Compiler Project Documentation

Overview

This project is a basic compiler implementation that includes the following core functionalities:

1. **Lexical Analysis**: Tokenizes the source code.
2. **Parsing**: Validates syntax based on grammar rules.
3. **Semantic Analysis**: Ensures type correctness and other logical constraints.
4. **Symbol Table Management**: Manages variable declarations and type checking.

Project Files

1. main.c

- **Purpose**: Acts as the entry point of the compiler.
- **Responsibilities**:
    - Initializes the compilation process.
    - Calls lexer, parser, and semantic analysis functions.
    - Displays error messages or success notifications.
- **Key Functions**:
    - int main(): Drives the compilation process.

2. lexer.l

- **Purpose**: Implements lexical analysis (tokenization).
- **Responsibilities**:
    - Scans the source code to identify tokens such as keywords, identifiers, operators, and numbers.
    - Handles invalid token errors.
- **Key Features**:
    - Token definitions for IDENTIFIER, NUMBER, operators (+, -, *, etc.), and delimiters.
    - Integration with yylval for passing token values to the parser.

3. parser.y

- **Purpose**: Implements parsing using context-free grammar rules.
- **Responsibilities**:
    - Validates the syntax of the source code based on predefined grammar rules.
    - Constructs a parse tree or performs semantic actions during parsing.
- **Key Features**:
    - Rules for variable declarations, assignments, expressions, and control structures.
    - Error handling for syntax errors.

4. semantic.c

- **Purpose**: Implements semantic analysis and symbol table management.
- **Responsibilities**:
  - Ensures type correctness during variable declarations and assignments.
  - Manages a symbol table to store variable names, types, and scopes.
- **Key Functions**:
  - semantic_insert_symbol(): Adds a variable to the symbol table.
  - semantic_lookup_symbol(): Retrieves the type of a variable.
  - semantic_check_type(): Verifies type compatibility between variables and expressions.

5. semantic.h

- **Purpose**: Header file for semantic analysis.
- **Responsibilities**:
  - Declares data structures and functions for symbol table management and type checking.
- **Key Contents**:
  - struct Symbol: Represents an entry in the symbol table.
  - Function prototypes for semantic analysis functions.

6. Makefile

- **Purpose**: Automates the build process.
- **Responsibilities**:
  - Defines rules to compile the lexer, parser, and other C files into an executable.
- **Key Features**:
  - Commands to generate lex.yy.c and y.tab.c using flex and bison.
  - Includes clean-up rules for removing intermediate files.

7. test_cases.txt

- **Purpose**: Contains sample source code for testing the compiler.
- **Responsibilities**:
  - Provides input examples to test variable declarations, assignments, type checking, and error handling.
- **Examples**:

  c
  CopyEdit
  ```
  int a = 10;
  int b = 20;
  float c = a + b;
  ```

Build Instructions

1. Ensure flex and bison are installed on your system.
2. Compile the project:

   bash
   CopyEdit
   ```
   make
   ```

3. Run the compiler with a test file:

   bash
   CopyEdit
   ```
   ./compiler < test_cases.txt
   ```

Clean-Up

To remove generated files, use:

bash
CopyEdit
```
make clean
```

Testing

- Use the test_cases.txt file to verify correct behavior.
- Modify or add test cases to cover additional scenarios.

By:

1.	Abdelaziz Ebrahim	1404559
2.	Guyo Dido	1410553
3.	Danya Abdella	1403750
4.	Sintayehu Getahun	1404276
5.	Tsigereda Habtamu	1404605