Introduction to Language Theory and Compilation: Exercises

Session 7: `Semantic Analysis`

ULB

Introduction

Stages of compiling:

1. *Preprocessing*
2. Lexical analysis
3. Parsing (syntactical analysis)
4. **Semantic analysis**
5. Code generation
6. *Code optimization*
7. *Linker*

The italic steps are optional or depend of the input/output language.

Role of of the Semantic analysis

The **Semantic analysis** checks if lexical units make sense in their context.

- Resolve names: association between the labels (lexical unit) and the identifiers (from the table of symbols).
- Construct the table of symbols: add all required information (identifier, type, ...) into the table of symbols.
- Check type of expressions: conversion between integer/real, strings, ...
- Check if the constants are well defined (non-null).

Parsing versus Semantic analysis

Parser

Check if the words respect the grammar, apply the rules of the grammar.

Semantic analysis

Check if the words make sense in the matched rule of the grammar, observe some mechanisms to give a sense.

$\Longrightarrow$ Parser for grammar and Sementic analysis for sense.

ULB

Exercise 1

Assuming that the Java grammar respects:

<IF>	→	**if** <EXPRESSION> <END_IF>
<EXPRESSION>	→	<BOOLEAN_EXPRESSION>
<EXPRESSION>	→	<NUMERIC_EXPRESSION>
<EXPRESSION>	→	<CHAR_EXPRESSION>
<EXPRESSION>	→	<OBJECT_EXPRESSION>

Explain why these methods are (not) semantically correct.

```
public class Exercise1{
    public static void exerciseA(String[] args){
        if(!args) throw new IllegalArgumentException();
        System.out.println("All right!");
    }
    public static void exerciseB(String[] args){
        if(new Boolean(true))
            throw new IllegalArgumentException();
        System.out.println("All right!");
    }
}
```

Exercise 2

Write the expression tree with the type in each node. Do not forgot to respect the arithmetical priority of operators.

<EXP>	→	<EXP> + <EXP>
<EXP>	→	<EXP> * <EXP>
<EXP>	→	<EXP> - <EXP>
<EXP>	→	<EXP> / <EXP>
<EXP>	→	**(int)** <EXP>
<EXP>	→	**(** <EXP> **)**
<EXP>	→	**ID**
<EXP>	→	**Bool**
<EXP>	→	**Real**
<EXP>	→	**Int**
<EXP>	→	**String**

```
public class Exercise2{
    public static void main(String[] args){
        System.out.println(3+4+"."+(1+2*3.0));
    }
}
```

Exercise 3

Identify the scope of all variables

- Give the Table of Symbols (ToS)
- Give the parse tree of each numerical expression
- Annotated the parse trees with changes of ToS

Report any semantic error.

```
public class Exercise3{
    public static final double PI = 3.141592653589793;
    public static double diameter;
    public static void main(String[] args){
        double perimeter = 50;
        diameter = perimeter / PI;//15.915494309189533
        final int diameter = Exercise3.diameter;
        System.out.println(((int)(diameter*100))/100.0);
    }
}
```