

**TC3048 Diseño de Compiladores**

**Nansus Programming Language**

***Project Technical Description and Documentation***

**Authors:**

Juan Fernando Ulloa Morán A00817807

Jesús Idilio Guadiana Cantú A00814770

***21/11/2018***

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

Juan Fernando Ulloa Morán Jesús Idilio Guadiana Cantú

Table of Contents

[I. Project Description 4](#_Toc530563150)

[a. Purpose 4](#_Toc530563151)

[b. Objectives 4](#_Toc530563152)

[c. Scope 5](#_Toc530563153)

[d. Requirement Analysis 5](#_Toc530563154)

[e. Use Cases 6](#_Toc530563155)

[f. Development Process 13](#_Toc530563156)

[1. Description 13](#_Toc530563157)

[2. Logbook 13](#_Toc530563158)

[3. Individual Reflections 15](#_Toc530563159)

[II. Language Description 16](#_Toc530563160)

[a. Name 16](#_Toc530563161)

[b. Main Characteristics 16](#_Toc530563162)

[c. Error Type List 17](#_Toc530563163)

[1. Compilation 17](#_Toc530563164)

[2. Execution 17](#_Toc530563165)

[III. Compiler Description 18](#_Toc530563166)

[a. Computing Equipment, Language and Additional Props 18](#_Toc530563167)

[b. Lexical Analysis Description 18](#_Toc530563168)

[c. Syntax Analysis Description 21](#_Toc530563169)

[d. Intermediate Code Generation and Semantic Analysis Description 23](#_Toc530563170)

[1. Operation Codes and Virtual Address Association 23](#_Toc530563171)

[2. Syntax Diagrams 25](#_Toc530563172)

[26](#_Toc530563173)

[26](#_Toc530563174)

[26](#_Toc530563175)

[3. Semantic Action Descriptions 27](#_Toc530563176)

[4. Semantic Consideration Table (Semantic Cube) 29](#_Toc530563177)

[e. Detailed Description of Memory Management in Compilation 33](#_Toc530563178)

[IV. Virtual Machine Description 36](#_Toc530563179)

[a. Computing Equipment, Language and Additional Props 36](#_Toc530563180)

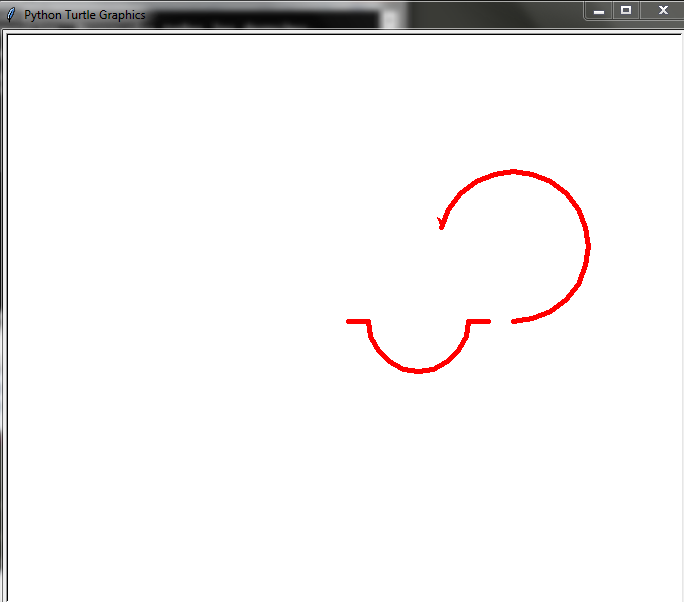
[b. Detailed Description of Execution Memory Management Process 36](#_Toc530563181)

[Virtual to Real Memory Association 37](#_Toc530563182)

[V. Functionality Tests 38](#_Toc530563183)

[1. Test Code 38](#_Toc530563184)

[2. Compiler and Virtual Machine Results 40](#_Toc530563185)

[ 41](#_Toc530563186)

[VI. Listings 44](#_Toc530563187)

[VII. Quick Reference Manual (Also in the environment) 68](#_Toc530563188)

# Project Description

## Purpose

The purpose of the “Compiler Design Final Project” is to help us, as developers, to understand the underlying function of a compiler whenever we use it. Understanding the way compilers work will result in the following: better understanding of the internal function of a programming language, quicker understanding of new languages that we wish to learn based on our newfound detailed understanding of the underlying structure of the compiler, as well as better use of language standards for consistent program development. During the development of this project we will be putting into practice all the theoretical topics seen in class, and we will be taking a closer look into the compilation stages:

* 1. Lexical Analysis
  2. Syntax Analysis
  3. Semantic Analysis
  4. Intermediate Code Generation
  5. Code Optimization
  6. Virtual Machine

## Objectives

The project’s main objective is to generate a general-purpose language in terms of its basic functions, where you can program any basic programs that support components such as:

1. Variables
2. Types
3. Arrays/Vectors and Matrices
4. Expressions
5. Binary Operators
6. Statements such as: Assignments, Conditions, Cycles, Read and Write.
7. Basic Syntax
8. Basic Semantics

Our program is strongly based, syntax-wise, to those that we would usually use most throughout our chosen career. We chose to design a language which receives a code input and has a graphical output that demonstrates the function of the program based on the given code.

## Scope

Our project contemplates within its scope the following standard capabilities of programming language:

* 1. Basic Operations (Arithmetic, Logic, Assignment, Print)
  2. Function Activation Register (Function Calls and Context Management)
  3. Recursive Behavior (Nested Function Calls)
  4. Multi-dimensional Variable Management (Arrays and Matrices)

We do not manage within our project things such as:

1. Classes/Objects
2. Polymorphism

## Requirement Analysis

* Functional Requirements:
  + Implementation of arithmetical operations that include: addition, subtraction, multiplication, division.
  + Implementation of relational operations that include: greater than, equal, less than, different, less than or equal to, greater than or equal to.
  + Implementation of logical operations that include: AND and OR.
  + Implementation of function definitions and function calls (context management)
  + Implementation of arrays (multidimensional variables)
  + Implementation of some form of input/output function
  + Implementation of graphical output functions.
* Non-Functional Requirements:
  + Allocation of virtual memory for operations.
  + Optimization of memory resources on execution.
  + Error display on execution.
  + Display of the graphical output.

## Use Cases

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 1 | | |
| Use Case Name: | Arithmetic Operation | | |
| Created By: | Juan Fernando Ulloa | Last Updated By: | Juan Fernando Ulloa |
| Date Created: | 21/11/2018 | Date Last Updated: | 21/11/2018 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | The user provides code for the addition/subtraction binary operations. |
| Preconditions: | The input code must have the operations written and the operands for the operation be the appropriate types for the operation. |
| Postconditions: | The result should show the result of the operation, and it should be a numeric value. |
| Priority: | High |
| Frequency of Use: | High |
| Normal Course of Events: | 1. The numbers/identifiers are given for the operation in the code. 2. The information is processed and solved. 3. The information is displayed. |
| Exceptions: | Division by zero will result in an error, Type Mismatch of operands will result in an error |
| Special Requirements: | None |
| Assumptions: | The user will provide numerical values for the operations, since they are arithmetical (variables in the form of int and float variables) |
| Test Case Description: | For the main test case, you will want to verify results of the known arithmetic operations, such as 4+3 should be 7 (you can use print in order to display the result) |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 2 | | |
| Use Case Name: | Relational Operation | | |
| Created By: | Juan Fernando Ulloa | Last Updated By: | Juan Fernando Ulloa |
| Date Created: | 21/11/2018 | Date Last Updated: | 21/11/2018 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | The user provides code for the relational binary operations. |
| Preconditions: | The input code must have the operations written and the operands for the operation be the appropriate types for the operation. |
| Postconditions: | The result should provide the result of the operation as a Boolean operation. |
| Priority: | High |
| Frequency of Use: | High |
| Normal Course of Events: | 1. The identifiers are given for the operation in the code. 2. The information is processed and solved. 3. The information is displayed. |
| Exceptions: | Type mismatch will result in an error |
| Special Requirements: | None |
| Assumptions: | The user will compare only elements of the same type in an effort to avoid type errors. |
| Test Case Description: | For test cases in relational operations you want to try out interactions with the relational operators, and you will only be able to make these via while, do while and if, since you need to have the Boolean produce a result and these are not displayable in Nansus. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 3 | | |
| Use Case Name: | Logic Operations | | |
| Created By: | Juan Fernando Ulloa | Last Updated By: | Juan Fernando Ulloa |
| Date Created: | 21/11/2018 | Date Last Updated: | 21/11/2018 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | The user provides code for the logical binary operations. |
| Preconditions: | The input code must have the operations written and the operands for the operation be the appropriate types for the operation. |
| Postconditions: | The result should produce the result of the operation, and it should be a Boolean value. |
| Priority: | High |
| Frequency of Use: | High |
| Normal Course of Events: | 1. The identifiers are given for the operation in the code. 2. The information is processed and solved. 3. The information is displayed. |
| Exceptions: | Type mismatch will result of any operation but Boolean with Boolean |
| Special Requirements: | Both operands must be Booleans. |
| Assumptions: | The user will provide Boolean values for the comparison and will receive a Boolean as a product. |
| Test Case Description: | For logic operations you want to use a follow up of the test used in relational, again since you can’t declare Boolean values within Nansus, you have to test this case by stating a two relational and then logical solution (a < b && b> a). |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 4 | | |
| Use Case Name: | Function Definition / Procedure Calls | | |
| Created By: | Juan Fernando Ulloa | Last Updated By: | Juan Fernando Ulloa |
| Date Created: | 21/11/2018 | Date Last Updated: | 21/11/2018 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | The user provides code for function definition. |
| Preconditions: | The input code must have a proper syntactical format for the declaration of the function in Nansus. |
| Postconditions: | The function will be stored and ready for use. |
| Priority: | High |
| Frequency of Use: | High |
| Normal Course of Events: | 1. The function is properly written in the code. 2. The function is properly stored in the function Directory. 3. The function is later accessed by the code and used appropriately. |
| Exceptions: | Functions are not defined with proper syntax; function calls are not with the correct number of parameters/or not in proper syntax |
| Special Requirements: | None |
| Assumptions: | The user knows how to declare and call functions in Nansus language. |
| Test Case Description: | For function calls you want to test the function of them by making sure to try both void function calls and return function calls (note to reader, as of the time of this document being written, return functions do not actually work, however, they can be called in the same manner as a function with return.  Functionname(1,2)  A = functionname(2,1) |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 5 | | |
| Use Case Name: | Input | | |
| Created By: | Juan Fernando Ulloa | Last Updated By: | Juan Fernando Ulloa |
| Date Created: | 21/11/2018 | Date Last Updated: | 21/11/2018 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | The user provides code for input (ReadInput) |
| Preconditions: | The input code must have a proper syntactical format for the input read in Nansus. |
| Postconditions: | The information will be read and stored within specified id. |
| Priority: | Medium |
| Frequency of Use: | Medium |
| Normal Course of Events: | 1. The read input is properly written in the code. 2. A variable is provided for storage. 3. The type of the storage is correct. 4. The content is stored into the variable. |
| Exceptions: | Exceptions whenever attempting to assign a different type of value to the variable. |
| Special Requirements: | None |
| Assumptions: | The user will typically provide input of the same type as the variable he/she wishes to store in. |
| Test Case Description: | For input testing, try out any variable and try to associate its corresponding value (int, 1; float 1.2; char, ‘a’), and any of these values should work. No other value should be allowed to go through. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 6 | | |
| Use Case Name: | Output | | |
| Created By: | Juan Fernando Ulloa | Last Updated By: | Juan Fernando Ulloa |
| Date Created: | 21/11/2018 | Date Last Updated: | 21/11/2018 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | The user provides code for an output (print) |
| Preconditions: | The output code must have a proper syntactical format for the declaration of the function in Nansus. |
| Postconditions: | The output will be displayed. |
| Priority: | Low |
| Frequency of Use: | Low |
| Normal Course of Events: | 1. The print will be properly declared in the code. 2. The user will provide either a variable name or a literal string to output. 3. The output will be displayed. |
| Exceptions: | Output is attempted with non-declared variables; no element is provided for the output |
| Special Requirements: | None |
| Assumptions: | The user will provide content to the output and knows the format for the print. |
| Test Case Description: | For this test case, you can use any predefined variable and place it as the argument in the print function. You can also write a literal string in one of the arguments and the output should be the same literal string once again. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 7 | | |
| Use Case Name: | Special Functions | | |
| Created By: | Juan Fernando Ulloa | Last Updated By: | Juan Fernando Ulloa |
| Date Created: | 21/11/2018 | Date Last Updated: | 21/11/2018 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | The user provides code for special functions (involving graphical output) |
| Preconditions: | The code must respect the special function in question and provide the correct number of arguments for its use. |
| Postconditions: | The graphical output for the function will be displayed. |
| Priority: | High |
| Frequency of Use: | High |
| Normal Course of Events: | 1. The function is properly written in the code. 2. The function properly executes its commands. |
| Exceptions: | Special function called without proper format or required types, attempts to use without correct parameters |
| Special Requirements: | None |
| Assumptions: | The user knows which special functions Nansus allows for him/her to use. |
| Test Case Description: | For special functions, try out all those that are available, by typing jedo.functionname for each of the different functions that we predefine, and you should see a special graphical output interaction corresponding to it. |

## Development Process

### Description

For the development process, we mostly tried to stay true to the process of building the compiler as we went through the corresponding topics during the week and implementing that part of the solution whenever we had the means to do it. In our process, we didn’t start testing until very late into the project, reason for which we had conflicts at later weeks, extending our workload to fix the incongruences up to that moment. During the last two weeks, we took one week to fix the incongruences up to that point, and the second week was used to wrap up the project and write the documentation. The process was mostly done in person as a pair, once we realized that working remotely resulted in us not completing the elements on time, and at the end, we mostly separated the implementation into one of us checking for compiler development and the other focusing on the virtual machine implementation.

