Compiler Construction (G1)

Project Phase 2

Submitted by: Ali Abdul Jabbar

Reg ID: L1F22BSCS0501

Submitted to: Sir Asif Farooq

Date: 11/01/2025

1. Introduction

The objective of Phase 02 is to design and implement a syntax analyzer for the custom "Harry Potter" themed language designed in Phase 01. The parser is implemented using **Bison (Yacc)** and validates the grammatical structure of the source code using tokens provided by the Phase 01 Flex scanner.

2. Context-Free Grammar (CFG)

The following Context-Free Grammar (CFG) has been implemented in parser.y. It supports variable declarations, struct-like "Houses", functions, loops (loopcharm, spellcycle), and conditionals (ifcharm)

Program Structure:

Program → FunctionList

FunctionList → FunctionList Function

| FunctionList HouseDefinition

| FunctionList GlobalDeclaration

| ε

Declarations & Structures:

HouseDefinition → HOUSE IDENTIFIER LBRACE HouseBody RBRACE

HouseBody → HouseBody HouseMember | ε

HouseMember → DataType IDENTIFIER DOLLAR

GlobalDeclaration → DataType IDENTIFIER DOLLAR

| DataType IDENTIFIER ASSIGN_OP Expression DOLLAR

Declaration → DataType IDENTIFIER OptionalInit

OptionalInit → ASSIGN_OP Expression | ε

Functions:

Function → DataType IDENTIFIER LPAREN ParameterList RPAREN Block

| VOIDCHARM IDENTIFIER LPAREN ParameterList RPAREN Block

| VOIDCHARM BEGINMAGIC LPAREN RPAREN Block

ParameterList → Parameters | ε

Parameters → Parameter | Parameters COMMA Parameter

Parameter → DataType IDENTIFIER

Statements:

Block → LBRACE StatementList RBRACE

StatementList → StatementList Statement | ε

Statement → Declaration DOLLAR

| Expression DOLLAR

| ConditionalStmt

| LoopStmt

| OutputStmt DOLLAR

| InputStmt DOLLAR

| ReturnStmt DOLLAR

| BreakStmt DOLLAR

| ContinueStmt DOLLAR

| Block

Control Flow:

ConditionalStmt → IFCHARM LPAREN Expression RPAREN Block

| IFCHARM LPAREN Expression RPAREN Block ELSECHARM Block

LoopStmt → LOOPCHARM LPAREN Expression RPAREN Block

| SPELLCYCLE LPAREN ForInit DOLLAR Expression DOLLAR Expression RPAREN Block

OutputStmt → REVEAL Expression

InputStmt → LISTEN IDENTIFIER

ReturnStmt → RETURNCHARM Expression | RETURNCHARM

3. FIRST and FOLLOW Sets

Below are the FIRST and FOLLOW sets for two major non-terminals in the grammar

3.1 Non-Terminal: Block

FIRST(Block) = { LBRACE }

- *Explanation:* A block definition in the grammar explicitly begins with an opening brace {.

FOLLOW(Block) = { RBRACE, ELSECHARM, VOIDCHARM, NUMSPELL, TEXTSPELL, FLOATSPELL, TRUTHCHARM, EOF }

- *Explanation:* A block can be followed by ELSECHARM (in an if-else structure), a closing brace (if nested), or the start of a new function/declaration.

3.2 Non-Terminal: Statement

- **FIRST(Statement)** = { NUMSPELL, TEXTSPELL, FLOATSPELL, TRUTHCHARM, IDENTIFIER, IFCHARM, LOOPCHARM, SPELLCYCLE, REVEAL, LISTEN, RETURNCHARM, BREAKCURSE, SKIPCURSE, LBRACE }
    - *Explanation:* A statement can begin with a data type (declaration), an identifier (assignment), a keyword (control flow), or a block starter.
- **FOLLOW(Statement)** = { RBRACE, NUMSPELL, TEXTSPELL, FLOATSPELL, TRUTHCHARM, ... }
    - *Explanation:* A statement is immediately followed by the next statement in the list or the end of the block (RBRACE).

4. Integration of Phase 01 Tokens

The Phase 02 Parser reuses the Phase 01 Lexical Analyzer without modification, ensuring consistency.

1. **Token Declaration:** All tokens defined in the scanner (e.g., NUMSPELL, REVEAL, DOLLAR) are declared in parser.y using the %token directive.
2. **Shared Header:** The Bison compilation generates parser.tab.h, which is included in scanner.l
   This allows the lexer to return the correct integer values for tokens recognized by the parser.
3. **Error Tracking:** The yyerror function in the parser utilizes yylineno and yytext from the scanner to report the exact location and content of syntax errors.

5. Parse Tree Representation

Sample Code: numspell x = 10 $

Derivation Tree:

1. **Statement** derives Declaration DOLLAR
2. **Declaration** derives DataType IDENTIFIER OptionalInit
3. **DataType** derives NUMSPELL (token: numspell)
4. **IDENTIFIER** matches token x
5. **OptionalInit** derives ASSIGN_OP Expression
6. **ASSIGN_OP** matches token =
7. **Expression** derives NUMBER_INT matches token 10
8. **DOLLAR** matches token $

6. Test Results

The parser was tested against valid and invalid programs as required.

6.1 Valid Program (test_valid.txt)

The valid test file contains house definitions, global variables, and a beginmagic function with nested loops and conditionals

Output:

Syntax Analysis Complete

Status: Syntax analysis successful

Invalid Program 1: Missing Token (test_invalid1.txt)

This test intentionally omits a closing parenthesis) in an ifcharm statement. **Code:** ifcharm (x > 5 {

Output:

LINE NUMBER: 6

ERROR: syntax error, unexpected LBRACE, expecting RPAREN

TOKEN: {

Status: Failed

6.3 Invalid Program 2: Invalid Structure (test_invalid2.txt)

This test omits the mandatory terminator $ after a variable declaration. **Code:** numspell y = 20 (newline) reveal y $

Output:

LINE NUMBER: 6

ERROR: syntax error, unexpected REVEAL, expecting DOLLAR

TOKEN: reveal

Status: Failed