Instituto Tecnológico y de Estudios Superiores de Monterrey

Proyecto Final - Diseño de Compilador

JADE

Diseño de Compiladores - Grupo 1

Equipo

Ana Carolina

Ana Lizbeth Zermeño Torres
A00824913

Ana Carolina Arellano Álvarez
A01650945

Noviembre 22, 2022

Index

a). PROJECT DESCRIPTION

a.1) Purpose and scope of the project

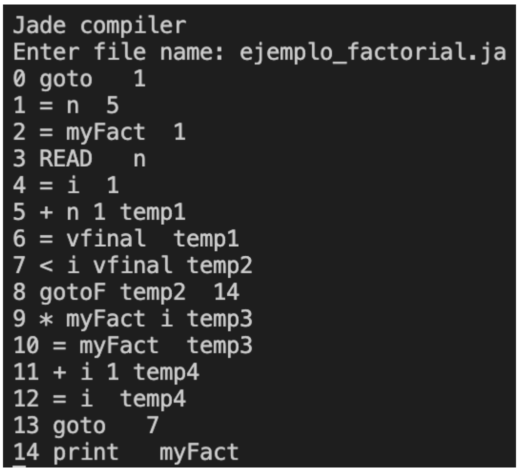
Our goal by developing Jade is to create a new programming language that based on our own experiences we would consider ideal to teach how to program to people that do not necessarily have a tech background, including kids. Although the main purpose is a product for people of all ages, different skill sets, etc., we want to provide a programming language that can support multiple operations that programmers need. We will achieve this by creating a clear and explicit grammar that will guide the user through their coding process by creating fluid structures.

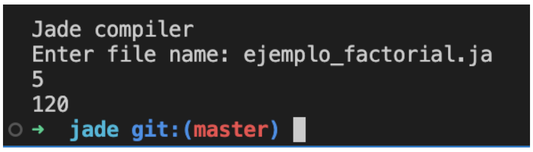
a.2) Requirement analysis and principal Test Cases description

JADE is a programming language that will support math expressions (arithmetic, logical and relational), statements (assignment, cycles, conditions, read, write), modules, arrays and object oriented programming. Below are representative examples that allow you to see the main functionalities and syntax of the program.

Test case 1 - Factorial Iterative

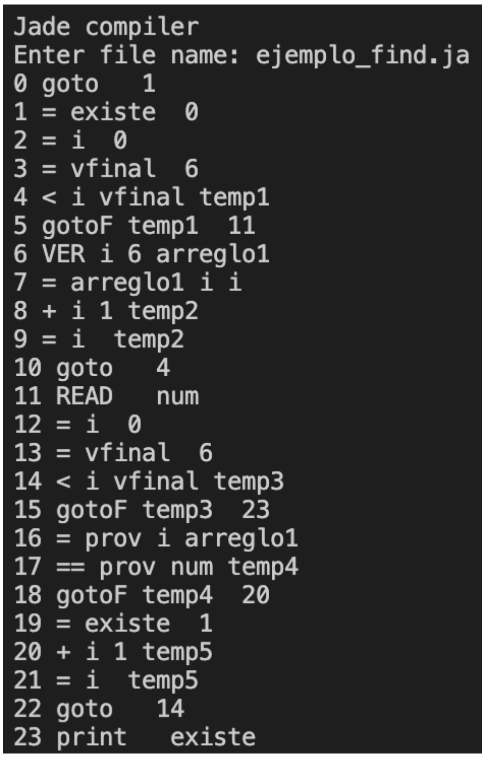
Description	User enters a number and the program calculates the Factorial of said number
Code	**Generated Quadruples**

