Computer Architecture and Programming Languages    Course: CO20-320241
Jacobs University Bremen    9 December, 2018
Romelda Blaceri

# Homework 12

## Problem 12.1
**Solution:**

| Language | Generation | Type of Language | Type Checking |
|---|---|---|---|
| C | 3rd | Imperative, Von Neumann | Static |
| C++ | 3rd | Imperative, Object-oriented | Static |
| B | 3rd | Imperative | Typeless |
| Java | 3rd | Imperative, Object-Oriented | Static |
| Python | 3rd | Imperative, Object-Oriented, Scripting | Dynamic |
| Ruby | 3rd | Imperative, Object-Oriented, Scripting | Dynamic |
| Pascal | 3rd | Imperative | Static |
| Basic | 3rd | Imperative | Dynamic |
| Smalltalk | 4th | Object-Oriented, Scripting | Dynamic |
| Perl | 3rd | Imperative, Object-Oriented, Scripting | Dynamic |
| PHP | 3rd | Imperative, Object-Oriented, Scripting | Dynamic |
| Prolog | 5th | Declarative | Dynamic |

Note: Considering the following reference
*https://en.wikipedia.org/wiki/Fourth-generation_programming_language*
some advanced 3GLs (3rd Generation Languages) like Python, Ruby, and Perl combine some 4GL abilities within a general-purpose 3GL environment. Also, libraries with 4GL-like features have been developed as add-ons for most popular 3GLs. This has blurred the distinction of 4GL and 3GL. Therefore, I have considered them as 3GL languages, but in many websites it is also mentioned that we can't actually make a proper classification.

## Problem 12.2
**Solution:**

```
<expression> = <variable> | <expression> ″+″ <variable >| <expression> ″−″ <variable> |
               <expression> ″*″ <variable> | <expression> ″/″ <variable>

<condition> = <variable> | <condition> ″<″ <variable> | <condition> ″>″ <variable> |
              <condition> ″<=″ <variable> | <condition> ″>=″ <variable> |
              <condition> ″==″ <variable> | <condition> ″!=″ <variable>

<ternary−operator> = <variable> ″=″ <condition> ″?″ <expression> ″:″ <expression>

// Expression and condition can also be expressed in the following form:
// <expression> = <variable> [(″+″ | ″−″ | ″*″ | ″/″) <variable >]
// <condition> = <variable> [(″<″, ″>″, ″<=″, ″>=″, ″==″, ″!=″) <variable >]
```

## Problem 12.3
**Solution:**

```
<id> = <identifier> [(″<″, ″>″, ″<=″, ″>=″, ″==″, ″!=″) (<identifier> | <constant >)]

<variable> = <identifier> | <constant>

<expression> = <variable> [(″+″ | ″−″ | ″*″ | ″/″) <variable >] | <identifier> ″++″ |
               <identifier> ″−−″

<statement> = <identifier> ″=″ <expression> | <identifier> ″++″ | <identifier> ″−−″

<statement/s> = <loop> | <statement> | <statement/s> <statement>

<loop> = ″while″ ″(″ <id> ″)″ ″{″ <statement/s> ″}″
```

Explained syntax:
- usage of [] - means that the expression inside the brackets is optional (in our case, <expression> can either be just a <variable>, or a <variable> followed by what is inside the brackets)
- usage of () without quotes - means that only one of the operators/expressions at a time written inside can be part of our expression

Note: I have assumed there is no need for semicolon in the end of every statement as it is not specified which language does the while loop belong to. In case it is needed, just add ";" in the end of every statement.