Calculator for crazy mathematician

Imagine that you've just met a crazy mathematician. This crazy mathematician wants you to write a program for him which work as a calculator. Here is what he wants:
1. the calculator works only on integer numbers,
2. the calculator has only one operator which is minus (-),
3. the calculator accepts a string with all of the operations at once,
4. the calculator should calculate and print the result.

Here are some examples of the correct input for the calculator:

a. input: - - - - 2
   result: 2
b. input: 3 - 2
   result: 1
c. input: - 3 - - 2
   result: -1
d. - - - - - - - 23 - - - 3 - -2
   result: -24
e. input: 2
   result: 2

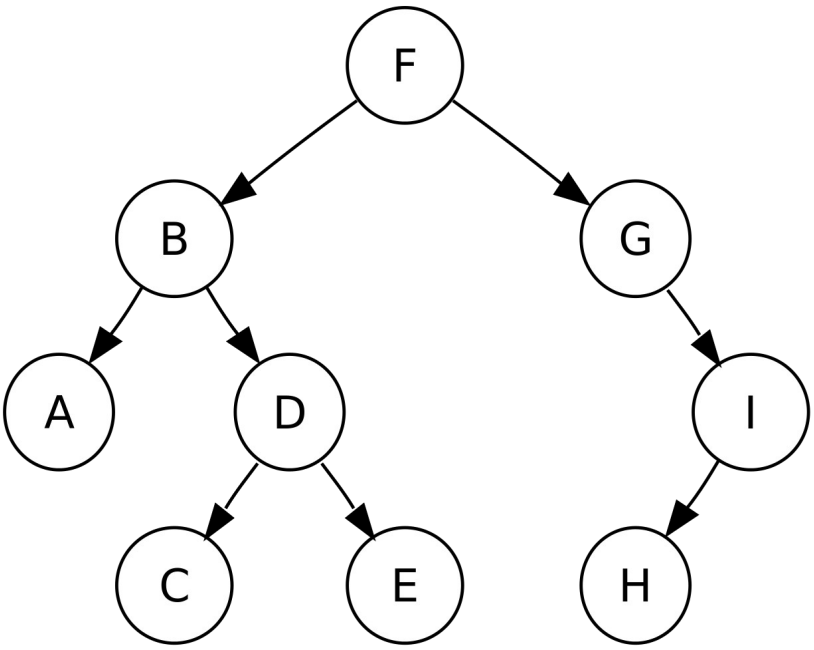
Here are some examples of incorrect input:

a. input: - - - -
b. input: 2 - -
c. input: 2 3

Notice that minus (-) operator may be a subtraction (when there are 2 numbers/parameters) or a negation (when there is only one number/parameter).

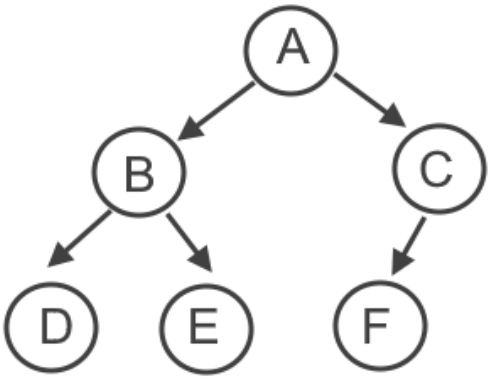
You are going to implement a parser that reads an input ad constructs an expression tree, which when correctly constructed can calculate the result of the expression.

Let's concentrate on what is an expression tree. It is a data structure, which is constructed from nodes and edges. Here is a generic example a tree (circles are nodes, and lines are edges):



In computer science trees are upside-down. The node in the highest level of the tree is a root. In our example it is node F. We say that a node has children if there are nodes below this node that are connected by edges with this node. For example node B has children: A and D, while node E has no children. We also say that the node connected with our node, that is in the higher level is it's parent. For example node B is parent of node D and node D is child of node B. The root node has no parent.

An expression tree is a tree that has numbers or operators in nodes. For example the expression: **3 – 1 – – 2** could be represented by such tree:



where:

A = –
B = –
C = –
D = 3
E = 1
F = 2
Notice that A and B represent subtraction with 2 parameters (in tree node A has 2 children), and C represents negation with 1 parameter (in tree node C has 1 child).

Trees are very convenient to represent arithmetic formulas, programs etc. When you write a program for example in C++, the compiler will parse it and construct a tree structure representing your code as part of the compilation process. Going back to our formulas and trees: implementing calculation of any correct input converted into tree is very easy. Let's see it with our previous example. Suppose that we have such classes representing operators and numbers that can be nodes of the tree:
node A = object of ExprTreeSubtraction class with children: B and C
node B = object of ExprTreeSubtraction class with children D and E
node C = object of ExprTreeNegation class with only one child F
node D = object of ExprTreeNumber class with value 3
node E = object of ExprTreeNumber class with value 1
node F = object of ExprTreeNumber class with value 2

Each of the classes SubtractionNode, NegationNode and NumberNode implement a method **compute()**. For SubtractionNode this method should ask the first child to compute (call compute() method on first child and collect result), then ask the second child to compute (call compute()

method on second child and collect result), and then subtract collected results and return the result of subtraction. The compute() method in Negation node is simpler: it only calls the compute() method on its child and multiplies the result by -1 before returning. The compute() method of NumberNode is the simplest: it just returns the value of the node.
In our example:
1. We only call compute() on node A
2. Node A calls compute() on nodes B and C
3. Node B calls compute() on nodes D and E
4. Node D returns its value 3
5. Node E returns its value 1
6. Now node B has values of D and E, computes their subtraction (3 – 1 = 2) and return 2
7. Node C calls compute() on node F
8. Node F returns its value 2
9. Node C negates result from F and returns -2
10. Node A takes values returned from nodes B and C and subtracts them (2 - -2 = 4) and returns result 4

There are more than one way to implement such a tree in C++, but you are going to implement it as 4 classes:
1. ExprTree – an abstract class defining the interface for other parts of the program which use an expression tree
2. ExprTreeSubtraction – a class deriving from ExprTree, which represents subtraction and has 2 pointers (left and right) to its children.
3. ExprTreeNegation – a class deriving from ExprTree, which represents negation and has 1 pointer (child) to its child.
4. ExprTreeNumber – a class deriving from ExprTree, which represents a number and has only a value.

You should test all 4 above classes in main.cxx. Uncomment parts of the main function.

The last part of your task is the implementation of the **parse()** method of the parser. Inside of this method call provided methods: **addMinustoTree()** and **addNumberToTree()**. Again uncomment last part of the code of the main() function to test it.