Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A11

Language Specification(REVISED)

Team:

[Ahnaf Kamal] - Id: [041034479] / [Skylar Phanenhour] - Id: [041024402]

Language Name: Zeus

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

**EXPLANATION**

Zeus is a Julia-like GPL language built using ANSI C. It doesn’t have any cool features, other than some built in math formulas. The coolest part about the language is the name. This document covers the basics of the language. Keywords in yellow in the code snippets are reserved keywords.

* 1. **User Manual**

**Element 1: Name / Extension**

Name: Zeus

Extension: .zs

**Element 2 – Comments**

All comments will have a specific foreground/text color that will be easily differentiated from the other portions of code.

Single line comments will be written with a single # in the beginning.

# This is a single line comment

Multi line comments will be written with an opening #= and ending with a =#.

#=

In Zeus, a multi-line comment

Works like this.

=#

**Element 3 – Keywords**

Keywords: int, real, string, char, if, elseif, else, while, do, for, in, print, readline, function, end, return, for, while, void, const, true, false

**Element 4 – Datatypes**

***character***: 32 bit primitive type.

char grade = ‘A’

***integer***: int32, 32 bits.

int bird\_population = 3000

int bird\_area = 5

***real:*** float32, 32 bits.

real bird\_speed = 0.5423

real bird\_wing\_span = 5.6

***string:***

Finite sequence of one or more chars, enclosed within a pair of double quotes (“)

* Double quotes within double quotes must be protected using a backslash (\)

string dialogue = “And he said: \”cheeseburger\””

print(dialogue)

And he said: “cheeseburger”

* Strings are immutable. They cannot be changed when created. If a string is modified, a new string is made, and the reference is changed.
* ‘\n’ can be used within strings to go to the next line.

string intro = “ Hello?\n hi\n”

print(intro)

Hello?

hi

* Strings can be concatenated using \*

string one = “photos”

string two = “printed”

println(one \*\* “, “ \*\* two) # Concatenates

Photos printed

***Bool***: true = 1 (int), false = 0 (int)

if true

print(“true”)

end

true

**Element 5 – Variables**

The data type of the variable does not need to the specified when defining it. The datatype of the variable on the left is inferred from the item on the right side of the = sign (assignment operator).

string language\_name = “Zeus” # stored as string

int language\_birthday = 2023 # stored as integer

real something = 5.5 # stored as real

The maximum length of a variable name is 31 characters. Variable names must begin with [a-z] or [A-z] or underscore. No numbers are allowed in variable naming.

**Element 6 - Commands**

**Attribution/assignment:**

New variables are defined by the following syntax:

*variable\_name = value*

1. ***variable\_name***: name of the variable
2. ***=*** : the assignment operator
3. ***value***: the value being assigned. ***value*** can be any of the previously defined datatypes (integer, real, string, boolean).

The programmer will assign a value to a variable by using the = sign. The value on the left side (lvalue) will be equal to the value on the right side of the equal sign (rvalue)

int x = 5 # assigning integer 5 to x

string y = “hello world!” # assigning string to y

z = 1.1 # assigning real 5.5 to z

Our language will handle simple math, such as simple addition, subtraction and multiplication.

int x = 52 + 2 # addition

real y = 3 \* 9 # multiplication

real z = 14 – 5 # subtraction

real cool\_greek\_name = 25/5 # division

real taxes = 0.5 \* x # multiplication using variables

String concatenation is allowed, as explained in the strings section in Element 4: Datatypes.

**Selection:**

***Comparison***

> Left hand greater than Right hand.

< Left hand less than Right hand.

== Equal to

!= Not Equal to

>= Greater than or equal to

<= Less than or equal to

20 < 200

false

8 < 9

true

5 == 5

true

5 != 5

false

5 >= 2

true

3 <= 5

true

***Boolean Operations***

&& Logical AND

| | Logical OR

! Logical NOT

***Boolean operations examples:***

true && true # logical AND

\

true

(false && true)|| (true)

true

!true # logical NOT

false

**Flow construction**

**Note: The indentation has to be either 4 spaces or a tab delimiter.**

**If statement**

if condition

statement

end

**if, else statement**

if condition

statement

else condition

statement

end

**if, elseif, else statement**

if condition

statement

elseif condition

statement

else

statement

end

**Interaction:**

Our code will handle looping with for-loops and while loops.