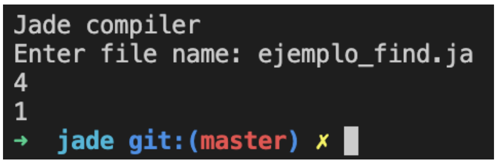
program test12{
 	
main{
var int n;
assign n = 5;
var int myFact;
assign myFact = 1;
var int i;
 	
read n;
 	
for(i=1 : n+1){
assign myFact = myFact * i;
}
print(myFact);
}
	
	Jade compiler
	Enter file name: ejemplo_factorial.ja
	0 goto 1
	1 = n 5
	2 = myFact 1
	3 READ n
	4 = i 1
	5 + n 1 temp1
	6 = vfinal temp1
	7 < i vfinal temp2
	8 gotoF temp2 14
	9 * myFact i temp3
	10 = myFact temp3
	11 + i 1 temp4
	12 = i temp4
	13 goto 7
	14 print myFact

}	
Expected result - input: 5	**Final Result**
120	 Jade compiler Enter file name: ejemplo_factorial.ja 5 120 ○ → jade git:(master)

Test case 2 - Operation with arrays

Description	Find element in a defined array, returns 0 if not found or return 1 if element is found within the array
Code	**Generated Quadruples**

program find {
    main {
        var array int arreglo1 = [6];
        var int i;
        var int num;
        var int prov;
        var int existe;
        assign existe = 0;
 	
        for(i = 0 : 6){
            assign array arreglo1[i] = i;
        }
 	
        read num;
 	
        for(i = 0 : 6){
            assign prov = arreglo1[i];
            if(prov == num){
                assign existe = 1;
            }
        }
        print(existe);
    }
}
	
	Jade compiler
	Enter file name: ejemplo_find.ja
	0 goto 1
	1 = existe 0
	2 = i 0
	3 = vfinal 6
	4 < i vfinal temp1
	5 gotoF temp1 11
	6 VER i 6 arreglo1
	7 = arreglo1 i i
	8 + i 1 temp2
	9 = i temp2
	10 goto 4
	11 READ num
	12 = i 0
	13 = vfinal 6
	14 < i vfinal temp3
	15 gotoF temp3 23
	16 = prov i arreglo1
	17 == prov num temp4
	18 gotoF temp4 20
	19 = existe 1
	20 + i 1 temp5
	21 = i temp5
	22 goto 14
	23 print existe

Expected result - input: 4	**Final Result**
1	 Jade compiler Enter file name: ejemplo_find.ja 4 1 → jade git:(master) ✗

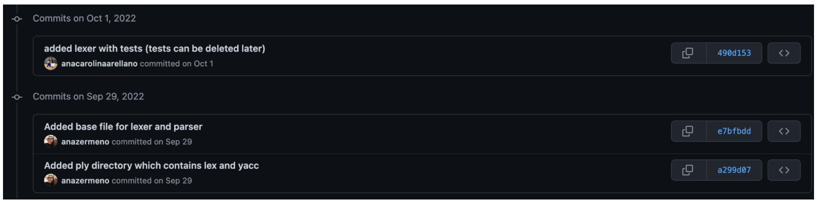
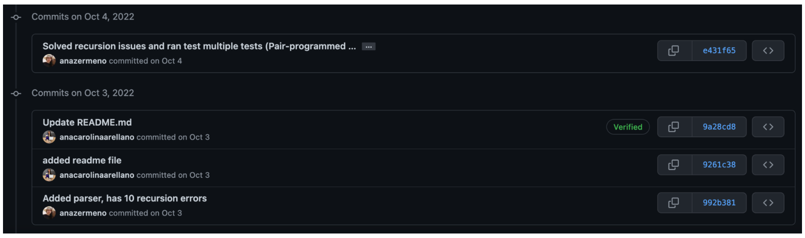
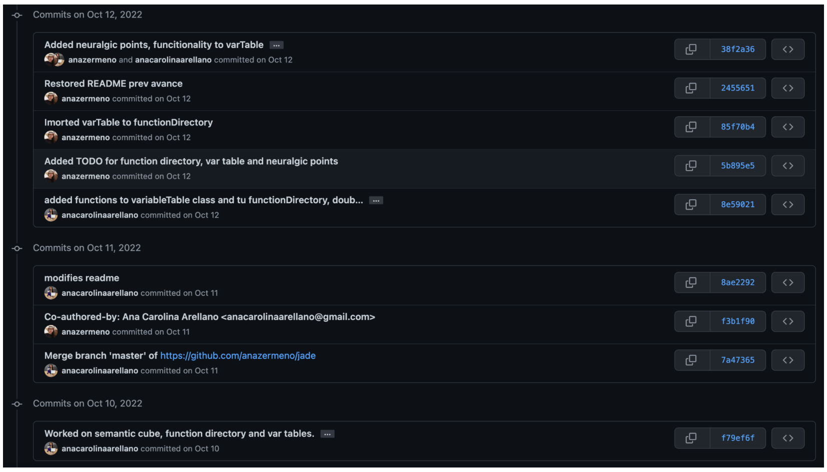
a.3) Project development process

