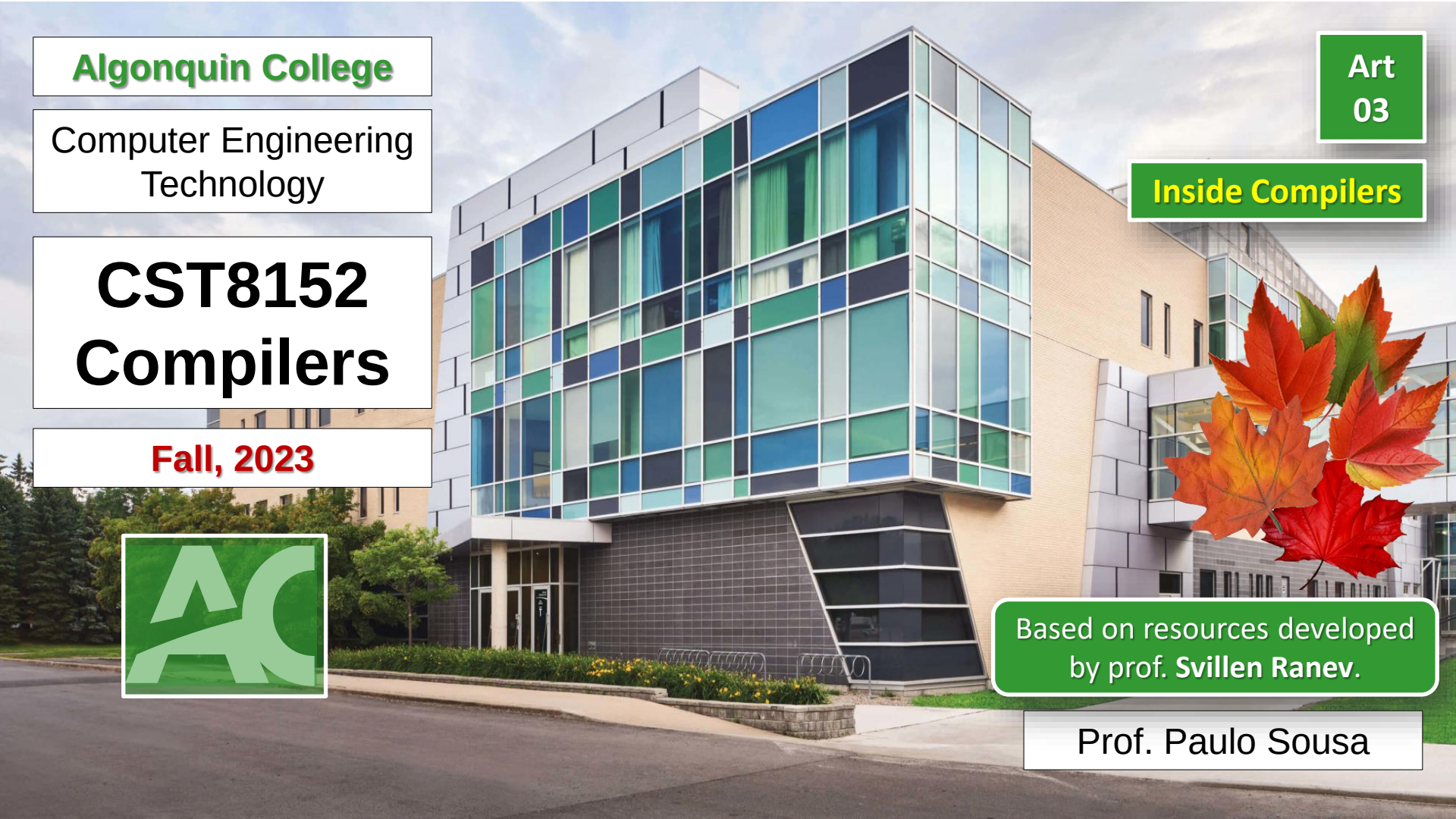
CST8152
Compilers


Algonquin College
Computer Engineering Technology
CST8152 Compilers
Fall, 2023
Art 03
Based on resources developed by prof. Svillen Ranev.
Prof. Paulo Sousa



Algonquin College

Computer Engineering Technology

CST8152 Compilers

Fall, 2023

Art 03

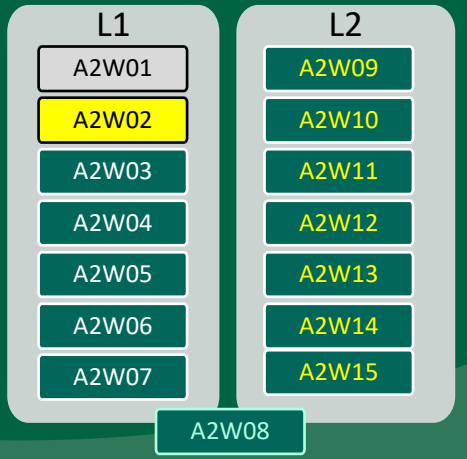
Inside Compilers

Based on resources developed by prof. **Svillen Ranev**.

