Algorithm example (infallible)

```
greet
if response_is_positive // by_response
    speak_in_tone
    tell_a_joke
else if response_is_negative
{
    wonder
    ask_what_happened
    sympathize
}
else
    ask_of_mood
wish_good_luck
```

Trace example *(may contain mistakes made by the student in making up the trace)*

```
program began
    greet executed 1st time
    alternative by_response began 1st time  // begin of alternative
        condition (response_is_positive) executed 1st time - true
        condition (response_is_negative) executed 1st time - false
        branch of condition (response_is_negative) began 1st time
            wonder executed 1st time
            ask_what_happened executed 1st time // any comments beside the trace lines are
allowed
            sympathize executed 1st time
        branch of condition (response_is_negative) ended 1st time
    alternative by_response ended 1st time
    wish_good_luck executed 1st time
program ended
```

```
greet
ask_of_mood
sympathize       // this would be skipped
tell_a_joke
wish_good_luck
```

```
program began
    greet executed 1st time
    ask_of_mood executed 1st time
    tell_a_joke executed 1st time
    wish_good_luck executed 1st time
program ended
```

```
sympathize executed 1st time
```
the act ^ must be here

Algorithm syntax features:

- Case-sensitive
- Indent-sensitive
- Line separators (*<newline>*) must be set as specified (because parsing is performed in line-by-line style)
- Block braces '{' and '}' can be omitted both, but when used must occupy the whole line and the block content still must be indented
- Empty lines are not allowed
- Single-line comments must follow the specification (multiline comment is not defined)
- Arbitrary comments are not allowed

Short algorithm syntax specification:

<Algorithm> ::= *<statement>* **+** ; One or more statements

<statement> ::= *<action>* | *<alternative>* | *<loop>* | *<named_sequence>* ;

\# *«Primitives»*

<**user_ID**> ::= [\w\d=*+-]**+** ; an identifier defined by user: a word consisting of letters, digits and some non-space chars, like: 'my-while-1', 'color==yellow', 'over_color'.

<comment start> ::= '#' | '//' ;

<indent> ::= [space char]**+** | [tab char]**+** ;

<newline>::= '\n'

<number>::= [-] **?** [\d] **+** ; one of more digits (integer) following optional minus

<opt_condition_values> ::= '->' *<condition_value>***+** | ; a construction like '-> 101' or nothing (optional)

<condition_value> ::= '**1**' | '**0**' ; a chain of values an expression takes like '100011100'

\# *Common actions*

<action> ::= <**user_ID**> *<newline>* ; (a word occupying the whole line)

<condition> ::= <**user_ID**> ;

<block> ::= (*<indent>* *<statement>*) **+** ; (1) indented statements nested to a control structure

<block>::= '{' *<newline>* # (2) indented statements wrapped in '{' and '}'
(*<indent>* *<statement>*) **+**
'}' *<newline>* ;

\# *Verbose sequence*

<named_sequence> ::= '{' *<comment start>* <**user_ID**: named_sequence_name> *<newline>*
<block>
'}' *<newline>* ;

\# *Alternative (if – elseif – else)*

<alternative> ::= *<if_branch>*
<else-if_branch> ***** # Zero or more branches
<else_branch> **?** # Zero or one branch
;

<if_branch> ::= '**if**' *<condition>* < *opt_condition_values>* *<comment start>* < **user_ID**: *alternative _name>* *<newline>*
<block>;

<else-if_branch> ::= '**else**' '**if**' *<condition>* *<opt_condition_values>*<newline>
<block>;

<else_branch> ::= '**else**' *<newline>*
<block>;

\# *Loops*

<loop> ::= *<while_loop>*| *<do_loop>* | *<do-until_loop>*| *<for_loop>* | *<foreach_loop>*

<while_loop> ::= '**while**' *<condition>* *<opt_condition_values>* *<comment start>* <**user_ID**: loop_name> *<newline>*
<block>;

<*do_loop*> ::= ‘**do**’ <*comment start*> <**user_ID**: *loop_name*> <*newline*>
<*block*>
‘**while**’ <*condition*> <*opt_condition_values*> ;

<*do-until_loop*> ::= ‘**do**’ <*comment start*> <**user_ID**: *loop_name*> <*newline*>
<*block*>
‘**until**’ <*condition*> <*opt_condition_values*> ;

<*for_loop*> ::= ‘**for**’<**user_ID**: *variable_name*> ‘**from**’<*number*> ‘**to**’<*number*> ‘**step**’<*number*> <*opt_condition_values*><*comment start*> <**user_ID**: *loop_name*>
<*newline*>
<*block*>;

< *foreach_loop*> ::= ‘**for each**’<**user_ID**: *variable_name*> ‘**in**’<**user_ID**: *container_name*> <*opt_condition_values*><*comment start*> <**user_ID**: *loop_name*> <*newline*>
<*block*>;

Trace syntax features:

- Case-sensitive
- Indent-*in*sensitive
- Line separators (<*newline*>) must be set as specified (because parsing is performed in line-by-line style)
- Empty lines are not allowed
- Single-line comments at end of line are allowed but mean nothing (multiline comment is not defined)

Short trace syntax specification:

```
<Trace>  ::=    ‘program began’ <newline>
                ( <act_line><newline> ) *    # Zero or more acts
                ‘program ended’ ;

<nth_time> ::=[\d]+ ( ‘st’ | ‘nd’ | ‘rd’ | ‘th’ ) ‘time’   ; # ‘1st time’, ‘5th time’ or so

<value>::=      ‘true’ | ‘false’ ;

<struct> ::=    ‘sequence’ | ‘alternative’ | ‘loop’ ;

<act_line>::=   <user_ID: action_name> ‘executed’ <nth_time> |          # simple act
                <struct> <user_ID: statement_name> ‘began’ <nth_time> |          # begin of complex act
                <struct> <user_ID: statement_name> ‘ended’ <nth_time> |          # end of complex act
                ‘condition’ [ ‘of’<struct>] ‘(’<user_ID: condition_name> ‘)’ ‘evaluated’ <nth_time> ‘—‘ <value> | # evaluation-of-condition act
                ‘branch of condition’ ‘(’<user_ID: condition_name> ‘)’ ‘began’ <nth_time>|
                ‘branch of condition’ ‘(’<user_ID: condition_name> ‘)’ ‘ended’ <nth_time> |
                ‘else branch’ ‘(’<user_ID: condition_name> ‘)’ (‘began’ | ‘ended’) <nth_time> |
                ‘iteration’ <number> ‘of loop’ ‘(’<user_ID: loop_name> ‘)’ ‘began’ <nth_time> |
                ‘iteration’ <number> ‘of loop’ ‘(’<user_ID: loop_name> ‘)’ ‘ended’ <nth_time> |
                ‘executed’ ‘initialization’ ‘(’<user_ID: loop-init_name> ‘) <nth_time>|
                ‘executed’ ‘update’ ‘(’<user_ID: loop-update_name> ‘)’ <nth_time>
                ;

        # Example:     condition of loop (not_green) evaluated 1st time - true
```