Asian Institute of Technology

Calculator Project

An assignment submitted in partial fulfillment
of the requirements for AT70.07
Programming Language and Compilers

Submitted To:

Professor Phan Minh Dung

Submitted By:

Aung Htet Lwin(st125773)

31st March 2025

1 Introduction

This report presents the implementation details of a calculator project developed for the Programming Language and Compilers course. The project involves parsing **prefix expressions and converting them to infix** using a bottom-up SLR(1) parser.

The primary objectives of this project include:

- Implementing a parser for **prefix** mathematical expressions.
- Supporting arithmetic operations such as addition and multiplication.
- Converting prefix expressions to infix notation.
- Ensuring correct operator precedence and associativity.

2 Chosen Parsing Approach

2.1 Bottom-up SLR(1) Parsing

The project utilizes an SLR(1) parser, which is a type of bottom-up parsing strategy. The parsing table is constructed based on grammar rules and follows shift-reduce operations. The implemented grammar supports basic arithmetic operations.

2.2 Grammar Rules

The grammar rules for parsing prefix arithmetic expressions are as follows:

```
Rule 0     S' -> statement
Rule 1     statement -> expr
Rule 2     expr -> NUMBER
Rule 3     expr -> + expr expr  [precedence=left, level=1]
Rule 4     expr -> * expr expr  [precedence=left, level=2]
```

Hence the canonical LR(0) items are:

State 0

```
S' -> . statement
statement -> . expr
expr -> . NUMBER
expr -> . expr MINUS expr
expr -> . expr + expr
```

State 1

```
S' -> statement .
```

State 2

```
statement -> expr .
expr -> expr . MINUS expr
expr -> expr . + expr
```

State 3

```
expr -> NUMBER .
```

State 4

```
expr -> expr MINUS . expr
expr -> . NUMBER
expr -> . expr MINUS expr
expr -> . expr + expr
```

State 5

```
expr -> expr + . expr
expr -> . NUMBER
expr -> . expr MINUS expr
expr -> . expr + expr
```

State 6

```
expr -> expr MINUS expr .
expr -> expr . MINUS expr
expr -> expr . + expr
```

State 7

```
expr -> expr + expr .
expr -> expr . MINUS expr
expr -> expr . + expr
```

2.3 Terminals and Non-Terminals

- **Terminals:** NUMBER, +, *
- **Non-Terminals:** statement, expr

3 Parsing Table

State	+	*	NUMBER	$end	statement	expr
0	s5	s6	s3		1	2
1				acc		
2	s7	s8				
3	r2	r2				
4	r3	r3				
5	s4	s9	s3			
6	s4	s10	s3			
7	r1	r1				
8	r4	r4				

Table 1: State Transition Table

4 Semantic Rules for Translation

4.1 Prefix to Infix Conversion

Production	Semantic Rule
expr → + expr expr	expr.infix := (’(’ —— expr_1.infix —— ’ + ’ —— expr_2.infix —— ’)’)
expr → * expr expr	expr.infix := (’(’ —— expr_1.infix —— ’ * ’ —— expr_2.infix —— ’)’)
expr → NUMBER	expr.infix := NUMBER.lexval

Table 2: Semantic Rules for Prefix to Infix Conversion

5 Implementation Details

5.1 Key Components

- **Lexer:** Tokenizes input prefix expressions.
- **Parser:** Implements grammar rules and parsing logic.
- **Conversion Methods:** Converts prefix expressions into infix notation.

5.2 Main Features Implemented

- Parsing prefix mathematical expressions.
- Converting prefix expressions to infix.
- Ensuring correct operator precedence.
- Displaying results on a graphical user interface (GUI).

5.3 Challenges and Solutions

Challenges	**Solutions**
Handling incorrect input	Added error handling in the parser.
Operator precedence issues	Implemented a precedence table in the parser.
Prefix to infix conversion logic	Used a recursive approach to ensure correctness.

6 Conclusion

This project successfully implements an SLR(1) parser for **prefix expressions** and converts them into **infix notation**. The implemented parsing table ensures correct precedence and associativity of operators. Future improvements could include support for additional mathematical functions such as minus and division operation.

This project has deepened our understanding of parsing techniques, syntax-directed translation, and compiler design principles.

7 References

- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools*. Pearson.
- Appel, A. W. (2002). *Modern Compiler Implementation in C*. Cambridge University Press.
- Course Lecture Notes on Parsing Techniques.