Prof. Paulo Sousa

Art 4: Inside Compilers

- *General View*
- *Hybrid Compilation*

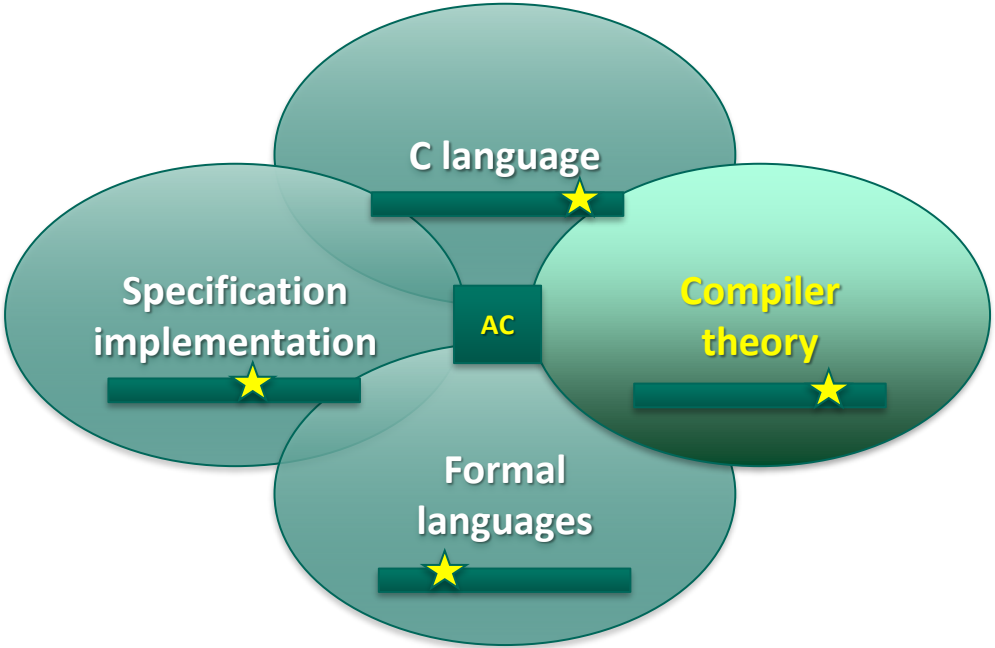


ALGONQUIN COLLEGE

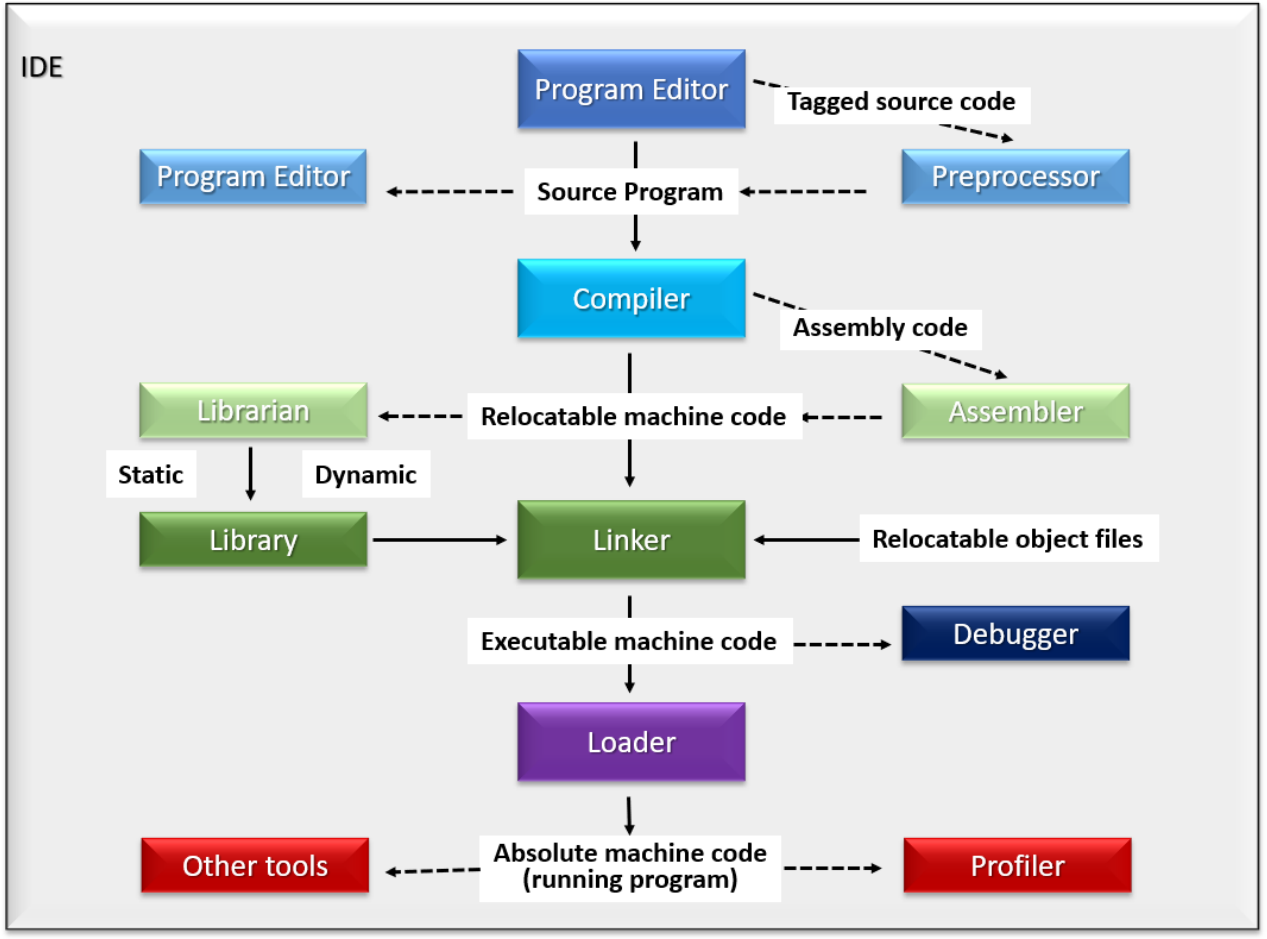
Compilers – Art. 3

General View

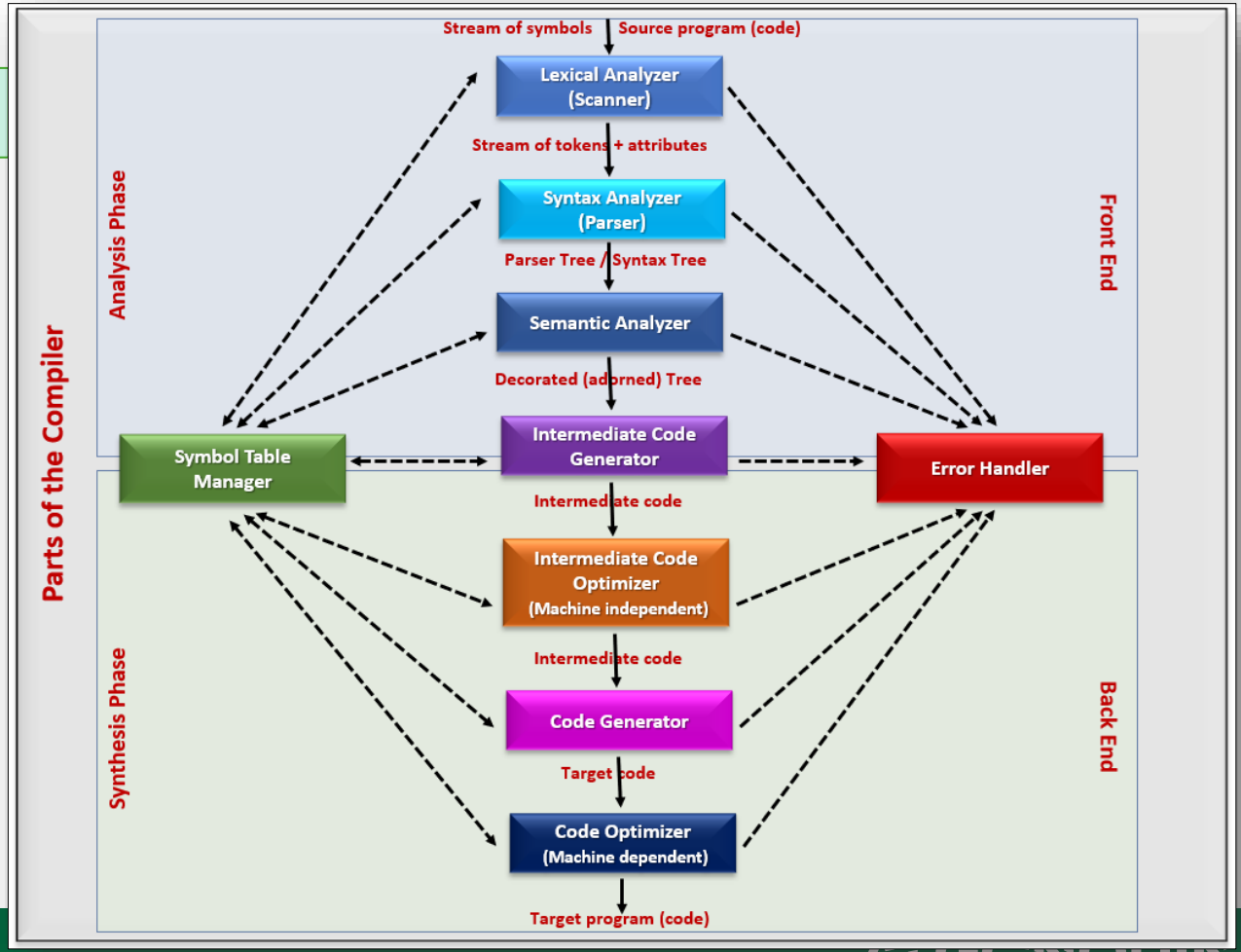
Let's start…


C language
Specification implementation
AC
Compiler theory
Formal languages

4.1 - Review



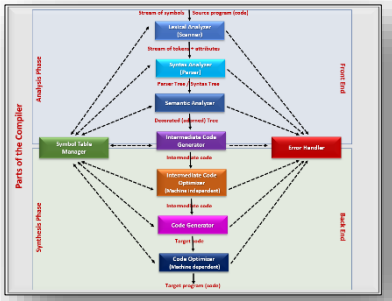
4.1 - Review


Stream of symbols
Source program (code)
Lexical Analyzer (Scanner)
Stream of tokens + attributes
Syntax Analyzer (Parser)
Parser Tree / Syntax Tree
Semantic Analyzer
Decorated (adorned) Tree
Intermediate Code Generator
Intermediate code
