

# Assignment 3

COMP9021, Session 1, 2014

**Aims:** The main purpose of the assignment is to let you

- perform operations on files;
- manipulate structures;
- manipulate pointers;
- implement an interface.

## General description

Your program will read two kinds of data from two files: names and predicates. It will read a formal expression from standard input and check whether that expression is syntactically correct, *i.e.*, built from the names and predicates that have been read, together with a few boolean operators. Then your program will read from a third file a set of basic facts assumed to be true, and check whether the formal expression itself is true. Finally, it will find out whether it is possible to make some basic facts true to make the formal expression itself true (this being trivially the case if the answer to the previous question is positive), and in case it is, write a possible solution to a file. The program will use an abstract interface that you will have to implement. The interface and the client program will be provided.

## Detailed description

We suppose that there is in the working directory a file `names.txt` that contains sequences of symbols called *names*. The file contains at least one name. A name follows the syntactic rules that characterize C identifiers (it is a nonempty sequence of symbols that are uppercase letters, lower case letters, digits or underscores, and that does not start with a digit). Names are separated by any nonnull number of blanks, tabs and new line characters. Intuitively, names denote persons, animals, objects or whatever. The following is a possible contents for `names.txt`:

```
juliet paul
fido      peter

melissa
```

We suppose that there is in the working directory another file `predicates.txt` that contains nonempty sequences of symbols called *predicates*, in the form

`predicate_name/arity`

where `predicate_name` follows the syntactic rules that characterize C identifiers, and `arity` is a nonnegative integer (possibly equal to 0). The file contains at least one predicate. Predicates are separated by any nonnull number of blanks, tabs and new line characters. Intuitively, predicate names denote properties or relations between things denoted by names; the arity represents the number of things to which the property or relation applies. Different predicates names may have the same arity. The following is a possible contents for `predicates.txt`:

```
taller_than/2

temperature_is_now_above_25/0    rich/1
```

From the names in `names.txt` and predicates in `predicates.txt`, we define as follows the *formulas*.

- If `predicate_name` is the name of a predicate of arity  $n$  and if `names_1`, ..., `name_n` are names then `predicate_name(name_1, ..., name_n)` is a formula, called an *atom*. Note that an atom contains no space.
- If `formula`, `formula_1` and `formula_2` are three formulas then
  - `not formula`
  - `[ formula_1 and formula_2 ]`
  - `[ formula_1 or formula_2 ]`
  - `[ formula_1 implies formula_2 ]`
  - `[ formula_1 iff formula_2 ]`

are all formulas, where any number of spaces (including 0) can occur before and after the `[` and `]` symbols.

For example:

- `temperature_is_now_above_25`
- `rich(melissa)`
- `[rich(melissa) or taller_than(paul,peter)]`
- `[ temperature_is_now_above_25 and  
                  [rich(melissa) or not taller_than(paul,peter)]        ]`
- `[[rich(juliet) and rich(juliet)] iff  
                  [taller_than(paul,peter) iff rich(juliet)]]`

are all formulas, whereas

- `temperature_is_now_above_30`

- `rich(gina)`
- `rich(melissa) or taller_than(paul,peter)`
- `[ temperature_is_now_above_25 and [rich(melissa) or  
not taller_than(paul,peter,juliet)] ]`
- `[[rich(juliet) and rich(juliet)] iff  
not [[taller_than(paul,peter) iff rich(juliet)]]]`

are not formulas.

We suppose that there is in the working directory a third file `true_atoms.txt` that contains atoms, representing an *interpretation*. The file might contain no atom at all. No atom occurs twice in the file. Atoms are separated by any nonnull number of blanks, tabs and new line characters. The following is a possible contents for `true_atoms.txt`:

```
temperature_is_now_above_25

taller_than(paul,peter)      rich(paul)
rich(fido) rich(peter)
```

It is then possible to establish whether a formula is true or false in the interpretation as follows:

- An atom is true just in case it occurs in `true_atoms.txt`.
- A formula of the form `not formula` is true just in case `formula` is false.
- A formula of the form `[ formula_1 and formula_2 ]` is true just in case `formula_1` and `formula_2` are both true.
- A formula of the form `[ formula_1 or formula_2 ]` is true just in case at least one of `formula_1` and `formula_2` is true.
- A formula of the form `[ formula_1 implies formula_2 ]` is false just in case `formula_1` is true and `formula_2` is false.
- A formula of the form `[ formula_1 iff formula_2 ]` is true just in case `formula_1` and `formula_2` are both true, or `formula_1` and `formula_2` are both false.

