Compiler Project

Authors: Arlind Lacej, Ridvan Plluzhina
Course: Formal Languages and Compilers
Academic Year: Second Semester 2024/25

Table of Contents

Project Overview

This project implements a **lexical analyzer** and **parser** for a simple programming language using:

- **FLEX** for lexical analysis (tokenization)
- **Bison/YACC** for syntax analysis (parsing)
- **Custom symbol table** for semantic analysis

Key Features

Variable declarations (`int` , `float`)
Arithmetic expressions (`+` , `-` , `*` , `/`)
Assignment operations with type checking
Comprehensive error reporting
Symbol table management

Language Specification

Grammar Rules

```
program  → program stmt | ε
stmt     → decl ';' | assign ';'
decl     → INT_TYPE ID | FLOAT_TYPE ID
assign   → ID '=' expr
expr     → expr '+' expr
         | expr '-' expr
         | expr '*' expr
         | expr '/' expr
         | NUM
         | ID
```

Supported Data Types

- **`int`** - Integer variables
- **`float`** - Floating-point variables

Operators

Operator	Description	Example
`+`	Addition	`x + y`
`-`	Subtraction	`x - y`
`*`	Multiplication	`x * y`

Operator	Description	Example
`/`	Division	`x / y`
`=`	Assignment	`x = 5`

Input Format & Examples

Basic Syntax Rules

- Each statement must end with a **semicolon (`;`)**
- Variables must be **declared before use**
- Comments start with `//` and continue to end of line

Valid Input Examples

```
// Variable declarations
int x;
float y;

// Assignments
x = 10;
y = 3.14;

// Expressions
x = 5 + 3;
y = x * 2.5;
```

Sample Test Files

test1.txt:

```
float x;
float y;
x = 2.0 + 2.0;
y = 3.25 + 3.25;
```

test2.txt:

```
int a;
a = 5.5;  // Type warning: float assigned to int
```

test3.txt:

```
z = 8;  // Error: undeclared variable
```

Installation & Usage

Prerequisites

- **MSYS2** environment
- **GCC** compiler
- **Flex** (lexical analyzer generator)
- **Bison** (parser generator)

Compilation Steps

```
# 1. Generate lexer
flex lexer.l

# 2. Generate parser
bison -d parser.y

# 3. Compile everything
gcc -Wall -g -o my_compiler parser.tab.c lex.yy.c symbol_table.c -lfl

# 4. Run with test file
./my_compiler < tests/test1.txt
```

Quick Build with Makefile

```
# Build everything
make

# Run tests
make test

# Clean generated files
make clean
```

Test Cases

Test File	Purpose	Expected Output
`test1.txt`	Basic arithmetic	`Assignment OK: x = 4.000000`
`test2.txt`	Type checking	`Type error: assigning float to int`
`test3.txt`	Undeclared variables	`Error: Undeclared variable z`
`test4.txt`	Multiple variables	Multiple successful assignments
`test5.txt`	Complex expressions	Expression evaluation results

Sample Output

```
$ ./my_compiler < tests/test1.txt
Assignment OK: x = 4.000000
Assignment OK: y = 6.500000

$ ./my_compiler < tests/test2.txt
Type error: assigning float to int variable a
Assignment OK: a = 5.500000
```

⚙ Technical Implementation

File Structure

```
compiler_project/
├── lexer.l           # Flex lexical analyzer
├── parser.y          # Bison parser grammar
├── symbol_table.c    # Symbol table implementation
├── symbol_table.h    # Symbol table header
├── tests/            # Test input files
│   ├── test1.txt
│   ├── test2.txt
│   └── ...
└── build/            # Compiled binaries
    └── my_compiler
```

Symbol Table Operations

- `insert(name, type)` - Add new variable
- `lookup(name)` - Get variable type
- `set_value(name, value)` - Update variable value
- `get_value(name)` - Retrieve variable value

Token Types

```
NUM         // Numeric literals (123, 3.14)
ID          // Identifiers (variable names)
INT_TYPE    // "int" keyword
FLOAT_TYPE  // "float" keyword
```

Error Handling

The compiler provides comprehensive error detection:

Lexical Errors

- Invalid characters or tokens

Syntax Errors

- Missing semicolons
- Invalid expression structure
- Malformed statements

Semantic Errors

- **Undeclared variables:** `Error: Undeclared variable x`
- **Type mismatches:** `Type error: assigning float to int variable x`

Example Error Output

```
Error: Undeclared variable z
Type error: assigning float to int variable a
```

Learning Outcomes

This project demonstrates understanding of:

- **Lexical Analysis** - Converting source code into tokens
- **Syntax Analysis** - Parsing token streams using formal grammars
- **Semantic Analysis** - Type checking and symbol management
- **Compiler Construction** - End-to-end compilation process
- **Tool Usage** - Flex/Bison integration and workflow