Intermediate Code Optimizer (Machine independent)
Intermediate code
Code Generator
Target code
Code Optimizer (Machine dependent)
Target program (code)
Symbol Table Manager
Error Handler
Parts of the Compiler
Analysis Phase
Synthesis Phase
Front End
Back End

4.1. Some concepts

Analysis part (phase)

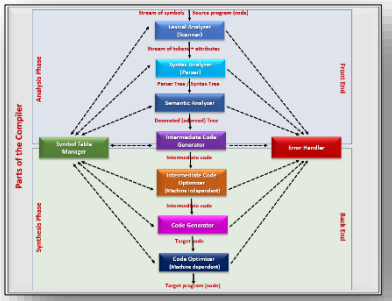
- The first phase of a compiler is called ***lexical analysis*** or ***scanning***.
  - The lexical analyzer reads the stream of characters making up the source program and groups the characters into meaningful sequences called lexemes.
  - For each lexeme, the lexical analyzer produces as output a token of the form: (token - name (code), attribute - value).



4.1. Some concepts

Analysis part (phase)

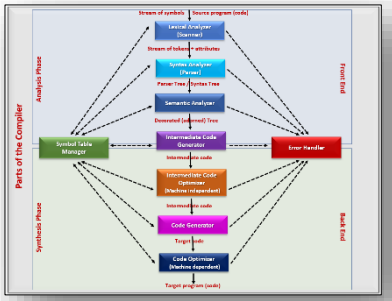
- The second phase of the compiler is ***syntax analysis*** or ***parsing***.
  - The parser uses the first components of the tokens produced by the lexical analyzer to create a tree-like intermediate representation called ***parse tree*** that depicts the grammatical structure of the token stream.
  - Typically the parse tree is reduced to another form of representation called ***syntax tree*** in which each interior node represents an operation and the children of the node represent the arguments of the operation.