### Logbook

|  |  |
| --- | --- |
| Expected progress | Syntactical and Lexical Analysis (Scanner and Parser) |
| Date | 17/9/18 - 21/9/18 |
| Actual Progress | * We designed the grammar at this point and implemented, at the time, an incomplete version of the lexer and the parser that couldn’t properly read the tokens for the code. * We didn’t execute any tests at this point. |

|  |  |
| --- | --- |
| Expected progress | Basic Semantics: Function Directory and Variable Table |
| Date | 24/9/18 - 28/9/18 |
| Actual Progress | * We added structures merely as concepts for both the function directory and variable table but didn’t really implement any function for them. * No tests * No changes to Lexer/Parser |

|  |  |
| --- | --- |
| Expected progress | Basic Expression Semantics: Semantic Cube  Code Generation for Arithmetic Operations |
| Date | 1/10/18 - 5/10/18 |
| Actual Progress | * We designed a structure for the Semantic Cube but didn’t really implement any function for it. * No tests * No further changes |

|  |  |
| --- | --- |
| Expected progress | Code Generation for Conditions |
| Date | 8/10/18 - 12/10/18 |
| Actual Progress | * We didn’t turn in a report for the week on this occasion. We didn’t develop any valuable information. |

|  |  |
| --- | --- |
| Expected progress | Function Code Generation |
| Date | 15/10/18 - 19/10/18 |
| Actual Progress | * After speaking with Dr. Hector, we realized that we were supposed to be testing up to that point, so after testing we soon realized that the parser was failing at the basic level, and not allowing syntax analysis at all. |

|  |  |
| --- | --- |
| Expected progress | Memory Mapping  Virtual Machine: Arithmetic Expressions and Sequential Statements |
| Date | 22/10/18 - 26/10/18 |
| Actual Progress | * Still in our slump, we continue to search the reason why the lexer and parser aren’t even allowing input. |

|  |  |
| --- | --- |
| Expected progress | Code Generation for Structured Types  Virtual Machine: Conditional Statement Execution |
| Date | 29/10/18 - 2/11/18 |
| Actual Progress | * At last, we realize the code-breaking error that we had: We had implemented “EMPTY” as a token, and that token was causing us to expect the string “empty” into the compiler as a token, since we didn’t realize that the empty value in parsing was simply left blank (neither of us solved the “Little Duck” Grammar) * We solve and now can read all the input within the syntax. |

|  |  |
| --- | --- |
| Expected progress | Syntactical and Lexical Analysis (Scanner and Parser) |
| Date | 5/11/18 - 9/11/18 |
| Actual Progress | * We map out the work we have yet to complete (basically everything, and proceed to start out during the Friday 11, resulting in about half of the regular actions to map the information to memory. |

|  |  |
| --- | --- |
| Expected progress | Syntactical and Lexical Analysis (Scanner and Parser) |
| Date | 12/11/18 - 16/11/18 |
| Actual Progress | * We work to start implementing the virtual machine in theory, but don’t go deep into it just yet. * We start working out the logic for the implementation of the now allocated memory spaces for the variables and begin with code generation. |

|  |  |
| --- | --- |
| Expected progress | Syntactical and Lexical Analysis (Scanner and Parser) |
| Date | 19/11/18 - 23/11/18 |
| Actual Progress | * After the weekend, we finally have functioning at compiling level everything but structured elements (arrays and matrices) and start testing the virtual machine. * Monday, we have the virtual machine working, and realize the need for string in the compilation (not as a storable variable, but as a token) and proceed to implement it. Arrays and Matrices have yet to work at this point. * The rest of the functionality and documentation is left to be completed during Tuesday 20, 2018. |

### Individual Reflections

The statement “You won’t finish this project in a day.” is to be feared even more that it seems. The project is extremely complex, however in a good sense. A better management would most definitely have had an impact in our workload at the end of the semester, but it still handles decently. I did learn extreme amounts of information from developing the project, things like just how complex memory allocation can be, and how that information is used by the interpreter, how the theory of a subject can so drastically impact the actual practical implementation. I think that the project is the culmination of everything we learn in class and is worthy of being the “last hurdle” for most of the people who study computer science.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Juan Fernando Ulloa

The compilers design project involves all the information we learned through our years studying computer science. We finally get our chance to design our own compiler and understand all the logic behind a compiler and truly understand the implications involving our coding style. The main resource you need for this project is time and patience, this project is not an easy task and it requires a lot of investigation and understanding of all the theory seen through class. Strong knowledge of object-oriented programming and data structures are your best friends through this project. This project helps you get a better idea of the reasons behind how popular languages are design the way they are why there and the reasons why so many languages exists out there, every single programming language has its pros and cons and all of this is because of the logic/strategy behind it and the functionality the designer were looking to fulfill.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jesus Guadiana

# Language Description

## Name

Our proposed language is called “Nansus”, a blend word from Juan Fernando’s pseudonym “Nando” and Jesus’ name.

## Main Characteristics

Our programming language takes a lot of inspiration from languages that follow similar structures to C++.

* The language expects input code to include a name for the program in the format that an identifier would usually have (“program name;”). All statements or commands are expected to end with a “;” except for the cycle (while, do while) structures and function definitions.
* We only manage three types of assignable variables – int, float, char -, however Boolean types are used when evaluating condition expressions and strings can be directly input into print commands and for certain special functions.
* Variables in our language are initialized by default with value of 0 for int, 0.0 for float, and “” for char; it is important to note that variables cannot be initialized while they are declared, only after.
* Any functions that wish to be included in the program must be declared before the main, as the main is the end of the program (must include “;” after closing the main context).
* Only single dimension dimensioned matrices were implemented, even though our original plan was to include matrices also, so only those are available.
* Inputs can be used by the three assignable type of variables, and outputs can use both the assignable variables as well as literal strings (“Like this for example”).
* Because the nature of our project is towards generating a graphical output, we implemented basic functions from languages such as LOGO that permit a cursor (which we call “jedo”), which permits functions such as: drawing lines, squares, circles, triangles, rectangles, lifting the pen and placing it again, creating colors, etc).

## Error Type List

### Compilation

* Variable “Name” is not defined: Access to a variable without declaring it first.
* Variable “Name” is not dimensioned: Trying to access a variable by providing indexes to a non-dimensional variable.
* Variable “Name” is not an array: Trying to access a variable by providing a second index to a variable that is a vector.
* Type Mismatch: This error can manifest when trying using every type of value – int, float, char, bool – at respective sections of the code that it is generated at (conditions, indexes, variable assignment, special functions etc.)
* Variable “Name” already defined: Whenever a variable already has a name allocated and it is attempted to assign it once again within the same context.
* Function is not defined: Whenever a function that is defined is attempted to be accessed.
* Number of arguments mismatch: Whenever a function is called without the proper number of arguments.
* Address not within usable range: Whenever the space for a memory partition (global, local, temporary and constant) runs out
* No remaining space in memory partition: Whenever a type of sub-memory (int, float, bool, string, char) runs out of space in the indicated context.
* Invalid Address in memory: When the memory type is not compatible with the operation in play.
* Address Type does not match value type: Value and address are of different types and therefore incompatible.
* Syntax Error: Tokens not received in predefined patterns.

### Execution

* Division by Zero: Undefined Operation.
* Input Type Error: Input Type does not match the type of the variable used for storage.
* Index out of bounds: Attempt to access a non-existent cell of an array.

# Compiler Description

## Computing Equipment, Language and Additional Props

The development of our language was mainly tested on Windows OS, 64-bit systems. We designed our compiler using Python 3.7, and since we used this language, our additional prop or main engine to produce the lexer and parser is Python Lex & Yacc Library.

## Lexical Analysis Description

|  |  |  |
| --- | --- | --- |
| Token | Regular Expression | Example |
| PRGM | r’program’ | program |
| MAIN | r’main’ | main |
| FUNCTION | r’function’ | function |
| WHILE | r’while’ | while |
| IF | r’if’ | if |
| DO | r’do’ | do |
| ELSE | r’else’ | else |
| ELSEIF | r’elseif’ | elseif |
| PRINT | r’print’ | print |
| JEDO | r’jedo’ | jedo |
| CIRCLE | r’circle’ | circle |
| SQUARE | r’square’ | square |
| RECTANGLE | r’rectangle’ | rectangle |
| TRIANGLE | r’triangle’ | triangle |
| FORWARD | r’forward’ | forward |
| BACK | r’back’ | back |
| TURNRIGHT | r’turnRight’ | turnRight |
| TURNLEFT | r’turnLeft’ | turnLeft |
| COLOR | r’color’ | color |
| THICKNESS | r’thickness’ | thickness |
| STARTPEN | r’startpen’ | startpen |
| STOPPEN | r’stoppen’ | stoppen |
| STARTFILL | r’startfill’ | startfill |
| FILLSHAPE | r’fillshape’ | fillshape |
| STOPFILL | r’stopfill’ | stopfill |
| RESTART | r’restart’ | restart |
| CREATE | r’create’ | create |
| DRAWDOT | r‘drawDot’ | drawDot |
| TYPEINT | r’int’ | int |
| TYPEFLOAT | r’float’ | float |
| TYPECHAR | r’char’ | char |
| NOTYPE | r’void’ | void |
| RETURN | r’return’ | Return |
| READINPUT | r’readInput’ | readInput |
| ID | r"[a-zA-Z][a-zA-Z0-9]\*" | a, B, c9, hello25 |
| PLUS | r’\+’ | + |
| MINUS | r’-’ | - |
| MULTIPLY | r’\\*’ | \* |
| DIVIDE | r’/’ | / |
| CSTI | r'\d+' | 12, 25, 0 |
| CSTF | r'\d\*\.\d+' | 0.0, 25.1 |
| CSTC | r'\'.\'' | ‘a’, ‘b’, ‘c’ |
| CSTS | r'\".\*\"' | “Hello World”, “Nansus” |
| EOS | r’;’ | ; |
| LEFTP | r’(’ | ( |
| RIGHTP | r’)’ | ) |
| LEFTB | r’[’ | [ |
| RIGHTB | r’]’ | ] |
| LEFTBRACE | r’{’ | { |
| RIGHTBRACE | r’}’ | } |
| AND | r'\&\&' | && |
| OR | r'\|\|' | || |
| SEPARATOR | r'\,' | , |
| POINT | r'\.' | . |
| EQUALS | r'\=' | = |
| GREATER | r'\>' | > |
| GREATEREQUAL | r'\>\=' | >= |
| LESS | r'\<' | < |
| LESSEQUAL | r'\<\=' | <= |
| EQUAL | r'\=\=' | == |
| NOTEQUAL | r'\!\=' | != |

# -----------------------------------------------------------------------------

# Juan Fernando and Jesus’ Programming Language

# lex.py

# Last edit: 18/11/2018

# -----------------------------------------------------------------------------

# Build the lexer

from functions.printToOutputFile import \*

import ply.lex as lex

import os

import sys

# Reserved Words

reservedWords = {

'program' : 'PRGM',

'main' : 'MAIN',

'function' : 'FUNCTION',

'while' : 'WHILE',

'if' : 'IF',

'do' : 'DO',

'else' : 'ELSE',

'elseif' : 'ELSEIF',

'print' : 'PRINT',

'jedo' : 'JEDO',

'circle' : 'CIRCLE',

'square' : 'SQUARE',

'rectangle' : 'RECTANGLE',

'triangle' : 'TRIANGLE',

'forward' : 'FORWARD',

'back' : 'BACK',

'turnRight' : 'TURNRIGHT',

'turnLeft' : 'TURNLEFT',

'color' : 'COLOR',

'thickness' : 'THICKNESS',

'startpen' : 'STARTPEN',

'stoppen' : 'STOPPEN',

'startfill' : 'STARTFILL',

'fillshape' : 'FILLSHAPE',

'stopfill' : 'STOPFILL',

'restart' : 'RESTART',

'int' : 'TYPEINT',

'float' : 'TYPEFLOAT',

'char' : 'TYPECHAR',

'void' : 'NOTYPE',

'return' : 'RETURN',

'readInput' : 'READINPUT'

}

#Token Definition

tokens = ['ID', 'PLUS', 'MINUS', 'MULTIPLY', 'DIVIDE',

'CSTI','CSTF', 'CSTC', 'CSTS', 'EOS','LEFTP', 'RIGHTP', 'LEFTB',

'RIGHTB', 'LEFTBRACE', 'RIGHTBRACE', 'AND', 'OR',

'SEPARATOR', 'POINT', 'EQUALS', 'GREATER', 'GREATEREQUAL',

'LESS', 'LESSEQUAL', 'EQUAL', 'NOTEQUAL']

#Add the reserved words to the token definition

tokens += reservedWords.values()

# Tokens Regulare Expressions for accepted values

t\_PLUS = r'\+'

t\_MINUS = r'-'

t\_MULTIPLY = r'\\*'

t\_DIVIDE = r'/'

t\_CSTI = r'\d+'

t\_CSTF = r'\d\*\.\d+'

t\_CSTC = r'\'.\''

t\_CSTS = r'\".\*\"'

t\_EOS = r'\;'

t\_LEFTP = r'\('

t\_RIGHTP = r'\)'

t\_LEFTB = r'\['

t\_RIGHTB = r'\]'

t\_LEFTBRACE = r'\{'

t\_RIGHTBRACE = r'\}'

t\_AND = r'\&\&'

t\_OR = r'\|\|'

t\_SEPARATOR = r'\,'

t\_POINT = r'\.'

t\_EQUALS = r'\='

t\_GREATER = r'\>'

t\_GREATEREQUAL = r'\>\='

t\_LESS = r'\<'

t\_LESSEQUAL = r'\<\='

t\_EQUAL = r'\=\='

t\_NOTEQUAL = r'\!\='

#Identifier token has to check for reserved words prior to storing as ID

def t\_ID(t):

r"[a-zA-Z][a-zA-Z0-9]\*"

t.type = reservedWords.get(t.value, 'ID')

return t

# Ignored characters (tabs, blanks, new line)

t\_ignore = " \t\n"

def t\_newline(t):

r'\n+'

t.lexer.lineno += t.value.count("\n")

def t\_error(t):

print("Illegal character %s" % t.value[0])

t.lexer.skip(1)

#Executes Lexer

lexer = lex.lex()

