ALGONQUIN COLLEGE

SCHOOL OF ADVANCED TECHNOLOGY

ICT - Applications & Programming
Computer Engineering Technology – Computing Science

A11

Language Specification

Team:

[Cameron Ewing] - Id: **[041037946]** / **[Megan Clinch]** - Id: **[041043369]**

Language Name:

Velvet

The smoothest running program.

Part 1 Language User Reference

1.1. User Manual

Element 1: Name / Extension / Font

Velvet is a language that is rooted in C with inspiration from both Julia and Python. It is unique in that it has been designed with the intention of user simplicity and elegance. Velvet code will be structured using indentation.

- Name: Velvet
- Extension: .vlt
- Font: Our language will be written in `Courier New`

Element 2 – Comments

We have decided to write our comments using a '//' (double forward slash).

- i.e., // Single line comment
- i.e., //. Multi-line comment .//

Element 3 – Keywords

The list of keywords that we have defined include:

- method – this keyword will replace 'function' in Julia
- if / elseif / else
- for
- when – this keyword will replace 'while' in Julia
- struct
- close – this keyword will replace 'end' in Julia
- true / false
- send – this keyword will replace 'return' in Julia
- print
- cast – this keyword will replace 'convert' in Julia

- input – this keyword will replace ‘readline’ in Julia
- ent / ENT – this keyword will replace ‘int’ in Julia
- decimal / DECIMAL – this keyword will replace ‘float’ in Julia
- chain / CHAIN – this keyword will replace ‘string’ in Julia

Element 4 – Datatypes

The list of datatypes that we have defined include:

- Entero – ranges between -2^{63} and $2^{63} – 1$, 64 bits per number
- Decimal – 64 bits per number
- Chain – ranges between 1 and 256 characters, 32 bits per character

Element 5 – Variables

Our language will define variables as follows:

- Entero – ent variableName
  - Constant: ENT variableName
- Decimal – decimal variableName
  - Constant: DECIMAL variableName
- Chain – chain variableName
  - Constant: CHAIN variableName

Element 6 - Commands

- **Attribution / Assignment**
  - Entero – ent variableName
    - Initialization: i.e., ent number = 10
    - Assignment: i.e., number = 10
    - Constant: ENT variableName, i.e., ENT number = 10
  - Decimal – decimal variableName
    - Initialization: i.e., decimal number = 10.1
    - Assignment: i.e., number = 10.1
    - Constant: DECIMAL variableName, i.e., DECIMAL number = 10.1
  - Chain – chain variableName, i.e., chain name = “George”
    - Initialization: i.e., chain name = “George”

        - Assignment: i.e., name = “George”
        - Constant: CHAIN variableName, i.e., CHAIN name = “George”
    - Casting will automatically create a new variable under the casted type
        - The following pseudocode provides an example on how we will implement this logic
        - Note! Chain types can only be cast into entero or decimal if they are one character in length
            - Entero to decimal: cast(decimal, 4) will turn 4 into 4.0
            - Entero to chain: cast(chain, 4) will turn 4 into “4”
            - Chain to entero: cast(ent, “h”) will turn “h” into 104
            - Chain to decimal: cast(decimal, “h”) will turn “h” into 104.0
            - Decimal to entero: cast(ent, 2.6) will turn 2.6 into 3
            - Decimal to chain: cast(chain, 2.6) will turn 2.6 into “2.6”
    - In our language, math will be handled using arithmetic operators – see below
    - In our language, strings can be concatenated using the ‘+’ (plus sign)
        - Note! Concatenation can only occur with chain types
- **Selection**
    - Conditionals – if-style logic
        - Our language is implementing if/elseif/else conditional logic
        - The following pseudocode provides an example on how we will implement this logic

```
ent number = 10
if number > 10
        print(“Your number is greater than 10”)
elseif number < 10
        print(“Your number is less than 10”)
else
        print(“Your number is 10”)
close
```

    - Arithmetic operators
        - Add: +
        - Subtract: –
        - Multiply: *
        - Divide: /
        - Modulus: %

- Relational operators
  - Equal: ==
  - Not equal to: !=
  - Less than: <
  - Less than or equal to: <=
  - Greater than: >
  - Greater than or equal to: >=

