COS**341** **Project 3** (2018): ***Executable Code*** *for* ***SPL***

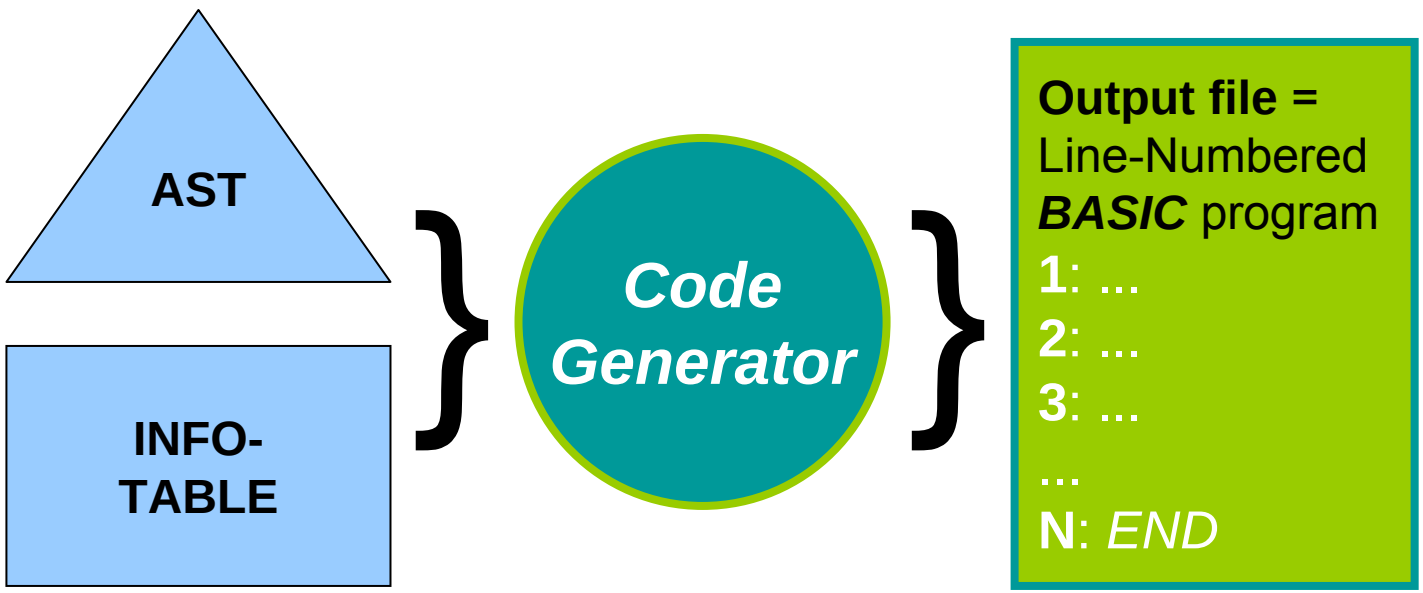
(Part 3***b***)

Project 3: Overview

- Part 3*a*: Intermediate Code **Preparation**
  (done)
- Part 3***b***: Intermediate Code Generation
  (**THIS**)
- Part 3*c*: Liveness-Analysis and
  (later) Intermediate Code Optimisation

Project 3b: Overview

- We assume that a given ***SPL*** program has been successfully parsed and also has its static semantics checked to be correct. Its abstract syntax tree and its info-table exist.



TASK

- **Apply your knowledge** of book-chapter 6 to generate LOW-LEVEL ***BASIC*** code from ***SPL*** source code**!**
  - Implement a recursive **Trans**-function for this purpose (similar to the examples shown in the book).
  - In particular:
    - **IF-statements and WHILE-statements** must be translated as per Figure 6.5 on page 129
      - *Modern BASIC's "fancy" high-level if-then{}-else{} concept may NOT be used!*
    - **Composite Boolean conditions** (e.g.: (**B_1 or B_2**)) must be "decomposed" into "cascading" GOTO jumps, as per Figure 6.8 on page 133.

Further advice follows ➔

Further Advice

- **"Split" the Code Generation into two subsequent phases**:
  - **Phase A**: Creation of "symbolic addresses" without BASIC LINE NUMBERS, as in book! Write the output into an **Auxiliary File A**.
  - **Phase B**: Read the Auxiliary File A, and then generate the proper **LINE NUMBERS from 1 to N** to obtain a proper BASIC program. Write this into the final **Output File B**.
    - *The Line Numbers from 1 to N will be needed later in Project 3**c** for the purpose of Liveness Analysis.*

Further Advice

- For SPL's String variables use the BASIC string variables that are indicated with **$**
- For SPL's constant truth values { **T**,**F** } you ***may*** use numeric representation { **0**,**1** } in BASIC
- SPL variable declarations (e.g.: **num *x***) are not translated into BASIC at all (because they have no computational effects)
  - *We only used those declarations to "fill" the static semantic information table.*

Further Advice

- **For the translation of SPL's CALL commands you *may* use BASIC's gosub/return "syntactic sugar" ☺**
  - This will make your Trans-function somewhat easier to implement for the translation of SPL procedure bodies and procedure calls ☺ **1Footnote**
    - *Later, however, in Project Part 3c (Liveness Analysis), you must do some "additional thinking" in order to find out what Line-of-Code is **SUCC[i]** when Line **i** contains a gosub or a return command*

1Footnote: Otherwise you would have to 'close' the bodies of procedures with IF-conditioned GOTO-statements, whereby the IF-conditions would somehow need to represent the various locations from where the procedure can have been called

Further Advice

- For the translation of SPL's halt command, use **goto *M*** in BASIC, where *"M"* denotes your BASIC program's maximal (largest) line-number.
  - At this maximal (largest) line-number, generate the BASIC command END.

Further Advice

- Translate ***SPL***'s **FOR**-statements similar to the translation of ***SPL***'s WHILE, whereby you should "exploit" the **semantic equivalence** ≈ between the following two constructs:

```
for ( c := 0, c < N, c := c + 1) do
        { BODY // without c }
```

≈

```
c := 0
while ( c < N ) do
            { c := c + 1 ;
              BODY // without c }
```

Assessment (Tuesday 15th May)**:**

- The generation of BASIC code alone will **not** be sufficient**!**
- In the assessment it will be tested if your generated BASIC program **runs correctly** and delivers the **right output** *according to the meaning of the original SPL program**!***
  - In other words: we want a **correct translator** (*not* just "any" translator).
- **And now**: HAPPY PAIR-PROGRAMMING! ☺☺