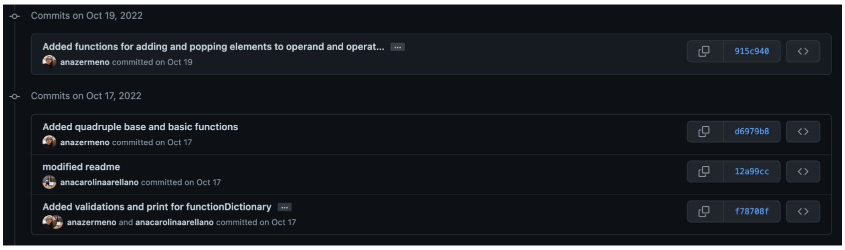
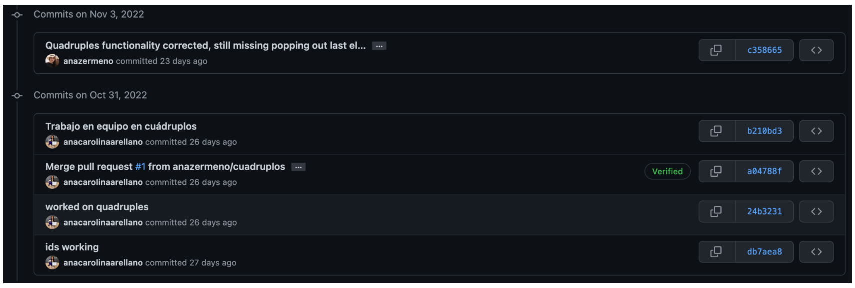
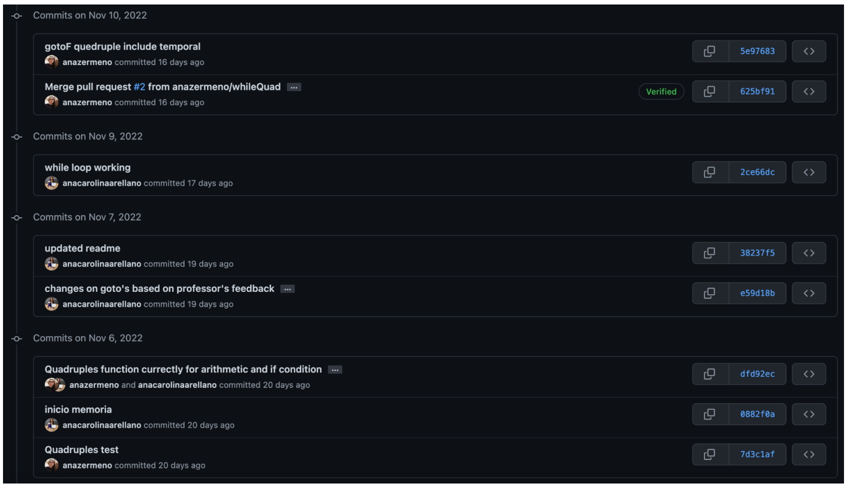
Expected milestones to be achieved per week

