Intermediate Code Generation

We could translate the source code directly into the target Language, but there are **benefits** to having an intermediate, Machine independent code:

- A clear distinction between the machine-independent and machine-dependent parts of the compiler.
- We could apply machine independent code optimization techniques.

The syntax tree formed after Syntax and Semantic Analysis is used for Intermediate Code generation.

Three Address Code

Three address Code RULES:

- At most 3 Operands.
- At most 1 Operator on the right side of the equation.

Three address Code contains statements of the form:

- Assignment statements of the form x := y op z
- Assignment statements of the form x := op z where op is a unary operation (e.g. unary minus, logical negation, shift and convert operators)
- Copy statements of the form x:=y
- Here x, y, z are names, constants and compiler generated temporary variables.
- Op stands for Arithmetic or Logical operator, only one operator is permitted.

Three Address Code conversion & optimization Example:

- t1 = intToFloat(60)
- t2 = id1 * t1
- t3 = id2 + t2
- id3 = t3

So the optimized code is:

- t1 = id3 * 60.0
- id1 = id2 * t1

Summary:

- Intermediate code generation is concerned with the production of a simple machine independent representation of the source program.
- We saw three-address code as an example of such intermediate code and how structures can be translated into it.