ALGONQUIN COLLEGE

SCHOOL OF ADVANCED TECHNOLOGY

ICT - Applications & Programming
Computer Engineering Technology – Computing Science

A11

Carthage Language Specification

Name:
Megan Machkouri- Id: **41003409** **Section: 013 Due: January 22 Date: January 22 Professor : Paulo Sousa**

Language Name Carthage

PART 0: Important Patchnotes

1. **The language will no longer support long datatypes and constants! (Removed Big)**
2. **EVERY Keyword must be earmarked by "#"**
   Ex. #prime#, #IF#, #VAR#, #WHILE#, #output#, #char#, #text# …
3. **The language will no longer support function/method declarations (prime will be only method).**
4. **The language will no longer require variable identifier keywords.**
   Ex. Char, Int, Float String NOT REQUIRED/SUPPORTED
5. **Comment Identifier has been modified to "?" instead of "\\" because the '?' is a better representation of comments in my opinion.**
6. **Negation Operator has been switched to 'N'. For clarity and is more descriptive since '!' is used frequently for variable identification.**
7. **Not Equal Operator has been switched to '~' since it is more descriptive .**
8. **All logical operators only require single char**
   Ex. & |

Part 1 Language User Reference

1.1. General Language Overview

Name

Carthage

Extension

*Extension = **.car***

*MyCarthageProgram**.car***

Genealogy

Subset of C – Abides to C language specification (especially regarding storage and memory of Datatypes) however it closely follows SOFIA in terms of language implementation.

Inspired by the ancient civilization of Carthage a renowned trading hub in its glory days.

Carthage will be defined as a DSL that will be built specifically for young children learning to program. This

language will have no use case outside of this sphere and will have limited functionality.

Brief Overview

The goal of Carthage is to design a simplified language that is ideal for teaching young children how to program. In other words, to provide a smaller, simpler and less complex subset of C ideal for basic operations.

A Carthage program consists of a single file that follows a specific strict format:

```
 ? Main method ?
#prime#{
        ? Variable Declarations ?
        #VAR#
            ?
             int !price# = 10;
             dec !total$;
             dec !tax$ = 1.13;
             string !name@;
            ?
        ? Code execution ?
        #RUN#
            ?
            !total$ = !price# * !tax$;
            ?
        ? Return Statement ?
        #YIELD#
            ? !total$! ?
}
```

KEY RULES

- ALL FUNCTIONS (PRIME INCLUDED) MUST HAVE 3 SECTIONS: VAR, RUN & YIELD ENCLOSED By #
    - VAR : Section where variables are declared and/or initialized
    - RUN : Section where code is executed and manipulation occurs (May be empty)
    - YIELD: Section where you specify what data is to be outputted/returned
- ALL FUNCTIONS ARE ENCLOSED BY **$** Ex. $prime$
- ALL VARIABLES ARE PRECEDED BY **!** and must END with either: **# $ @ ? !**
- ALL Code Statements must be delimited by semicolon **;**
- Main(prime) method does not allow variables to be passed into it

1.2. User Manual

Element 0 – Comments / Keywords

Comments must be enclosed by forward slashes ? comment ? (can be single or multiple line)

Reserved Keywords

WHILE	***STOP***	~~***dec***~~	***RUN***
DO	***LOOP***	~~***big***~~	***YIELD***
IF	~~***char***~~	~~***text***~~	***output***
EL	~~***int***~~	***VAR***	***Input***

KEY RULE

- None of the words listed in the above table may be used as variable or function identifiers

Element 1 – Variables and Datatypes

Carthage abides to C language specification in terms of datatype memory and range

Datatype	*Memory (bytes)*	*Range*	*Type*
char	*1*	***-128 to 127 or 0 to 255***	***Single ASCII value***
int	*4*	***-2,147,483,648 to 2,147,483,648***	***Integer number***
dec	*8*	***2.3E-308 to 1.7E+308***	***Fractional number***
big	*8*	***-9223372036854775808 to 9223372036854775807***	***Integer number***
text	*Number of Chars * Size of Char (1)*	***128 to 127 or 0 to 255***	***Char array delimited by null***

KEY RULES

- Char must be single ASCII value enclosed by single quotes (‘’)
- Text is a char array enclosed by double quotes (“”) and null terminated
- Dec can only be assigned to fractional numbers (Ex 1.3)
- Int can only be assigned to non-fractional numbers (Ex 1)
- Big corresponds to long in C language and inherits its range and memory attributes

Element 2 – Assignment

KEY POINTS

- Datatype must be declared **explicitly**
- All variables must be preceded by **!** and ended by either ; **# , $, @, ? or !**
- If preceded by **~!** variable is constant and is immutable
    - **char variables must end with ?**
    - **int variables must end with #**
    - **dec variables must end with $**
    - **big variables must end with !**
    - **text variables must end with @**
    - ~~**constants must be preceded by ~!**~~
- Semicolon delimited statements
- Casting will not be supported since this DSL is designed to educate children making it important to correctly classify datatypes. No use case for this added complexity.
- Only assignment operator supported is **"="** no use case for others implemented in C
    - Assigns value from right side operand to left side operand value
    - Used to initialize variables