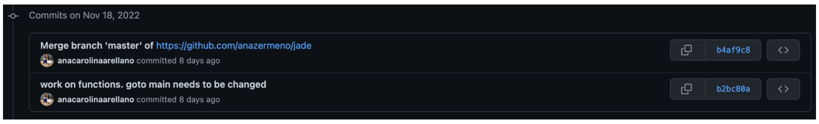
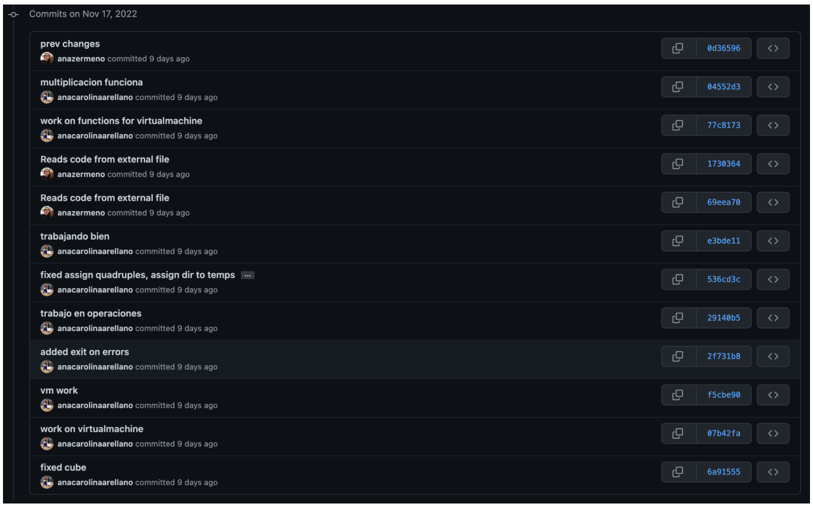
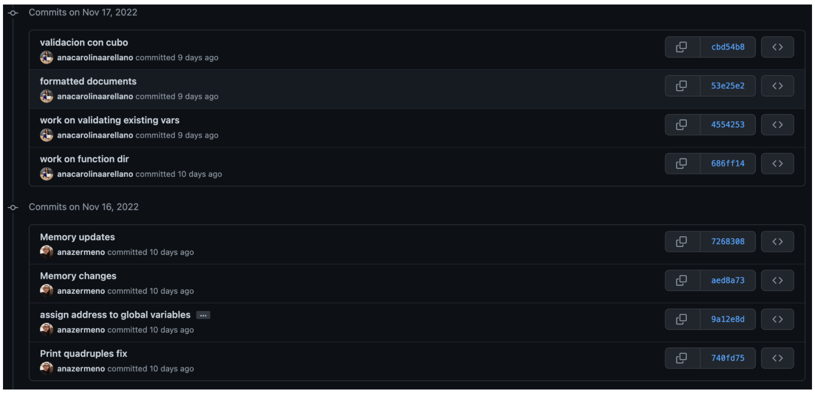
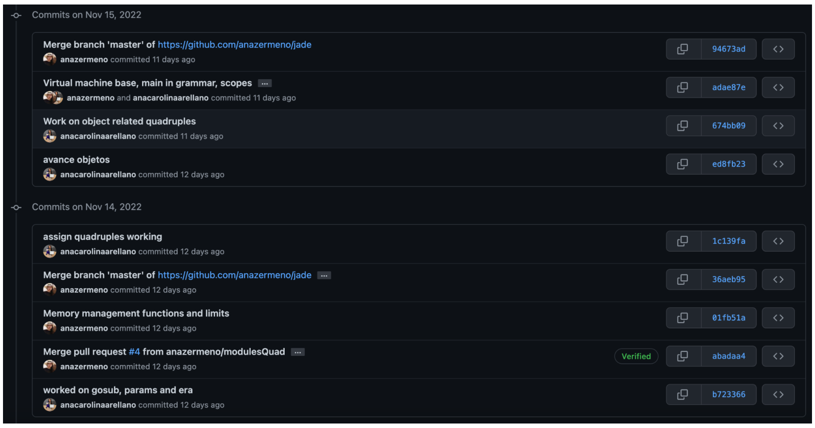
Week	Project updates
1 September 26 - 30	- Defined project scope - Selection of project tools and environment - General characteristics - Tokens and grammatical diagrams
2 October 3 - 7	- Generated repository - Updated tokens and grammatical diagrams - Functional first iteration of lexer and parser
3 October 10 - 14	- Function directory - Var table fundaments - Semantic cube
4 October 17 - 21	- Var table implementation - Function directory implementation
October 24 - 28	**SEMANA i**
5 October/November 31 - 4	- For loop quadruples - If statement quadruples - If else statement quadruples
6 November 7 - 11	- While quadruples - Documentation starts
7 November 14 - 18	- Function quadruples - Semantic cube fix - Virtual memory implementation - Virtual machine implementation

