COS341 Practical 5.

When a given SPL program went passed through all your foregoing practical software, then we know that it is "good enough" for translation to target code: We had made sure that all procedures are well "scoped", and we had made sure not to encounter value-less variables.

The translation itself follows along the lines of chapter **6** of our book and uses a *fragment* of BASIC as "target code".

See also:

- Task number **#15** in your ClickUp folder,
- The attached **slide-show** (Appendix to this sheet) for an example.

What is *NOT allowed* to be used are all the *higher-level Elements* of BASIC which would make the target code too similar to SPL itself; instead we want to use only the low-level elements of BASIC such as to "mimic" something similar to a "machine code". These **forbidden** elements include:

- BASIC's **For**-Loop command,
- BASIC's **While**-Loop command,
- BASIC's **Else**-Branch (after IF-statements).

WARNING: 0 Points *if* *your generated code contains these forbidden high-level elements!*

Translation Hints:

For SPL Procedure Calls, use BASIC's **GOSUB** command (as shown in the slide-show Appendix). The body-code of a procedure must end with BASIC's **RETURN** command for this purpose.

SPL's *composite LOGIC* Expressions (from the grammar of Practical 2) must be translated into ***cascading* GOTO *jumps*** along the lines of Fig.6.8 in our book, such that the logical operators (and, not, or) will vanish entirely from the target code.

SPL's truth values {**T**,**F**} you may translate into integer constants {**1**,**0**} which is the usual number representation of Boolean constants.

Since BASIC's high-level "**ELSE**" is also forbidden, an SPL *BRANCH* of the form

i (BOOLEXPR**) t {**ALGO*1*** } e {**ALGO*2*** }**

will have a target code structure in BASIC which looks similar like this:

```
someLineNumber : IF logicalConditionInTargetCode GOTO otherLineNumber
nextLineNumber  : targetCodeOfAlgo2
                  ...
                  GOTO exitLineNumber
otherLineNumber : targetCodeOfAlgo1
                  ...
exitLineNumber  : ...
```

SPL's boolean variables you translate into BASIC's number variables (numeric truth representation)

For String variables please keep in mind that oldschool BASIC uses the **$** indicator.

For SPL's halt command h, use BASIC's **STOP**.

Turn the page →

For SPL's numeric expressions (add., multipl., div.) use the usual mathematical symbols (+, * , /).

For SPL's output command use BASIC's **PRINT**.

For SPL's input command use BASIC's **INPUT**.

What you get from the Tutor:

You will get the **AST of an SPL Program** that is *guaranteed to be semantically correct* (as checked in the two foregoing practicals P4 and P3).

Thus you can work on this P5 even if you had nothing done in P3 nor in P4 :)

What your Software must do:

Your software must "crawl" along the AST and apply the techniques of Syntax-directed Translation in order to produce the desired BASIC program as target code, also with all the line numbers nicely in place.

How you can check for yourself whether you did well:

From the Internet you can download a BASIC Emulator in which you can *run* the BASIC program which your software has generated from the given SPL AST. If the generated BASIC program runs correctly in the emulator and correctly carries out all the intended operations, then chances are high that you have programmed a good Translator Tool.

And now: **HAPPY CODING :)**