CSC 345 Lab – Recursive Descent Parser

Overview

Write a recursive descent parser. You will need to create a class named LabParser that will contain the parsing code.

Part 1

Create a class named LabScanner. It should recognize +, -, id, and end of file.

- Token Enum. Add an enum for the tokens as a member variable. It should contain enum values for PLUS, MINUS, ID, and EOF.
- PushbackReader.
  - Add a member variable for a PushbackReader. This will be the input stream that is read from by the scanner.
- Constructor. Add a constructor that takes a PushbackReader as a parameter. It should initialize the PushbackReader member variable.
- readNextChar Method. Write a method to read the next character from the input stream and return that value. It should read data from the PushbackReader member variable. Here is the method header: int readNextChar()
- scan Method. Write the scan method. Here is the method header: TOKEN scan()
  - There should be a while loop that keeps going until the end of file. A character value of -1 means the end of file has been reached.
  - The scan method should ignore all white space. You can write a method to check if a character is white space and call it from scan() as necessary. If a white space character is encountered, it should read the next character and go back to the loop test. Here is pseudocode for a method to check for whitespace.

    ```
    IsWhiteSpace(int c) return boolean
    If (c == 32) or (c == 9) or (c == 10) or (c == 13)
      return true
    Else
      return false
    ```
  - Check if + was the character just read in. If it is, then return the token corresponding to it.
  - Check if - was the character just read in. If it is, then return the token corresponding to it.
  - Check for a letter (use the Character.isLetter() method). If a letter is encountered, then it should process as an id and return the ID token. You do not need a token string buffer for this lab.

    Hint: There needs to be a loop to process the characters for the id.
  - Add code after the loop to return the end of file token.

Part 2

Create a class named LabParser.

- Declare a scanner member variable.
- Declare a scanner token member variable to hold the nextToken.
- There should be a method named parse. Here is the method header:
  void parse(String filename)
  - The parse method should open the input file with a FileReader.
  - Create a local variable in parse for a PushbackReader that is connected to the file that was just opened.
  - Create a new instance of LabScanner for the scanner member variable and pass the PushbackReader as a parameter to its constructor.
- Create a getNextToken method. Here is the method header:
  void getNextToken()
  It should get the next token from the scanner and store it in the parser's nextToken member variable.
- Create a match method. Here is the method header:
  boolean match(LabScanner.TOKEN expectedToken)
  If the expectedToken is the same as nextToken it should get the next token and return true otherwise it should return false.
- There should be a method named parse. Here is the method header:
  void parse(String filename)
  - The parse method should open the input file with a FileReader.
  - Create a local variable in parse for a PushbackReader that is connected to the file that was just opened.
  - Create a new instance of LabScanner for the scanner member variable and pass the PushbackReader as a parameter to its constructor.
  - Call getNextToken().

Part 3

Write a recursive descent parser for the following grammar.

Expr → id + id
Expr is the start symbol.

1. Add methods to LabParser to create a recursive descent parser for the grammar.
2. Update parse so that it begins parsing from the start symbol and matches the EOF token. It should print the message "Parsing successful" if it matched the EOF token otherwise it should print "Parsing failed".
3. Create an input file that follows this grammar.
4. Add code in main to create an instance of LabParser and parse the input file.

5. Run the program with different inputs and make sure that it can correctly produce successful and failed messages.

Part 4

Write a recursive descent parser for the following grammar (update the input file and main as necessary).

Expr → id ExprEnd
ExprEnd → + id

Part 5

Write a recursive descent parser for the following grammar (update the input file and main as necessary).

Expr → id ExprEnd
ExprEnd → + id | - id

Part 6

Write a recursive descent parser for the following grammar (update the input file and main as necessary).

Expr → Fact ExprEnd
ExprEnd → + Fact | - Fact
Fact → id

Part 7

Write a recursive descent parser for the following grammar (update the input file and main as necessary). You should now be able to parse the following: a+b+c-d-e.

Expr → Fact ExprEnd
ExprEnd → + Fact ExprEnd | - Fact ExprEnd | ε
Fact → id