CS 1720 Programming Languages

Fall 2023, Exercises Rust 3 - Recursive Descent Parser

Synopsis In this exercise, your goal is to modify an existing recursive descent parser. The recursive descent parser needs to operate on the tokens from your lexer. Also, you will add some additional functionality to the parser and grow a proper parse tree.

1 Integration of Lexer

The existing recursive descent parser has a mockup lexer.

(i) Integrate your handwritten lexer into the parser's code.

(ii) Lexer and Parser should reside in different files; use Rust's module system.

2 Parsing Additional Constructs

The given recursive descent parser can parse a single function with a nested block body. Modify the parser such that it can parser the following addition constructs.

(iii) Your parser should support multiple function definitions. Also, a function may be a procedure and thus have no return type; modify the parser accordingly.

(iv) Your parser should support `let`, `return`, and `if-then-else` statements. The statements need not yet support expressions other than a single identifier or literal.

```
func add(x : int32) -> int32
[
    let value : int32 = 35;
    if true [
       let a = 5;
    ] else [
       let b = 7;
    ]
    return value;
]

func main()
[
    let sum : int32 = 42;
    return sum;
]
```

(v) Write EBNF grammar rules for above statements in the comments of corresponding parse functions.

3 Growing A Parse Tree

(vi) Implement a `ParseTree` struct to create abstract parse trees. Each tree node should hold a token and may have zero, one, two, or n child notes.

(vii) Finally, your parser should grow a `ParseTree` while parsing. Note that the parse tree can be already a more abstract syntax tree.

4 Submission Format

Submit your code in a single `ER3_<first_last>.zip` file where first and last are your name (e.g., `ER3_Stephan_Ohl.zip`). The zip file should also contain a README file with your name and, if you want to add, additional comments.