EXAMPLES

FORMAT

***\\ datatype $name$/#/@/! = value; ***

Examples

char !a? = 'x' \\ character value

text !b@ = "Hello World"; \\ string literal

int !c# = 23; \\integer value

dec !d$ = 3.3; \\floating point value

~~***big !e! = 999999999999; \\ long value***~~

dec ~!tax$ = 1.13; \\ constant

int ~!price# = 3; \\ constant

Element 3 – Selection Structure

```
#IF#(Conditional Statement) {
#TRUE#
        Execute if true
#STOP#
}
#IF#(Conditional Statement ){
#TRUE#
        Execute if true
#STOP#
}
#EL# {
#TRUE#
        Execute if true
#STOP#
}
```

KEY POINTS

- TRUE must be present and **may be empty** for both IF and EL statement
- Must be terminated by STOP command
- There can be **any number** of IF statements and EL statements **are optional** not required
- **Only one** EL statement that cannot be empty and **must be preceded** by IF statement
- TRUE and STOP must be enclosed by #

Element 4 – Loops

```
#WHILE#(Conditional Statement) {
#LOOP#
                    \\ Execute Code \\
#STOP#
}
```

```
#DO#{
    #LOOP#
                    \\Execute Code \\
    #STOP#
} #WHILE#( Conditional Statement)
```

KEY RULES

- Does not support FOR loops
- Must contain LOOP and STOP section both must be enclosed by #
- LOOP section may be empty
- Conditional Statement may not be empty and must include at least one conditional expression
- LOOP section will only execute if conditional expression returns TRUE
- LOOP section will continue executing until conditional statement returns FALSE
- WHILE LOOP section only executes if pre conditional statement is TRUE
- DO-WHILE Statement will execute at least once up until post conditional is FALSE
- SECTIONS MUST BE ENCLOSED BY BRACKETS { }

Element 5 – Arithmetic Operators

Operator Symbol	Name	Example
+	*Addition*	*1 + 2 = 3*
-	*Subtraction*	*2 - 1 = 1*
*	*Multiplication*	*2 * 2 = 4*
/	*Division*	*9 / 3 = 3*
%	*Modulus*	*8 % 3 = 2*
**	*Exponentiation*	*2 ** 2 = 4*
++	*Increment*	*++!A#; Add 1*
--	*Decrement*	*--!A#; Sub 1*

KEY POINTS

- Arithmetic Operators follow same standard associativity and order of precedence as C
- Arithmetic Operators may only be applied to numeric values
- Only pre increment/decrement are supported, post increment/decrement is illegal
- Unary sign and SAME SIGN binary sign operators of type '+' and '-' are legal
    - **++x#, --x#, +x#, -x# are all legal**
    - **+-x#, -+x#, %%x are all illegal**

Element 6 – Relational Operators

Operator	Name	Example
==	*Equality*	*X == Y*
!!	*Not Equal*	*X !! Y*
>	*Greater*	*X >Y*
<	*Lesser*	*X < Y*
>=	*Greater or Equal*	*X >= Y*
<=	*Lesser or Equal*	*X <= Y*

Element 7 – Logical Operators

Operator	Description	Example
&&	True if both statements true	2 < 3 && 4 > 5 = FALSE
||	True if one statement true	2 < 3 || 4 > 5 = TRUE
!!	Reverses result	!!TRUE = FALSE

KEY POINTS

- Implemented in conditional statements
- Parentheses may be used to enclose a relational expression only when followed by logical operator
  - Ex. (2 < 3) && (4 > 5)
- Logical Operators must be used in combination with at least two relational expressions
- Relational Operators take precedence over logical operators
- Relational expressions evaluate to 0 or 1
- Relational Operators may only be used to compare same datatypes
- Relational Operators may only be used to compare numeric values
- STRING COMPARISION IS NOT SUPPORTED
- Conditional Statements are left associative
- Conditional expressions only evaluate to 1 (TRUE) or 0 (FALSE)
- Conditional expression must include at least one Relational Operator
- All Conditional expressions must be enclosed within parentheses ()

EXAMPLES

IF(!x# >> !y#) \\ legal
IF(!x# >> !y$) \\ illegal to compare float and int
IF(!x# >> !y# && !x# !! 0) \\ legal

Element 8 – Input and Output

Input Function

- ***text !x@ = $input$(); \\read until next white***
- ***$input$(!x#);***

Output Function

Text !string@ = "GoodBye";

- ***$output$(); \\ illegal***
- ***$output$@(!string@); \\ output var value***
- ***$output$(!string@ << "Hello"); \\ output with space in between***
- ***$output$(!string@ < "Hello); \\ output with no space in between***
- ***$output$(!string@ <n< "Hello"); \\ output second arg on newline***
- ***$output$(!string@ <t< "Hello"); output with tab in between***

