ECE650 Programming Assignment 1: Parser and Boolean Evaluator

The skeleton for this assignment is available at the master branch of `https://git.uwaterloo.ca/ECE650-F2022/skeleton` in directory `pa1`. Follow the instructions in Assignment 0 to correctly fetch and merge the files from the skeleton!

This is the first in a series of assignments that is part of a single large project. The project is to build a SAT solver, and the purpose of this assignment is (1) to build a parser that parses a string representing the combination of a Boolean formula and an assignment; (2) to evaluate the truth value of the formula under this assignment; (3) error handling of erroneous inputs.

Your Program

For this assignment, you need to write a program that

- Parse the input of a combination of (1) a string representing a Boolean formula and (2) a string representing an assignment into a Boolean formula syntax tree and an assignment map. Output the evaluation result of the formula on the assiginment.
- Continuously take input from standard input, and output to standard output. Errors shall also be output to standard output, but should always start with "`Error:`" followed by a brief description. Your program should terminate gracefully (and quietly) once it sees `EOF`. Your program should not generate any extraneous output; for example, do not print out prompt strings such as "`please enter input`" and things like that.
- Write your code in C++.
- Ensure that it compiles with the C++ compilor on

  `eceubuntu.uwaterloo.ca` and runs on `eceubuntu.uwaterloo.ca`. (Use `eceterm.uwaterloo.ca` to log-in from off-campus; then follow the instructions to connect to `eceubuntu`.)
- You can modify the skeleton as you wish, but you may only `#include` the following libraries (and no others!) for the current assignment: `iostream`, `fstream`, `sstream`, `iomanip`, `string`, `utility`, `exception`, `vector`, and `map`.

Context-free Grammar for Valid Inputs

One valid input shall be a string generated from the Context-Free Grammar G below:

```
S ::= Formula ';' Assignments
Formula ::= ConjTerm ( '+' ConjTerm )*
ConjTerm ::= Term ( '*' Term )*
Term ::= Constant | VarName | '-'Term | '(' Formula ')'
Assignments ::= an empty string | Assignment (',' Assignment)*
Assignment ::= VarName ':' Constant
VarName ::= a continuous sequence of letter (upper- or lowercase) or digits (0-9);
the first character cannot be a digit; the total length shall be no longer than 10
Constant ::= '0' | '1'
```

* Blue represents non-terminals and red represents terminals.
* Arbitrary amounts of whitespace are permitted before, after, or in between any of these terms.

Semantics of the Input

One valid input consists of a string representation of a Boolean formula S_F and a string representation of an assignment S_A, separated by a semicolon.

In S_F, "1" and "0" represent Boolean constants $TRUE$ and $FALSE$, respectively; a Boolean variable is a sequence of letters (either in upper- or lowercase) or digits (0-9); however, the variable name cannot start with a digit and its total length cannot be longer than 10. "$+$" represents the 2-argument infix Boolean function OR; "$*$" represents the 2-argument infix Boolean function AND; "$-$" represents the prefix 1-argument Boolean function NOT. The order of operations is: $parenthesis > NOT > AND > OR$.

Programming Guide

See the provided skeleton and comments in the skeleton for guidance.

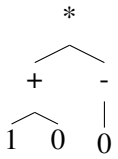
Examples

Example 1

Input:

$$(1 + 0) * -0;$$

Boolean Formula Syntax Tree:



Assignment Map:

$$\{\}$$

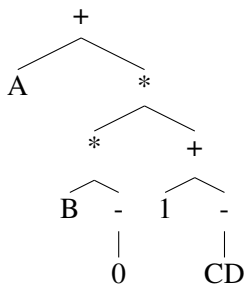
Output:
1

Example 2

Input:

$$A + B * -0 * (1 + -CD); A : 1, B : 0, CD : 1, B : 0$$

Boolean Formula Syntax Tree:



Assignment Map:

$$\{A : 1, B : 0, CD : 1\}$$

Output:
1

Sample Run

The executable shall be called `pa1`.

```
$ ./pa1
1 * VAr ; VAr: 1, VAr: 0
Error: contradicting assignment
1
1 * VAr;VAr : 0
0
Error: invalid input
 (1 * (a1 +VAr)) ; VAr: 1, a1: 0, VAr: 0
Error: contradicting assignment
1 *(VAr + 0 );
Error: incomplete assignment
```

You can see test cases in `test\test.in` and expected outputs in `test\test.out` for more examples. Also, an automatic testing script `eval.sh` is provided. See `README.md` for more information.

Also, you can play with the provided sample executable `sample-pa1` on `eceubuntu`.

Error Handling

You program should be able to identify the following types of error and print an error message starting with "`Error:`":

- Invalid input (any input that is not in the Context-free language defined by the grammar G)
- Incomplete assignment (at least one variable is not assigned to a truth value).
- Contradicting assignment (one variable is assigned to contradicting truth values).

Marking

Your output has to perfectly match what is expected. You should also follow the submission instructions carefully. It discusses how to name your files, how to submit, etc. The reason is that our marking is automated.

- Does not compile/make/crashes: automatic 0
- Your program runs, awaits input and does not crash on input: + 20
- Passes Test Case 1: + 10
- Passes Test Case 2: + 10
- Passes Test Case 3: + 10
- Passes Test Case 4: + 10
- Passes Test Case 5: + 10
- Passes Test Case 6: + 5
- Correctly detects errors: + 20
- Programming style: + 5

Submission

You should place all your files at the `pa1` directory in your GitLab repository. The directory should contain:

- All your source-code files.
- A `Makefile`, that builds a final executable named `pa1`
- A file `user.yml` that includes your name, WatIAM, and student number.

See `README.md` for any additional information.

The submitted files should be in `pa1` directory in the `master` branch of your repository.