Assignment No. 8
EECS 468
Programming Language Paradigms
Due: 11:59 PM, Monday, November 27, 2023
Submit deliverables in a single zip file to Canvas
Files in other formats (e.g., .tar) will not be graded
Name of the zip file: FirstnameLastname_Assignment8 (with your first and last name)
Name of the Assignment folder within the zip file: FirstnameLastname_Assignment8

Deliverables:

1. Copy of Rubric8.docx with your name and ID filled out (do not submit a PDF).
2. Haskell script file(s).
3. Screen print as described below (Copy and paste the output to a Word document and PDF it).

Assignment:

- The aim of this project is to create a Haskell program that can parse and evaluate arithmetic expressions containing operators +, -, *, /, %, and **, as well as numeric constants. The program should be able to handle expressions with parentheses to define precedence and grouping.
- In this project, you will build a versatile arithmetic expression evaluator using Haskell. The program will take an arithmetic expression as input, parse it, and calculate the result according to the order of operations (PEMDAS). This project will help you reinforce your understanding of parsing techniques, functional programming, and algorithm design.
- Key Features:
    - Expression Parsing:
        - Your program should be able to parse arithmetic expressions entered by the user, considering operator precedence and parentheses.
    - Operator Support:
        - + (addition)
        - - (subtraction)
        - * (multiplication)
        - / (division)
        - % (modulo)
        - ** (exponentiation)
    - Parenthesis Handling:
        - Ensure that your program can handle expressions enclosed within parentheses to determine the order of evaluation.
    - Numeric Constants:
        - Recognize and calculate numeric constants within the expression.
    - Error Handling:
        - Implement robust error handling to manage scenarios like division by zero or invalid expressions.

- Test your program with the following valid and invalid expressions; copy and paste the output from your program into a PDF and turn it as described above.
- Your program should take in a string and output the result or an error message as described below. For example:

  > parse “3 + 4”
  7
  > parse “2 * (4 + 3 – 1”
  Error: unmatched parentheses

  - Valid Expressions:
    1. Addition:
       - Expression: 3 + 4
       - Result: 7
       - Explanation: This expression adds two numeric constants, resulting in a valid calculation.
    2. Subtraction with Parentheses:
       - Expression: 8 - (5 - 2)
       - Result: 5
       - Explanation: The parentheses ensure that the subtraction inside them is performed first, leading to the correct result.
    3. Multiplication and Division:
       - Expression: 10 * 2 / 5
       - Result: 4
       - Explanation: The multiplication and division operators are applied from left to right, resulting in the final answer.
    4. Exponentiation:
       - Expression: 2 ** 3
       - Result: 8
       - Explanation: The ** operator calculates 2 raised to the power of 3.
    5. Mixed Operators:
       - Expression: 4 * (3 + 2) % 7 - 1
       - Result: 5
       - Explanation: This expression combines multiple operators and parentheses to correctly calculate the result step by step.
  - Invalid Expressions (use the description, e.g., “unmatched parentheses” as the error message):
    1. Unmatched Parentheses:
       - Expression: 2 * (4 + 3 - 1
       - Explanation: This expression has unmatched opening and closing parentheses, making it invalid.
    2. Operators Without Operands:
       - Expression: * 5 + 2
       - Explanation: The * operator lacks operands on the left, making the expression invalid.
    3. Incorrect Operator Usage:
       - Expression: 4 / 0

    - Explanation: Division by zero is undefined in mathematics, so this expression is invalid.
4. Missing Operator:
    - Expression: 5 (2 + 3)
    - Explanation: The expression lacks an operator between 5 and (2 + 3), making it invalid.
5. Invalid Characters:
    - Expression: 7 & 3
    - Explanation: The & character is not a valid arithmetic operator, so this expression is invalid in the context of arithmetic operations.
6. Mismatched Parentheses:
    - Expression: (((3 + 4) - 2) + (1)
    - Explanation: The parentheses are not properly matched, with one closing parenthesis missing, making the expression invalid.
7. Invalid Operator Usage:
    - Expression: ((5 + 2) / (3 * 0))
    - Explanation: This expression attempts to divide by zero, which is mathematically undefined, rendering the expression invalid.
8. Invalid Operator Sequence:
    - Expression: ((2 -) 1 + 3)
    - Explanation: The expression contains an operator - without a valid operand on its left, making it invalid.
9. Missing Operand:
    - Expression: ((4 * 2) + (-))
    - Explanation: There is a missing operand after the - operator, making the expression invalid.
10. Invalid Characters:
    - Expression: ((7 * 3) ^ 2)
    - Explanation: The ^ character is not a valid arithmetic operator in this context, causing the expression to be invalid.

- Provide comments for the Haskell code that explain what each line of code is doing. See rubric below.

Rubric for Program Comments		
Exceeds Expectations (90-100%)	**Meets Expectations (80-89%)**	**Unsatisfactory (0-79%)**
Software is adequately commented with prologue comments, comments summarizing major blocks of code, and comments on every line.	Prologue comments are present but missing some items or some major blocks of code are not commented or there are inadequate comments on each line.	Prologue comments are missing all together or there are no comments on major blocks of code or there are very few comments on each line.

Adequate Prologue Comments:

- Name of program contained in the file (e.g., EECS 468 Assignment 7 - Replicate)

- Brief description of the program, e.g.:
    - Haskell function for replicate
- Inputs,e.g.,:
    - Number of replications
    - Element to replicate
- Output, e.g.,
    - List of replicated elements
- All collaborators
- Other sources for the code ChatGPT, stackOverflow, etc.
- Author's full name
- Creation date: The date you first create the file, i.e., the date you write this comment

Adequate comments summarizing major blocks of code and comments on every line:

- Provide comments that explain what each line of code is doing.
- You may comment each line of code (e.g., using --) and/or provide a multi-line comment (e.g., using {- and -}) that explains what a group of lines does.
- Multi-line comments should be detailed enough that it is clear what each line of code is doing.
- Each block of code must indicate whether you authored the code, you obtained it from one of the sources listed in the prolog, or one of your collaborators authored the code, or if it was a combination of all of these.

Collaboration and other sources for code:

- When you collaborate with other students or use other sources for the code (e.g., ChatGPT, stackOverflow):
    - Your comments must be significantly different from your collaborators.
    - More scrutiny will be applied to grading your comments in particular explaining the code "in your own words", not the source's comments (e.g., ChatGPT's comments).
- Failure to identify collaborators or other sources of code will not only result in a 0 on the assignment but will be considered an act of Academic Misconduct.
- Students who violate conduct policies will be subject to severe penalties, up through and including dismissal from the School of Engineering.