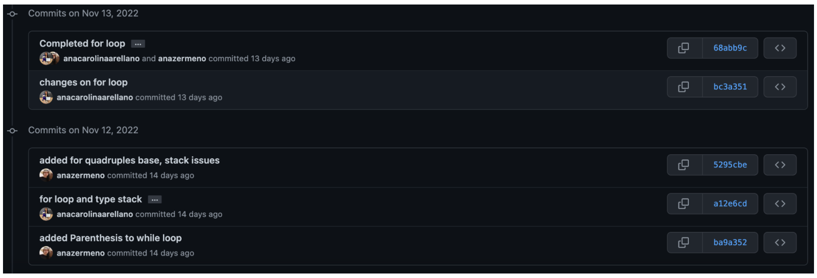
Binnacle and commit log

Week	Project updates
1 September 26 - 30	Through this week we made decisions regarding the tools we were going to use for the development of the project, we decided to develop it using PLY in python. During this week we also delimited the scope of the project, the main objectives, team logistics, grammar and tokens in a document and we also had time to create the repository and pass a first version of our grammatic. It is important to mention that we held 2 meetings in person during that week and split the tasks when it came to transferring what was agreed to the project repository. We also

	tested said grammatic and it passed the basic tests we included at the moment.
Commits Week 1	 Commits on Oct 1, 2022 added lexer with tests (tests can be deleted later) anacarolinaarellano committed on Oct 1 · 490d153 Commits on Sep 29, 2022 Added base file for lexer and parser anazermeno committed on Sep 29 · e7bfbdd Added ply directory which contains lex and yacc anazermeno committed on Sep 29 · a299d07
2 October 3 - 7	During week 2 we tested more complex test cases and started to notice some recursion errors which were solved by the second day of work by adjusting some of the previously stated grammatical rules, this led to our first working version of our parser.
Commits Week 2	 Commits on Oct 4, 2022 Solved recursion issues and ran test multiple tests (Pair-programmed ... ··· anazermeno committed on Oct 4 · e431f65 Commits on Oct 3, 2022 Update README.md anacarolinaarellano committed on Oct 3 · Verified · 9a28cd8 added readme file anacarolinaarellano committed on Oct 3 · 9261c38 Added parser, has 10 recursion errors anazermeno committed on Oct 3 · 992b381
3 October 10 - 14	With our parser complete and working we were all set to start creating the first versions of the data structures to store the basic required information during the compilation process, such as the var table and the program (function) directory, we designed the data structures as objects with their corresponding methods to allow us easier access to attributes in order to add or delete when needed. With these structures created we also added the first neuralgic points in which we primarily focused on solving expressions, adding functions into the program directory and their corresponding variables to the associated var table. Finally we developed our first version of the semantic cube.
Commits Week 3	 Commits on Oct 12, 2022 Added neuralgic points, funcitionality to varTable ··· anazermeno and anacarolinaarellano committed on Oct 12 · 38f2a36 Restored README prev avance anazermeno committed on Oct 12 · 2455651 Imorted varTable to functionDirectory anazermeno committed on Oct 12 · 85f70b4 Added TODO for function directory, var table and neuralgic points anazermeno committed on Oct 12 · 5b895e5 added functions to variableTable class and tu functionDirectory, doub... ··· anacarolinaarellano committed on Oct 12 · 8e59021 Commits on Oct 11, 2022 modifies readme anacarolinaarellano committed on Oct 11 · 8ae2292 Co-authored-by: Ana Carolina Arellano <anacarolinaarellano@gmail.com> anazermeno committed on Oct 11 · f3b1f90 Merge branch 'master' of https://github.com/anazermeno/jade anacarolinaarellano committed on Oct 11 · 7a47365 Commits on Oct 10, 2022 Worked on semantic cube, function directory and var tables. ··· anacarolinaarellano committed on Oct 10 · f79ef6f
4 October 17 - 21	On week 4 we continued with the implementation of the function directory and var table by creating more functions to manipulate its elements and started to create the first functions required to create