4.1. Some concepts

Analysis part (phase)

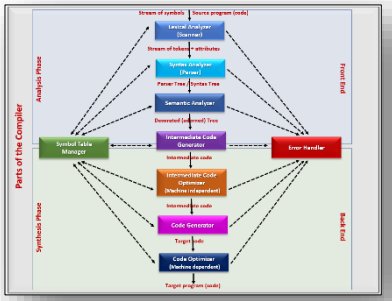
- The ***semantic analyzer*** uses the syntax tree and the information in the ***symbol table*** to check the source program for semantic consistency with the language definition.
    - It also gathers type information and saves it in either the syntax tree or the symbol table, for subsequent use during intermediate-code generation.



4.1. Some concepts

Synthesis part (phase)

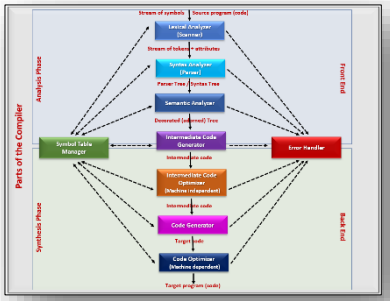
- After syntax and semantic analysis of the source program, many compilers generate an explicit low-level or ***machine-like intermediate representation***, which we can think of as a program for an *abstract machine*.
  - This intermediate representation should have two important properties:
    - it should be easy to produce and
    - it should be easy to translate into the target machine.
  - Typically, the intermediate representation separates the front end from the back end.



4.1. Some concepts

Synthesis part (phase)

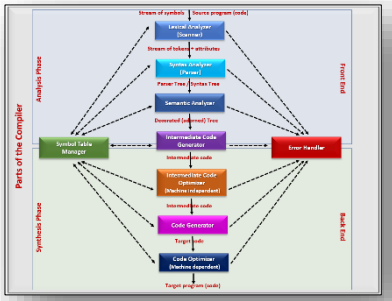
- The **machine-independent code-optimization** phase attempts to improve the intermediate code so that better target code will result.
- There is a great variation in the amount of ***code optimization*** different compilers perform. In those that do the most, the so-called "optimizing compilers," a significant amount of time is spent on this phase.
  - There are simple optimizations that significantly improve the running time of the target program without slowing down compilation too much.
- The ***code generator*** takes as input an intermediate representation of the source program and maps it into the target language



4.1. Some concepts

Synthesis part (phase)

- The ***symbol table*** is a data structure containing a record for each variable name, with fields for the attributes of the name.
- The ***error handler*** is responsible to report to the programmer the lexical, syntactical, and the semantic error discovered during the compilation process.
  - It is also responsible to prevent the compiler from producing a target code is an error has been detected.