## Syntax Analysis Description

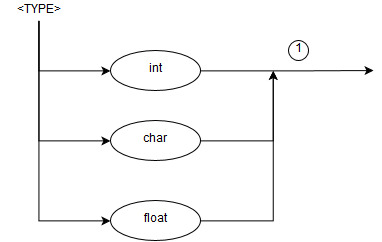
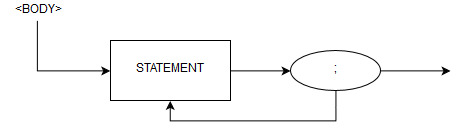
* 1. PROGRAM
     1. P -> program id ; P’
     2. P’ -> V P’
     3. P’ -> M P’
     4. P’ -> P’’
     5. P’’ -> main { P’’’
     6. P’’’ -> V P’’’
     7. P’’’ -> B } ;
  2. VARS
     1. V -> TY V’ V’’’
     2. V’ -> id V’’
     3. V’’ -> [ EXP ]
     4. V’’ -> ε
     5. V’’’ -> , V’ V’’’
     6. V’’’ -> V’’’’
     7. V’’’’ -> ,
  3. TYPE
     1. TY -> int
     2. TY -> float
     3. TY -> char
  4. BODY
     1. B -> S B’
     2. B’ -> B
     3. B’ -> ε
  5. MODULES
     1. M -> function M’ id ( M’’
     2. M’ -> TY
     3. M’ -> void
     4. M’’ -> T id M’’’
     5. M’’’ -> , M’’
     6. M’’’ -> ) { M’’’’
     7. M’’’’ -> V M’’’’
     8. M’’’’ -> B M’’’’’
     9. M’’’’’ -> return EXP ; }
     10. M’’’’’ -> }
  6. STATEMENT
     1. S -> A ;
     2. S -> PT ;
     3. S -> FC ;
     4. S -> C
     5. S -> SF ;
     6. S -> I ;
  7. COMPOUNDEXP
     1. CE -> EXPRES CE’
     2. CE’ -> && CE
     3. CE’ -> || CE
     4. CE’ -> ε
  8. EXPRESSION
     1. EXPRES -> EXP EXPRES’
     2. EXPRES’ -> > EXP
     3. EXPRES’ -> < EXP
     4. EXPRES’ -> == EXP
     5. EXPRES’ -> != EXP
     6. EXPRES’ -> >= EXP
     7. EXPRES’ -> <= EXP
  9. EXP
     1. EXP -> TERM EXP’
     2. EXP’ -> + EXP
     3. EXP’ -> - EXP
     4. EXP’ -> ε
  10. TERM
      1. TERM -> F TERM’
      2. TERM’ -> \* TERM
      3. TERM’ -> / TERM
      4. TERM’ -> ε
  11. FACTOR
      1. F -> ( EXPRES )
      2. F -> O
      3. F -> + O
      4. F -> - O
  12. OPERAND
      1. O -> csti
      2. O -> cstf
      3. O -> cstc
      4. O -> id O’
      5. O’ -> [ EXP ]
      6. O’ -> ( EXP O’’
      7. O’ -> ε
      8. O’’ -> , EXP O’’
      9. O’’ -> )
  13. ASSIGNMENT
      1. A -> id A’ = EXP
      2. A’ -> EXP
      3. A’ -> function FC
      4. A’’ -> [ EXP ]
      5. A’’ -> ε
  14. CONDITION
      1. C -> while ( CE ) { B }
      2. C -> do { B } while ( CE )
      3. C -> if C’
      4. C’ -> ( CE ) C’’
      5. C’’ -> S C’’’
      6. C’’ -> { B } C’’’
      7. C’’’ -> ELSEIF C’
      8. C’’’ -> ELSE C’’’’
      9. C’’’ -> ε
      10. C’’’’ -> { B }
      11. C’’’’ -> S
  15. FUNCTIONCALL
      1. FC -> id ( EXP FC’
      2. FC’ -> , EXP FC’
      3. FC’ -> )
  16. PRINT
      1. PT -> print ( PT’
      2. PT’ -> EXP PT’
      3. PT’ -> csts PT’’
      4. PT’’ -> , EXPRES PT’’
      5. PT’’ -> )
  17. SPECIALFUNCTION
      1. SF -> circle ( EXP )
      2. SF -> square ( EXP )
      3. SF -> rectangle ( EXP )
      4. SF -> drawDot ( EXP ) (csts)
      5. SF -> triangle ( EXP )
      6. SF -> forward ( EXP )
      7. SF -> back ( EXP )
      8. SF -> turnRight ( EXP )
      9. SF -> turnLeft ( EXP )
      10. SF -> color csts
      11. SF -> thickness (EXP)
      12. SF -> create ( )
      13. SF -> startpen ( )
      14. SF -> stoppen ( )
      15. SF -> startfill ( )
      16. SF -> fillshape ( csts )
      17. SF -> stopfill ( )
      18. SF -> restart ( )
  18. INPUT
      1. I -> readInput ( id )

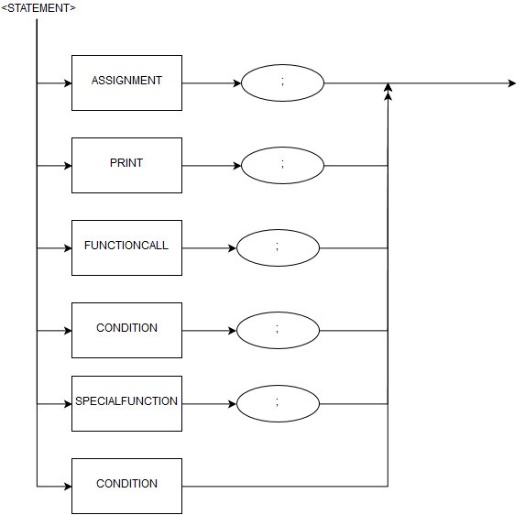
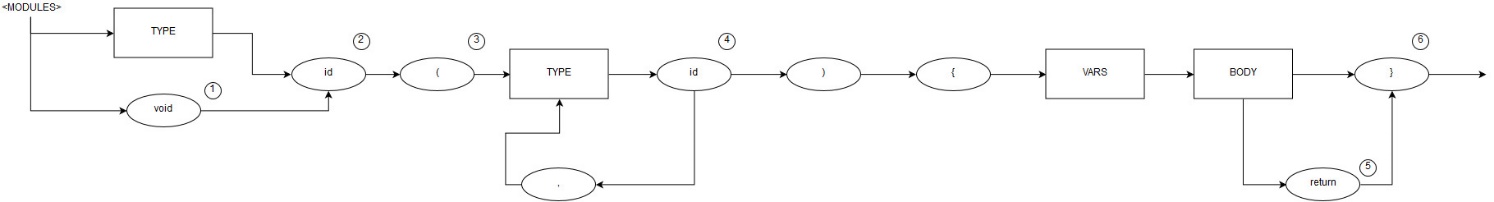
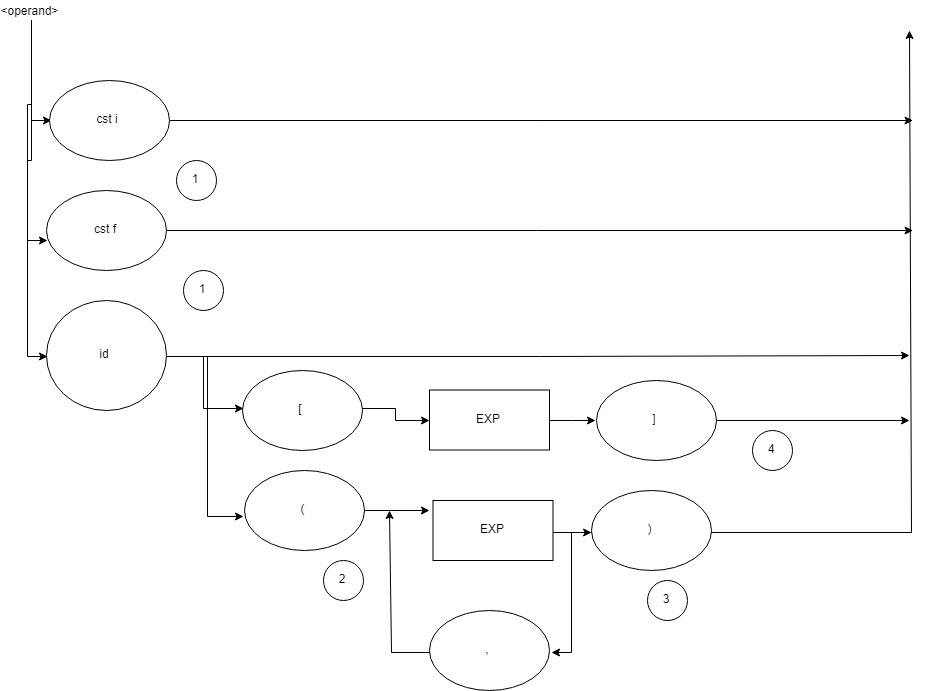
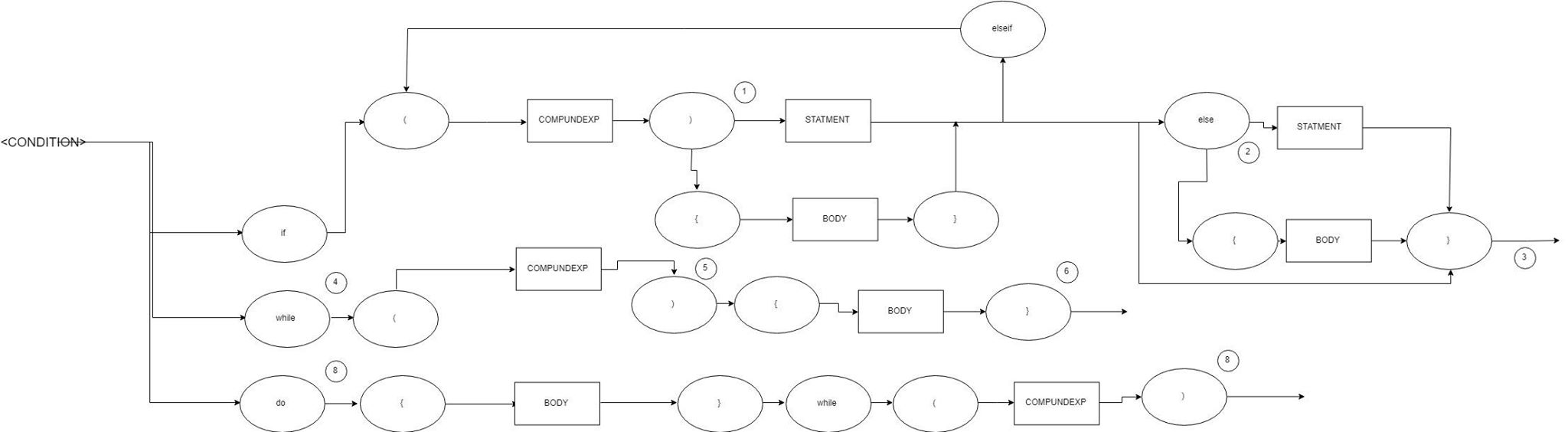
## Intermediate Code Generation and Semantic Analysis Description

### Operation Codes and Virtual Address Association

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Left Operand | Right Operand | Result |
| create | None | None | None |
| turnright | MemAddress (Degrees to turn) | None | None |
| turnleft | MemAddress(Degrees to turn) | None | None |
| forward | MemAddress(Amount of steps) | None | None |
| Circle | MemAddress(Radius) | None | None |
| Arch | MemAddress(Arch Size) | Direction to do arch | None |
| Square | MemAddress (Length of the sides) | None | None |
| Triangle | MemAddress (Length of the sides) | None | None |
| Rectangle | Memaddress (Length of x) | MemAddress(Length of y) | None |
| back | MemAddress(Amount of steps) | None | None |
| Color | MemAddress(String with jedo’s new color) | None | None |
| Thickness | MemAddress(Jedo’s pen thickness) | None | None |
| drawDot | MemAddres(Dot radius) | String containing the color | None |
| Startpen | None | None | None |
| Stoppen | None | None | None |
| Startfill | None | None | None |
| Fillshape | MemAddress(Color string for the shape) | None | None |
| stopFill | None | None | None |
| Restart | None | None | None |
| + | MemAddress(int or float) | MemAddress(int or float) | MemAddress |
| - | MemAddress(int or float) | MemAddress(int or float) | MemAddress |
| / | MemAddress(int or float) | MemAddress(int or float) | MemAddress |
| \* | MemAddress(int or float) | MemAddress(int or float) | MemAddress |
| = | MemAddress(variable’s address) | None | MemAddress |
| == | MemAddress(int,char or float to compare) | MemAddress(int,char, or float to compare) | MemAddress |
| < | MemAddress(int or float to compare) | MemAddress(int or float to compare) | MemAddress |
| > | MemAddress(int or float to compare) | MemAddress(int or float to compare) | MemAddress |
| >= | MemAddress(int or float to compare) | MemAddress(int or float to compare) | MemAddress |
| <= | MemAddress(int or float to compare) | MemAddress(int or float to compare) | MemAddress |
| != | MemAddress(int,float or char to compare | MemAddress(int or float to compare) | MemAddress |
| && | MemAddress(boolean value) | MemAddress(Boolean value) | MemAddress |
| || | MemAddress(Boolean value) | MemAddress(Boolean value) | MemAddress |
| PRINT | MemAddress(Value to print) | None | None |
| GOTO | None | None | Quad to jump |
| GOTOF | MemAddress(Result of Boolean operator) | None | Quad to jump |
| GOTOV | MemAddress(Result of Boolean operator) | None | Quad to jump |
| ENDPROC | None | None | None |
| ERA | FunctionName | None | None |
| GOSUB | None | None | Quad to jump |
| PARAMETER | MemAddress (parameter value) | None | None |
| RETURN | MemAddress(value) | None | MemAddress |
| VER | MemAddress(value to compare lower and upper indexes) | Lower index | Upper index |
| READINPUT | Var type | None | MemAddress |
| GOTO | MAIN |  | Quadtojump |

### C:\Users\user\AppData\Local\Temp\Untitled Diagram.jpgSyntax Diagrams

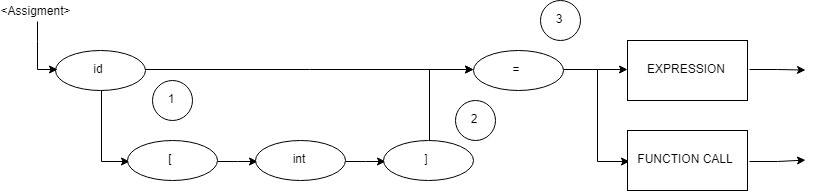
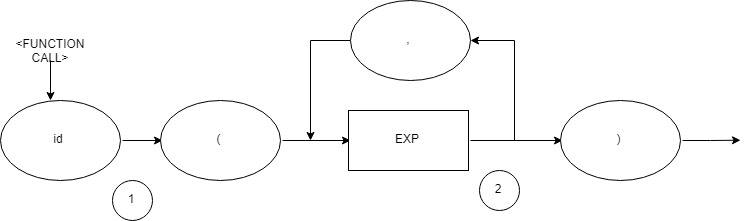
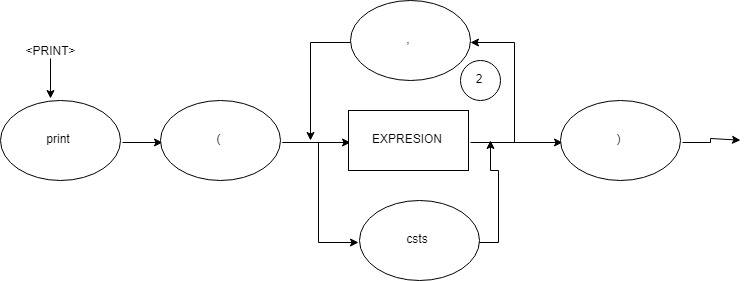
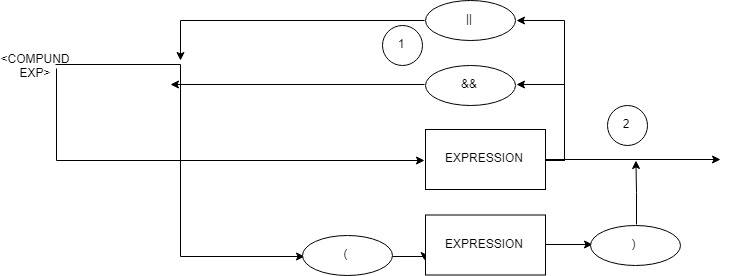
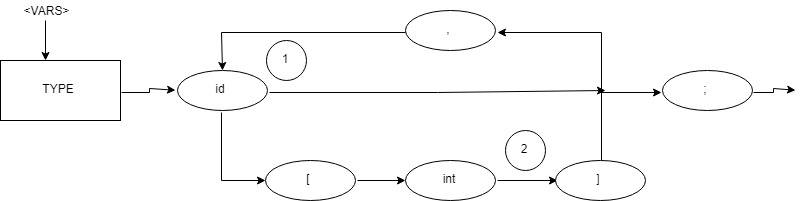