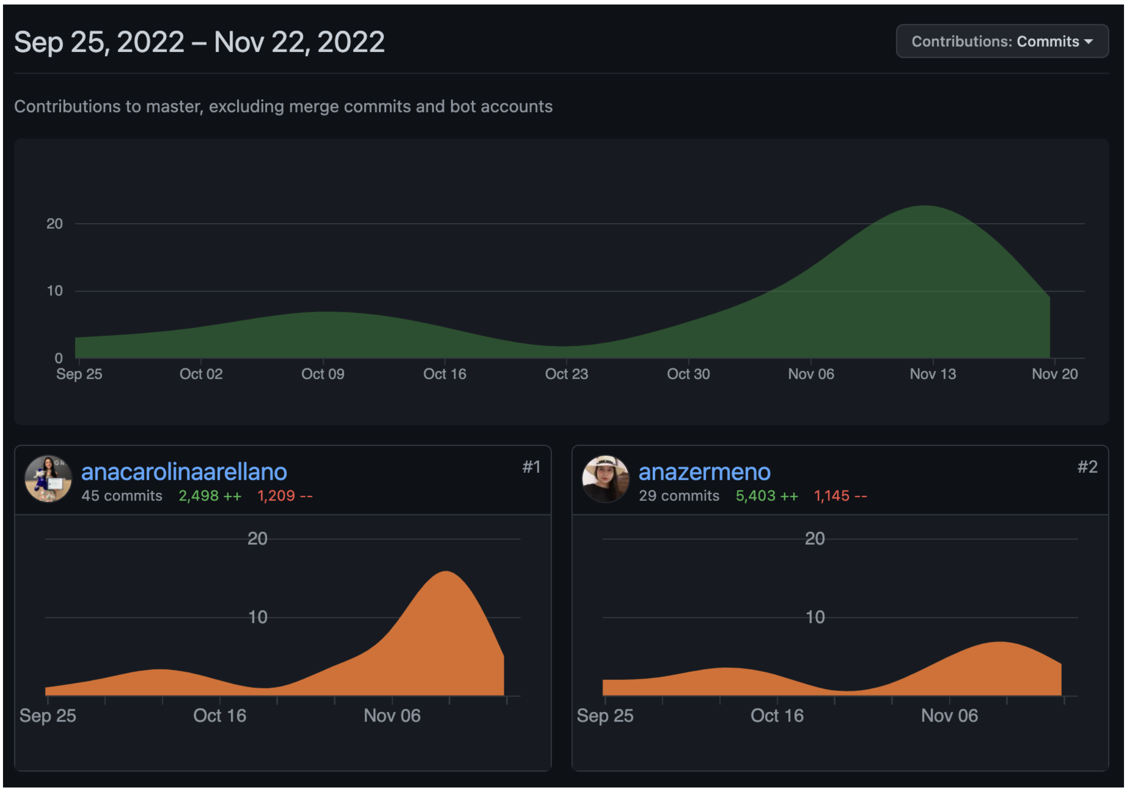
	quadruples for expressions and arithmetic sequences, we developed said quadruple functions in a peer programming session and solved bugs individually through the week.
Commits Week 4	 Commits on Oct 19, 2022 Added functions for adding and popping elements to operand and operat... ··· 915c940 anazermeno committed on Oct 19 Commits on Oct 17, 2022 Added quadruple base and basic functions d6979b8 anazermeno committed on Oct 17 modified readme 12a99cc anacarolinaarellano committed on Oct 17 Added validations and print for functionDictionary ··· f78708f anazermeno and anacarolinaarellano committed on Oct 17
October 24 - 28	**SEMANA i**
5 October/November 31 - 4	In week five we kept working on quadruple functions for conditional statements and corrected previous mistakes made for the arithmetic and expressions quadruple. During this week we also tested the data structures which were previously implemented.
Commits Week 5	 Commits on Nov 3, 2022 Quadruples functionality corrected, still missing popping out last el... ··· c358665 anazermeno committed 23 days ago Commits on Oct 31, 2022 Trabajo en equipo en cuádruplos b210bd3 anacarolinaarellano committed 26 days ago Merge pull request #1 from anazermeno/cuadruplos ··· Verified a04788f anacarolinaarellano committed 26 days ago worked on quadruples 24b3231 anacarolinaarellano committed 26 days ago ids working db7aea8 anacarolinaarellano committed 27 days ago
6 November 7 - 11	Through this week we worked exclusively on quadruples for more complex data structures, such as quadruples for for loops and while loops and conditional statements with else if and else. We also tested everything we had developed so far and concluded that the current version was able to handle the stated scenarios. Finally we also started writing the structure for the project documentation and then created the base for the virtual memory distribution.
Commits Week 6	 Commits on Nov 10, 2022 gotoF quedruple include temporal 5e97683 anazermeno committed 16 days ago Merge pull request #2 from anazermeno/whileQuad ··· Verified 625bf91 anazermeno committed 16 days ago Commits on Nov 9, 2022 while loop working 2ce66dc anacarolinaarellano committed 17 days ago Commits on Nov 7, 2022 updated readme 38237f5 anacarolinaarellano committed 19 days ago changes on goto's based on professor's feedback ··· e59d18b anacarolinaarellano committed 19 days ago Commits on Nov 6, 2022 Quadruples function currectly for arithmetic and if condition ··· dfd92ec anazermeno and anacarolinaarellano committed 20 days ago inicio memoria 0882f0a anacarolinaarellano committed 20 days ago Quadruples test 7d3c1af anazermeno committed 20 days ago
7	As we were getting closer to the deadline and the classes finished we

