Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A11

Language Specification

Team:

[Aeham Alobidat] - Id: [040897798]

[Bradley Cliche] - Id: [041040049]

Language Name [Phoenix]

***This template is suggested (not mandatory) to answer A11 Specification.***

|  |  |
| --- | --- |
| **Part**  **1** | **Language User Reference** |

**EXPLANATION**

*The purpose of this assignment is to invent a new computer language.*

* *This language can have the syntax and structure of your choosing.*
* *Option 1: Adapt the ‘Julius’ language to be Julia compatible (see*[*https://julialang.org/*](https://julialang.org/)*).*
* *Option 2: Define a****DSL****– Proper to solve specific problems (ex: science, economy, music, etc.)..*
* *This is going to be a fairly basic language. There's a lot of functionality that we'll be skipping over, while we implement the basics. You will need to tell me those basics, of course. In this document, I'm going to explain the steps of what to do with a bit of detail.*
  1. **User Manual**

**Element 1: Name / Extension**

*[Name your language! We suggest you use one "word" related to your “Julia-like” language or DSL] 🡪 My new language name is Phoenix*

*[What is the filename extension of your language? For example, for C it is .c, and for Professor Paulo's “Julius”****language****it is ".****jul****".] My filename extension for this language going to be Nix*

*[What is your language patterned after, or what is it similar to? What languages are inspiring your choice? It's okay if you're following Julia closely.]*

*Following* Python

**Element 2 – Comments**

*[Comments: I want to do comments in your language. How do I write them?]*

*I will use this Sign for the Comments ‘#/’*

*The Multi-line Comment*

**Element 3 – Keywords**

*[Keywords: List the sequence of reserved / key words from your language]*

begin

break

const

continue

do

else

elseif

end

export

false

finally

for

function

global

if

import

return

true

type

using

where

while

**Element 4 – Variables and Datatypes**

*[Datatypes: Define integers, real numbers (float points) and strings]*

* *How many bytes are you needing for your variables? This determines their ranges. (Chambly, for instance, has a special 64-byte integer. This is ridiculously huge for most purposes.)*

Phoenix supports several data types, including integers (Int32 or Int64 bits), floating-point numbers and strings.

Integers use 4 bytes of memory and have a range of -2^31 to (2^31)-1.

Floating-point numbers use 8 bytes of memory and have a range of 1.8 x 10^-308 to 1.8 x 10^308 with a precision of around 15-17 decimal points.

strings variables generally use 1 byte per character and the size of the string can be of any length.

*[Remember to define the number of bytes – and, if possible, range]*

On most modern systems, int (integer) variables typically use 4 bytes of memory. The range of integers that can be represented by 4 bytes is typically -2^31 to (2^31)-1.

float (floating-point number) variables typically use 8 bytes of memory. The range of floating-point numbers that can be represented by 8 bytes is typically around 1.8 x 10^-308 to 1.8 x 10^308 with precision of around 15-17 decimal points.

str (string) variables generally use 1 byte per character in the string and the size of the string can be of any length.

**Element 5 – Variables and Datatypes**

*[Variables: How would a programmer define variables that can hold integer numbers (numbers with no decimal point), floating point numbers (numbers with a decimal point) or text (ie: strings in Java). This is element 1. Consider if you want to flag the variables in a special way, like SOFIA or BASIC, or not, like C or Java.]*

*To define a variable that can hold an integer number (no decimal point), you can simply assign an integer value to a variable name: EX : me\_int = 5*

*To define a variable that can hold an float number (with decimal point), you can simply assign an float value to a variable name: EX: me\_ponto = 3.14*

*To define a variable that can hold string you can simply assign an string value to a variable name:*

*EX: me\_text = "Olá mundo!"*

*If I want to flag variables in a special way, I can use a keyword or special symbol to indicate the data type of a variable.*

*# Defining an integer variable in Phoenix*

*int meu\_int = 5*

*# Defining a float variable in Phoenix*

*float meu\_ponto\_flutuante = 3.14*

*# Defining a string variable in Phoenix*

*string meu\_texto = "Olá mundo!"*

**Element 6 - Commands**

* ***Attribution****: How does your language let a programmer assign a value to a variable? (Will you allow casting? If so, how will it work?) How will your language handle math, and will it allow strings to be concatenated (merged)?*

In Phoenix, a variable is assigned a value using the assignment operator, which is typically the equals sign (=). For example

x = 5

Phoenix also allows casting, which refers to converting a value of one data type to another. This can be done by using casting functions or operators. For example

x = int(5.0) # x will be 5

y = float(5) # y will be 5.0

z = str(5) # z will be "5"

Strings in Phoenix can also be concatenated, which means merging two or more strings into a single string. This is typically done using the plus sign (+) or a concatenation operator. Here's an example:

name = "John"

greeting = "Hello, " + name + "!"

