A picture containing logo

Description automatically generated

**SCHOOL OF ADVANCED TECHNOLOGY**

**ICT - Applications & Programming**

**Computer Engineering Technology – Computing Science**

Graphical user interface, application

Description automatically generated

A11

Language Specification

**Team:**

**Aliab Eman**- Id: 041000420

**Language Name [ApouC]**

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

**EXPLANATION**

The purpose of this assignment is to invent our own computer language which is supposed to be a sub-version of Python through the C-implementation of the available 'BOA' language. In A11, we will introduce some aspects of our computer language that we will aim to implement this semester.

* **User Manual**

**Element 1: Name / Extension**

My language is named "ApouC", pronounced "App-Wash", which was the name of my father. The filename extension for my language is ".apc". My language will be patterned after Python to resemble the BOA language. The end goal of my language is to create a high-level language like Python.

**Element 2 – Comments**

Comments will be written as single line comments with "//" before the intended comment.

Multi-line comments will begin with "/\*" and end with "\*/".

Example:

Single line comment, once written, everything for the rest of that specific line becomes a comment until the new-line sequence (\n) is interpreted:

// This is a comment

Multi-line comment:

/\* This is a

multi-lined comment \*/

**Element 3 – Keywords**

defFunc – to define a function

gloVar – to declare a global variable

nonGlo – to declare a non-local variable

defData - defining type of data-input

defCode – statements of code

true – Boolean value, result of comparison operations

false – Boolean value, result of comparison operations

**Element 4 – Variables and Datatypes**

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

int – datatype for integers, 2 bytes, [0,255]

real – datatype for floating point numbers, 4 bytes, [0, 65535]

string – datatype for strings

char – datatype for characters, 1 byte

* *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.)*

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

**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.]*

Variables and constants will be defined outside of our functions either globally or locally (if they need to be destroyed after use).

Factors to consider:

* Numbers that are actually string values, as numbers that appear within the print stream function
* Converting non-string valued numbers into numbers within the print stream function.

**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)?*

I will allow for casting via function calls to the type of cast the user wants to return.

Types of casting available:

* + - String casting: str(casting\_val)
    - Integer casting: int(casting\_val)
    - Float casting: float(casting\_val)

Bitwise conversions will be necessary as well as warnings/ data retention methods to make sure memory loss is minimal. The classic example, casting a float as an integer for example, would need to be regarded accordingly, whether that means giving a warning upon execution, or giving an opportunity to exit the casting opportunity and keeping the variable as is (potential use of conditional blocks if I want to go through with down/up casting exceptions).

* Handling math:
  + Arithmetic Operators usable: “+”, “-“, “\*”, “/”, “%”, “\*\*”, “//”
  + Assignment Operators usable: “=”, “+=”, “-=”, “/=”, “\*=”
  + Comparison Operators usable: “==”, “!=”, “>”, “<”, “>=”, “<=”
  + Logical Operators usable: “and”, “or”, “not”
  + In a calculation we cannot have two consecutive integer/ floating point values that occupy different memory spaces (i.e., we cannot have “5 6 + 8”, the values need to be accounted by positions and operators in between then in specific orders)
  + We will need to be able to allow for concatenation of multiple strings together (ex. variableX = “string1” + “string2”)
* Variable initialization is the second step after variable type initialization.
  + When initializing the variable, we will have an assignment operator assigning the variable on the left of the operator to the value on the right of the operator.
  + Since we will deal with integers, floating-point and strings:
    - Integers: values that do not start/ end with quotation marks (or whatever expression is decided to represent quotation marks)
    - Floating point numbers will not start/ end with quotation marks, they can either be a set of integers or separated by only ONE period mark (XXX.xxx) so that it can accept decimal value numbers
    - Integers and floating-point numbers will need to be able to understand negative numbers as well as positive numbers
    - Strings will turn any inputs within the string quotation marks/ lexemes associated with creating strings into sequences of characters regardless of if the characters within the start/ stop lexemes integer are based, or character-based values.
* Negative numbers in integer or floating-point variables will need proper conversions when using them in calculations
  + We should use bitwise operations to calculate values in our compiler.
  + Recognition of the byte sizes for our number values will be important so our iterations/ calculations do not overflow.
* How will my language do if-style logic:

If – to make a conditional statement

Else – used in conditional statements

Elif – used in conditional statements, same as else if

* Potentially use the Python dictionary to implement a switch/case logic…

The use of keywords that acknowledge logical statements will be needed in order to control the sequential executions of code. I am considering making my boolean operations as strings moving forward to help with the high-level code aspect that I’m aiming for, else I’d keep them as generic symbols.