### https://scontent.fmfe1-1.fna.fbcdn.net/v/t1.15752-9/46521369_257734881589119_5139793590825779200_n.jpg?_nc_cat=107&_nc_ht=scontent.fmfe1-1.fna&oh=efe709e16616920a89eead34214a19aa&oe=5CAEEA21https://scontent.fmfe1-1.fna.fbcdn.net/v/t1.15752-9/46510860_1774452476015265_3525300386233581568_n.jpg?_nc_cat=103&_nc_ht=scontent.fmfe1-1.fna&oh=64a65b2e910f632cbafbb01a19ae7d73&oe=5C7F93A8https://scontent.fmfe1-1.fna.fbcdn.net/v/t1.15752-9/46495339_556237931456808_4675426137713672192_n.jpg?_nc_cat=109&_nc_ht=scontent.fmfe1-1.fna&oh=fd4057e404e6b4b5f17781cc53fc067c&oe=5CAB5057

### https://scontent.fmfe1-1.fna.fbcdn.net/v/t1.15752-9/46507536_925029241023636_8840490945255309312_n.jpg?_nc_cat=100&_nc_ht=scontent.fmfe1-1.fna&oh=251ee33955ef29d483152ed182c77973&oe=5C7B06EDhttps://scontent.fmfe1-1.fna.fbcdn.net/v/t1.15752-9/46512225_374081590026294_1045292291343777792_n.jpg?_nc_cat=109&_nc_ht=scontent.fmfe1-1.fna&oh=20a112b593a29c2b59f87201a6add3f2&oe=5C67404E

### https://scontent.fmfe1-1.fna.fbcdn.net/v/t1.15752-9/46507990_626782277719284_3429308365384187904_n.jpg?_nc_cat=111&_nc_ht=scontent.fmfe1-1.fna&oh=9082c6754133a83936ec08806c50b25a&oe=5C72F179

### Semantic Action Descriptions

* + 1. Program
       1. The program is initialized and the instance of the funcDirectory is generated. (Main quad created on the side)
       2. The id is associated and stored as the first function and set as the current context for working in (both global and current).
       3. The quadruple list is transferred to the virtual machine and the local information no longer has any use.
    2. Type
       1. Set current\_type variable to the value of the type
    3. Modules
       1. In case the type is void, set current\_type to void (avoids void variables)
       2. One the id is received, check for the function in the directory, and if it doesn’t exist, create it.
       3. Prepare to receive parameters by instantiating the functions VarTable.
       4. Read every id one by one and register them as parameters, store their values in VarTable and in signature (inner variables), store number of parameters.
       5. Generate a RETURN quadruple and store in it values for the return address as well as the value to store in said address.
       6. Generate ENDPROC quadruple and eliminate local information (Activation Record).
    4. Operand
       1. Pushes the id or constant into the operand stack (types into type stack)
       2. Fake bottom insert
       3. Pop Fake bottom
       4. Add the values for the multi-dimensioned variables.
    5. Condition
       1. Mark the spot to fill the quad number with the goto value
       2. Fill the goto value and mark the spot for the goto value to fill (else)
       3. Fill the else goto value
       4. Mark the spot for return after while
       5. Mark the spot to fill the quadruple for GOTOF
       6. Fill the GOTO and the GOTOF based on location
       7. Mark the spot to return on GOTOV
       8. Fill the spot in the newly made GOTOV
    6. Factor
       1. Fake bottom
       2. Pop Fake bottom
       3. Push operator to stack
    7. Term
       1. Push operator to stack
       2. Evaluate if \* or /
    8. Expression
       1. Push operator
       2. Evaluate if relational
    9. ReadInput
       1. Verify type of variable
    10. EXP
        1. Add operator to stack
        2. Evaluate if + or -
    11. Function Call
        1. Verify function id existence
        2. Validate every parameter
    12. Print
        1. Assign local memory if string
    13. Assign
        1. Push ID
        2. Add values for dimensioned variable
        3. Push operator
    14. CompoundExp
        1. Push logical operator
        2. Evaluate if the top is && or ||
    15. VARS
        1. Verify id isn’t in use, then add to table
        2. Add dimensioned variable additional directions

### Semantic Consideration Table (Semantic Cube)

|  |  |  |  |
| --- | --- | --- | --- |
| Type 1 | Type 2 | Operator | Result |
| Int | Int | + | Int |
| Int | Int | - | Int |
| Int | Int | / | Int |
| Int | Int | \* | Int |
| Int | Int | = | Int |
| Int | Int | < | Bool |
| Int | Int | > | Bool |
| Int | Int | == | Bool |
| Int | Int | <= | Bool |
| Int | Int | >= | Bool |
| Int | Int | != | Bool |
| Int | Int | && | Error |
| Int | Int | || | Error |
| Int | Float | + | Float |
| Int | Float | - | Float |
| Int | Float | / | Float |
| Int | Float | \* | Float |
| Int | Float | = | Float |
| Int | Float | < | Bool |
| Int | Float | > | Bool |
| Int | Float | == | Bool |
| Int | Float | <= | Bool |
| Int | Float | >= | Bool |
| Int | Float | != | Bool |
| Int | Float | && | Error |
| Int | Float | || | Error |
| Int | Char | + | Error |
| Int | Char | - | Error |
| Int | Char | / | Error |
| Int | Char | \* | Error |
| Int | Char | = | Error |
| Int | Char | < | Error |
| Int | Char | > | Error |
| Int | Char | == | Error |
| Int | Char | <= | Error |
| Int | Char | >= | Error |
| Int | Char | != | Error |
| Int | Char | && | Error |
| Int | Char | || | Error |
| Int | Bool | + | Error |
| Int | Bool | - | Error |
| Int | Bool | / | Error |
| Int | Bool | \* | Error |
| Int | Bool | = | Error |
| Int | Bool | < | Error |
| Int | Bool | > | Error |
| Int | Bool | == | Error |
| Int | Bool | <= | Error |
| Int | Bool | >= | Error |
| Int | Bool | != | Error |
| Int | Bool | && | Error |
| Int | Bool | || | Error |
| Float | Int | + | Float |
| Float | Int | - | Float |
| Float | Int | / | Float |
| Float | Int | \* | Float |
| Float | Int | = | Float |
| Float | Int | < | Bool |
| Float | Int | > | Bool |
| Float | Int | == | Bool |
| Float | Int | <= | Bool |
| Float | Int | >= | Bool |
| Float | Int | != | Bool |
| Float | Int | && | Error |
| Float | Int | || | Error |
| Float | Float | + | Float |
| Float | Float | - | Float |
| Float | Float | / | Float |
| Float | Float | \* | Float |
| Float | Float | = | Float |
| Float | Float | < | Bool |
| Float | Float | > | Bool |
| Float | Float | == | Bool |
| Float | Float | <= | Bool |
| Float | Float | >= | Bool |
| Float | Float | != | Bool |
| Float | Float | && | Error |
| Float | Float | || | Error |
| Float | Char | + | Error |
| Float | Char | - | Error |
| Float | Char | / | Error |
| Float | Char | \* | Error |
| Float | Char | = | Error |
| Float | Char | < | Error |
| Float | Char | > | Error |
| Float | Char | == | Error |
| Float | Char | <= | Error |
| Float | Char | >= | Error |
| Float | Char | != | Error |
| Float | Char | && | Error |
| Float | Char | || | Error |
| Float | Bool | + | Error |
| Float | Bool | - | Error |
| Float | Bool | / | Error |
| Float | Bool | \* | Error |
| Float | Bool | = | Error |
| Float | Bool | < | Error |
| Float | Bool | > | Error |
| Float | Bool | == | Error |
| Float | Bool | <= | Error |
| Float | Bool | >= | Error |
| Float | Bool | != | Error |
| Float | Bool | && | Error |
| Float | Bool | || | Error |
| Char | Int | + | Error |
| Char | Int | - | Error |
| Char | Int | / | Error |
| Char | Int | \* | Error |
| Char | Int | = | Error |
| Char | Int | < | Error |
| Char | Int | > | Error |
| Char | Int | == | Error |
| Char | Int | <= | Error |
| Char | Int | >= | Error |
| Char | Int | != | Error |
| Char | Int | && | Error |
| Char | Int | || | Error |
| Char | Float | + | Error |
| Char | Float | - | Error |
| Char | Float | / | Error |
| Char | Float | \* | Error |
| Char | Float | = | Error |
| Char | Float | < | Error |
| Char | Float | > | Error |
| Char | Float | == | Error |
| Char | Float | <= | Error |
| Char | Float | >= | Error |
| Char | Float | != | Error |
| Char | Float | && | Error |
| Char | Float | || | Error |
| Char | Char | + | Error |
| Char | Char | - | Error |
| Char | Char | / | Error |
| Char | Char | \* | Error |
| Char | Char | = | Char |
| Char | Char | < | Error |
| Char | Char | > | Error |
| Char | Char | == | Error |
| Char | Char | <= | Error |
| Char | Char | >= | Error |
| Char | Char | != | Error |
| Char | Char | && | Error |
| Char | Char | || | Error |
| Char | Bool | + | Error |
| Char | Bool | - | Error |
| Char | Bool | / | Error |
| Char | Bool | \* | Error |
| Char | Bool | = | Error |
| Char | Bool | < | Error |
| Char | Bool | > | Error |
| Char | Bool | == | Error |
| Char | Bool | <= | Error |
| Char | Bool | >= | Error |
| Char | Bool | != | Error |
| Char | Bool | && | Error |
| Char | Bool | || | Error |
| Bool | Int | + | Error |
| Bool | Int | - | Error |
| Bool | Int | / | Error |
| Bool | Int | \* | Error |
| Bool | Int | = | Error |
| Bool | Int | < | Error |
| Bool | Int | > | Error |
| Bool | Int | == | Error |
| Bool | Int | <= | Error |
| Bool | Int | >= | Error |
| Bool | Int | != | Error |
| Bool | Int | && | Bool |
| Bool | Int | || | Bool |
| Bool | Float | + | Error |
| Bool | Float | - | Error |
| Bool | Float | / | Error |
| Bool | Float | \* | Error |
| Bool | Float | = | Error |
| Bool | Float | < | Error |
| Bool | Float | > | Error |
| Bool | Float | == | Error |
| Bool | Float | <= | Error |
| Bool | Float | >= | Error |
| Bool | Float | != | Error |
| Bool | Float | && | Bool |
| Bool | Float | || | Bool |
| Bool | Char | + | Error |
| Bool | Char | - | Error |
| Bool | Char | / | Error |
| Bool | Char | \* | Error |
| Bool | Char | = | Error |
| Bool | Char | < | Error |
| Bool | Char | > | Error |
| Bool | Char | == | Error |
| Bool | Char | <= | Error |
| Bool | Char | >= | Error |
| Bool | Char | != | Error |
| Bool | Char | && | Bool |
| Bool | Char | || | Bool |
| Bool | Bool | + | Error |
| Bool | Bool | - | Error |
| Bool | Bool | / | Error |
| Bool | Bool | \* | Error |
| Bool | Bool | = | Error |
| Bool | Bool | < | Error |
| Bool | Bool | > | Error |
| Bool | Bool | == | Error |
| Bool | Bool | <= | Error |
| Bool | Bool | >= | Error |
| Bool | Bool | != | Error |
| Bool | Bool | && | Bool |
| Bool | Bool | || | Bool |

## Detailed Description of Memory Management in Compilation

* 1. Function Directory: Our function directory is responsible for managing the information for every context that the program is manages at compilation, this includes the following elements:
     1. Identifier: The name given for the function
     2. Parameters:
        1. Parameter Types: A list that manages the types of the parameters that are stored (for the signature).
        2. Parameter Addresses: A list that works alongside the previous one, this one storing the virtual addresses for the parameters as they are stored.
        3. Parameter Counter: Serves as part of the signature at compilation level, and verifies that the number of parameters during a function call matches the number that is defined for the function
     3. Local Variables: An instance of the Variable Table Structure that is unique to the defined function.
     4. Local Variable Counter:
        1. Int: Counts the number of variables of type int inside (for activation record) of this function
        2. Float: Counts the number of variables of type float inside (for activation record) of this function
        3. Char: Counts the number of variables of type char inside (for activation record) of this function
     5. Temporary Variable Counter:
        1. Int: Counts the number of temporary variables of type int inside (for activation record) of this function
        2. Float: Counts the number of temporary variables of type float inside (for activation record) of this function
        3. Char: Counts the number of temporary variables of type char inside (for activation record) of this function
        4. Bool: Counts the number of temporary variables of type bool inside (for activation record) of this function (This one doesn’t exist in local variable counter since there is no bool type variable assignment established)
     6. Return Type: For functions that have a return, this stores the type of value that is meant to be returned for verifications within operations.

This value can be void.

* + 1. Return Address: An address reserved for information that the function returns on call. It is set to -1 whenever a function is void.
    2. Quad Number: Used for management of context/flow change for function calls to know to what quadruple to go to.
  1. Variable Table:
     1. Identifier: Name for a variable that is defined. These must not be repeated within the same context.
     2. Type: Stores the type of content that the variable is compatible with.
     3. Address: For accessing content, this value contains the address where the information for the variable is stored to be used by operations.
     4. Dimension One: Stores the value for the size of a vector/array, this value is -1 whenever a variable is not dimensioned.
  2. Quadruples:
     1. Quad Number: The number that represents its place in the sequence of quadruples to execute.
     2. Operator: Any of the predefined operators (+, -, /, \*, etc.) that is stored in the first space of the quadruple.
     3. Left Operand: Depending on the situation, it takes different meanings, but typically for binary operations, the left operand of the operation.
     4. Right Operand: Depending on the situation, it takes different meanings, but typically for binary operations, the right operand of the operation.
     5. Result: Typically, a spot for the temporal space to store the iresult of a binary operation, however its meaning can change based on the operation executed.
  3. Current Program:
     1. Scope g and Scope l: These variables are meant to store the function name for the context. Scope g will always retain the name of the program, which is required at the start of the input, and Scope l will change name based on the context that is currently being evaluated.
     2. Array Control:
        1. Vec Mat First Dimension: Holds the value for the size of the - currently being evaluated – array.
        2. Vec Mat Variable Flag: A flag that is changed to True whenever an array is beginning to get declared (when it finds a bracket after an id)
     3. Temporary Management Attributes:
        1. Current Value Attributes: This includes id, function id, type, parameter identifier, parameter type, parameter counter, parameter evaluation counter, and current dimension. All these variables store temporary information mainly used to fill in the values or content of the elements that they are named after.
        2. Temporary Arg Types: Used to evaluate the types of a function when it is being called.
     4. Stacks:
        1. Operand Stack: Stores operands for binary operations.
           1. Type Stack: Stores types of operands for binary operations.
           2. Operator Stack: Stores operators for binary operations.
           3. Jump Stack: Stores the quadruple number for GOTO’s for later assignment to quadruples.
           4. Return Stack: Stores the values for returns within. (Main use is for recursion)
        2. Quadruple Attributes:
           1. Quad Number: Tracks the number for the next quadruple to generate.
           2. Quad List: Stores the intermediate code until it is sent to the virtual machine.
        3. Structures:
           1. Func Directory: An instance of the function directory structure.
           2. Sem Cube: An instance of the Semantic Cube structure.
           3. Mem: An instance of the Memory Interface Virtual Memory Structure. (More on that below)

# Virtual Machine Description

## Computing Equipment, Language and Additional Props

The development of our virtual machine was mainly tested on Windows OS, 64-bit systems. We continued using Python 3.7, and we used the Turtle library from Python in order to generate the graphical output.

