Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A11

Language Specification

Lab Professor / Lab Session:

[Lab Prof : Paulo Sousa / Lab Session: 311]

Team:

[Student Name] - Id: [041121881] / [Student Name] –: [Huijun Bu]

Language Name [PowerR]

***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 ‘Sofia language to be R compatible (see* <https://www.r-project.org/>*).*
* *Option 2: Define a* ***DSL*** *– Proper to solve specific problems (ex: science, economy, music, etc.)..*

*This is going to be a 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 of my language: PowerR*

*The Extension Name : .pr*

*[What is your language patterned after, or what is it similar to?*

*This language is similar to R.*

***Element 2 – Comments***

*[Comments: A. Using # to make a single line comment.*

*B. Using ## to make multiple lines comments.*

***Element 3 – Keywords***

*[Keywords:]*

*A: Condition keywords:*

*if ------ used to perform a block of code if the condition is true.*

*else ---- used to perform a block of code when “if” statement is false.*

*else if ----- used to build a nested conditional structure to implement different code when some certain conditions are true.*

*B: Loop*

*for --- used to build a structure to perform a block of code constantly until the condition is true.*

*while --- used to build a structure to perform a block of code when the condition is true.*

*infiniteLoop --- used to build a structure to perform a block of code without any conditional statement, and it must be exited explicitly by using the key word “break”.*

*break --- used to explicitly break loop.*

*continue --- used to skip a certain step of iteration in a loop.*

*C: Function Keyword*

*function --- to define a block of code to perform a certain behavior, such as: receive the input from user and output some text word or result to the console.*

*D: Output keyword*

*print*

*E: Input Keyword*

*readLine*

***Element 4 – Datatypes***

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

*[Remember to define the number of bytes – and, if possible, range] Range should be as same as C*

1. *Integer Number*
2. *Keyword: int*

*Range: 4 bytes , from -2,147,483,648 to 2,147,483,647.*

1. *Keyword: short*

*Range: 2 bytes , from -32768 to 32767.*

1. *Keyword: long*

*Range: 8 bytes, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.*

1. *Floating points Number*
2. *Keyword: float*

*Range: 4 bytes, provide precision up to 7 decimal place, from ±3.4e±38*

1. *Keyword: double*

*Range: 8 bytes, provide precision up to 15-16 decimal place, from ±1.7e±308*

1. *Strings*
2. *Keyword: text*

*Description: The string data type stores sequences of characters, such as letters and numbers. ASCII-encoded strings are particularly used for handling English text and standard control characters.*

*Range: each character in string occupies 1byte.*

**Element 5 – Variables**

*[Variables:*

*Define 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.]*

*Integer numbers:*

*int --- example: x <- 20*

*short – example, y <- 60*

*long --- example, z <- 78749473L*

*Floating points numbers:*

*float ---- example: a <- 15.2823568.*

*double ---- example: b <- 17.895623148543566*

*String:*

*text ---- example: a <- “cc”*

**Element 6 – Methods / Functions**

*[Variables: How would a programmer define methods]*

*Keyword: function*

*Example: function adding (x, y) {*

*}*

*“adding” is the name of this method.*

*x, y are parameters.*

*( ) --- parameter must be placed in brackets.*

*{ } --- method body code must be enclosed in curly braces.*

**Element 7 - Commands**

* ***Attribution / assignment****:*

***Assignment: <- is used to be a symbol of assign value to a variable.***

***For example:***

***x <- 42***

***Casting: explicitly casting is allowed in this language.***

***For example:***

***Cast to float***

***x <- 42***

***z <- float(x)***

***Cast to string:***

*y <- 52.68*

*w <- text(y)*

*Concatenate:*

*“ + ” --- used to concatenate strings.*

*For example: name <- “ my name” + “ hello world”*

*The strings in name variable is my name hello world now*

* ***Selection****:*

***The logic of if statement is following****:*

*If(condition) {# when condition is true*

*#some code*

*}else{ #when condition is false*

*#some code*

*}*

***The Boolean operator is following****:*

*Logic “or” ---- using “ | ”*

*Logic “and” --- using “ & ”*

*Logic “not” --- using “ ! ”*

***Condition operator:***

*Equals --- using “ == ”. For example: if( x == y )*

*Not Equals --- using “ != ”. For example: if( x != y )*

*Greater and greater than --- using “ > ” and “ >= ”. For example: if(x > y) , if(x >= y)*

*Less and less than --- using “ < ” and “ <= ”.For example: if (x < y) or if (x <= y)*