print(greeting)

# Output: "Hello, John!"

* ***Selection****: How does your language do if-style logic? (Optional: Do you want to do some kind of switch/case as well?). You will need to explain how "conditionals" work in your language. How do you write Boolean operations, such as "or", "and", "not", and other conditions, such as less than, greater than, etc?*

***Basically, like other language, I said that because most of the language are sharing the same strategy***

In Phoenix, like in many other programming languages, selection or conditional statements are used to perform different actions based on different conditions

if condition:

# code to be executed if the condition is true

For example, the following code checks if the variable x is greater than 5, and if it is, it prints "x is greater than 5"

x = 7

if x > 5:

print("x is greater than 5")

----------------------------------------------------------------------------------

else statement in combination

if condition:

# code to be executed if the condition is true

else:

# code to be executed if the condition is false

x = 3

if x > 5:

print("x is greater than 5")

else:

print("x is not greater than 5")

----------------------------------------------------------------------------------

if condition1:

# code to be executed if condition1 is true

elif condition2:

# code to be executed if condition1 is false and condition2 is true

else:

# code to be executed if condition1 and condition2 are false

You can also use these operators to chain multiple conditions together.

The "and"/"or"/not" operators

In Phoenix, there is no equivalent to a switch statement like in some other languages

x = 7

y = 4

z = 3

if (x > 5 and y < 10) or not z == 5:

print("x is greater than 5, y is less than 10 or z is not equal to 5")

* ***Interaction****: How will your code handle looping? (You can do one or more of a for-style loop, a while/do loop, etc.)*

Yes, I can do the loop styles

The for-style loop is used to iterate over a sequence of elements

for variable in sequence:

# code to be executed for each element in the sequence

*----------------------------------------------------------------------------------*

numbers = [1, 2, 3, 4, 5]

for number in numbers:

print(number)

*----------------------------------------------------------------------------------*

*The while-style loop is used to execute a block of code repeatedly as long as a certain condition is true.*

*while condition:*

*# code to be executed while the condition is true*

*----------------------------------------------------------------------------------*

*x = 0*

*do:*

*print(x)*

*x += 1*

*while x < 5*

* ***Input****: How does your program get input from the keyboard? (Strings are easiest.)*

In Phoenix, you can get input from the keyboard by using the built-in input() function, which reads a line of text from the user and returns it as a string. The general syntax for the input() function is as follows:

response = input("Prompt")

name = input("What is your name? ")

* ***Output****: What would a programmer type to put output on the screen? What sort of variables or data will your code take?*

In Phoenix, you can put output on the screen by using the built-in print() function. The print() function takes one or more arguments and displays them on the screen.

----------------------------------------------------------------------------------

print(expression1, expression2, ..., expressionN)

name = "John"

print("Hello, ", name)

----------------------------------------------------------------------------------

x = 5

y = 3.14

print("The value of x is:", x, "and the value of y is:", y)

----------------------------------------------------------------------------------

* *The general syntax for the print() function is as follows:*
* ***Functions****: [Function definition: parameters and returning types]*
  + *What will be the syntax for making a function or subroutine?*

In Phoenix, functions are defined using the def keyword, followed by the name of the function, a pair of parentheses that may contain any number of parameters, and a colon. The code that makes up the function is indented underneath the definition line How will it take parameters?

def function\_name(parameter1, parameter2, ..., parameterN):

# Function body

* + *How will it return results?*

*In Phoenix, a function returns a value using the return statement. The return statement is used to specify the value that the function should return when it is called. The value that is returned by a function can be assigned to a variable or passed as an argument to another function.*

def my\_function(x, y):

return x + y

----------------------------------------------------------------------------------

result = my\_function(5, 3)

print(result) # Output: 8

**Element 7 – Proper elements**

*[Include specific features / elements to be included in your language]*

* *What you could include / modify? Think about new datatypes / structures / commands, etc.*

**Boolean; Strings; Hashes; Arrays; Symbols; Numbers**

* *Note: Do not share this info (it is supposed to be your proper elements in the language.*

|  |  |
| --- | --- |
| **Part**  **2** | **Examples** |

**Option 1: Julia-like**

**Hello World**

|  |  |  |
| --- | --- | --- |
|  | [Your Code here]  # printing Hello World in PHX language  print("Hello World") |  |

