SORBONNE UNIVERSITÉ

Autumn 2023

Ternary Logic

TP 2/3

Aims

- ✯ Let familiarized with modern C++ syntax.
- ✯ Understanding the compiler
- ✯ Implementing a hierarchy

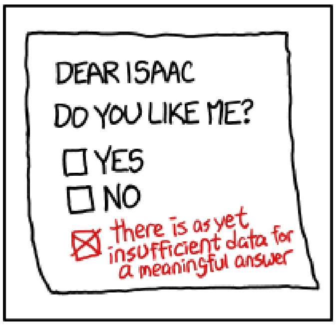
Constraints

- Indent your source files.
- Evaluation will take into account the briefness of the methods you wrote (avoid to write functions that are more than 25 line long) ; never hesitate to split a method into multiple shorter sub-methods (privates).
- Your code must not have errors in Valgrind (neither memory leak, nor other error).
- Vous ne devez pas utiliser de fonction C quand un équivalent C++ existe.
- For UML, you can use UMLet or Umbrello.
- Class names must start with a uppercase letter.
- You must provide a Makefile compile each source file and contains a rule called `clean` that delete intermediary files and a rule called `test` that execute the testcase binary.
- The **source and diagrams** must be *pushed* on your practical git.

Practical preparation

- A set of unit tests is provided in the folder `tests`, you can compile them with `make testcase`.
  The tests are using the library `doctest`, to learn more you can go on the project website `https://github.com/doctest/doctest`.
- The practical must be put on moodle at the end of the practical. It could be updated until the day before the next practical.

Concept



As it is illustrated in *strip* from xkcd, we need sometimes to have more than 2 choices such as (yes/no or truefalse). Sometimes the answer may be *I dont know*.
In mathematics, ternary logic theory exists (`https://fr.wikipedia.org/wiki/Logique_ternaire`), It is this logic that we will implement in this practical. The following figure gives the truth tables of operators `NOT`, `AND` & `OR`.

A	B	NOT A	A AND B	A OR B
T	T	F	T	T
T	F	F	F	T
T	U	F	U	T
F	T	T	F	T
F	F	T	F	F
F	U	T	F	U
U	T	U	U	T
U	F	U	F	U
U	U	U	U	U

Table 1: T : true ; F : false ; U : Unknown

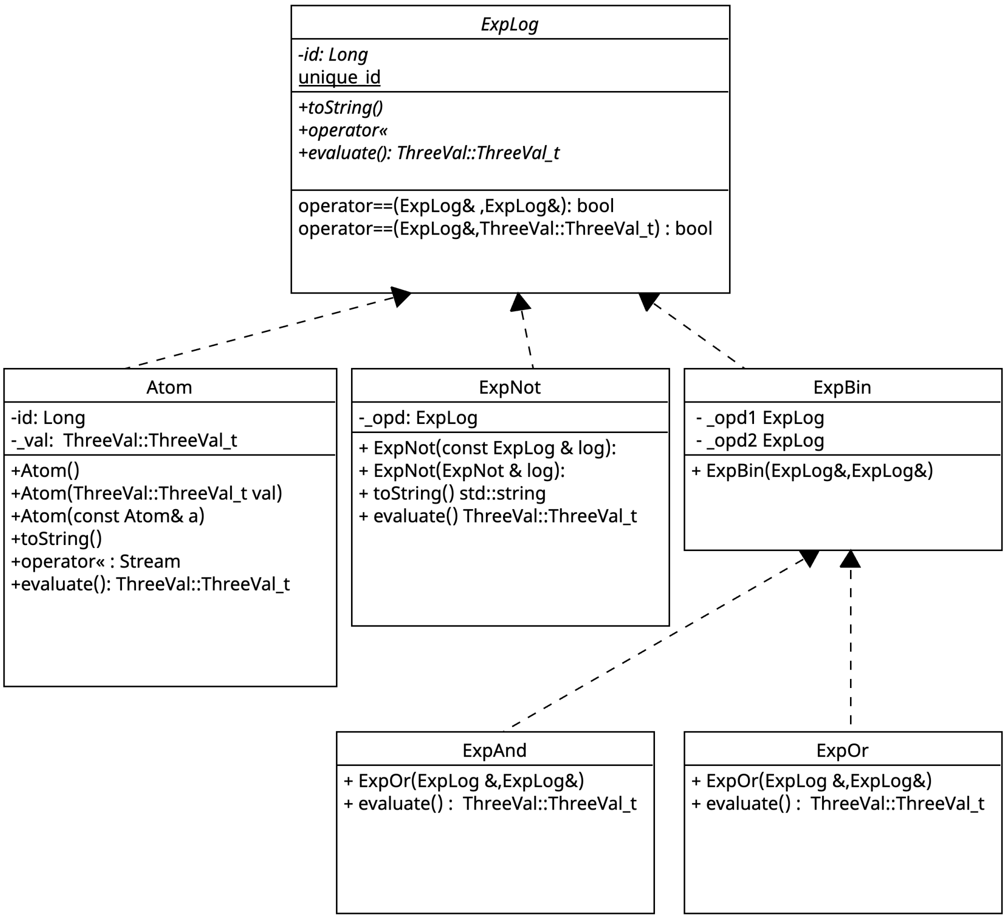
We will represent logic expression of ternary logic. A logical expression is composed of a set of logical variables combined with logical operations. Each logical can have the value (*true*, *false* or *unknown*). The variables are composed with logical operators : OR, AND & NOT. An example of logical expression *exp* could be *exp=*(a_1 *AND ((NOT* a_2*) OR* a_3*))*, where a_1, a_2 et a_3 are logical variables. If a_1 is equal to *true*, a_2 equals *false* and a_3 vaut *unknown*, then the expression *exp* is equal to *(true AND ((NOT false) OR unknown)*, as a consequence it is *true*.