**Note: The indentation has to be either 4 spaces or a tab delimiter. Conditions have to in parenthesis.**

for 1 in 1:5

statement

end

while condition

statement

end

**input:**

To read input from the keyboard, we are going to use readline. We are going to read the input and store the result in the matching datatype.

print(“Enter name:”)

string name = readline()

print(“Enter age:”)

int age = readline()

print(“Enter money:”)

real money = readline()

**Output:**

The user can type printf or print to output to the console/screen. The escape char \n is supported.

print(“Big frog”)

int kilometer = 50

print(kilometer)

**Functions:**

The syntax to make a function starts with the keyword function, followed by the datatype, the name of the function and a set of parentheses that holds the parameters of the functions, separated by commas.

function function\_name (param1, param2)

a = 2 + 2

return a

end

The syntax will be separated into multiple parts. First it will need to start with the keyword “function” to identify that a function is being created. Next will be the return type of the function, which is explained more below. Then will be the name of the function, which will not be surrounded by any parenthesis to avoid any confusion with reading parameters. Lastly will be the parameters, which will be explained more below.

It will take parameters within the parenthesis; the datatype and the variable name will have to be included. To accept multiple parameters, a comma will have to be included to separate each parameter.

(datatype param1name, datatype param2name)

We will have to make the return a keyword for our language to recognize that we are returning a variable. When creating the function, the return type does not need to be specified. When returning, the programmer needs to make sure that they include the return keyword followed by the correct datatype if they have a return type. The return type of the function will not be surrounded by any parenthesis to avoid any confusion with reading parameters. Only one value can be returned with the return keyword.

**Element 7 – Proper elements**

We would include separate math functions for squaring a number, cubing. We will also have some built-in constant variables such as PI.

Example: Squaring

x = square(5) # Calculates 5x5 and stores it in x

y = cube(5) # Calculates 5x5x5 and stores it in y

Example: Cubing

vol = 4/3\*PI\*cube(r) # Calculates the volume of a

# sphere, where r = radius

Example: Pi

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

**Option 1: Julia-like**

**Hello World**

|  |  |  |
| --- | --- | --- |
|  | print(“Hello World”) |  |

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

|  |  |  |
| --- | --- | --- |
|  | function real sphere\_volume(real r)  return 4/3\*PI\*r^3 # **^ operator not implemented**  end  volume = sphere\_vol(5)  printf(“volume = %0.4f\n”, volume) **# %0.4f unsupported now  NEW (ADDED A22)**  print("Input x: ")  x = readline()  print("Input mu: ")  mu = readline()  print("Input standard deviation: ")  std = readline()  zScore = (x-mu)/std  printf(zScore) |  |

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

**Advantages**

The goal of the language is to be as close as possible to Julia, with a little bit of our own stuff added, but keeping in line with Julius.

**Strategy: C Implementation**

**Data types and structures**

Since we are using ANSI C to implement the language, the data types that the language will be using are derived from it. So an int in Zeus is the same int in C. For the case of an integer, we will be implementing int32. When parsing, identifiers and reserved keywords will be recognized using regular expressions. The tokens extracted will be stored in a data structure (table/list/tree), which will be built using C. The most frequently used data type will most likely be pointers, since the allocation, re-allocation and deallocation of memory is guaranteed. The datatype string in Zeus will be supported using an array of chars in C.

String literals will be captured using regular expressions that detects “”. Other constants such as integers, will be caught using the same strategy.

**Output**

When printing variables, we can print the contents of the memory address of the variable.

We will have a function called “print”/”printf”/”println” which will be reserved keywords, so the compiler knows what to do when this is encountered. The captured special function will need to be matched with the list of reserved words.

**Scopes**

Our language will identify the scopes based on the indentations of the code. We will have the block start with the next line being indented by an extra tab delimiter. The end of the scope will be defined with a “end” keyword.

**Basic ideas about C implementation**

For the symbol table, a data structure will need to be used, possibly a tree. The pointers data type will be essential throughout the entire language implementation process.

The hardest part of the parsing process is making sure that the program is foolproof from an infinite number of combinations that can arise from a source program. It will need to detect errors and logical problems, so the intermediate code is identical to the source. We will need to make sure that we perform excellent memory management, failure to do so may make the program unusable. Another challenge will be detecting and navigating scopes, since this would affect the variables, symbol table and logic.

***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**

C doesn’t have string, but this language will have it, so the implementation of a string with C will be difficult. Strings will have to be built using an array of chars and would follow the string termination method of attaching the terminating character \0.

Making sure that we don’t cause any memory leaks to occur. Since pointers will be such an important part of the project, ensuring that memory is allocated and deallocated properly and on time. Otherwise, it wouldn’t be functional.

**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]*

* *NOTE: Even if you use any tool (ex: ChatGPT), report here.*

Algonquin College

Winter, 2023