**Sphere Volume Expression (or any other example)**

|  |  |  |
| --- | --- | --- |
|  | [Your Code here]  from math import pi  def sphere\_volume(radius:float)-> float:  """  This function calculates the volume of a sphere  given the sphere's radius  """  volume = (4/3) \* pi \* radius\*\*3  return volume  # The sphere radius  r = 3  # Calculate the volume of the sphere  vol = sphere\_volume(r)  # Print the volume  print("The volume of the sphere with radius", r, "is", vol)  # Get sphere radius from user input  r = float(input("Enter the sphere radius: "))  # Calculate the volume of the sphere  vol = sphere\_volume(r)  # Print the volume  print("The volume of the sphere with radius", r, "is", vol) |  |

*[TIP: See examples in the Lecture Notes –* ***Appendix 1****]*

**Option 2: DSL**

**[Your example here]**

|  |  |  |
| --- | --- | --- |
|  | [Your Code here] |  |

|  |  |
| --- | --- |
| **Part**  **3** | **Architectural Aspects** |

**Advantages**

*[What's the goal of your language? Are you trying to make something simple, fun, complicated? My personal language, Chambly, is based around being useful to scientists. (You can just make something up here, honestly. Think about it a little bit, have a little fun.)]*

*My goal of this language to pass the course, not more then that. Because whatever I am going to do in this course I will forget about in the next few months.*

**Strategy: C Implementation**

*[How your language can be implemented in C – ex: datatypes]*

* *In plain English, or maybe even some high-level pseudocodes, how are you going to parse your language? You will be writing a compiler for your language, so these are some things you need to think about.*

The process of converting code written in Phoenix, into machine code that can be executed by a computer (also called "compilation") can be divided into several stages, such as:

Lexical analysis: In this stage, the source code is read and broken into smaller pieces called tokens. Each token represents a single element of the language, such as keywords, operators, and identifiers. The lexer or tokenizer would work by reading the source code character by character and grouping them into tokens.

Syntax analysis: In this stage, the tokens generated by the lexer are used to build an abstract syntax tree (AST). The syntax analyzer would use the tokens to check that the source code adheres to the grammatical rules of the language and to build a tree structure representing the source code. This tree structure can later be used for code generation or interpretation.

Semantic analysis: In this stage, the AST is examined to ensure that the source code is semantically correct. This involves checking that the types of variables and expressions match the expected types, that variables are declared before they are used, and that the source code follows other rules of the language.

Code Generation: In this stage, the target code is generated based on the AST generated by the previous stages. This code is typically in a lower-level language such as C. The code generator would convert the tree structure of the AST into a sequence of instructions that the target computer architecture can understand.

Optimization: In this stage, the generated code is optimized in order to make it run faster, use less memory, or to follow best practices.

For the data types of Phoenix has some similarities with C, Phoenix has variables of types int, float, str, tuple, list, and etc. The data types in Phoenix can be mapped to the similar data types in C. For example, an int in Phoenix can be mapped to an int in C, a float in Phoenix can be mapped to a float in C, a string in Phoenix can be mapped to a char array in C.

Please keep in mind that the process of converting a programming language into machine code is a complex task and different languages have different requirements

*[Your ideas about how to identify elements from language]*

* *Consider your "write to the console" command as an example. How will your compiler detect it? How will it sort out what to write to the console? What if there's some literal text (ie: "this is going to get printed") instead of variables?*

*[Your ideas about how to identify scope (ex: blocks between conditionals or functions)]*

* *How do you mark a block of code? If I use your loop logic, how do I control what portion of code gets looped through? In C, you might use { and }. In Python, the indentation is what matters. How does it work in your language?*

**Basic ideas about C implementation**

*[Which structures or datatypes you imagine to use in your language implementation]*

* *What do you think is going to be really hard about this? What would be, in your opinion, the hardest part of parsing your own new language? You don't have to write an essay, a paragraph or two will be fine.*

***Note 1: C Datatypes***

*Remember that you are implementing your language in ANSI C. For this reason, you cannot create arbitrarily your language (from scratch). You need to use what is already provided by C Compiler. For this reason, think about using and defining the language obeying the datatypes.*

**Problems when using C implementation**

*[Your vision about main problems / difficulties when implementing a new language (ex: memory allocation, range of datatypes]*

**FINAL SUGGESTIONS**

*Here some ideas to think about your language....*

* *Don't make this assignment harder than it needs to be on yourself. Focus on making the syntax for your language that meets our requirements. Worry about extra features later.*
* *Don’t worry if your new language winds up having really difficult parts. You'll be allowed to change your language as you go along, as long as you make "patch notes" to explain those changes. We'll tell you about this later.*
* *There's a marking key at the end of****CST8152\_Compilers\_W23-A11-Specification****that should steer you along for grades. Focus your efforts on where you'll get the best results.*
* *Finally, think about creating an “master-piece”: until now, you have used several languages. And if you have conditions to define yours, how it could be?*

**References**

*[Include eventual references used here]*

*ChatGPT*

Algonquin College

Fall, 2022