## Detailed Description of Execution Memory Management Process

* 1. Mem Interface: The mem interface is responsible for separating the memory into different segments in the following way:
     1. Global Memory: The memory spaces assigned to the global memory are the following - 10000 to 15999.
     2. Local Memory: The memory spaces assigned to the local memory are the following - 16000 to 22199.
     3. Temporary Memory: The memory spaces assigned to the temporary memory are the following - 22000 to 27999.
     4. Constant Memory: The memory spaces assigned to the constant memory are the following - 28000 to 33999.
  2. Mem Manager: The mem Manager works with the memory spaces defined for each of the Global, Local, Temporary and Constant memories. Partitioning each individual memory into int, char, float, bool, or string memory spaces for each of the types of memory. Some of its characteristic elements are as follows:
     1. Identifier: The name of the memory space (Global, Local, Temporary, Constant)
     2. Initial: The initial memory block that will contain the memory blocks.
     3. Final: The final memory block for the given scope
     4. Partition Type: The partition type is defined by the number of addresses of the scope divided by 5 (available types) which in total gives us 1200 memory spaces per type value.
     5. Int Initial: Int initial is given by the initial value of the memory because ints are stored in the first available blocks for memory within the scope.
     6. Int Final: Int Final is the result of adding the initial memory value plus the partition size -1 or (Base + 1199)
     7. Char Initial: Char initial is the memory consisting of the base memory plus the partition size(Base + 1200)
     8. Char Final: is the result of adding the initial memory value plus the partition size \* 2 -1 or (Base + 2399)
     9. Float Initial: is the memory consisting of the base memory plus the partition size \* 2(Base + 2400)
     10. Float Final: is the result of adding the initial memory value plus the partition size \* 3 -1 or (Base + 3599)
     11. Bool Initial: is the memory consisting of the base memory plus the partition size \* 3 or (Base + 3600)
     12. Bool Final: is the result of adding the initial memory value plus the partition size \* 4 -1 or (Base + 4799)
     13. String Initial: is the memory consisting of the base memory plus the partition size \* 4 or (Base + 4800)
     14. String Final: is the result of adding the initial memory value plus the partition size \* 4 -1 (Base + 5999)
     15. Int current: Which defines the pointer of the current int memory which is initialized at the position of int initial
     16. char current: Which defines the pointer of the current int memory which is initialized at the position of char initial
     17. float current: Which defines the pointer of the current int memory which is initialized at the position of float initial
     18. bool current: Which defines the pointer of the current int memory which is initialized at the position of bool initial
     19. string current: Which defines the pointer of the current int memory which is initialized at the position of string initial
     20. int partition: Which is a subsegment dictionary in the form of a list for every int address and its value
     21. char partition: Which is a subsegment dictionary in the form of a list for every char address and its value
     22. float partition: Which is a subsegment dictionary in the form of a list for every float address and its value
     23. bool partition: Which is a subsegment dictionary in the form of a list for every bool address and its value
     24. string partition: Which is a subsegment dictionary in the form of a list for every string address and its value

### Virtual to Real Memory Association

There is no remapping in our project in terms of memory address within execution and within compilation, they follow the same partitions of 4 types of memory handled, with 5 sub-partitions representing the 5 types of memory that can be allocated (int, float, char, string, bool). We do however use information based off the attributes in the function directory to know what size of Register we need to activate on function call.

# Functionality Tests

### Test Code

Test 1:

program p;

int a, b, x, y;

float c;

main

{

a = 2;

b = 1;

c = 20;

jedo.create();

jedo.circle(25\*5);

jedo.restart();

x = 25;

y = 35;

jedo.rectangle(x, y);

jedo.restart();

jedo.triangle(x+y);

jedo.restart();

jedo.startfill();

jedo.fillshape("red");

jedo.circle(500/5);

jedo.stopfill();

jedo.forward(20);

jedo.restart();

while(a < 200){

jedo.square(a);

a = a + 10;

}

jedo.restart();

while(b < 10){

jedo.forward(10);

jedo.stoppen();

jedo.drawDot(5, "blue");

jedo.startpen();

b = b + 1;

}

jedo.arch(50,"right");

b = 0;

while(b < 10){

jedo.forward(10);

jedo.stoppen();

jedo.drawDot(5, "blue");

jedo.startpen();

b = b + 1;

}

jedo.restart();

a = 2;

b = 5;

if(a > b){

jedo.thickness(a);

jedo.color("green");

} else {

jedo.thickness(b);

jedo.color("red");

}

jedo.forward(2000/100);

jedo.arch(50, "right");

jedo.forward(2000/100);

jedo.stoppen();

jedo.forward(25);

jedo.startpen();

jedo.circle(75);

};

Test 2:

program p;

function void sorry(int j)

{

print("I can go in, but not recursively :C.");

}

main

{

int x, y;

int a, b, c, n, m;

int j, k;

x = 1;

y = 1;

b = 1;

c = 1;

print("");

print("Enter the number to calculate the factorial:");

print("");

readInput(x);

while(x > 1)

{

y = y \* x;

x = x - 1;

}

print("");

print("The while loop says that your value is:");

print(y);

print("");

print("Enter the number of elements you want from the base fibonacci sequence: ");

print("");

readInput(n);

m = n;

print("");

print("The do while, if, elseif and else say that your answer is: ");

print("");

do{

if(n == m)

{

print(1);

n = n - 1;

}

elseif(n == m - 1)

{

print(1);

n = n - 1;

}

else

{

a = b + c;

b = c;

c = a;

print (a);

n = n - 1;

}

}while (n > 0)

j = 1;

};

### Compiler and Virtual Machine Results

Test 1:

INTERMEDIATE CODE:

1 | GOTO MAIN None 2

2 | = 28000 None 10000

3 | = 28001 None 10001

4 | = 28002 None 12400

5 | create None None None

6 | \* 28003 28004 22000

7 | circle 22000 None None

8 | restart None None None

9 | = 28005 None 10002

10 | = 28006 None 10003

11 | rectangle 10003 10002 None

12 | restart None None None

13 | + 10002 10003 22001

14 | triangle 22001 None None

15 | restart None None None

16 | startfill None None None

17 | fillshape 26800 None None

18 | / 28007 28008 22002

19 | circle 22002 None None

20 | stopfill None None None

21 | forward 28009 None None

22 | restart None None None

23 | < 10000 28010 25600

24 | GOTOF 25600 None 29

25 | square 10000 None None

26 | + 10000 28011 22003

27 | = 22003 None 10000

28 | GOTO None None 23

29 | restart None None None

30 | < 10001 28012 25601

31 | GOTOF 25601 None 39

32 | forward 28013 None None

33 | stoppen None None None

34 | drawDot 28014 "blue" None

35 | startpen None None None

36 | + 10001 28015 22004

37 | = 22004 None 10001

38 | GOTO None None 30

39 | arch 28016 "right" None

40 | = 28017 None 10001

41 | < 10001 28018 25602

42 | GOTOF 25602 None 50

43 | forward 28019 None None

44 | stoppen None None None

45 | drawDot 28020 "blue" None

46 | startpen None None None

47 | + 10001 28021 22005

48 | = 22005 None 10001

49 | GOTO None None 41

50 | restart None None None

51 | = 28000 None 10000

52 | = 28022 None 10001

53 | > 10000 10001 25603

54 | GOTOF 25603 None 58

55 | thickness 10000 None None

56 | color 26801 None None

57 | GOTO None None 60

58 | thickness 10001 None None

59 | color 26802 None None

60 | / 28023 28024 22006

61 | forward 22006 None None

62 | arch 28025 "right" None

63 | / 28026 28027 22007

64 | forward 22007 None None

65 | stoppen None None None

66 | forward 28028 None None

67 | startpen None None None

68 | circle 28029 None None

### 

Test 2:

INTERMEDIATE CODE:

1 | GOTO MAIN None 4

2 | PRINT 26800 None None

3 | ENDPROC None None None

4 | = 28000 None 16001

5 | = 28000 None 16002

6 | = 28000 None 16004

7 | = 28000 None 16005

8 | PRINT 26801 None None

9 | PRINT 26802 None None

10 | PRINT 26803 None None

11 | READINPUT int None 16001

12 | > 16001 28000 25600

13 | GOTOF 25600 None 19

14 | \* 16002 16001 22000

15 | = 22000 None 16002

16 | - 16001 28000 22001

17 | = 22001 None 16001

18 | GOTO None None 12

19 | PRINT 26804 None None

20 | PRINT 26805 None None

21 | PRINT 16002 None None

22 | PRINT 26806 None None

23 | PRINT 26807 None None

24 | PRINT 26808 None None

25 | READINPUT int None 16006

26 | = 16006 None 16007

27 | PRINT 26809 None None

28 | PRINT 26810 None None

29 | PRINT 26811 None None

30 | == 16006 16007 25601

31 | GOTOF 25601 None 35

32 | PRINT 28000 None None

33 | - 16006 28000 22002

34 | = 22002 None 16006

35 | - 16007 28000 22003

36 | == 16006 22003 25602

37 | GOTOF 25602 None 42

38 | PRINT 28000 None None

39 | - 16006 28000 22004

40 | = 22004 None 16006

41 | GOTO None None 49

42 | + 16004 16005 22005

43 | = 22005 None 16003

44 | = 16005 None 16004

45 | = 16003 None 16005

46 | PRINT 16003 None None

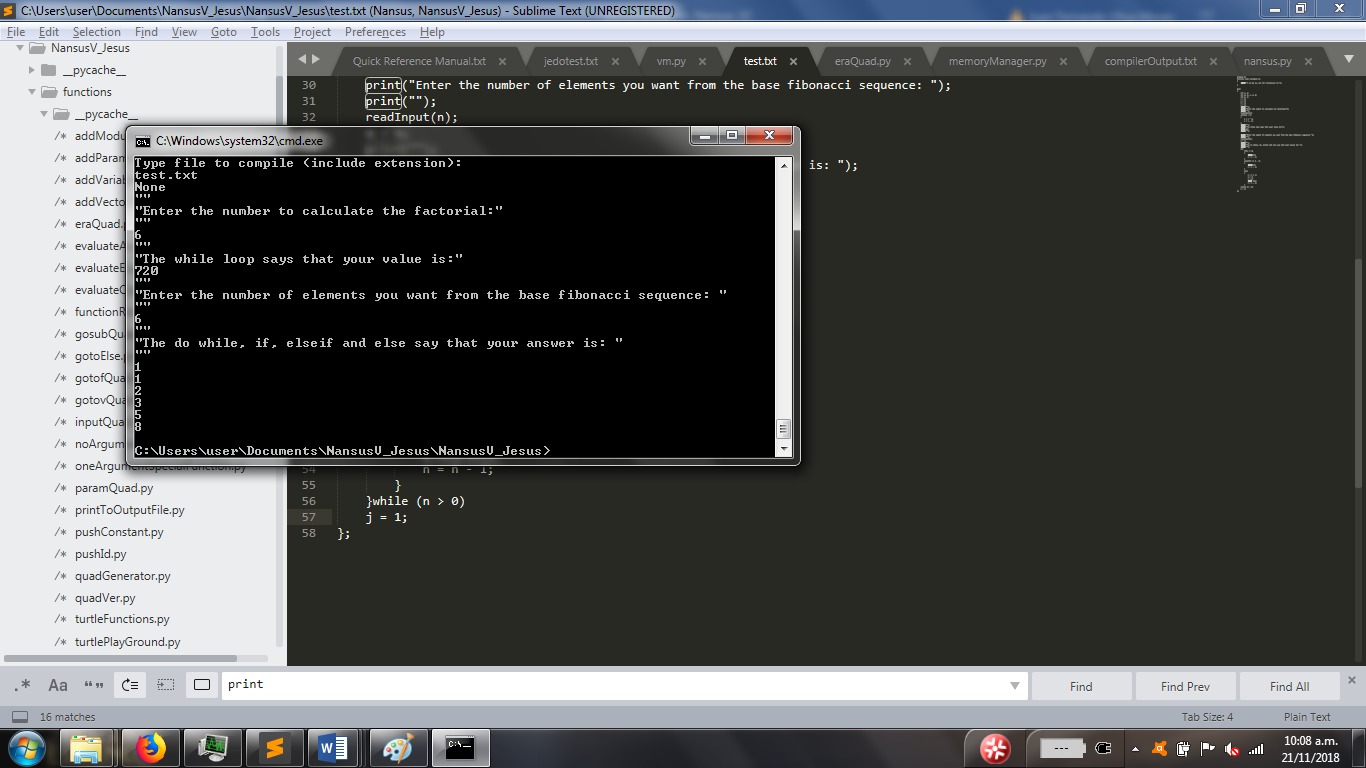
47 | - 16006 28000 22006

48 | = 22006 None 16006

49 | > 16006 28001 25603

50 | GOTOV 25603 None 30

51 | = 28000 None 16008



# Listings

# -----------------------------------------------------------------------------

# Juan Fernando and Jesus’ Programming Language

# nansus.py

# Last edit: 21/11/2018

# -----------------------------------------------------------------------------

#Python Lexer & Yacc

from ply import \*

#Main Program Structure Import

from currentProgram import CurrentProgram

#Import Virtual Machine

from vm import Machine

#Structure imports

from structures.funcDirectory import FuncDirectory

from structures.quad import Quad

from structures.semCube import SemanticCube

#Function imports

from functions.addModules import add\_module\_to\_directory

from functions.addParameter import add\_parameter\_to\_function

from functions.addVariable import add\_variable\_to\_directory

from functions.addVectorOrMatrix import add\_vector\_or\_matrix

from functions.eraQuad import era\_quad

from functions.evaluateAssign import evaluate\_assignment

from functions.evaluateExpression import evaluate\_binary\_operation

from functions.evaluateOperation import evaluate\_operation

from functions.functionReturn import return\_from\_function

from functions.gosubQuad import gosub\_quad

from functions.gotoElse import goto\_else\_function

from functions.gotofQuad import gotof\_quad\_function

from functions.gotovQuad import gotov\_quad\_function

from functions.inputQuad import input\_quad\_function

from functions.noArgumentSpecialFunction import no\_argument\_special\_function

from functions.oneArgumentSpecialFunction import one\_argument\_special\_function

from functions.paramQuad import param\_quad

from functions.printToOutputFile import print\_to\_output\_file

from functions.pushConstant import push\_constant\_to\_stack

from functions.pushId import push\_id\_to\_stack

from functions.quadGenerator import quad\_append

from functions.quadVer import quad\_ver\_function

from functions.twoArgumentSpecialFunction import two\_argument\_special\_function

from functions.twoArgumentSpecialFunctionIntString import two\_argument\_string\_special\_function

#Import Lexer and System Functions

import lex

import sys

#Obtain Tokens and Instantiate the Program Structure

tokens = lex.tokens

current\_program = CurrentProgram()

#Default syntax error message

def p\_error(p):

print("Syntax error at %s" %p.value)

#Program Syntax Diagram Grammar Representation

#Receive Program id and place a ;

def p\_program(p):

'program : PRGM ID main\_quad add\_program\_function EOS program\_prime'

#Cycle through global variables and modules declaration (order doesn't matter)

def p\_program\_prime(p):

'''program\_prime : vars program\_prime

| modules program\_prime

| program\_second\_prime'''

#Declares the main function space (change local context to main in fill\_main) and opens the body

def p\_program\_second\_prime(p):

'program\_second\_prime : MAIN fill\_main LEFTBRACE program\_third\_prime'