- **Interaction**
  - For loop: The following pseudocode provides an example on how we will implement this logic

    for i in 1:4

        print(i + " ")

    close

    Output: 1 2 3 4
  - When loop: The following pseudocode provides an example on how we will implement this logic

    ent number = 0

    when number < 5

        print(number + " ")

        number = number + 1

    close

    Output: 0 1 2 3 4

- **Input**
  - In order to get input from the keyboard, we will create, and implement an input() method
    - i.e., chain name = input()

- **Output**
  - In order to print output to the screen, we have decided to keep the print() method
    - i.e., print("We love compilers!")

- **Functions**
  - What will be the syntax for making a function or subroutine?
    - The syntax for making a function will be: method **methodType* methodName(parameters) and will end using the 'close' keyword
      - *Our functions will only require a methodType when returning a value
      - *Our functions can only take predefined datatypes as their methodType

- How will it take parameters?
  - Our functions will take parameters in brackets following the method initialization
  - Our function parameters will require to have their data types specified
- How will it return results?
  - Our functions will return results using the 'send' keyword
- The following pseudocode provides an example on how we will implement this logic

```
method name()
        chain foo = "bar"
        send(foo)
close
```

Element 7 – Proper elements

Specific Features

- Our language will be set apart from other languages due to its simplicity in formatting and in syntax
  - i.e., not requiring the use of {} (or parentheses) to signify a function/class
  - i.e., not requiring the use of the ; (or semi-colon) to end a statement
  - i.e., emphasizing indentation for ease of reading
    - Indentation will be done using a tab (by hitting the 'tab' key)
    - When there are multiple blocks of code, nested blocks will have their own indentation
    - The following pseudocode provides an example on how we will implement this logic

```
method numbers()
        ent number = 10
        when number < 5
                print(number + " ")
                number = number + 1
        close
close
```

Part 2 Examples

Option 1: Julia-Like

Hello World

```
method main()

     print("Hello World!")

close
```

Sphere Volume Expression

```
method main()

     print("Please enter the radius of your sphere: ")
     decimal radius = input()
     print("The volume of your sphere is: " +
sphereVolume(radius))

close


method sphereVolume(decimal radius)

     decimal pi = 3.14159265
     decimal volume = 4/3*pi*(radius*radius*radius)
     send volume

close
```

Part 3 Architectural Aspects

Advantages

Velvet is an all-purpose language with an emphasis on 'Keeping It Simple Student'. We have created this language to be easy for beginner programmers to learn, using simple and straightforward syntax while maintaining the basic principles of programming.

Strategy: C Implementation

Implementing C Language

- Our language will implement ANSI C via its data types (i.e., integer, character array (string), float)
- We have decided to parse our language in a top-down format

Feature Integration		
Julia	Velvet	C
int	ent	int
float	decimal	float
string	chain	char[]
function	method	function
if / elseif / else	if / elseif / else	if / else if / else
for	for	for
while	when	while
struct	struct	struct
end	close	} (closing parentheses)
true	true	1 (or any real number)
false	false	0
return	send	return
print	print	printf
convert	cast	(typeName) variableName
readline	input	scanf

Identifying Elements

- Our compiler will detect reserved functions or keywords (i.e., print(), send, etc.) and read the subsequent data until the end of the statement
- Our language will not allow literal text – it will be flagged as an error

Identifying Scope

- In order to identify blocks of code, our language will implement indentation as a convention, as well as the 'close' keyword to signify the end of the block

Basic Ideas About C Implementation

Incorporating Datatypes

- Our language will use the framework already laid out by ANSI C. In other words, there is no reason to reinvent the datatypes when the types that we plan to use have already been made (i.e., integer, character array (string), float).

Problems When Using C Implementation

Difficulties of Parsing

- Considering we have not yet learned about the innerworkings of a parser, we are unsure as to what the hardest parts of parsing our own language will be. However, we believe that identifying keywords and reading blocks of code may be challenging, especially in a language based on the simplicity of code.

References

- CST8152_Compilers_W22-LectureNotes-v1
- The Julia Language by the Julia Project

Algonquin College
Winter, 2023