Compilers – Art. 3

Example

Code Example

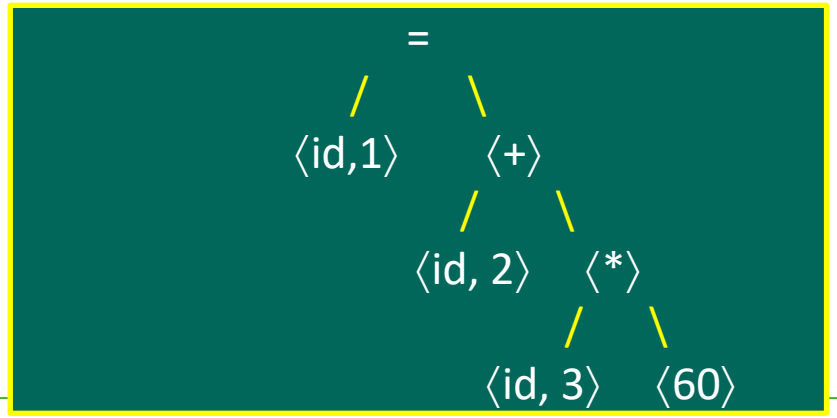
Instruction

position = initial + rate * 60

Lexical Analyzer

⟨id,1⟩ ⟨=⟩ ⟨id,2⟩ ⟨+⟩ ⟨id,3⟩ ⟨*⟩ ⟨60⟩

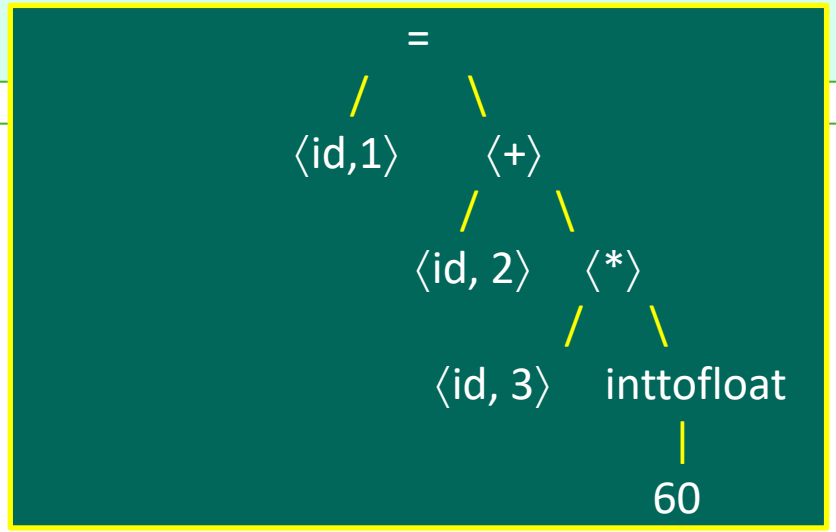
Syntax Analyzer



Symbol Table	
position	…
initial	…
rate	…

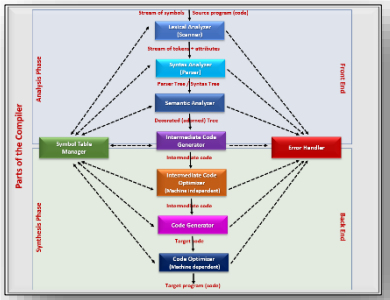
Code Example

Semantic Analyzer



Intermediate Code Generator

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```



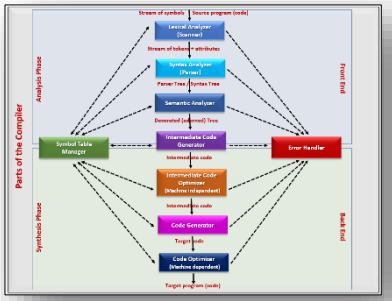
Code Example

Code Optimizer

```
t1 = id3 * 60.0
id1 = id2 + t1
```

Intermediate Code Generator

```
LDF     R2, id3
MULF    R2, R2, #60.0
LDF     R1, id2
ADDF    R1, R1, R2
STF     id1, R1
```



