Lab Assignment 9

💡 Due September 26 23:59 on moodle

The task involves defining a mini-language for polynomials and implementing operations like addition, subtraction, and evaluation based on the given input values. Below is an outline for the assignment, broken into clear steps to follows.

Objective:

In this assignment, you will design and implement a simple language that can represent polynomials with variables `x`, `y`, and `z` using **Lex** for tokenization and **Yacc** for parsing. You will also implement polynomial addition, subtraction, and evaluation at specific variable values.

Step-by-Step Instructions:

1. Design the Polynomial Grammar

You will define a grammar to represent polynomials using **Yacc**, and use **Lex** to tokenize the input expressions. Your grammar should handle the following:

- Polynomials with terms like `3x^2`, `4y`, or `5`.
- Polynomial variables are limited to `x`, `y`, and `z`.
- Polynomial expressions can include addition (`+`) and subtraction (`-`).
- Polynomials can be assigned to identifiers like `p1`, `p2`, etc.

---

2. Define the Lexical Tokens (Lex)

Use **Lex** to define tokens for polynomial elements:

- **Integer**: Coefficients like `3`, `5`.
- **Variable**: The variables `x`, `y`, and `z`.
- **Operator**: The operators `+`, `-`.

- **Exponent**: Representing powers such as `^2`.

3. Define the Grammar Rules (Yacc)

Using **Yacc**, define the grammar for handling polynomials, including rules for parsing terms and expressions. You will create a set of rules to handle polynomials, variables, coefficients, powers, and the operations `+` and `-`.

4. Implementing Polynomial Evaluation

Once the parsing is complete, implement the logic to evaluate the polynomials at specific values for the variables (`x`, `y`, or `z`). This should involve substituting the values into the polynomial and calculating the result.

5. Example Expressions

Using your **Lex** and **Yacc** definitions, define the following polynomials:

- `p1 = 3x^2 + 4x - 5`
- `p2 = 10x - 9x^2`
- `p3 = y - y^2 - y^3`

Perform the following operations:

- `p4 = p1 - p2`
- `p5 = p3 + p1`

6. Given values of the variables

Print the results of the following operations:

- for `p4`
  - `x = 2`
- for `p5`
  - `x = 0` and `y = 1`

Sample input:

```
p1 = 3x^2 + 4x - 5
p2 = 10x - 9x^2
p3 = y - y^2 - y^3
```

```
p4 = p1 - p2
p5 = p3 + p1
p1 : x = 2
p4 : x = 2
p5 : x = 0, y = 1
```

Sample output:

```
p1: 15
p4: 31
p5: -6
```

---

7. Optional (ungraded)

- Add more functionality such as handling parentheses for grouping terms.

Deliverables:

- Lex and Yacc files implementing the parser.
- Code for evaluating the parsed polynomials.

Submission Guidelines:

1. Create a new directory and rename it to your roll number, in the format CSXXBXXX (e.g., CS12B345).
2. Ensure that the **l9.l and l9.y files** of your project are placed **inside this CSXXBXXX directory** and there is no other directory inside CSXXBXXX
3. Write a Makefile and place it inside the CSXXBXXX directory. **The executable must be named as *'l9'*.**
4. Zip the directory using the following command:

   zip CSXXBXXX.zip -r CSXXBXXX
5. Unzipping CSXXBXXX.zip **must give a folder CSXXBXXX containing only l9.l, l9.y file, and Makefile.**

6. **Running *./l9* after *make* should execute the a.out executable**. Do **not** add any other extra flags.
7. **All submissions will be evaluated using a script. Failure to follow these instructions may result in a score of 0.**

Additional notes

1. The terms will have a single variable of any power i.e. there won't be any term like xy + 4.
2. statements p1 + x^2 is allowed
3. p1 = - x + y can be a statement **(**UPDATE**)**
    a. p1 = x - - y is a **valid** statement
    b. p2 = - x is a **valid** statement
    c. p3 = x + - y is a **valid** statement
    d. p4 = x --+y is a **invalid** statement
    e. p5 = + y is a **valid** statement
4. statements will be identified by p#, where # can be a number
5. Any statement will not be redefined
    a. Example
        i. `p1 = x`
        ii. `p2 = y`
        iii. `p1 = p1 + p2 (WILL NOT BE THERE)`
6. same polynomials can be evaluated with different values
    a. Example (**VALID**)
       `p1 = x`
       `p1 : x = 1`
       `p1 : x = 3`
7. All the coefficients will be on the left of the variable in the term and will only be a single number(**not** single digit).
8. There won't be any negative **exponent**.

9. All p# definitions will be before the value assignment to p#

    a. Example (**INVALID**)

       `p1= x`

       `p1 : x = 9  (this is value assignment statement)`

       `p2 = x^2`