#Cycle through variable declaration (only allowed at the start) and then proceed to cycle through

#statements (body)

def p\_program\_third\_prime(p):

'''program\_third\_prime : vars program\_third\_prime

| body RIGHTBRACE EOS'''

#Calls function to create the default GOTO MAIN quadruple

def p\_main\_quad(p):

'main\_quad : '

current\_program.quad\_number += quad\_append(current\_program, Quad(current\_program.quad\_number,

"GOTO", "MAIN", None, None))

#Calls function to add the program function to the function directory

def p\_add\_program\_function(p):

'add\_program\_function : '

current\_program.scope\_g = p[-2] #Both of these functions are at the moment equal to the global context

current\_program.scope\_l = p[-2] #whose name is defined by the program name that comes in the test file

current\_program.func\_directory.new\_function(current\_program.scope\_g, 'void')

#Completes the main quad with the quadruple to which it has to go to at the start of program

#and then proceeds to make the main context the current local scope, as well as adding it to

#the function directory

def p\_fill\_main(p):

'fill\_main : '

quad = current\_program.quad\_list[0]

quad.set\_quad\_goto(current\_program.quad\_number)

current\_program.scope\_l = p[-1]

current\_program.func\_directory.new\_function(current\_program.scope\_l, 'void')

#Vars Syntax Diagram Grammar Representation

#Redirects to type, and then continues with variable declaration

def p\_vars(p):

'vars : type vars\_prime vars\_third\_prime '

#Reads the id, saves it to a variable within the CurrentProgram structure for use

def p\_vars\_prime(p):

'vars\_prime : ID save\_id vars\_second\_prime'

#After entering here, if it finds square brackets it will change the dimension flag to indicate that

#it is currently handling an array

def p\_vars\_second\_prime(p):

'''vars\_second\_prime : change\_dimension LEFTB exp RIGHTB first\_dimension dimensional\_address\_allocation

| '''

#Here it will look to see if more than one variable was declared, and cycle through the previous

#instructions, otherwise it will store the variable and proceed

def p\_vars\_third\_prime(p):

'''vars\_third\_prime : SEPARATOR store\_variable vars\_prime vars\_third\_prime

| store\_variable vars\_fourth\_prime'''

#Ends the declaration of variables

def p\_vars\_fourth\_prime(p):

'vars\_fourth\_prime : EOS'

#Save the id name in the global structure for easy usage

def p\_save\_id(p):

'save\_id : '

current\_program.current\_id = p[-1]

#Change flag for dimensioned variable within the main program structure

def p\_change\_dimension(p):

'change\_dimension : '

current\_program.vec\_mat\_variable\_flag = True

#Adds the variable to the funcDirectory in the current function

def p\_store\_variable(p):

'store\_variable : '

if not current\_program.vec\_mat\_variable\_flag:

add\_variable\_to\_directory(current\_program)

current\_program.vec\_mat\_variable\_flag = False

#Handles information to store the the indexed dimension of the array

def p\_first\_dimension(p):

'first\_dimension : '

exp\_type = current\_program.type\_stack.pop()

if exp\_type == 'int':

result = current\_program.operand\_stack.pop()

current\_program.vec\_mat\_first\_dimension = result

else:

print("Type mismatch at index.")

sys.exit()

#Creates memory location for the array and assigns it the value of its indexed dimension

def p\_dimensional\_address\_allocation(p):

'dimensional\_address\_allocation : '

base\_address = add\_vector\_or\_matrix(current\_program)

current\_program.func\_directory.set\_local\_variable\_dimension\_one(

current\_program.scope\_l, current\_program.current\_type, current\_program.current\_id,

current\_program.mem.get\_content(current\_program.vec\_mat\_first\_dimension))

#Type Syntax Diagram Representation

#Three types handled: int, float, char

def p\_type(p):

'''type : TYPEINT

| TYPEFLOAT

| TYPECHAR'''

#Changes the current type for variable analysis

current\_program.current\_type = p[1]

#Body Syntax Diagram Representation

#This is a representation of the content within function/main to cycle through statements

def p\_body(p):

'body : statement body\_prime'

def p\_body\_prime(p):

'''body\_prime : body

| '''

#Module Syntax Diagram Representation

#Names function and provides id, following into parameter declaration

def p\_modules(p):

'modules : FUNCTION modules\_prime ID add\_module LEFTP modules\_second\_prime'

#Assigns either a type or void type (important to avoid void variables) to the function

def p\_modules\_prime(p):

'''modules\_prime : type

| NOTYPE void\_type'''

#States the type for the first parameter and provides its id, which is then added

def p\_modules\_second\_prime(p):

'modules\_second\_prime : type ID add\_parameter modules\_third\_prime'

#Cycles through the parameter declaration and ends with the right parentheses

def p\_modules\_third\_prime(p):

'''modules\_third\_prime : SEPARATOR modules\_second\_prime

| RIGHTP LEFTBRACE modules\_fourth\_prime'''

#Represents the body, restricts variable declaration to beginning of the body

def p\_modules\_fourth\_prime(p):

'''modules\_fourth\_prime : vars modules\_fourth\_prime

| body modules\_fifth\_prime'''

#Generates return or procedes to close a return-less function

def p\_modules\_fifth\_prime(p):

'''modules\_fifth\_prime : RETURN exp return\_quad EOS RIGHTBRACE endproc\_quad

| RIGHTBRACE endproc\_quad'''

#Sets the current type to void (not a variable type)

def p\_void\_type(p):

'void\_type : '

current\_program.current\_type = "void"

#Adds the function to the FuncDirectory structure

def p\_add\_module(p):

'add\_module : '

current\_program.scope\_l = p[-1]

add\_module\_to\_directory(current\_program)

#Add each parameter one by one as they are analized

def p\_add\_parameter(p):

'add\_parameter : '

add\_parameter\_to\_function(current\_program, p[-1])

#Generates both the return quad and the GOTO to return to functioncall location

def p\_return\_quad(p):

'return\_quad : '

return\_from\_function(current\_program)

#Generates the ENDPROC to represent end of function

def p\_endproc\_quad(p):

'endproc\_quad : '

current\_program.quad\_number += quad\_append(current\_program,

Quad(current\_program.quad\_number, "ENDPROC", None, None, None))

#Statement Syntax Diagram Representation

#Goes to any of the six type of elements (conditions do not end with ";")

def p\_statement(p):

'''statement : assignment EOS

| print EOS

| functioncall EOS

| condition

| specialfunction EOS

| input EOS'''

#This function is used by every one of the following syntax diagram representations

#to evaluate binary operations of logical, relational, multiplication/division, and

#addition/subtraction operations

def evaluate\_expression(p):

current\_program.quad\_number += evaluate\_binary\_operation(current\_program)

#Compoundexp Syntax Diagram Representation

#Solves binary expressions at logical level (&& and ||)

def p\_compoundexp(p):

'compoundexp : expression eval\_logic compoundexp\_prime'

def p\_compoundexp\_prime(p):

'''compoundexp\_prime : AND push\_operator compoundexp

| OR push\_operator compoundexp

| '''

#Checks that the operator is for a logic operation

def p\_eval\_logic(p):

'eval\_logic :'

if len(current\_program.operator\_stack) > 0 and len(current\_program.operand\_stack) > 1:

if current\_program.operator\_stack[-1] == '&&' or current\_program.operator\_stack[-1] == '||':

evaluate\_expression(p)

#Expression Syntax Diagram Representation

#Solves binary expressions at relational level (>, <, ==, etc.)

def p\_expression(p):

'expression : exp eval\_relational expression\_prime'

def p\_expression\_prime(p):

'''expression\_prime : GREATER push\_operator expression

| LESS push\_operator expression

| EQUAL push\_operator expression

| NOTEQUAL push\_operator expression

| GREATEREQUAL push\_operator expression

| LESSEQUAL push\_operator expression

| '''

#Checks that the operator is for a relational operation

def p\_eval\_relational(p):

'eval\_relational :'

if len(current\_program.operator\_stack) > 0 and len(current\_program.operand\_stack) > 1:

if current\_program.operator\_stack[-1] == '>' or current\_program.operator\_stack[-1] == '<' or \

current\_program.operator\_stack[-1] == '>=' or current\_program.operator\_stack[-1] == '<=' or \

current\_program.operator\_stack[-1] == '==' or current\_program.operator\_stack[-1] == '!=':

evaluate\_expression(p)

#Exp Syntax Diagram Representation

#Solves binary expressions at term level (+ and -)

def p\_exp(p):

'exp : term eval\_term exp\_prime'

def p\_exp\_prime(p):

'''exp\_prime : PLUS push\_operator exp

| MINUS push\_operator exp

| '''

#Checks that the operator is for a arithmetical sum/subtraction operation

def p\_eval\_term(p):

'eval\_term :'

if len(current\_program.operator\_stack) > 0 and len(current\_program.operand\_stack) > 1:

if current\_program.operator\_stack[-1] == '+' or current\_program.operator\_stack[-1] == '-':

evaluate\_expression(p)

#Term Syntax Diagram Representation

#Solves binary expressions at factor level (\*) and /)

def p\_term(p):

'term : factor eval\_factor term\_prime'

def p\_term\_prime(p):

'''term\_prime : MULTIPLY push\_operator term

| DIVIDE push\_operator term

| '''

#Checks that the operator is for a arithmetical multiplication/division operation

def p\_eval\_factor(p):

'eval\_factor :'

if len(current\_program.operator\_stack) > 0 and len(current\_program.operand\_stack) > 1:

if current\_program.operator\_stack[-1] == '\*' or current\_program.operator\_stack[-1] == '/':

evaluate\_expression(p)

#Factor Syntax Diagram Representation

#Manages content at the lowest level

def p\_factor(p):

'''factor : LEFTP fake\_bottom expression RIGHTP pop\_fake\_bottom

| operand

| PLUS operand

| MINUS operand'''

#Adds a fake bottom to the operand stack

def p\_fake\_bottom(p):

'fake\_bottom : '

current\_program.operator\_stack.append('[(')

#Pops a fake bottom to the operand stack

def p\_pop\_fake\_bottom(p):

'pop\_fake\_bottom : '

current\_program.operator\_stack.pop()

#Operand Syntax Diagram Representation

#Inserts values for expression into operand stack

def p\_operand(p):

'''operand : CSTI evaluate\_operation\_int

| CSTF evaluate\_operation\_float

| CSTC evaluate\_operation\_char

| ID push\_id operand\_prime '''

def p\_operand\_prime(p):

'''operand\_prime : LEFTB exp RIGHTB quad\_ver\_one

| LEFTP exp operand\_second\_prime

| '''

def p\_operand\_second\_prime(p):

'''operand\_second\_prime : SEPARATOR exp operand\_second\_prime

| RIGHTP'''

#Pushes an id (address) for a variable into the operand stack

def p\_push\_id(p):

'push\_id : '

push\_id\_to\_stack(current\_program, p[-1])

#Pushes a constant into the operand stack (by type)

def p\_evaluate\_operation\_int(p):

'evaluate\_operation\_int : '

push\_constant\_to\_stack(current\_program, p[-1], 'int')

def p\_evaluate\_operation\_float(p):

'evaluate\_operation\_float : '

push\_constant\_to\_stack(current\_program, p[-1], 'float')

def p\_evaluate\_operation\_char(p):

'evaluate\_operation\_char : '

push\_constant\_to\_stack(current\_program, p[-1], 'char')

#Assignment Syntax Diagram Representation

def p\_assignment(p):

'assignment : ID push\_id save\_assignment\_id assignment\_second\_prime EQUALS push\_operator assignment\_prime'

def p\_assignment\_prime(p):

'''assignment\_prime : exp eval\_assignment

| FUNCTION functioncall eval\_assignment'''

def p\_assignment\_second\_prime(p):

'''assignment\_second\_prime :

| LEFTB exp RIGHTB quad\_ver\_one'''

current\_program.current\_dim = 1

#Generates VER quadruple for index management in arrays

def p\_quad\_ver\_one(p):

'quad\_ver\_one :'

quad\_ver\_function(current\_program)

current\_program.quad\_number += 1

#Evaluates assignment

def p\_eval\_assignment(p):

'eval\_assignment : '

evaluate\_assignment(current\_program)

current\_program.quad\_number += 1

#Useful to find the data when using it for comparisons

def p\_save\_assignment\_id(p):

'save\_assignment\_id : '

current\_program.current\_id = p[-2]

#Condition Syntax Diagram Representation

#Full declaration of both while and do, start of if

def p\_condition(p):

'''condition : WHILE save\_jump LEFTP compoundexp RIGHTP gotof\_quad LEFTBRACE body RIGHTBRACE goto\_while\_fill

| DO save\_jump LEFTBRACE body RIGHTBRACE WHILE LEFTP compoundexp RIGHTP gotov\_quad

| IF condition\_prime'''

#Open if parentheses

def p\_condition\_prime(p):

'condition\_prime : LEFTP compoundexp RIGHTP gotof\_quad condition\_second\_prime' #Here the gotof is generated

#The content of the if can either be a single statement or a braced series of statements

def p\_condition\_second\_prime(p):

'''condition\_second\_prime : statement condition\_third\_prime

| LEFTBRACE body RIGHTBRACE condition\_third\_prime'''

#Fills the goto values since it finishes the section, and gives the option to proceed to an elseif or else

def p\_condition\_third\_prime(p):

'''condition\_third\_prime : goto\_if\_fill ELSEIF condition\_prime

| ELSE goto\_else condition\_fourth\_prime

| goto\_if\_fill'''

#A single statement or a group of braced statetements followed by the action of filling in the gaps within the goto

def p\_condition\_fourth\_prime(p):

'''condition\_fourth\_prime : statement goto\_if\_fill

| LEFTBRACE body RIGHTBRACE goto\_if\_fill'''

#Adds content to the jump stack for filling GOTO quads

def p\_save\_jump(p):

'save\_jump :'

current\_program.jump\_stack.append(current\_program.quad\_number)

#Generates the GOTOF quad for while, if, and elseif

def p\_gotof\_quad(p):

'gotof\_quad :'

gotof\_quad\_function(current\_program)

#Generates the GOTO whenever an else appears or at the end of while

def p\_goto\_else(p):

'goto\_else : '

goto\_else\_function(current\_program)

#Generates the GOTOV for the do while condition

def p\_gotov\_quad(p):

'gotov\_quad : '

gotov\_quad\_function(current\_program)

#Fills the jump value of the if version of the goto

def p\_goto\_if\_fill(p):

'goto\_if\_fill :'

end = current\_program.jump\_stack.pop();

quad = current\_program.quad\_list[end]

quad.set\_quad\_goto(current\_program.quad\_number)

#Fills the jump value of the while version of the goto

def p\_goto\_while\_fill(p):

'goto\_while\_fill :'

end = current\_program.jump\_stack.pop();

ret = current\_program.jump\_stack.pop();

current\_program.quad\_number += quad\_append(current\_program, Quad(current\_program.quad\_number, "GOTO", None, None, ret))

quad = current\_program.quad\_list[end]

quad.set\_quad\_goto(current\_program.quad\_number)

#Functioncall Syntax Diagram Representation

#The function call declares the id and opens the parenthesis, every parameter is verified

def p\_functioncall(p):

'functioncall : ID verify\_function LEFTP fake\_bottom exp verify\_parameter functioncall\_prime'

#Keeps verifying parameters, ends when it finds the end of the fake bottom and proceeds to jump to function

def p\_functioncall\_prime(p):

'''functioncall\_prime : SEPARATOR exp verify\_parameter functioncall\_prime

| verify\_param\_count RIGHTP pop\_fake\_bottom store\_memory\_data go\_sub\_quad'''

#Generates the ERA quad, which will have information for the Activation Register at the time of execution

def p\_verify\_function(p):

'verify\_function : '

current\_program.current\_func\_id = p[-1]

current\_program.quad\_number += era\_quad(current\_program, p[-1])

current\_program.param\_evaluation\_counter = 0

#One by one, verifies that each paramter is satisfactory to the signature of the function

def p\_verify\_parameter(p):

'verify\_parameter : '

current\_program.quad\_number += param\_quad(current\_program)

current\_program.param\_evaluation\_counter += 1

#One the params are all received, this command checks to see that the number of parameters matches the signature

def p\_verify\_param\_count(p):

'verify\_param\_count : '

current\_function = current\_program.func\_directory.get\_function(current\_program.current\_func\_id)

if current\_program.param\_evaluation\_counter != current\_function['parameters']['parameter\_counter']:

print("Error: Expected " + str(current\_function['parameters']['parameter\_counter'])

+ " arguments for call to function " + p[-6] + ".")

