EECS 293

Software Craftsmanship

2019 Fall Semester

Programming Assignment 5

Due at the beginning of your discussion session on

February 18-22, 2019

Reading

- Read the quick reference on routine names (canvas module).
- Read Sections 7 (preamble), 7.1, 7.2, 7.3, 7.5, 7.6, and 19.6 in Code Complete.

Grading Guidelines

An automatic C (or less) is triggered by:

- Any routine with complexity greater than 4, or by
- Any piece of code that is essentially repeated.

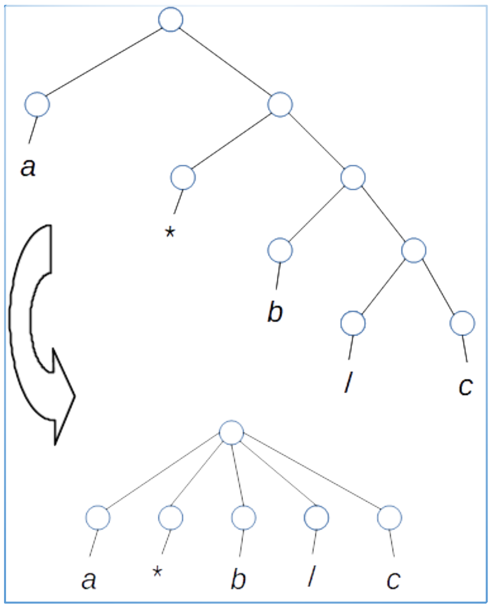
Starting with Programming Assignment 6, an automatic C (or less) will be triggered by improperly named routines.

Programming

First, make all the changes discussed in your discussion section. Additionally, you should refactor your code to make sure that it adopts the principles covered in the reading assignments.

Parse Tree Simplification

In the previous project, the parse tree can contain a long sequence of `EXPRESSION_TAIL`, representing a sequence of addition or subtractions, or `TERM_TAIL`, representing a sequence of multiplications and divisions. However, the tree can be expressed more compactly by bringing all the operators and variables as children of the original EXPRESSION or TERM. For example, the figure shows a parse tree for the expression *a**b*/c* and its simplification.



Modify InternalNode.Builder::simplify to replace an InternalNode with its children whenever the InternalNode starts with an operator (+, -, *, /). Note that the simplification should be made only if the previous InternalNode is not an operator, or else a UNARY may be simplified improperly. For example, the expression *a+-b* should not be simplified.

You can add additional helper functions to aid in the simplification process, such as:

- In Token and its derived classes, introduce a boolean isOperator() which returns true if this Token is a +, -, *, or /, and false otherwise.
- In Node and its derived classes, introduce
    - o boolean isOperator() that returns true if the node is a leaf corresponding to an operator, and false otherwise, and
    - o boolean isStartedByOperator() that returns true if the node's first child is an operator, and false otherwise (therefore, a leaf can be an operator, but never started by an operator).

Another simplification arises when a node is the father of an internal node whose only child is a leaf. In this case, the child can be replaced with the grandchild (leaf). Implement this simplification of the parse tree. You can add additional helper functions, such as:

- Optional<Node> firstChild() in Node and its derived classes, which returns the first child of this node, or empty if the node is either a leaf or unfruitful.
- boolean isSingleLeafParent() in Node and its derived classes, which returns true if this node's only child is a leaf, and false otherwise.

User Interface

For extra credit, implement a command line interface to the expression project. Its input is an expression list, and its output is a textual representation of the corresponding tree (or an explanation if an error occurred). Alternatively, your implementation could display the tree representation in a graphical interface.

Group Evaluation

Programming Assignment 5 wraps up the project. Along with the regular submission, you are required to insert a private comment on canvas on the group dynamics, and in particular on the percentage effort that you feel was put together by the individual team members.

General Considerations

This assignment concludes the Boolean expression project. In the next assignment, we will turn to a different project.

These classes may contain as many auxiliary private and package-private methods as you see fit, and additional package-private helper classes may be defined. However, any modification or addition to public classes and methods must be approved by the instructors at the discussion board.

You should write JUnit tests to make sure that your primary methods work as intended. However, we will revisit testing later on in the course, so extensive testing is not yet recommended. Similarly, your code should have a reasonable number of comments, but documentation is going to be the topic of a future assignment. As a general guideline at this stage of the course, comments and tests should be similar to those accepted in EECS 132.

Discussion Guidelines

The discussion will focus on routines, including but not limited to appropriate routine names.

Submission

Bring a copy to discussion to display on a projector. Additionally, submit an electronic copy of your program to Canvas. In addition to your code, include a README file explaining how to compile and run the code. The code should be handed in a zip, tar.bz2, or tar.gz archive. Archives in 7z cannot be accepted.