A logical expression will not be directly manipulated. We will use the following abstractions :

- an *Atom* represents a logical atom. It is constituted of a unique identifier and a ternary value. The ternary value *val* must provide the ability to be reinitialised by the user.
- a *Not* represents the negation of a logical expression. It is composed of its unique operand *opd* that is a logical expression of the name “NOT”.
- a binary expression *ExpBin* corresponds to a logical expression. It is constituted of two operands, *opd1* et *opd2*, that are logical expressions.
- a *And* is a binary logical expression that corresponds to the logic named “AND”.
- a *Or* is a binary logical expression that corresponds to the “OR” logic.

A logical expression is represented by an abstract class *ExpLog*, in the header file `ExpLog`.hh.

➟ Look at see the drawing of this class hierarchy.



UML Class Diagram of the practical

1 Les valeurs logiques

You can use it as follow :

```
ThreeVal_t t = F, l = T, c = U;
std::cout << t << "\n" << l << "\n"<< c << "\n" << std::endl;
```

Display :

```
F
T
U
```

2 Class Atom

Write the Atom class that passes tests 1, 2 and 3. The tests assume that the ternary values chosen are T, F and U. The function `toString()` returns a character string of the form form `(a_1 = val)` where `a_1` is the atom's (unique) name and generated automatically) and `val` its value. Implement the `evaluate()` method, which returns the atom's value of the atom.

3 ExpNot

The class ExpNot represents a logical expression with one operand. Write the corresponding class passing test 4. The function `toString()` returns a chain of character with the shape `!(a_1 = val)`. Implement the method `evaluate()` that return the ternary negation of the atome.

4 Class ExpBin

The abstract class `ExpBin` contins the constructor of a 2 operand expression.

4.1 Class ExpOr and ExpAnd

Write two classes that arebinary logical expressions and allows to pass the test 5. The function `toString()` will return a character chain with the following shape :

```
Atom a(T);
Atom b;
Atom c(F);
ExpNot n1(a);
ExpAnd and1(n1,b);
ExpAnd and2(c,b);
ExpOr or1(and1,and2);
cout << or1.toString() << std::endl;
```

```
((!(a_0 = T) AND (a_1 = U)) OR ((a_2 = F) AND (a_1 = U)))
```

5 Equality operator

Add the missing operations that will make the test 6 pass.

6 DIMACS file reading

We will use our structures to represent logical formulas in a conjunctive normal form (CNF). The formula is a conjunction of terms where each term is a literal disjunction. Literals are our boolean variables.

For example :

`(a_0 OR a_1 OR a_2) AND (a_3 OR !(a_4)) AND (NOT(a_0) OR a_3)` is a formula in CNF.

A classical problem is to find a value for each variable that make the formula true. In the preceding example, `a_0 = T`, `a_1 = F`, `a_2=T`, `a_3 = T` et `a_4 = F` solve the problem. This problem is a problem NP-complete classical but it exists the multiple solver (called SAT-solver) that are capable to solve formulas with millions of literals.

It exists a file format to define formulas under the form CNF, the DIMACS format. This is defined as presented in figure 1

```
File format
The benchmark file format will be in a simplified version of the DIMACS format:

c
c start with comments
c
c
p cnf 5 3
1 2 3 0
4 -5 0
-1 4 0

- The file can start with comments, that is lines begining with the character c.
- Right after the comments, there is the line p cnf nbvar nbclauses
indicating that the instance is in CNF format; nbvar is the exact
number of variables appearing in the file; nbclauses is the exact
number of clauses contained in the file.
- Then the clauses follow. Each clause is a sequence of distinct
non-null  numbers between -nbvar and nbvar ending with 0 on the same
line;  it cannot contain the opposite literals i and -i
simultaneously.
- Positive numbers denote the corresponding variables.
- Negative numbers denote the negations of the corresponding variables.
```

Figure 1: Format DIMACS

Add missing methods to build a logical expression from a DIMACS file format.

Bonus: Using the visitor pattern to save your logical expression under the DIMACS, we will suppose that the logical expression under the format CNF.