COS341 Semester Project 2025 : The Students' Programming Language "SPL"

A fully operational compiler must be implemented as a group project with four students per group. The exact submission deadline will be announced via ClickUp. This work-sheet provides grammar and vocabulary for the compiler's front-end, such that you can start working. Further information concerning the compiler's back-end will be released later in separate work-sheets, whenever some relevant topics have been treated in the weekly lectures. You have plenty of "**creative freedom**" implement everything by means of your own methods, as you like, as long as the compiler emits correct target-code at the end of the project. And now: HAPPY WORKING **:)**

General assumption: No token contains any blank_space. In other words: *if* the compiler encounters any blank_space while "eating" the input string, the compiler assumes that another "new" token has its beginning "behind" the blank_space. As *"blank_space"* we *also* regard *"tabs"* or *"new_line"*. On the basis of thes assumptions, this year's version of the "**Students' Programming Language**" (**SPL**) has the following **Context-Free Grammar** in which the **terminal** symbols are shaded with green colour for the sake of clarity:

```
SPL_PROG  ::=       glob { VARIABLES }  // comment: may have globals
                    proc { PROCDEFS }  // comment: without any return
                    func { FUNCDEFS }  // comment: return something
                    main { MAINPROG }

VARIABLES ::=       // comment: nothing, nullable
VARIABLES ::=       VAR VARIABLES

VAR       ::=       user-defined-name  // comment: vocabulary: see below
NAME      ::=       user-defined-name

PROCDEFS  ::=       // comment: nothing, nullable
PROCDEFS  ::=       PDEF PROCDEFS

PDEF      ::=       NAME ( PARAM ) { BODY }  // procedures return nothing

FDEF      ::=       NAME ( PARAM ) { BODY ; return ATOM }

FUNCDEFS  ::=       FDEF FUNCDEFS
FUNCDEFS  ::=       // comment: nothing, nullable

BODY      ::=       local { MAXTHREE } ALGO    // may have local variables

PARAM     ::=       MAXTHREE     // We do not want to make the language SPL too complicated

MAXTHREE  ::=       // comment: nothing, nullable
MAXTHREE  ::=       VAR
MAXTHREE  ::=       VAR VAR
MAXTHREE  ::=       VAR VAR VAR// comment: for simplicity not more than three

MAINPROG  ::=       var { VARIABLES } ALGO    // may have local variables

ATOM      ::=       VAR
ATOM      ::=       number // comment: vocabulary: see below
```

```
ALGO      ::=     INSTR // in SPL we do not allow "empty" algorithms
ALGO      ::=     INSTR ; ALGO

INSTR     ::=     halt
INSTR     ::=     print OUTPUT
INSTR     ::=     NAME ( INPUT )        // comment: procedure call
INSTR     ::=     ASSIGN
INSTR     ::=     LOOP
INSTR     ::=     BRANCH

ASSIGN    ::=     VAR = NAME ( INPUT ) // comment: Function call
ASSIGN    ::=     VAR = TERM

LOOP      ::=     while TERM { ALGO }
LOOP      ::=     do { ALGO } until TERM

BRANCH    ::=     if TERM { ALGO }
BRANCH    ::=     if TERM { ALGO } else { ALGO }

OUTPUT    ::=     ATOM
OUTPUT    :=      string  // comment: Vocabulary: see below

INPUT     ::=     // comment: nothing, nullable
INPUT     ::=     ATOM
INPUT     ::=     ATOM ATOM
INPUT     ::=     ATOM ATOM ATOM // comment: must match parameters!

TERM      ::=     ATOM
TERM      ::=     ( UNOP TERM )
TERM      ::=     ( TERM BINOP TERM )

UNOP      ::=     neg
UNOP      ::=     not

BINOP     ::=     eq
BINOP     ::=     >
BINOP     ::=     or
BINOP     ::=     and
BINOP     ::=     plus
BINOP     ::=     minus
BINOP     ::=     mult
BINOP     ::=     div
```

Rules of the Vocabulary

1. A user-defined-name may not be identical with any green keyword of the Grammar!
2. Under the condition that Rule 1 is not violated, a user-defined-name is described by the regular expression [a...z]{a...z}*{0...9}*
3. A constant number is has the regular expression (0 | [1...9][0...9]*)
4. string : *any sequence of digits or letters* between "quotationmarks" : max. length 15!