For/ in loops – iterations over a sequence within a list

* + Sequences can be a sequence of characters or a sequence of elements in a list…

The program will get input from the keyboard through the strings defined on the keyboard input stream.

There will be a scan function, known as input() in Python, available to take the input stream and store the keyboard input accordingly.

We will need to have our scan function recognize when to stop reading the keyboard input.

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

Print() – displays input on the screen

The print function should be able to display integer, floating point and string values.

The print function will need a way to know when to start/ stop displaying the keyboard input onto the screen.

**Syntax for Function Definitions**

1. Function Definition
   1. Parameterized functions
   2. Functions with return values or no return values
2. Block associated with the function’s subroutines
3. Need a way to recognize the ending of the function’s subroutines
4. We should be able to pass by value or by reference.
5. Functions with return values should have a return statement within the function subroutine.

**Element 7 – Proper elements**

1. I want to modify the way my Python in C compiler will read comments, making comments understandable as single line or multi-line using Java conventions with the “//” and “/\* … \*/” expressions.
2. I want to modify the structure of functions, using a character-compiler friendly form of block statement recognitions.
3. I want to include a way for the reader to bypass the tabulation necessary for Python.

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

**Option 1: Python-like**

**Hello World**

|  |  |  |
| --- | --- | --- |
|  | print(“Hello, world!”); |  |

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

|  |  |  |
| --- | --- | --- |
|  | pi = 3.14159265;  r= 6.0;  V= 4.0/3.0\*pi\*r\*\*3;  print('The volume of the sphere is: ',V); |  |

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

|  |  |
| --- | --- |
| **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.)]*

The goal of my language is to recreate elements of the Python language using the C-language as my front-end compiler. I don’t want to make it too simple but I want it to have a respectable range of elements available to make the compiler a good interpretation of Python using C.

**Strategy: C Implementation**

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

* *In plain English, or maybe even some high-level pseudocode, 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 ability to create an efficient lexical analyzer will allow for me to parse my language. I want to make sure that my compiler is able to take the input stream from the keyboard and create groups of identifiable strings/ elements to send to my syntax analyzer.

*[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?*

My write to the console command, input() function, will be recognized as a keyword/ function by my compiler so that it understands to take the user input upon command and end the user input via the end-line, \n, for example.

For the input() function it needs to both return void or return a value/ set of values to its destination. I’m still unsure how this will work.

*[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?*

In my language I want to use a sort of “braced” system to control the loop structure for my code. The reason why I want to consider this is for ease of understanding for the user, instead of enforcing them to follow the Python tabulation standard.

**Basic ideas about C implementation**

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

* *What do you think is going to be 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.*

I believe the hardest part of parsing my own new language will be understanding the complexities of error handling and memory management within my new language. Ideally, we all want to create an new version standard of the original code we aim to design our compiler from, but the possibility of that happening are slim. I do hope to create a respectable compiler that can handle basic errors and respond according to the exceptions it faces. Being able to not only compile and catch compilation errors, but also being able to define the types of errors are some of the things I hope to learn how to do with my Apouc language.

***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]*

The C compiler is structured very closely to assembly language in the sense that it is capable of optimally using heap and stack memory if the programmer is efficient. Problems I predict for myself include being able to manage blocks of memory through my compiler while simultaneously processing user inputs/ texts of information within the constraints of the language I’m creating. I consider myself a novice when it comes to the C language and the fact that I do not have the assistance of a Java compiler when it comes to things like garbage collection, etc., are aspects I will need to learn on the go as I develop my language.

**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\_F22-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**

1. W3Schools Python Tutorial. <https://www.w3schools.com/python/>
2. CST8152 F22 Lecture Notes for Compilers.
3. Variables and Keywords in C. [Variables and Keywords in C - GeeksforGeeks](https://www.geeksforgeeks.org/variables-and-keywords-in-c/#_=_)

Algonquin College

Fall, 2022