#A jump stack for the function to return to

def p\_store\_memory\_data(p):

'store\_memory\_data : '

current\_program.jump\_stack.append(current\_program.quad\_number)

#Generates the GOsub quadruple and adds it to the list

def p\_go\_sub\_quad(p):

'go\_sub\_quad : '

target = current\_program.current\_func\_id

current\_program.quad\_number += gosub\_quad(current\_program, target)

#Pushes an operator into the stack

def p\_push\_operator(p):

'push\_operator : '

current\_program.operator\_stack.append(p[-1])

#Print Syntax Diagram Representation

#Declares print and opens a fake bottom

def p\_print(p):

'print : PRINT LEFTP fake\_bottom print\_prime'

#Followed by an expression of any type of the three accepted variables, as well as literal strings

def p\_print\_prime(p):

'''print\_prime : expression print\_quad print\_second\_prime

| CSTS print\_string\_quad print\_second\_prime'''

#Separates the outputs the user wishes and treats every one indiviudually

def p\_print\_second\_prime(p):

'''print\_second\_prime : SEPARATOR expression print\_quad print\_second\_prime

| RIGHTP pop\_fake\_bottom'''

#Generates the PRINT quad

def p\_print\_quad(p):

'print\_quad : '

print\_content = current\_program.operand\_stack.pop()

print\_type = current\_program.type\_stack.pop()

current\_program.quad\_number += quad\_append(current\_program,

Quad(current\_program.quad\_number, 'PRINT', print\_content, None, None))

#Generates the specific type of PRINT quad that uses a string (since it is uniquely stored)

def p\_print\_string\_quad(p):

'print\_string\_quad : '

temporary\_string = current\_program.mem.temporary\_memory\_assign('string', p[-1])

current\_program.quad\_number += quad\_append(current\_program,

Quad(current\_program.quad\_number, 'PRINT', temporary\_string, None, None))

#SpecialFunction Syntax Diagram Representation

#Jedo.specialfunction

def p\_specialfunction(p):

'specialfunction : JEDO POINT specialfunction\_prime'

#Every single function that jedo can execute (based on turtle)

def p\_specialfunction\_prime(p):

'''specialfunction\_prime : CIRCLE LEFTP exp RIGHTP one\_argument\_quad

| SQUARE LEFTP exp RIGHTP one\_argument\_quad

| RECTANGLE LEFTP exp SEPARATOR exp RIGHTP two\_argument\_quad

| DRAWDOT LEFTP exp SEPARATOR CSTS RIGHTP two\_argument\_string\_quad

| ARCH LEFTP exp SEPARATOR CSTS RIGHTP two\_argument\_string\_quad

| TRIANGLE LEFTP exp RIGHTP one\_argument\_quad

| FORWARD LEFTP exp RIGHTP one\_argument\_quad

| BACK LEFTP exp RIGHTP one\_argument\_quad

| TURNRIGHT LEFTP exp RIGHTP one\_argument\_quad

| TURNLEFT LEFTP exp RIGHTP one\_argument\_quad

| COLOR LEFTP CSTS RIGHTP special\_string\_quad

| THICKNESS LEFTP exp RIGHTP one\_argument\_quad

| STARTPEN LEFTP RIGHTP no\_argument\_quad

| CREATE LEFTP RIGHTP no\_argument\_quad

| STOPPEN LEFTP RIGHTP no\_argument\_quad

| STARTFILL LEFTP RIGHTP no\_argument\_quad

| FILLSHAPE LEFTP CSTS RIGHTP special\_string\_quad

| STOPFILL LEFTP RIGHTP no\_argument\_quad

| RESTART LEFTP RIGHTP no\_argument\_quad

| '''

#For those functions that don't requiere parameters

def p\_no\_argument\_quad(p):

'no\_argument\_quad :'

special\_function = p[-3]

no\_argument\_special\_function(current\_program, special\_function)

#For those functions that require one int argument

def p\_one\_argument\_quad(p):

'one\_argument\_quad :'

special\_function = p[-4]

one\_argument\_special\_function(current\_program, special\_function)

#For those functions that require two int arguments

def p\_two\_argument\_quad(p):

'two\_argument\_quad :'

special\_function = p[-6]

two\_argument\_special\_function(current\_program, special\_function)

#For those functions that require one int argument and one string argument

def p\_two\_argument\_string\_quad(p):

'two\_argument\_string\_quad :'

special\_function = p[-6]

two\_argument\_string\_special\_function(current\_program, special\_function, p[-2])

#For those functions that require one string argument

def p\_special\_string\_quad(p):

'special\_string\_quad :'

temporary\_string = current\_program.mem.temporary\_memory\_assign('string', p[-2])

current\_program.quad\_number += quad\_append(current\_program,

Quad(current\_program.quad\_number, p[-4], temporary\_string, None, None))

#Input Syntax Diagram Representation

#ReadInput followed by the ID where the input will be stored and closed rightaway (no multi-input)

def p\_input(p):

'input : READINPUT LEFTP ID input\_quad RIGHTP'

#Generates the READINPUT quadruple

def p\_input\_quad(p):

'input\_quad : '

input\_quad\_function(current\_program, p[-1])

#Executes parser

import ply.yacc as yacc

import pprint

parser = yacc.yacc()

#Pretty Printer to make the format more pleasant

pp = pprint.PrettyPrinter(indent=4)

with open('test.txt','r') as f:

input = f.read()

pp.pprint(parser.parse(input))

print\_to\_output\_file("\nINTERMEDIATE CODE:\n")

#Prints intermediate code

index = 0

for i in range(len(current\_program.quad\_list)):

print\_to\_output\_file(current\_program.quad\_list[index])

index += 1

#print\_to\_output\_file("\nFUNCTION DIRECTORY:")

#current\_program.func\_directory.print\_directory()

#Executres the virtual machine with copies of the memory, function directory and list of quadruples

vm = Machine(current\_program.quad\_list, current\_program.mem, current\_program.func\_directory)

vm.run\_machine();

# -----------------------------------------------------------------------------

# Juan Fernando and Jesus’ Programming Language

# vm.py

# Last edit: 21/11/2018

# -----------------------------------------------------------------------------

from functions.turtleFunctions import Jedo

from memory.memoryInterface import MemInterface

from structures.funcDirectory import FuncDirectory

from ast import literal\_eval

import sys

import pprint

class Machine():

def \_\_init\_\_(self, quads, memory, funcDirectory):

self.quads = quads;

self.total\_quads = len(quads);

self.current\_quad = 0;

self.memory = memory;

self.funcDirectory = funcDirectory

def request\_local\_addresses(self, function):

for i in range(function['function']['local\_variable\_counter']['int']):

function['memory'].local\_memory\_assign('int')

for i in range(function['function']['local\_variable\_counter']['float']):

function['memory'].local\_memory\_assign('float')

for i in range(function['function']['local\_variable\_counter']['char']):

function['memory'].local\_memory\_assign('char')

def request\_temporal\_addresses(self, function):

for i in range(function['function']['temporary\_variable\_counter']['int']):

function['memory'].temporary\_memory\_assign('int')

for i in range(function['function']['temporary\_variable\_counter']['float']):

function['memory'].temporary\_memory\_assign('float')

for i in range(function['function']['temporary\_variable\_counter']['char']):

function['memory'].temporary\_memory\_assign('char')

for i in range(function['function']['temporary\_variable\_counter']['bool']):

function['memory'].temporary\_memory\_assign('bool')

def literal\_eval\_helper(self, type):

if type == "<class 'int'>":

type == "int"

def run\_machine(self):

pp = pprint.PrettyPrinter(indent=4)

memory = self.memory

function = {}

paramters = 0

local\_segment\_pointer\_list = []

temporal\_segment\_pointer\_list = []

instruction\_number\_to\_back\_list = []

while self.current\_quad < self.total\_quads:

quad = self.quads[self.current\_quad]

operator = quad.operator

l\_address = quad.left\_operand

r\_address = quad.right\_operand

result\_address = quad.result

# print(function)

#SPECIAL FUNCTIONS

#Creates jedo object with speed slow and increments current\_quad

if operator == "create":

jedo = Jedo("slow")

self.current\_quad += 1

#If the operator in the quad contains "circle"

elif operator == "circle":

#Get the value inside the memory address in the l\_address

#Send left\_operand (circle radius) as parameter to the function circle

left\_operand = memory.get\_content(l\_address)

jedo.circle(left\_operand)

#Procced to the next quad

self.current\_quad += 1

elif operator == "arch":

#Get the value inside l\_address which represents the size of the arch7

#Trim " " from the string in the r\_address which represents arch to the right or to the left

left\_operand = memory.get\_content(l\_address)

r\_string = r\_address.replace('"',"")

#if r\_string is right perfom an arch to the right of jedo

if r\_string == "right":

jedo.turnRight(90)

#Performs a semi circle with the size given by the user (left operand)

jedo.circle(left\_operand, 180)

jedo.turnRight(90)

#if r\_string is left perfomr an arch to the left of jedo

elif r\_string == "left":

jedo.turnRight(90)

#Performs a semi circle with the size given by the user (left operand)

jedo.circle(left\_operand, -180)

jedo.turnRight(90)

self.current\_quad += 1

#Creates a square in the turtle interface

elif operator == "square":

#Get the content in l\_address and insert it as parameter in the square function

left\_operand = memory.get\_content(l\_address)

jedo.square(left\_operand)

self.current\_quad += 1

elif operator == "triangle":

#Get the content in l\_address and insert it as parameter in the triangle function

left\_operand = memory.get\_content(l\_address)

jedo.triangle(left\_operand)

self.current\_quad += 1

#Creates a rectangle in the turtle interface

elif operator == "rectangle":

#Get the content in l\_addres and r\_address

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(l\_address)

#Perform a square x = left\_operand, y = right\_operand

jedo.rectangle(left\_operand, right\_operand)

self.current\_quad += 1

#Makes turtle move forward x steps

elif operator == "forward":

#Get the content in l\_address

left\_operand = memory.get\_content(l\_address)

#Move forward x positions

jedo.forward(left\_operand);

self.current\_quad += 1

elif operator == "back":

#Get the content in l\_address

left\_operand = memory.get\_content(l\_address)

#Move backwards x positions

jedo.back(left\_operand)

self.current\_quad += 1

#Turtle turns right x degrees

elif operator == "turnRight":

left\_operand = memory.get\_content(l\_address)

jedo.right(left\_operand)

self.current\_quad += 1

#Turtle turns left x degrees

elif operator == "turnLeft":

left\_operand = memory.get\_content(l\_address)

jedo.left(left\_operand)

self.current\_quad += 1

#Changes turtles's color to x color

elif operator == "color":

left\_operand = memory.get\_content(l\_address)

left\_operand = left\_operand.replace('"',"")

jedo.color(left\_operand)

self.current\_quad += 1

#changes pen's thichkness

elif operator == "thickness":

left\_operand = memory.get\_content(l\_address)

jedo.thickness(left\_operand)

self.current\_quad += 1

#---------------------------------------NEW SPECIAL FUNCTION (DOCUMENTATION REQUIRED) -------------------------------

#Draws a dot with x radius and y color

elif operator == "drawDot":

#Get the content in l\_addres which represents the radius value

left\_operand = memory.get\_content(l\_address)

#Get the string color in the r\_address and trim "" values

right\_operand = r\_address;

right\_operand = right\_operand.replace('"',"")

jedo.drawDot(left\_operand, right\_operand)

self.current\_quad += 1

#Enables turtle to draw on movement

elif operator == "startpen":

jedo.startPen()

self.current\_quad += 1

#Turtle is not able to draw

elif operator == "stoppen":

jedo.stopPen()

self.current\_quad += 1

#Tells turtle that an object will be filled with a color

elif operator == "startfill":

jedo.startFill()

self.current\_quad += 1

#Tells the turtle what color to fill the shape with

elif operator == "fillshape":

#Get the color in string format

left\_operand = memory.get\_content(l\_address)

left\_operand = left\_operand.replace('"',"")

#fill the shape with the given color

jedo.fillShape(left\_operand)

self.current\_quad += 1

#Tells the turtle to stop the fill"

elif operator == "stopfill":

jedo.stopFill()

self.current\_quad += 1

#Restar turtle interface

elif operator == "restart":

jedo.restart()

self.current\_quad += 1

#Execute the + operation for incoming quad

elif operator == "+":

#Gets the left and right values to add

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#Performs the SUM action

result = right\_operand + left\_operand

#Store the result of the operation on the result address given by the quad

memory.edit\_memory\_content(result\_address, result)

self.current\_quad += 1

elif operator == "=":

#Get the value in the l\_address

left\_operand = memory.get\_content(l\_address)

#Store the value in left\_operand

memory.edit\_memory\_content(result\_address, left\_operand)

self.current\_quad += 1

#Execute the - operation for incoming quad

elif operator == "-":

#Get the left and right values and perform the subtraction

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#Store the result of the - operation

result = left\_operand - right\_operand

#store the result of the operation in the result\_address

memory.edit\_memory\_content(result\_address, result)

self.current\_quad += 1

#Execute the \* operation for incoming quad

elif operator == "\*":

#Get the left and right values and perform the multiplication

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

result = right\_operand \* left\_operand

#Store the result in the result\_address

memory.edit\_memory\_content(result\_address, result)

self.current\_quad += 1

#Execute the / operation for incoming quad

elif operator == "/":

#Get the left and right values and perform the division

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#if the right operand is 0 raise an error and exit program

if right\_operand == 0:

print ('ZeroDivisionError: division by zero')

sys.exit()

else:

#Else perform the division

result = left\_operand / right\_operand

#Store the result in the result\_address

memory.edit\_memory\_content(result\_address, result)

self.current\_quad +=1

#Execute the == operation for the incomig quad

elif operator == "==":

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#Perform the == operation and get the result

boolean\_result = left\_operand == right\_operand

#return 1 or 0 to represent booleans

if boolean\_result == True:

result = 1

else:

result = 0

#Store the result in the result\_address

memory.edit\_memory\_content(result\_address, result)

self.current\_quad += 1

elif operator == "<":