***Arithmetic operator:***

*Multiply: “ \* ”. For example: x \* y*

*Division: “ / ”. For example: x / y*

*Addition: “ + ”. For example: x + y*

*Subtraction: “ - ”. For example: x – y*

*Remainder: “ % ”. For example: int x <-42, int y <-8, x % y. The result is 2;*

*Automatically increasing the value of a variable by 1: “ ++ ”.For example, int x <- 1, x++, the value of x is 2 now.*

*Automatically decreasing the value of a variable by 1: “ -- ”.For example, int x <- 4, x--, the value of x is 3 now.*

* ***Interaction****:*

***“for” loop:***

*for ( int i = 0; i < x; i++) { # execute the code in curly brace when condition is true. Stop execute the code # in curly brace when condition is false;*

*# some code*

*}*

***“while” loop:***

*while( i == 1 ){ # execute the code in curly brace when condition is true. Stop execute the code in curly # brace when condition is false;*

*#some code*

*}*

* ***Input****:*

***Keyword: readLine***

*For example:*

*text name <- readLine(“hello”)*

*print(“My name is : ” + name)*

* ***Output****:*

***Keyword: print***

*print(“Hello”)*

* ***Functions****:*

*Keyword:* ***void*** *stands for Return nothing*

*For Example:*

***#Syntax of defining a method and return nothing***

*function void adding (x, y){*

*#some code*

*}*

*“adding” is the name of this method.*

*x, y are parameters.*

*( ) --- parameter must be placed in brackets.*

*{ } --- method body code must be enclosed in curly braces.*

***#Syntax of defining a method and return a certain type of data.***

*function float adding (x, y){*

*#some code*

*float x <- 4.888*

*return x*

*}*

**Element 7 – Proper elements**

***Data Structure***

***Keyword: array***

***Index start from number 0.***

***For example:*** *xray<- array(1,3,4,5,6)*

*print(xray[ 0 ] )#output will be 1;*

***For String array example:***

*textArray <- array(“x”,”y”,”z”)*

*print(textArray[ 1 ])# output will be y*

*textArray[1] <- a. #the array is “x a z” now*

***Keyword: map***

*For example:*

*weekday <- map(key1=****”*** *Monday”,key2=”Tuesday”)*

*Print(weekday[“key1”])# output will be Monday*

*Weekday[“key1”] <- “Friday”# the key1 is Friday now*

***Error handling:***

***try – catch***

*Description:*

*Keyword: try – catch*

In PowerR language, the **try-catch** statement is used to catch and handle exceptions that may occur during code execution. This structure allows the program to continue running in the event of an error and provides an appropriate response to the error.

***Syntax:***

*try {*

*# some code that might cause error*

*}catch(# if there is a certain type of error ){*

*# prompt programmer to fix it.*

*}*

***For example:***

*try{*

*int x = 3*

*y <- x/0*

*}catch(ArithmeticException e){# ArithmeticException: type of error. e: type variable.*

*print(“cannot divide by 0”)*

*}*

|  |  |
| --- | --- |
| **Part**  **2** | **Language Comparison** |

**Comparing with C language**

**Differences**

|  |  |
| --- | --- |
|  | **Syntax and Usability:** PowerR is designed to be more user-friendly and accessible for data analysis and statistical computing, adopting a syntax closer to that of R, which is typically easier for statisticians and data analysts to grasp compared to C's more low-level, complex syntax.  **Memory Management:** C requires manual handling of memory through malloc and free, whereas PowerR handles memory management automatically, reducing the burden on the programmer and decreasing the likelihood of memory leaks and pointer errors. |

**Advantages / Disadvantages (in comparison with C)**

|  |  |  |
| --- | --- | --- |
|  | **Advantages:**  **Ease of Use**: PowerR offers higher-level abstractions and built-in functions specifically for statistical operations, making it easier to use for data analysis without the need for extensive programming background.  **Rapid Development**: Quick prototyping and interactive testing are possible in PowerR, which is advantageous for data science and statistics projects that require iterative exploration of data.  **Integrated Data Analysis Tools**: PowerR, unlike C, provides extensive built-in support for advanced data structures and operations used in data analytics.  **Disadvantages:**  Performance: While C is known for its high performance due to direct hardware manipulation capabilities, PowerR might not match C in performance due to its higher-level abstractions and automatic memory management. |  |

**Comparing with another language[[1]](#footnote-1)**

**Language Name:Java**

**Differences**

|  |  |  |
| --- | --- | --- |
|  | **Runtime Environment:** Java runs on the Java Virtual Machine (JVM), which provides platform independence. PowerR, being more directly modeled after R, does not use a virtual machine and is more directly compiled or interpreted, similar to standard R.  **Syntax and Paradigm:** Java is primarily object-oriented, encouraging encapsulation and complex class hierarchies. PowerR focuses more on functional programming aspects, which can be more intuitive for statistical analysis and matrix operations. |  |