For instance, with respect to the previous interpretation:

- `temperature_is_now_above_25` is true.
- `rich(melissa)` is false.
- `[rich(melissa) or taller_than(paul,peter)]` is true.

- `[ temperature_is_now_above_25 and  
          [rich(melissa) or not taller_than(paul,peter)] ]` is false.
- `[[rich(juliet) and rich(juliet)] iff  
      [taller_than(paul,peter) iff rich(juliet)]]` is true.

A formula is *satisfiable* if it is true in **at least** one interpretation. If a formula is true in the interpretation stored in the file `true_atoms.txt` then that formula is trivially satisfiable. Some formulas might be false in the interpretation stored in `true_atoms.txt`, but still be satisfiable. For example:

- `rich(melissa)` is satisfiable.
- `[ temperature_is_now_above_25 and  
      [rich(melissa) or not taller_than(paul,peter)] ]` is satisfiable.
- `[rich(juliet) and not rich(juliet)]` is not satisfiable.
- `[[not rich(juliet) and not rich(fido) ] and  
      [rich(juliet) implies rich(fido)]]` is satisfiable.
- `[[rich(juliet) and not rich(fido) ] and  
      [rich(juliet) implies rich(fido)]]` is not satisfiable.

Your program will:

- take a sequence of symbols from standard input (possibly over many lines), ending in Carriage return followed by Control D (Control Z in Windows) to signal end of input;
- determine if the sequence of symbols is a formula;
- if the answer to the previous question is yes, determine if the formula is true in the interpretation stored in `true_atoms.txt`;
- if the previous question has been addressed and has received a negative answer, determine if the formula is satisfiable and if it is, write a **minimal** interpretation that makes it true in a file `witnesses_satisfiability.txt`, one atom per line (in any order) with no repetition.

By a minimal interpretation that makes a formula true, we mean that if one atom is removed from the interpretation then the resulting interpretation does no longer make the formula true.

## Sample outputs

Here are the expected outputs of your program with the files considered above. Remember that the end of input is signaled by carriage return followed by Control D (or Control Z in Windows).

```
$ a.out
Input possible formula: temperature_is_now_above_25
Possible formula is indeed a formula.
Formula is true in given interpretation.
$ a.out
Input possible formula: rich(melissa) or taller_than(paul,peter)
Possible formula is not a formula.
$ a.out
Input possible formula: rich(melissa)
Possible formula is indeed a formula.
Formula is false in given interpretation.
Formula is satisfiable.
$ cat witnesses_satisfiability.txt
rich(melissa)
$ a.out
Input possible formula:
[ temperature_is_now_above_25 and
  [rich(melissa) or not taller_than(paul,peter)] ]
Possible formula is indeed a formula.
Formula is false in given interpretation.
Formula is satisfiable.
$ cat witnesses_satisfiability.txt
temperature_is_now_above_25
$ a.out
Input possible formula:
[[rich(juliet) and rich(juliet)] iff [taller_than(paul,peter) iff rich(juliet)]]
Possible formula is indeed a formula.
Formula is true in given interpretation.
$ a.out
Input possible formula:
[[rich(juliet) and not rich(fido) ] and [rich(juliet) implies rich(fido)]]
Possible formula is indeed a formula.
Formula is false in given interpretation.
Formula is not satisfiable.
$ a.out
Input possible formula: [ rich(melissa) and
  [ taller_than(paul,peter) or rich(paul) ] ]
Possible formula is indeed a formula.
Formula is false in given interpretation.
Formula is satisfiable.
$ cat witnesses_satisfiability.txt
rich(melissa)
taller_than(paul,peter)
```

## Templates

The files `reason.c` and `logic.h` are provided. You have to implement the interface given by `logic.h` in one or more files.