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#Perform the < operation and store the result

boolean\_result = left\_operand < right\_operand

if boolean\_result == True:

result = 1

else:

result = 0

memory.edit\_memory\_content(result\_address, result)

self.current\_quad += 1

#Execute the > operation

elif operator == ">":

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#Perform the > operation and store the result

boolean\_result = left\_operand > right\_operand

if boolean\_result == True:

result = 1

else:

result = 0

memory.edit\_memory\_content(result\_address, result)

self.current\_quad += 1

#Execure the >= operation for the incoming quad

elif operator == ">=":

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#Perform the >= operation and store the result

boolean\_result = left\_operand >= right\_operand

if boolean\_result == True:

result = 1

else:

result = 0

memory.edit\_memory\_content(result\_address , result)

self.current\_quad += 1

#Execute the <= operation for the incoming quad

elif operator == "<=":

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#Perform the <= operation and store the result

boolean\_result = left\_operand <= right\_operand

if boolean\_result == True:

result = 1

else:

result = 0

memory.edit\_memory\_content(result\_address , result)

self.current\_quad += 1

#Execute the != operation for the incomign quad

elif operator == "!=":

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#Perform the != operation and store the result

boolean\_result = left\_operand != right\_operand

if boolean\_result == True:

result = 1

else:

result = 0

memory.edit\_memory\_content(result\_address, result)

self.current\_quad += 1

elif operator == "&&":

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#Perform the $$ operation and store the result

boolean\_result = left\_operand and right\_operand

if boolean\_result == True:

result = 1

else:

result = 0

memory.edit\_memory\_content(result\_address, result)

self.current\_quad += 1

elif operator == "||":

left\_operand = memory.get\_content(l\_address)

right\_operand = memory.get\_content(r\_address)

#Perform the || operation and store the result

boolean\_result = left\_operand or right\_operand

if boolean\_result == True:

result = 1

else:

result = 0

memory.edit\_memory\_content(result\_address, result)

self.current\_quad += 1

#Execute the print operation for the incoming quad

elif operator == "PRINT":

left\_operand = memory.get\_content(l\_address)

#Print the l\_address content

print(str(left\_operand))

self.current\_quad += 1

elif operator == "GOTO":

#Redirect current\_quad to the quad number in the result\_address

self.current\_quad = result\_address - 1

#GOTOF operation

elif operator == "GOTOF":

left\_operand = memory.get\_content(l\_address)

if not left\_operand:

self.current\_quad = result\_address - 1

else:

self.current\_quad += 1

elif operator == "GOTOV":

left\_operand = memory.get\_content(l\_address)

#If left operand = 1 "true" return tu quad

if left\_operand == 1:

self.current\_quad = result\_address - 1

else:

#Else advance

self.current\_quad += 1

elif operator == "ENDPROC":

function.clear()

memory.local\_memory = local\_segment\_pointer\_list.pop()

memory.temporal\_memory = temporal\_segment\_pointer\_list.pop()

self.current\_quad = instruction\_number\_to\_back\_list.pop() + 1

elif operator == "ERA":

#Get the function's info and assign memory for its variables

function['function'] = self.funcDirectory.get\_function(l\_address)

function['memory'] = MemInterface()

parameters = 0

#Saves the local and temporaral variables of the function

self.request\_local\_addresses(function)

self.request\_temporal\_addresses(function)

self.current\_quad += 1

elif operator == "GOSUB":

# Stores the number of instruction we will return after the function execution ends

instruction\_number\_to\_back\_list.append(self.current\_quad)

# Change the local and temporal memory segments for the ones the in the function

local\_segment\_pointer\_list.append(memory.local\_memory)

temporal\_segment\_pointer\_list.append(memory.temporary\_memory)

memory.local\_memory = function['memory'].local\_memory

memory.temporary\_memory = function['memory'].temporary\_memory

self.current\_quad = result\_address - 1

elif operator == "PARAMETER":

# Gets the value of the parameter

left\_operand = memory.get\_content(l\_address)

#Get the address of the parameter

parameter\_address = function['function']['parameters']['parameter\_addresses'][parameters]

#Increment parameter counter

parameters += 1

#Edit the parameter\_address with the new parameter value

function['memory'].edit\_memory\_content(parameter\_address, left\_operand)

self.current\_quad += 1

elif operator == "RETURN":

left\_operand = memory.get\_content(l\_address)

memory.edit\_memory\_content(result\_address, left\_operand)

self.current\_quad += 1

elif operator == "VER":

#Get the index to get and the limits of the dimentioned variable

left\_operand = memory.get\_content(l\_address)

lower\_limit = r\_address

upper\_limit = result\_address

#Check if the value is within the limits of the dimentioned variable

if left\_operand >= lower\_limit and left\_operand < upper\_limit:

self.current\_quad += 1

else:

print("Index out of bounds")

elif operator =="READINPUT":

#Reads the input and stores the value in memory

var\_type = l\_address

user\_var = input()

try:

user\_var\_type = type(literal\_eval(user\_var))

if user\_var\_type == type(literal\_eval("1")):

user\_var\_type = "int"

elif user\_var\_type == type(literal\_eval("1.0")):

user\_var\_type = "float"

except(ValueError, SyntaxError):

if len(user\_var) == 1:

user\_var\_type = "char"

if var\_type == user\_var\_type:

memory.edit\_memory\_content(result\_address, user\_var)

else:

print("Expected type " + var\_type + " variable")

sys.exit()

self.current\_quad += 1;

# -----------------------------------------------------------------------------

# Juan Fernando and Jesus’ Programming Language

# funcDirectory.py

# Last edit: 18/11/2018

# -----------------------------------------------------------------------------

#Packages Imported

from structures.varTable import VarTable

from functions.printToOutputFile import print\_to\_output\_file

import json

import sys

#Class Name

class FuncDirectory():

#Base Constructor

def \_\_init\_\_(self):

self.list\_of\_functions = {}

#FuncDirectory attribute definition

def new\_function(self, identifier, return\_type, quad\_number = -1):

self.list\_of\_functions[identifier] = {

'identifier' : identifier,

'parameters' : {

'parameter\_types' : [],

'parameter\_addresses' : [],

'parameter\_counter' : 0

},

'local\_variables' : VarTable(),

'local\_variable\_counter' : {

'int' : 0,

'float' : 0,

'char' : 0,

},

'temporary\_variable\_counter' : {

'int' : 0,

'float' : 0,

'char' : 0,

'bool' : 0

},

'return\_type' : return\_type,

'return\_address' : -1,

'quad\_number' : quad\_number

}

#Verification of existance/non-existance prior to interacting with specified function

def function\_exists(self, identifier):

return identifier in self.list\_of\_functions.keys()

#Getter for the specified function

def get\_function(self, identifier):

if self.function\_exists(identifier):

return self.list\_of\_functions[identifier]

else:

print("Function " + identifier + " does not exist.")

return None

#Adds each parameter (one by one) as they are read in the syntactical analyzer for the specified function

def new\_function\_parameter(self, identifier, p\_type, p\_address):

current\_function = self.get\_function(identifier)

if current\_function is not None:

current\_function['parameters']['parameter\_types'].append(p\_type)

current\_function['parameters']['parameter\_addresses'].append(p\_address)

current\_function['parameters']['parameter\_counter'] += 1

else:

print("Function " + identifier + " does not exist.")

sys.exit()

#Function Directory's version of variable existance check from VarTable

def function\_variable\_exists(self, identifier, v\_identifier):

new\_function = self.get\_function(identifier)

if new\_function is not None:

if new\_function['local\_variables'].variable\_exists(v\_identifier):

return True

else:

return False

else:

print("Variable name already in use. (Come on, be creative.)")

sys.exit()

#Adds a new local variable (a non-dimensioned variable) to the specified function

#Local means "current active context" in my head in this context - Juan Fernando Ulloa

def new\_local\_variable(self, identifier, v\_type, v\_identifier, v\_address = 0):

new\_function = self.get\_function(identifier)

if new\_function is not None:

if not new\_function['local\_variables'].variable\_exists(v\_identifier):

new\_function['local\_variables'].new\_variable(v\_type, v\_identifier, v\_address)

new\_function['local\_variable\_counter'][v\_type] += 1;

else:

print("Variable name already in use. (Come on, be creative.)")

else:

print("Function " + identifier + " does not exist.")

sys.exit()

def set\_local\_variable\_dimension\_one(self, identifier, v\_type, v\_identifier, v\_address = 0):

current\_function = self.get\_function(identifier)

if current\_function is not None:

if current\_function['local\_variables'].variable\_exists(v\_identifier):

current\_function['local\_variables'].set\_dimension\_one(v\_identifier,

v\_address)

else:

print("Variable name already in use. (Come on, be creative.)")

sys.exit()

else:

print("Function " + identifier + " does not exist.")

sys.exit()

#Adds to the temporary variable counter

def new\_temporary\_variable(self, identifier, t\_type):

new\_function = self.get\_function(identifier)

if new\_function is not None:

new\_function['temporary\_variable\_counter'][t\_type] += 1

else:

print ("Function " + identifier + " does not exist.")

sys.exit()

#From this point on, these functions are either simple getters, or simple setters

#Type Getter for the specified function return type

def get\_function\_type(self, identifier):

current\_function = self.get\_function(identifier)

if current\_function is not None:

current\_function = current\_function['return\_type']

return current\_function

else:

print ("Function " + identifier + " does not exist.")

sys.exit()

#Parameter List of Types and Address Getter for the specified function

def get\_function\_parameters(self, identifier):

current\_function = self.get\_function(identifier)

if current\_function is not None:

return current\_function['parameters']

else:

print ("Function " + identifier + " does not exist.")

sys.exit()

#Variable Getter for single specified variable

def get\_function\_variable(self, identifier, v\_identifier):

current\_function = self.get\_function(identifier)

if current\_function is not None:

current\_variable = current\_function['local\_variables'].get\_variable(v\_identifier)

if current\_variable is not None:

return current\_variable

else:

#print ("Variable " + str(current\_variable) + " was not declared in this scope.")

return None

else:

print ("Function " + identifier + " does not exist.")

sys.exit()

#Function Quadruple Getter for specified function

def get\_quad\_number(self, identifier):

new\_function = self.get\_function(identifier)

if new\_function is not None:

return new\_function['quad\_number']

else:

print ("Function " + identifier + " does not exist.")

sys.exit()

#Function Quadruple Setter for specified function

def set\_quad\_number(self, identifier, quad\_number):

current\_function = self.get\_function(identifier)

if current\_function is not None:

current\_function['quad\_number'] = quad\_number

else:

print ("Function " + identifier + " does not exist.")

sys.exit()

#Function Address Setter for specified function

def set\_function\_address(self, identifier, address):

current\_function = self.get\_function(identifier)

if current\_function is not None:

current\_function['return\_address'] = address

else:

print ("Function " + identifier + " does not exist.")

sys.exit()

def print\_directory(self):

for function, properties in self.list\_of\_functions.items():

print\_to\_output\_file("\n")

print\_to\_output\_file("function : " + str(function))

for prop, value in properties.items():

if isinstance(value, VarTable):

print\_to\_output\_file(" " + str(prop) + " : " +

json.dumps(value.list\_of\_variables, indent=4))

elif isinstance(value, dict):

print\_to\_output\_file(" " + str(prop) + " : " +

json.dumps(value, indent=4))

else:

print\_to\_output\_file(" " + str(prop) + " : " + str(value))

print\_to\_output\_file("-" \* 80)

# Quick Reference Manual (Also in the environment)

Hi! This is the Quick Reference Manual for the Programming Language Nansus!

For starters, some important things you must remember while you are programming in Nansus:

1. You must always include your program name in the format “program name;” at the beginning of the document when you are preparing your code. The name is yours to choose.
2. At this point, you can declare global variables and functions in any order you like until you reach a new context (within a function or the main) but remember that you can only declare variables at the start within contexts!
3. Another important thing to remember, is that variables in Nansus do not allow you to pre-initialize them at the time of declaration, they will by default be initialized to 0, 0.0 and ‘’ respectively for each of the three types, do not forget to assign your own values!
4. Do not forget to write the word main when opening your main segment, or your code will crash.
5. One you are all done, remember that the main must be closed with a; after closing your braces, otherwise the code will not compile!

Now that we have gone over the basics, let’s divide this into some more specific sections:

Operators:

+: For sum of binary operations

-: For difference of binary operations

\*: For product of binary operations

/: For quotient of binary operations

>: For comparison in binary operations (Greater Than)

<: For comparison in binary operations (Less than)

>=: For comparison in binary operations (Greater than or Equal

<=: For comparison in binary operations (Less than or Equal)

==: For comparison in binary operations (Equal)

!=: For comparison in binary operations (Not Equal)

&&: For logic in binary operations (And operator)

||: For logic in binary operations (Or Operator)

Print(a): Prints value in the variable

readInput(a): Reads from input and stores in variable a

jedo.specialfunctions(): Will execute special drawing functions (more ahead).

Typical Format for each operation:

Assignment:

1. A = 0;
2. A = b + c;
3. A = function nando(1, 2)

Conditions:

1. While (a < b) {}
2. Do {} while (b > 8 && c < 10)
3. If (x == a)
   1. { a = b;}
4. elseif (x < a)
5. {b = b + a;}
6. Else
7. { print a;}

You can, of course use different formats, as long as they don’t deviate from this standard.

For function definition:

Function name (type name, type name){ a = 1; }

You can have multiple declarations, but again, remember that you can only declare variables at the beginning of each context.

For the special functions we have:

* jedo.circle(1) : You can indicate the size of the radius for the circle to draw.
* jedo.square(1): You can indicate the size of each of the sides of a square to draw.
* jedo.Rectangle(1, 2): You can indicate the size of length and width to draw a rectangle.
* jedo.drawDot(1, “blue”): You can specify the size of the dot and color in the second slot.
* jedo.arch(1, “left”): You can indicate the size of the arch and the direction (left of right) to which to draw it.
* jedo.triangle (1): You can indicate the height of the triangle to draw.
* jedo.forward(1): You can indicate the distance for jedo (the cursor) to move forward.
* jedo.forward(1): You can indicate the distance for jedo to move backwards (based on faced direction.
* jedo.turnLeft(1): You can indicate the angle at which you wish the cursor to move (in degrees) in the left direction.
* jedo.turnRight(1): You can indicate the angle at which you wish the cursor to move (in degrees) in the right direction.
* jedo.color(“red”): You can indicate with a string what color you wish to display on jedo (this will depend on availability)
* jedo.thickness(1):You can indicate the thickness of the line that jedo draws as it moves.
* jedo.startpen(): Restarts drawing the line behind jedo.
* jedo.create(): Creates the window for the jedo graphical output
* jedo.stoppen(): Ceases drawing lines behind jedo (jedo still moves)
* jedo.startfill(): Sets up the jedo drawing to know that it will be filled at some point of the figure.
* jedo.fillshape(“red”): This will paint the set-up jedo drawing from before and change it to a color that you specify with a string.
* jedo.stopfill(): Stop setting up figures to be colored.
* Jedo.restart(): Clears the screen and begins jedo at the center of the screen once again.

With all this data, you should be prepared to explore the possibilities of the Nansus Programming Language! For further information, look for the full documentation of the Language, Happy Coding!