README-A3

Recursive Descent Parser for Ada Subset - John Akujobi A3

jakujobi/
Ada_Compiler_Construc...

Compiler for ADA written in python for CSC 446

1 Contributor · 0 Issues · 2 Stars · 0 Forks

Ada_Compiler_Construction/A3_Recursive_Parser at main · jak...

Compiler for ADA written in python for CSC 446. Contribute to jakujobi/Ada_Compiler_Construction development by creating an account on GitHub.

github.com

Welcome to the Recursive Descent Parser project! This project is a continuation of a compiler construction course, focusing on parsing a subset of the Ada programming language. The parser takes a list of tokens generated by the Lexical Analyzer and verifies syntactic correctness based on a context-free grammar (CFG).

The parser is designed to be modular, error-resilient, and extensible, allowing future enhancements such as semantic analysis and code generation.

Overview

What This Parser Does

- Reads a list of tokens generated by the Lexical Analyzer.
- Uses Recursive Descent Parsing to verify syntactic correctness.
- Reports errors and optionally stops or attempts panic-mode recovery.
- Constructs and prints a Parse Tree with indentation and structured connectors.
- Produces a summary report with details of parsing success or failure.

Key Features

- Recursive Descent Parsing with methods corresponding to non-terminals in the CFG.
- Error Handling:
  - Option to stop on error and ask if the user wants to continue.
  - Panic-mode recovery (skips tokens until a safe recovery point).
- Modular Design *(Easily extendable for future Ada compiler features).*

Easter Egg!!! 🪺 🐣

Parse Tree Construction (🪺🐣):

- Displays parse structure using indented tree format.
- Uses `├─`, `└─`, and `│` for better readability.

Project Structure

- `RDParser.py` - Implements the Recursive Descent Parser.
  - Reads tokens from the Lexical Analyzer.
  - Implements parsing methods for each non-terminal in the CFG.
  - Supports error handling, panic-mode recovery, and parse tree printing.
- `ParseTreePrinter.py` - Handles the formatted printing of the parse tree with indentation and connectors.
- `JohnA3.py` - The driver program that:
  - Reads the source code file.
  - Invokes the Lexical Analyzer to generate tokens.
  - Calls the Recursive Descent Parser.
  - Prints tokens, errors, and the parse tree.
- `Logger.py` - A singleton logger that logs:
  - Parsing steps.
  - Matched tokens.
  - Errors and recovery attempts.
- `Definitions.py` - Defines token types and provides mappings for reserved words and regex patterns.
- `Token.py` - Represents each token with attributes such as:
  - `token_type`
  - `lexeme`
  - `line_number`, `column_number`

Grammar Rules (CFG)

The parser is based on the following context-free grammar (CFG) as provided in the last assignment, A2:

```
Prog           ->  procedure idt Args is
                   DeclarativePart
                   Procedures
                   begin
                   SeqOfStatements
                   end idt;
```

```

DeclarativePart -> IdentifierList : TypeMark ; DeclarativePart | ε

IdentifierList  -> idt | IdentifierList , idt

TypeMark       -> integert | realt | chart | const assignop Value

Value          -> NumericalLiteral

Procedures     -> Prog Procedures | ε

Args           -> ( ArgList ) | ε

ArgList        -> Mode IdentifierList : TypeMark MoreArgs

MoreArgs       -> ; ArgList | ε

Mode           -> in | out | inout | ε

SeqOfStatements -> ε
```

How to Run

Prerequisites

- Python 3.10+
- Ensure you have the Lexical Analyzer (`JohnA1.py`) working, as it generates the token list.

Installation

1. Clone the Repository:

```
git clone https://github.com/jakujobi/Ada_Compiler_Construction.git
```

2. Navigate to the Parser Directory:

```
cd Ada_Compiler_Construction/A3-RecursiveDescentParser
```

3. Set Up a Virtual Environment (Optional but Recommended):

```
python3 -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate
```

Running the Parser

To run the parser on an Ada source file:

```
python3 JohnA3.py <input_file.ada> [output_file.txt]
```

- `input_file.ada` - The Ada source file to be parsed.
- `output_file.txt` *(optional)* - Stores the token results.

Example

```
python3 JohnA3.py example.ada parse_output.txt
```

This will:

1. Tokenize the source code.
2. Parse the tokens using the **Recursive Descent Parser**.
3. Display **parsing steps**, **error reports**, and **parse tree**.
4. Save the output (if specified).

Example Output

Tokens Generated

```
Token Type      | Lexeme                   | Value
---------------------------------------------------
PROCEDURE       | procedure                 | None
ID              | my_program                | None
LPAREN          | (                         | None
...
END             | end                       | None
ID              | my_program                | None
SEMICOLON       | ;                         | None
EOF             | EOF                       | None
```

Parse Tree (Indented Format)

```
Prog
├── PROCEDURE: procedure
├── ID: my_program
├── Args
│   ├── ε
├── IS: is
├── DeclarativePart
│   ├── ε
```

```
├── Procedures
│   ├── ε
├── BEGIN: begin
├── SeqOfStatements
│   ├── ε
├── END: end
└── ID: my_program
    └── SEMICOLON: ;
```

Error Handling and Recovery

The parser supports two error-handling modes:

1. Stop-on-Error Mode (`stop_on_error=True`)

- If enabled, the parser stops immediately on an error.
- Prompts the user:

```
Error at line 4: Expected 'BEGIN', found 'END'
Stop on error? (y/n):
```

- If the user enters `y` , the parsing terminates.

2. Panic Mode Recovery (`panic_mode_recover=True`)

- Instead of stopping, the parser skips tokens until a safe point.
- Example:

```
Error: Unexpected token 'END', skipping until 'SEMICOLON'
```

Design Decisions

Recursive Descent Approach

- Each non-terminal has a corresponding method (`parseProg()` , `parseDeclarativePart()` , etc.).
- Uses backtracking-free LL(1) parsing.

Modular Structure

- Keeps parsing logic separate from lexical analysis and logging.

Error Resilience

- Implements Stop-on-Error and Panic Mode Recovery.

Parse Tree Construction

- Users can enable/disable tree printing.
- Uses clear indentation and connectors (`├──`, `└──`, `│`).

Acknowledgements and Ethical Use of AI

Use of AI in Development

- AI LLM Models like DeepSeek-R1-Distill-Qwen-14B (Locally run) and Cody AI (integrated into IDE) assisted in:
  - Code documentation including parts of this README.
  - Improving error handling strategies.
  - Suggested structured tree printing techniques.
- I manually reviewed all AI suggestions were manually reviewed before integrating them.
  - In most cases, I also created multiple suggestions by the different models, and selectively edited and combined the parts I deemed helpful.

Ethical Considerations

- The project is licensed under the Hippocratic License.
- Not for use in military, surveillance, or ecocide applications.
- Promotes ethical AI and open-source usage.

Contributing

Contributions are welcome! To contribute:

1. Fork the repository.
2. Implement your changes.
3. Submit a pull request with detailed documentation.

jakujobi/
Ada_Compiler_Construc...

Compiler for ADA written in python for CSC 446

1 Contributor · 0 Issues · 2 Stars · 0 Forks

GitHub - jakujobi/Ada_Compiler_Construction: Compiler for A...

Compiler for ADA written in python for CSC 446. Contribute to jakujobi/Ada_Compiler_Construction development by creating an account on GitHub.

github.com

License

Hippocratic License HL3-BOD-CL-ECO-LAW-MEDIA-MIL-SV

Contributors

Right now, it's just me, John Akujobi

jakujobi - Overview

jakujobi has 44 repositories available. Follow their code on GitHub.

github.com