November 14 - 18	had more space in our schedule to work daily on the project. We made lots of progress during the week, mainly in the remaining quadruples for functions, assign, read and print quadruples that were not implemented yet. Virtual memory implementation and assignment to variables was completed when working on the virtual machine. During this week we also realized that type validation was not complete in certain steps through the neuralgic points so it was also implemented.
Commits Week 7	 Commits on Nov 18, 2022 Merge branch 'master' of https://github.com/anazermeno/jade anacarolinaarellano committed 8 days ago b4af9c8 work on functions. goto main needs to be changed anacarolinaarellano committed 8 days ago b2bc80a  Commits on Nov 17, 2022 prev changes anazermeno committed 9 days ago 0d36596 multiplicacion funciona anacarolinaarellano committed 9 days ago 04552d3 work on functions for virtualmachine anacarolinaarellano committed 9 days ago 77c8173 Reads code from external file anazermeno committed 9 days ago 1730364 Reads code from external file anazermeno committed 9 days ago 69eea70 trabajando bien anacarolinaarellano committed 9 days ago e3bde11 fixed assign quadruples, assign dir to temps … anacarolinaarellano committed 9 days ago 536cd3c trabajo en operaciones anacarolinaarellano committed 9 days ago 29140b5 added exit on errors anacarolinaarellano committed 9 days ago 2f731b8 vm work anacarolinaarellano committed 9 days ago f5cbe90 work on virtualmachine anacarolinaarellano committed 9 days ago 07b42fa fixed cube anacarolinaarellano committed 9 days ago 6a91555  Commits on Nov 17, 2022 validacion con cubo anacarolinaarellano committed 9 days ago cbd54b8 formatted documents anacarolinaarellano committed 9 days ago 53e25e2 work on validating existing vars anacarolinaarellano committed 9 days ago 4554253 work on function dir anacarolinaarellano committed 10 days ago 686ff14 Commits on Nov 16, 2022 Memory updates anazermeno committed 10 days ago 7268308 Memory changes anazermeno committed 10 days ago aed8a73 assign address to global variables … anazermeno committed 10 days ago 9a12e8d Print quadruples fix anazermeno committed 10 days ago 740fd75  Commits on Nov 15, 2022 Merge branch 'master' of https://github.com/anazermeno/jade anazermeno committed 11 days ago 94673ad Virtual machine base, main in grammar, scopes … anazermeno and anacarolinaarellano committed 11 days ago adae87e Work on object related quadruples anacarolinaarellano committed 11 days ago 674bb09 avance objetos anacarolinaarellano committed 12 days ago ed8fb23 Commits on Nov 14, 2022 assign quadruples working anacarolinaarellano committed 12 days ago 1c139fa Merge branch 'master' of https://github.com/anazermeno/jade … anazermeno committed 12 days ago 36aeb95 Memory management functions and limits anazermeno committed 12 days ago 01fb51a Merge pull request #4 from anazermeno/modulesQuad … anazermeno committed 12 days ago Verified abadaa4 worked on gosub, params and era anacarolinaarellano committed 12 days ago b723366