Compilers – Art. 4

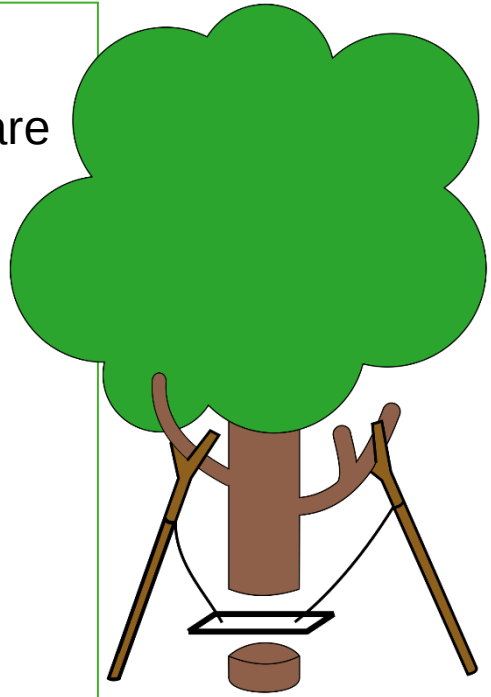
General Purpose Languages

4.2. The “ontological” problem

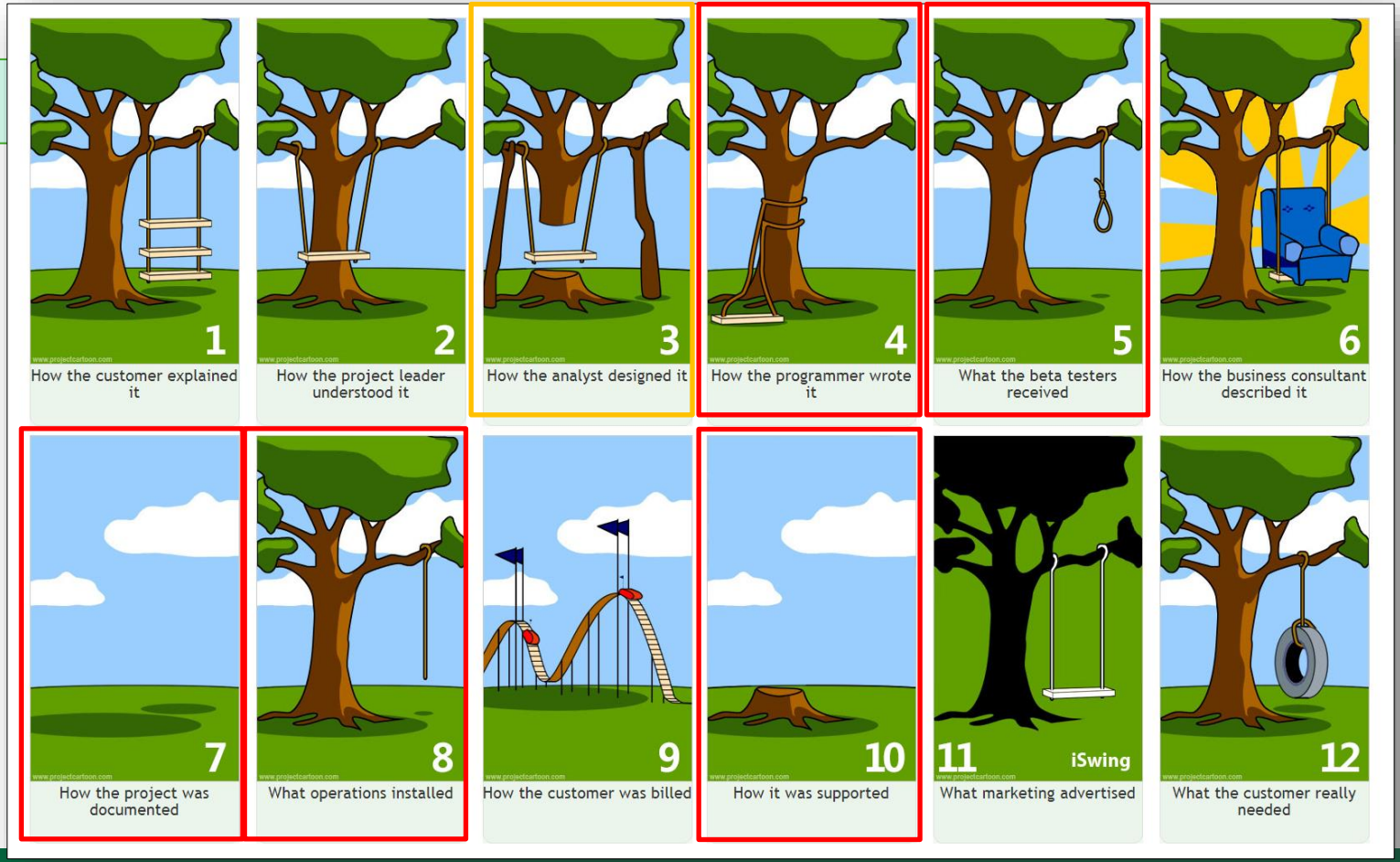
GPL (General Purpose Languages)

- Should be able to create “artefacts” for software development;
- But software needs to attend business needs;
- Business are domain-specific

Note: Part of the objective of SE (Software Engineering) is decrease the “gap” between the idea and the implementation…



I know!

