Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

Assignment 1: Lambda Calculus Syntax

CSI3120-B (Fall 2024) – Prof Karim Alghoul

In this assignment, you will write a program that lexically analyzes and parses a lambda calculus expression. The grammar is based on the lambda calculus, as explained in the lectures.

Program Requirements:

- **Input:** The program reads an expression from .txt file. The program may accept multiple expressions, one per line.
- **Lexical Analysis:** The input is broken down into tokens (such as variables, parentheses, and operators).
- **Parsing:** Use recursive descent (top-down parsing) or bottom-up parsing, to parse the tokens according to the lambda calculus grammar.
- **Outputs:**
    - The program will output if the expression is Valid or Not.
    - Output the parsed expression in a more structured format, such as an Abstract Syntax Tree (AST) instead of simple string output. This will require you to structure the data internally and display it in a way that reflects the relationships between expressions.
- **Error Handling:** The program must detect syntax errors and provide error messages if necessary.

Additionally, implement detailed error messages that include the position of the error and what the parser expected. For example:

- "Error at position 5: Expected closing parenthesis, found ')'."
- "Error at position 8: Missing operator between terms."

- **No External Libraries:** Do not use external libraries; rely only on standard library functions.

Submission Details:

- Include a README file with:
    - Student numbers.
    - Information on whether the program works or has known defects.
    - Any deviations from the assignment requirements, if applicable.
    - Reference all the websites and external sources you used for help, so you don't fall into plagiarism.
    - If you used any AI tool for help, like ChatGPT, you need to screenshot the whole chat and include it in the .zip file.
- Include your well commented code. Use the template provided. You can add functions to the template.
- Include Four .txt files:
    1. Valid examples given to you,
    2. Negative example given to you
    3. Extra valid examples you chose
    4. Extra negative examples you chose.

Grammar Rules:

- **Program structure:**

```
<expr> ::= <var> | '(' <expr> ')' | '\' <var> <expr> | <expr> <expr>
```

- **Variable names:**
    - Consist of letters a-z, A-Z, and digits 0-9, but must start with a letter.

- **Whitespace:**
    - The program is whitespace-insensitive, but whitespace should separate variables.
- **The Dot (.) After Lambda Abstraction:**
    - The dot (.) can be used after a lambda abstraction variable to clarify grouping, acting like parentheses. When a dot is used, it groups everything to the right of the dot with the closest lambda abstraction. It's as if you're inserting parentheses around the expression following the dot.
    - For example:
        - `\x. x y` should be equivalent to: `\x (x y z)`, where the dot groups x y z together as a single expression
        - `\x. x y z` should be equivalent to: `\x (x y z)`
        - `\x. \y. x y z` should be equivalent to: `\x (\y (x y z))`

        When you write in the .txt file an expression with a Dot , it should be parsed as the example above by your program.

Examples:

Valid Examples:

- (A B)
- abc
- a b c
- a (b c)
- (\x a b)
- \x. a b
- (\x((a) (b)))

Invalid Examples:

- \ (missing variable after lambda).
- \x (missing expression after lambda abstraction).
- ((x (missing closing parenthesis).

- () (missing expression).
- a (b (missing closing parenthesis).
- a (b c)) (input not fully parsed).

Additional Task for the report:

In your report, describe your work in a clear and organized manner, ensuring that anyone reading it can easily understand the following:

- **Problem Statement:**
  Clearly explain the problem your program is designed to solve. What is the main objective of the lambda calculus parser you implemented?
- **Parsing Method:**
  Describe the parsing method you used in detail. Did you use top-down parsing, bottom-up parsing, or a combination of both? Why did you choose this method?
- **Examples with Parsing Techniques:**
  Take all the positive and negative examples from the assignment and demonstrate how they are parsed using both top-down and bottom-up parsing methods.
  For each example, show the steps of the parsing process and explain how your program handles the input.

Bonus Task:

Task Description:

1. **Ambiguity Detection:**
   - Analyze the given lambda calculus grammar to determine if it can produce ambiguous expressions.
   - Provide at least **two examples** of expressions that have multiple valid parse trees (derivations), demonstrating ambiguity in the grammar.
   - Explain how and why ambiguity occurs in these examples.
2. **Ambiguity Resolution:**
   - Modify the grammar to make it **unambiguous**. You may introduce **associativity rules** or restrict certain expressions with additional parentheses to clarify how expressions should be parsed.
   - After modifying the grammar, provide examples of how the new grammar resolves the ambiguity (with updated parse trees).

Hints for Students:

- Focus on cases where expressions like A B C or (\x A B) could be parsed in different ways (left-associative vs. right-associative).
- Think about how parentheses can be used to enforce grouping of expressions.
- Consider how you might modify the rules for <expr> ::= <expr> <expr> to specify whether application is **left-associative** or **right-associative**.

Evaluation Criteria:

- **Correctness (50%):** Does the program correctly parse expressions and find Valid and invalid syntax?
- **Error Handling (15%):** Are errors identified clearly and with informative messages?
- **The Dot criteria (10%):** The Dot (.) After Lambda Abstraction works properly?
- **Comments (5%):** is the code WELL COMMENTED? Anyone who reads your comments should be able to understand what you are trying to do.
- **Report (20%):** Covers all the tasks asked from you in “Additional Task for the report”.
- **Bonus (10%):** Ambiguity, you don’t have to implement it.