**Advantages / Disadvantages (in comparison with this second language)**

|  |  |  |
| --- | --- | --- |
|  | Advantages:  **Specialized for Statistics:** PowerR is specifically designed for statistical analysis, offering functionalities that are either not available in Java or require external libraries.  **Simpler Syntax for Data Tasks**: PowerR's syntax is simpler and more direct for data manipulation and statistical analysis compared to Java, which often requires verbose coding to achieve similar tasks.  Disadvantages:  **Scalability:** While Java is known for robust performance in large-scale applications, PowerR might not be as scalable in handling very large datasets or highly complex computational tasks without optimization.  **Less Versatility:** Java can be used for a wide range of applications from web development to mobile apps, whereas PowerR is specialized and thus less versatile. |  |

|  |  |
| --- | --- |
| **Part**  **3** | **Architectural Questions** |

**Advantages**

***Goal:*** *PowerR aims to provide a user-friendly and powerful tool for data scientists and statisticians, prioritizing ease of use, data manipulation, and statistical analysis directly from the language's syntax. The design philosophy revolves around simplifying the coding process for data analysis, making it accessible even to those with minimal programming experience. The language seeks to combine the expressive power of R with improvements in readability and efficiency.*

*Ease of Use: PowerR is designed to be intuitive for users familiar with R, reducing learning time and making it straightforward to perform complex data manipulations and analyses.*

*Specialized Statistical Tools: Includes built-in functions and data structures specifically for statistical work, which are optimized for performance and ease of use in statistical computations.*

*Interactive Development: Supports an interactive environment that enables immediate feedback and iterative data exploration, which is crucial for statistical analysis and modeling.*

**Strategy: C Implementation**

***Parsing and Compiling:***

***Lexical Analysis:*** *Using tools like Flex to tokenize the input, recognizing keywords, identifiers, literals, and operators from the code written in PowerR.*

***Syntax Analysis:*** *Bison or a similar parser generator can be used to implement a parser that checks and builds a syntax tree based on the grammar of PowerR. This will involve defining grammatical rules that describe how tokens combine to form valid expressions and statements.*

***Semantic Analysis:*** *The compiler will check for semantic correctness (e.g., type checking, variable declarations before use) and translate the high-level code into an intermediate representation that closely aligns with C's capabilities.*

***Parsing:***

***Lexical Analysis***

*For example:*

*token\_list = lex(input\_code) // Lexical analysis*

*This line represents the process of lexical analysis. The lex function takes the raw input code (input\_code) as its parameter. The function's purpose is to decompose the input source code into a series of "tokens". Each token represents a fundamental element of the code, such as keywords (like print), identifiers, numeric values, symbols, etc. These tokens form the basis for further processing and will be used to construct the structure of the code, which is the next step: the syntax tree.*

***Syntax Analysis***

*For example: syntax\_tree = parse(token\_list) // Syntax analysis*

*After obtaining the list of tokens, the next step is syntax analysis. The parse function receives these tokens and organizes them into a syntax tree (syntax tree) according to the grammar rules of the language. The syntax tree is a structured representation that describes the syntactic structure of the code. Each node represents an element of the code, such as a statement or an expression.*

***Handling print Statements***

***Detection:*** *The parser recognizes print followed by an expression or a string literal.*

***Execution:*** *When the compiler encounters a print statement, it translates this into a C printf function, ensuring the arguments are correctly formatted according to their types.*

***Identifying Scope***

***Marking Blocks:*** *PowerR uses curly braces { } to denote blocks of code, similar to C. This ensures clarity in the beginning and end of control structures like loops and conditionals.*

***Scope Control:*** *Each block creates a new scope, and variables defined within a block are local to that block unless specified otherwise.*

***Write to Console:***

*Command Detection: Detect the print keyword and parse the subsequent expression.*

*Output Handling: If the expression is a literal string, pass it directly to printf. If it involves variables or computations, evaluate these first before passing the result to printf.*

***Identifying Elements from the Language:***

*Use a combination of regular expressions and token matching during lexical analysis to recognize language constructs and ensure they conform to the defined grammar.*

**References**

*[1] OpenAI, “ChatGPT session,” accessed May 14, 2024. [Online]. Available: https://www.openai.com/chatgpt*

|  |  |
| --- | --- |
|  | * ***NOTE****: Using chat GPT, google search;* |

Algonquin College

Summer, 2024

1. You can use any language (different from C). Ex: Java, Go, Python, etc. [↑](#footnote-ref-1)