# Laboratory 1

Statement: Considering a small programming language (that we shall call mini-language), write 3 small programs in this language.

Deliverables:
p1.* , p2.*, and p3.* and p1err.* - small programs written in your programming language (p1, p2, p3 should be lexically correct; p1err should contain 2 types of lexical errors).
For example:

- p1 and p2: compute de max/min of 3 numbers; verify if a number is prime, compute gcd of 2 numbers, compute the solutions for a 2nd order equation, aso
- p3: compute the sum of n numbers, compute the max/min of n numbers

---

## WideLang

A strongly typed language that takes up your whole screen in width, while also being extremely hard to remember. Tries to be grammatically correct, fails spectacularly in doing so.

**comment**: # or //

**multiple line comment:** /* code */

**types:** boolean (notTrue or absolutelyTrue), integer, unsigned integer, number (equivalent to float), unsigned number, string, char, nothing (equivalent to void)

**delimiter:** .

**keywords:**

- 'as' = variable type declaration
    - used as: **variableName is type**

- ‘returns’ = function type declaration
    - used as: **fun returns type functionName with parameters**
- ‘fun’ = function keyword
    - followed by the function name
    - parameters are specified after the ‘with’ keyword
    - multiple parameters are separated by comma
    - function body is between the keywords ‘does’ and ‘ifCalled’
- ‘with’ = parameter declaration
- ‘does’ = function body start
- ‘ifCalled’ =  function body end
- ‘inCase’ = conditional keyword
    - conditions must be terminated by a question mark ‘?’
    - if the condition is met, then the code between the keywords ‘thenThis’ and ‘happens’ is executed
    - if condition is not met, then the code between the keywords ‘otherwiseThis’ and ‘happens’ is executed
- ‘this … happens’ = code block
- ‘takes’ = assignment operator
- ‘return’ = return keyword
- ‘for every’ = for statement
    - expects a variable than can take a value
    - after the keyword ‘that’, WideLang expects a condition
- ‘list’ = array keyword
    - needs a type specified by ‘of’
    - ex: example as list of integer.
    - can access elements like list at index

**reserved words:**

- ‘chill’ = does nothing
- ‘startOfRunningProgram’ = main function, entry point
- ‘scream’ = prints to console
- ‘notTrue’ = false
- ‘absolutelyTrue’ = true
- ‘intimidateTheUserForInputInTheConsole’ = gets input from console
- ‘append’ = appends strings
- ‘at’ = used for accessing elements in a list
    - example: list at 3

**relational operators:**

- ‘greaterThan’ = equivalent to C’s ‘>’
- ‘greaterThen’ = also equivalent to C’s ‘>’, valid in WideLang because programmers cannot spell
- ‘equals’ = equality operator
- ‘smallerThanOrEqualTo’ = equivalent to C’s ‘<=’
- ‘smallerThenOrEqualTo’ = equivalent to C’s ‘<=’, valid in WideLang because programmers cannot spell

**algebraic operators:**

- ‘dividedBy’ : / division
- ‘plus’: + sum
- ‘times’: * multiplication

```
# p1. compute max of 3 numbers

fun returns integer MaximumOfThreeNumbers with number1 as intege
does
    maximumNumber as integer takes number1.
    inCase number2 greaterThan maximumNumber ?
    this
        maximumNumber takes number2.
    happens
    otherwise this
        chill
    happens

    inCase number3 greaterThan maximumNumber?
    this
        maximumNumber takes number3.
    happens

    return maximumNumber.
ifCalled

fun returns nothing StartOfRunningProgram
does
    scream MaximumOfThreeNumbers with 1, 3, 2.
ifCalled
```

```
# p2. verify if number is prime

fun returns boolean IsPrimeNumber with numberToCheck as integer
does
    inCase numberToCheck equals 2
    this
        return absolutelyTrue.
    happens
    inCase numberToCheck modulus 2 equals 0
    this
        return notTrue.
    happens
    for every divisor takes 3 that divisor smallerThanOrEqualTo
```

```
        inCase numberToCheck modulus divisor equals 0
        this
            return notTrue.
        happens
    return absolutelyTrue.
ifCalled

fun returns nothing StartOfRunningProgram
does
    scream IsPrimeNumber with 132.
ifCalled
```

```
# p3: compute the sum of n numbers

fun returns integer crunchTheSumOfNumbersFromArray with numbers
does
    sumOfNumbers as integer takes 0.
    for every i takes 1 that i smallerThanOrEqualTo n do i takes
        sumOfNumbers takes sumOfNumbers plus numbers at integer
    return sumOfNumbers.
ifCalled

fun returns nothing StartOfRunningProgram
does
    n as integer.
    numbers as list of integer.
    scream "GIVE ME THE NUMBER OF ELEMENTS: ".
    intimidateTheUserForInputInTheConsole with n.
    scream "NOW GIVE ME ALL YOUR NUMBERS".
    for every i takes 1 that i smallerThanOrEqualTo n do i takes
        scream "GIVE ME THE " append i append "ST/ND/RD/TH NUMBE
        intimidateTheUserForInputInTheConsole with numbers at in
    scream crunchTheSumOfNumbersFromArray with numbers, n.
ifCalled
```

```
# p1err: compute the multiplication of n numbers

fun returns integer crunchTheMultiplicationOfNumbersFromArray wi
does
    sumOfNumbers is integer takes 0.
    for every i takes 1 thet i smallerThanOrEqualTo n do i takes
        sumOfNumbers takes sumOfNumbers times numbers at integer
    return sumOfNumbers.
ifCalled

fun returns nothing StartOfRunningProgram
does
    n as integer.
    numbers as list of integer.
    scream "GIVE ME THE NUMBER OF ELEMENTS: ".
    intimidateTheUserForInputInTheConsole with n.
    scream "NOW GIVE ME ALL YOUR NUMBERS".
    for every i takes 1 that i smallerThanOrEqualTo n do i takes
        scream "GIVE ME THE " append i append "ST/ND/RD/TH NUMBE
        intimidateTheUserForInputInTheConsole with numbers at in

    scream crunchTheMultiplicationOfNumbersFromArray with number
ifCalled
```

Is and thet are lexical errors because the tokens cannot be classified.