NOTE BOTH FUNCTIONS ARE ENCLOSED By '$'

KEY RULES INPUT

- If parentheses are empty will read standard input until next new line (\n) and will assign to text variable.
- If parentheses are empty function may only be used if preceded by assignment operator and may only be assigned to datatype text.
- Otherwise, will assign values from standard input in corresponding variable list order

KEY RULES OUPUT

- Cannot be empty
- Multiple arguments must be separated by either **<, <<, <n< or <t<** and enclosed within parentheses
- Will accept and output **string literals** surrounded by **double quotes**, any **datatype** or **variable as arguments**.
- If outputting multiple variables use < or << in between variables or datatypes
  - If "<" is used there will be no space between arguments
  - If "<<" is used there will be single whitespace in between arguments
  - If "<n<" is used arguments will be outputted on newline
  - If "<t<" is used there will be tab in between arguments

Element 9 - Functions

```
Function Format
$name$ {
    #VAR#
        \\variable declarations\\
    #RUN#
        \\code statements\\
    #YIELD#
        \\return  statements\\
}
$example$(int !y#){
    #VAR#
        int  !x#;
    #RUN#
        !x# = !y# + 5;
    #YIELD#
        !x#;
}
```

KEY RULES

- Function name must be enclosed by **$**
- Must include 3 sections; **VAR, RUN, YIELD** enclosed by #
- **All sections must be enclosed within brackets {}**
- VAR and RUN cannot be empty but YIELD can be empty
- VAR Section is strictly for variable declarations/initializations
    - If variable is passed into function within parentheses it DOES NOT need to be declared
    - When passing variable to function datatype and name must be specified
- RUN Sections is for code statements and data manipulation
- YIELD Section can only return single variable and **will implicitly call output function**
- If YIELD is empty function will **implicitly return 0**

Part Examples

2

Hello World

```
#prime#{

   #VAR#

    #RUN#
          #output#(“Hello World!”);
    #YIELD#


}
```

Sphere Volume Expression

```
#calculateVolume# {

    #VAR#
          dec !volume$;

    #RUN#

           !volume$ = (4/3) * 3.14 * r**3;

   #YIELD#
           !volume$; \\ return\output volume
}
```

Part 3 Architectural Aspects

Strategy: C Implementation

This language will inherit certain aspects of C language. Datatypes will remain unaltered and will respect range and memory rules from C language. This will allow for easier implementation.

Datatype float will no longer be supported all decimal numbers will fall under the dec datatype which corresponds to double in C language.

Variable's scope will be the same as C, meaning brackets will delimit scope.

Variables defined within a program/function/loop/method won't be available outside of that scope.

A variable defined in a program will be available to functions declared within that program.

A variable defined in a function within a program won't be available outside of the function.

Syntax will be modified, meaning reserved keywords will differ in name however should be able to implement syntax rules using same logic.

Strategy to identify tokens will be using earmarks including; **!, $, @, ? and #.** This will make it much simpler to classify tokens to their correct type since all variables are preceded by **!** and all functions are enclosed by **$**. This will allow for easier identification and less complex algorithms.

Strategy: DSL BONUS

This language will be simplified and follow strict structure rules since it is a DSL that is designed specifically for teaching young children. Therefore, it is not designed with complexity in mind, rather simplicity since it is not meant to be used for complex operations. It is a language that is designed to educate children on the basic principles of programming and how to properly structure programs. It is not meant to be used for complex programs.

DEVELOP ALGORITHMS TO:

- Omit Comments
- Omit Whitespace
- Identify Reserved Keywords
- Identify Delimiters
- Identify Operators
- Identify and differentiate uppercase and lowercase digits
- Define numerical ranges

- Implement Syntax Rules
- Differentiate between keywords and other string input

Sample Pseudocode Logic for Identifying Keywords

```
char reservedKeywords[16][10] = {
                \\Replace with Carthage Keywords\\
}

for( i = 0; i < 16; i++) {

     if(Conditional statement that compares input to array
elements) {
      return 1; \\TRUE
      }
}

return 0; \\FALSE
```

Challenges when using C implementation

I envision many potential problems occurring including but not limited to;

- Identifying valid token patterns
- Imposing Syntax Rules
- Imposing Structure Rules
- Allocating memory correctly (stack/heap)
- Reading standard input and assigning them to the correct variable
- Reading multiple variables at once from standard input

Solution Strategies

I believe that the key to success with this project is to develop a well-organized and calculated implementation plan. The first step would be to develop a list of all valid identifiers and token patterns for your programming language since there are a multitude of possibilities. The next step would be to implement simple and efficient algorithms to detect valid patterns. In summary, only implement code after you've developed a plan since it might be necessary to implement certain functionalities before others.

References

https://preview.redd.it/r127wnxvxa141.jpg?auto=webp&s=2dbe6b11815472bfe827589eeb94d0ba3f9b57c7

Winter, 2022