ANTLR (Another Tool for Language Recognition):

- Lexer: ANTLR processes input text into tokens using rules defined in the grammar.
- Parser: It organizes these tokens into parse trees or abstract syntax trees (ASTs) based on context-free grammar rules.

Prototype in Prolog

- We started out with prolog, laid down the initial grammar rules for our programming language but we wanted to learn a new parser.

- ANTLR proved out to be much simpler in terms of translating our grammar rules and made the whole process much easier to understand and trace.

- We have out initial plan and prototype in the github repo.

Lyric Grammar

Data Types:

```
<data_type>::= num | bool | str

<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

<char> ::= 'a' | 'b' | … | 'z'

<num> ::= <num> <digit> | <digit>

<bool_value>::= yeah | nah

<str> ::= <str> ::= { <char> }
```

Lyric Grammar

Statements:

```
<stmts> ::= <stmts> <stmt> | ε
<stmt> ::= <expr> ';'
    |  <dec_stmt> ';'
    |  <loop_stmt>
    |  <repeat_stmt>
    |  <check_stmt>
    |  '{' <stmts> '}'
```

Expressions:

```
<expr> ::= <id>
    |  <assign_expr>
    |  <math_expr>
    |  <cpr_expr>
```

Lexical Analyzer and Parser

Evaluation and Runtime

Sample Programs

```
start


num x;

num y;


play x 7;

play y (x * 2) + (x - 3);


release x;

release y;


stop
```

```
start

num x;
str s;
bool b;

play x 5;
play s "testprint";

bool play y yeah;

check ((5 * 10) == (30 + 30 - 10)) here {
  repeat (3) {
    release s;
  }
} there {
  release "Notest";
}

stop
```

Future Scope

- Implementing Functional Programming Paradigm
- Object-Oriented Programming (OOP)
- Adding Structs for Complex Data Types
- Implementing Error Handling

Lessons Learnt

- Collaborative Teamwork
- Designing Grammar
- Using ANTLR to Generate Parser
- Lexical Analysis

THANK YOU!