Here is the contents of `reason.c`:

```

/* ****
 * Description:
 *
 * Written by *** for COMP9021
 *
 * Other source files, if any, one per line, starting on the next line:
 *     logic.c
 **** */

```

```
#include <stdio.h>
#include <stdlib.h>
#include "logic.h"

int main(void) {
    FILE *file = fopen("names.txt", "r");
    if (!file) {
        printf("Could not open names file. Bye!\n");
        return EXIT_FAILURE;
    }
    get_constants(file);
    fclose(file);
    file = fopen("predicates.txt", "r");
    if (!file) {
        printf("Could not open predicates file. Bye!\n");
        return EXIT_SUCCESS;
    }
    get_predicates(file);
    fclose(file);
    printf("Input possible formula: ");
    Formula form = make_formula();
    if (!form || !is_syntactically_correct(form)) {
        printf("Possible formula is not a formula.\n");
        return EXIT_SUCCESS;
    }
    printf("Possible formula is indeed a formula.\n");
    file = fopen("true_atoms.txt", "r");
    if (!file) {
        printf("Could not open interpretation file. Bye!\n");
        return EXIT_FAILURE;
    }
    Interpretation interp = make_interpretation(file);
```

```

    fclose(file);
    if (is_true(form, interp)) {
        printf("Formula is true in given interpretation.\n");
        return EXIT_SUCCESS;
    }
    printf("Formula is false in given interpretation.\n");
    if (is_satisfiable(form))
        printf("Formula is satisfiable.\n");
    else
        printf("Formula is not satisfiable.\n");
    return EXIT_SUCCESS;
}

```

And here is the contents of [logic.h](#):

```

#ifndef LOGIC_H
#define LOGIC_H

#include <stdbool.h>
#include <stdio.h>

typedef struct formula *Formula;
typedef struct interpretation *Interpretation;

void get_constants(FILE *);
void get_predicates(FILE *);
Formula make_formula();
Interpretation make_interpretation(FILE *);
bool is_syntactically_correct(Formula);
bool is_true(Formula, Interpretation);
bool is_satisfiable(Formula);

#endif

```

## Assessment

Up to eight marks will reward correctness of solutions by automatically testing your program on some tests, all different to the provided examples. More precisely:

- up to 3 marks will be awarded for determining whether the input is a formula;
- up to 3 marks will be awarded for determining whether a formula is true in given interpretation;
- up to 2 marks will be awarded for determining a formula false in given interpretation is satisfiable, and giving a suitable interpretation in [witnesses\\_satisfiability.txt](#).

Up to one mark will reward good formatting of the source code and reasonable complexity of the underlying logic as measured by the level of indentation of statements. For that purpose, the `mycstyle` script will be used, together with your customised `style_sheet.txt`, that you have to submit, and in which **Maximum level of indentation has to be set to 4** for this assignment. If the script identifies problems different to excessive indentation levels then you will score 0 out of 1. If the script identifies excessive indentation levels (which means that at least one line exhibits at least 5 indentation levels), then you will score 0.5 out of 1. If the script identifies no problem then you will score 1 out of 1. If your program attempts too little and contains too little code then the `mycstyle` script won't be used and you will score 0 out of 1.

Up to one mark will reward good comments, good choice of names for identifiers and functions, readability of code, simplicity of statements, compactness of functions. This will be determined manually.

Late assignments will be penalised: the mark for a late submission will be the minimum of the awarded mark and 10 minus the number of full and partial days that have elapsed from the due date.

## Submission

Your program will be stored in a number of files. You cannot modify `reason.c` nor `logic.h`. You have to submit all other files that make up your program, including (and possibly exclusively) `logic.c`. Go through `assignment_checklist.pdf` and make sure that you can tick all boxes (or at the least, are aware that you should tick them all). Then upload your files (the source code of your program and your customised `style_sheet.txt`) using WebCMS.

Of course, you should test your program and expect it to be tested with `names.txt`, `predicates.txt`, and `true_atoms.txt` having different contents to those provided.

Upload your files using WebCMS. Assignments can be submitted more than once: the last version is marked. Your assignment is due by June 8, 11:59pm.

## Reminder on plagiarism policy

You are permitted, indeed encouraged, to discuss ways to solve the assignment with other people. Such discussions must be in terms of algorithms, not C code. But you must implement the solution on your own. Submissions are routinely scanned for similarities that occur when students copy and modify other people's work, or work very closely together on a single implementation. Severe penalties apply to a submission that is not the original work of the person submitting it.