Paul Kummer

Dr. Hanku Lee

Compilers CSIS 455-01

10/04/2021

Assignment 3: Infix to Postfix

Description

Assignment 3 will take a user entered expression in infix notation and convert it to postfix notation. When entering an expression, the user can enter integers or a string of characters, which may include numbers after the first character. The expression must be properly formed otherwise the output will be incorrect. Additionally, the user may use parentheses to have some operations performed before others. When the user is done, they must enter zero to quit.

Design

This program is designed with two java classes. The first is “Parser.java” and the second is “PostFix.java”. The “PostFix.java” file will initiate the program and prompt the user to enter an expression until the user enters “0” as an expression, which will quit the program. The “Parser.java” file will read characters and convert them into a postfix notation as they are read.

Parsers.java

When a parser is created, it will read in the first character of the input stream and store it in its lookahead variable. This variable will be used to keep track of what the next character is in the input stream to determine the course of the program.

Once a parser object has its expr() method called, then the parsing will begin to happen. First, the first term will be created by calling the term() method. This method will look at the first character, which is currently stored in the lookahead variable, and determine if the program should exit, clear whitespace, make a number, make a word, start another expression, or throw an error. If a word or number is created, the lookahead will continuously be updated and appended to the word or number until the word or number is complete. The lookahead is updated by reading in a new character from the input stream with the use of the match() method.

After the first term() method is finished, the lookahead should then be a character that is an operand or closing a parenthesis. If an operand or closing parenthesis is found in a series of if else statement, the match() method is called updating the lookahead to the next character. Right after the match() method, the term() method is called again to find the second operand of the operation.

The expr(), term(), and match() methods will continuously be called until there are no more characters in the input stream, unless the expression is ill-formed. There is another method called isFinished() that keeps track of whether or not the user wants to exit the program and will return false when the user does want to quit.

Below are images of the Parser code.

C: > Users > pakum > Desktop > compilers > assign03 > Parser.java > Parser

```java
import java.io.*;
import java.util.*;

class Parser
{
    static int lookahead;
    public boolean notDone = true;

    public Parser() throws IOException
    {
        lookahead = System.in.read();
    }

    void expr() throws IOException
    {
        term();
        while(true && notDone)
        {
            if(lookahead == ' ' || lookahead == '\t')
            {
                lookahead = System.in.read();
            }

            else if( lookahead == '+' )
            {
                match('+');
                term();
                System.out.write('+');
            }

            else if( lookahead == '-' )
            {
                match('-');
                term();
                System.out.write('-');
            }

            else if( lookahead == '*' )
            {
                match('*');
                term();
                System.out.write('*');
            }

            else if( lookahead == '/' )
            {
                match('/');
                term();
                System.out.write('/');
            }

            else if( lookahead == '%' )
            {
                match('%');
                term();
                System.out.write('%');
            }

            else if( lookahead == '^' )
            {
                match('^');
                term();
                System.out.write('^');
            }

            else if(lookahead == ')')
            {
                match(')');
                System.out.write(')');
            }

            else
            {
                return;
            }
        }
    }
```

```java
    void term() throws IOException
    {
        if((char)lookahead == '0')
        {
            notDone = false;
            return;
        }

        if((char)lookahead == ' ' || (char)lookahead == '\t')
        {
            while(lookahead == ' ' || lookahead == '\t')
            {
                lookahead = System.in.read();
            }
        }

        if( Character.isDigit((char)lookahead) )
        {
            String num = "";
            while(Character.isDigit((char)lookahead))
            {
                num += (char)lookahead;
                match(lookahead);
            }
            System.out.print(num);
        }

        else if(Character.isLetter((char)lookahead))
        {
            String word = "";
            while(Character.isLetterOrDigit((char)lookahead))
            {
                word += (char)lookahead;
                match(lookahead);
            }
            System.out.print(word);
        }

        else if(lookahead == '(')
        {
            System.out.write('(');
            match('(');
            expr();
        }

        else
        {
            throw new Error("syntax error");
        }
    }
```

```java
    void match(int t) throws IOException
    {
        if( lookahead == t )
        {
            lookahead = System.in.read();
        }

        else
        {
            throw new Error("syntax error");
        }
    }

    boolean isFinished()
    {
        return notDone;
    }
}
```

PostFix.java

“PostFix.java” is a simple program that will continuously loop and ask the user to enter characters that make up an expression. Then, a new parser object will be created and have its expr() method called that reads in all the characters that the user just entered and convert them to postfix notation. Next, the parse objects isFinished() method is called to update the notComplete variable to determine if the loop should stop and quit the program.

C: > Users > pakum > Desktop > compilers > assign03 > PostFix.java > ...

```java
import java.io.*;
import java.util.*;

public class PostFix
{
    Run | Debug
    public static void main(String[] args) throws IOException
    {
        boolean notComplete;
        do
        {
            System.out.println("Enter Characters (0 to exit): ");
            Parser parse = new Parser();
            parse.expr();
            notComplete = parse.isFinished();
            System.out.write('\n');
        } while(notComplete);
    }
}
```

Bugs

Currently, the code does not support precedence of operators. Therefore, if higher level mathematical operations happen after a low-level operation like addition and subtraction, the low-level operation will occur before the high-level operation. This will result in an incorrect value, but if parentheses are used, the problem could be avoided.

Conclusion

This assignment showed a way to make a simple lookahead parser. However, this method is very naïve, and doesn’t make an abstract syntax tree. Its simplicity is nice if the program doesn’t require any advanced operations, but this limitation will prevent it from being used in most circumstances. The visitor pattern works much better than this algorithm. However, with the basic concepts of this parser, others parsers can be built and made more sophisticated to handle more complexity.