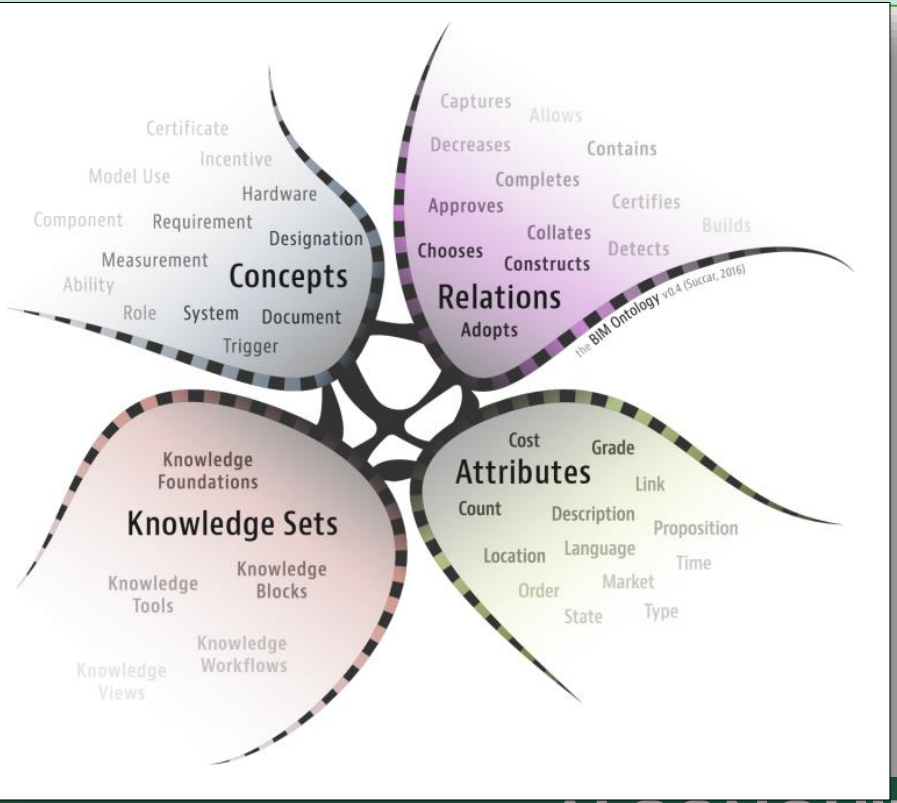

1
How the customer explained it
2
How the project leader understood it
3
How the analyst designed it
4
How the programmer wrote it
5
What the beta testers received
6
How the business consultant described it
7
How the project was documented
8
What operations installed
9
How the customer was billed
10
How it was supported
11
iSwing
What marketing advertised
12
What the customer really needed

4.2. The “ontological” idea

How to explain languages…

- For a “normal” people;

Note: Even OO is not a real “paradigm” to users, but for programmers.

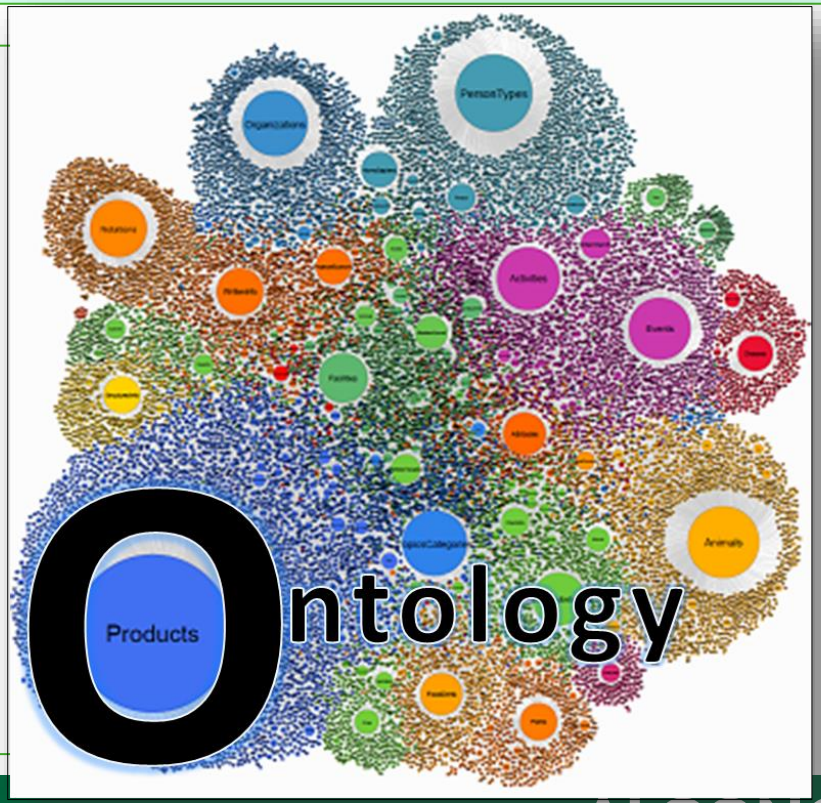


4.2. The “ontological” idea

How to explain languages…

Note: Is it possible to create high-level languages?

Imagine the complexity of concepts and associations required to implement “ontological” languages…



https://www.enterrasolutions.com/wp-content/uploads/2015/03/Ontology-01.png

Compilers – Art. 3

Concluding

Review

- *Define the elements of the context of the diagram.*

Source:
https://static.wixstatic.com/media/7594af_51a81a8ccc5f418281f52c8bdd2dd618~mv2.jpg

Some Questions

1. *Why to separate analysis from synthesis?*
2. *Explain the purpose of analysis and give one example (different from book);*
3. *Do the same for synthesis phase.*
4. *What is the complexity of GPL?*

Open questions…

- Any doubts / questions?
- How we are until now?

az.allevents.in/events3/banners/26b150363d5757da8578fa6a1481585a368f12d4247adfe95837e6ea6c5ab2af-rimg-w1200-h549-gmir.jpg?v=1569691155

Image URL: https://cdn-

Compilers – Art. 4

Thank you for your attention!