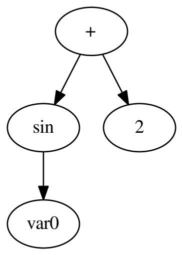
1 Introduction

Imagine the following situation: You are working in a large IT-company as a Senior C++ software developer. A colleague of you has written a small lousy library for building simple mathematical expressions, see `expressions.h`.

The library provides an interface for all expressions and some sub-classes for concrete operations like floating point constants, variables, unary functions (sin and cos), and the basic binary operators $+$, $*$, $/$, and $-$. For example the formula $\sin(x) + 2$ can be constructed via

```
auto expr1 = make_plus(
    make_sin( make_variable< 0 >() ) ,
    make_constant( 2.0 ) );
```

In tree like form this equation looks like



Besides that, your colleague has written some output methods for expressions, for example for polish notation and for graphviz, a graph visualization software.

2 Tasks

For the following tasks, use a version control system (VCS) of your choice and track your progress when solving the task. For example, you could create a github or bitbucket repository.

1. Enhance the expression interface by an `eval` method which lets you evaluate the tree. For example, any constant should evaluate to the value of the constant and a variable should evaluate to the value of an object in an evaluation context.
   Hint: The evaluation context is not yet present. It could be for example an array of doubles and `variable<0>` might return the first value of this array.

2. Write an algorithm, which linearizes all plus nodes in an expression. A linearized plus node is a node where the left child is not a plus node.

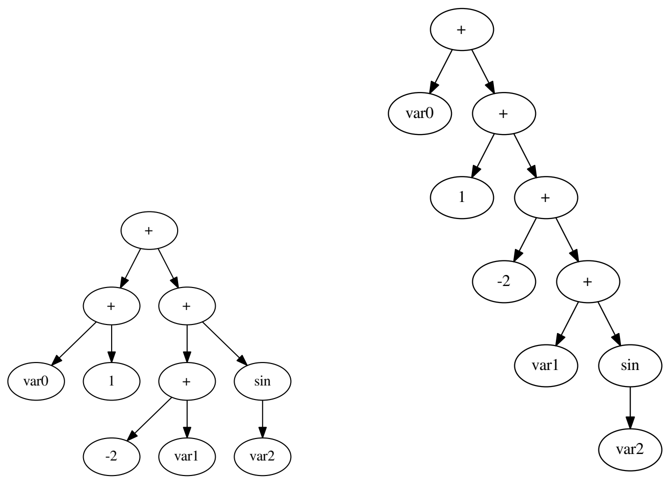


Figure 1: The graph on the left is the original graph of the equation $(x + 1) + ((-2 + y) + sin(x))$. The right graph is the linearized version of the original expression.

Of course, the evaluation of the expression should not be altered and the linearized expression should give the same evaluation result as the original expression. For example the expression in `main.cpp` will be transformed as shown in Figure 2.

3. Write a function `expr_ptr from_polish( std::string const& str , std::string const& separator )` to read from polish notation. That is, this function should create an expression from a string.

3 Optional tasks

1. Obviously, the design of the expression library is far from being optimal. What can be enhanced and optimized? Describe a few possibilities without putting them into source code.
2. Tree optimization: Write an algorithm which collapses all plus nodes which have two constants as children into one single constant.
3. Write a small benchmark suite to measure the performance and optimize the performance of the library.