LL(1)-parser

Compiler Construction Assignment 3

- Tauha Imran 22i-1239 | github
- Husain Ali 22i-0902 | github

LL(1) Parser

An LL(1) Parser implemented in C++ for a simplified programming language grammar.
This project was built as part of a Compiler Construction course assignment to demonstrate understanding of context-free grammars (CFG), LL(1) parsing tables, and predictive parsing techniques.

How to Run

```
g++ -o m main.cpp
./m
```

The parser reads an input source file (hardcoded as `input.txt`) and attempts to parse it according to the constructed LL(1) parsing table.

Input

Example input (`input.txt`):

```
id = number ;
if ( id == number ) {
  id = number ;
}
```

Grammar / CFG

The parser uses the following context-free grammar:

- **Start Symbol:** `S`
- **Production Rules:**

  ```
  S -> StmtList
  StmtList -> Stmt StmtList |
  Stmt -> id = Expr ;
        | if ( Cond ) { StmtList }
  ```

```
Expr -> Term ExprPrime
ExprPrime -> + Term ExprPrime
          | - Term ExprPrime
          |
Term -> id
     | number
Cond -> id ExprPrime RelOp Expr
     | number ExprPrime RelOp Expr
RelOp -> >
       | <
       | ==
       | !=
```

- **Non-Terminals:**

  `S, StmtList, Stmt, Expr, ExprPrime, Term, Cond, RelOp`

- **Terminals:**

  `id, number, =, +, -, ;, if, (, ), {, }, >, <, ==, !=`

LL(1) Parsing Table

The LL(1) parsing table was manually constructed based on the FIRST and FOLLOW sets of the grammar.
It supports all necessary terminals including parentheses `(`, `)`, braces `{`, `}`, relational operators like `==`, `!=`, and arithmetic operators `+`, `-`.

A snapshot of the table:

Non-Terminal Terminal	id	number	=	+	-	;	if	(	)	{	}	>	<	==	!=
S	StmtList	-	-	-	-	-	StmtList	-	-	-	-	-	-	-	-
StmtList	Stmt StmtList	-	-	-	-	-	Stmt StmtList	-	-	-		-	-	-	-
Stmt	id = Expr ;	-	-	-	-	-	if (Cond) { StmtList }	-	-	-	-	-	-	-	
Expr	Term ExprPrime	Term ExprPrime	-	-	-	-	-	-	-	-	-	-	-	-	-

Non-Terminal Terminal	id	number	=	+	-	;	if	(	)	{	}	>	<	==	!=
ExprPrime			-	+ Term ExprPrime	- Term ExprPrime		-	-		-		-	-	-	-
Term	id	number	-	-	-	-	-	-	-	-	-	-	-	-	-
Cond	Expr RelOp Expr	Expr RelOp Expr	-	-	-	-	-	-	-	-	-	-	-	-	-
RelOp	-	-	-	-	-	-	-	-	-	-	-	>	<	==	!=

Parsing Process (Summary)

The LL(1) parser uses a **stack-based predictive parsing** approach:

- Start with the stack initialized as `S $`.
- At each step:
  - Check the top of the stack.
  - If it's a terminal and matches the current input token, pop and consume the token.
  - If it's a non-terminal, expand using the parsing table entry for the current token.
- Repeat until both the stack and the input are reduced to the end marker `$`.

Example parsing steps (from provided input):

- `S` expands to `StmtList`
- `StmtList` expands to `Stmt StmtList`
- `Stmt` matches assignment: `id = Expr ;`
- `Expr` expands to `Term ExprPrime`
- `Term` matches `id` or `number`
- `ExprPrime` handles optional `+` or `-`
- Similarly, `if-else` blocks and relational conditions are parsed correctly using the `Cond` non-terminal and relational operator rules.

Features

Handles assignment statements and `if` conditional blocks.
Supports relational operations (`>`, `<`, `==`, `!=`).
Recognizes arithmetic operations (`+`, `-`) with proper precedence.

Processes both terminals and non-terminals in an LL(1) parsing approach.
Fully supports parentheses (,), and braces {, }.

Notes

- The parser assumes correct tokenization of the input.
- Parsing stops on successful reduction to $ or reports a parsing error otherwise.
- Extending the grammar with more constructs (like loops, functions) would involve updating the grammar and LL(1) parsing table accordingly.

Example Output (Partial)

```
Parsing content:
Input Tokens: id , = , number , ; , if , ( , id , == , number , ) , { , id , = number ; , }

Stack: S $
Top of Stack: S, Current Token: id
Expanding non-terminal 'S' with rule: StmtList
...
Matched terminal: id
Matched terminal: =
Expanding non-terminal 'Expr'
Matched terminal: number
Expanding non-terminal 'ExprPrime' with rule: epsilon
...
```

Author

Actually made by **Tauha Imran**
FAST NUCES | Compiler Construction Assignment

License

This project is open-source for learning and academic purposes.