	 Commits on Nov 13, 2022 Completed for loop anacarolinaarellano and anazermeno committed 13 days ago 68abb9c changes on for loop anacarolinaarellano committed 13 days ago bc3a351 Commits on Nov 12, 2022 added for quadruples base, stack issues anazermeno committed 14 days ago 5295cbe for loop and type stack anacarolinaarellano committed 14 days ago a12e6cd added Parenthesis to while loop anazermeno committed 14 days ago ba9a352

Commit log summary

Our commit log is extensive, for that reason we decided to attach the main development insights and contributions of the team members during the project in the repository starting on September 25th to present day November 22nd, for further details of the weekly progress consult the table above.



Sep 25, 2022 – Nov 22, 2022

Contributions: Commits

Contributions to master, excluding merge commits and bot accounts

anacarolinaarellano #1
45 commits 2,498 ++ 1,209 --

anazermeno #2
29 commits 5,403 ++ 1,145 --

Reflections

Ana Lizbeth Zermeño

Through the development of this project I was able to acquire a better understanding of some of the topics seen during my career and learn many other new things, this has been by far one of the most complex projects I've worked on in which I learned the importance of time management and creating a solid development plan (regarding data structures and code architecture). At some points I felt overwhelmed by all the parts that shape the compiler but at the same time I found this really interesting despite the fact that it was challenging. In the end even if this compiler isn't the final product I would have liked to present, it makes me feel proud about how much I have learned this semester and the work I got to do with my teammate Ana Carolina, it was nice sharing peer coding sessions and finding new ways to solve problems I wouldn't have thought of.

Ana Carolina Arellano

I consider that this project was very challenging in many ways, from understanding the theory seen in class, all the way to generating the intermediate code and executing it. That doesn't take away the good moments, such as generating the quadruples as expected, first the "simple" ones and then moving onto nonlinear statements, modules and arrays. Furthermore, when we managed to make the virtual machine produce the expected output I felt extremely happy and proud of what we had accomplished, because at the start of my career I would have considered accomplishing that almost impossible.

Nonetheless, there were some things that could have been improved if we had more time, because fixing things at this stage of the development meant potentially breaking things that were already working and we couldn't risk that.

Even though this was a very challenging project and that it could be improved, I feel very proud of what me and my teammate accomplished because we spent a long time in its development and it helped us to broaden our knowledge about the whole process of defining a programming language and executing it.

Signatures

Ana Carolina

Ana Lizbeth Zermeño Torres
A00824913

Ana Carolina Arellano Álvarez
A01650945

b). LANGUAGE DESCRIPTION

b.1) Language name: JADE

b.2) Generic description of the main features of the program

JADE is a highly explicit and descriptive language that, through keywords, helps to maintain clarity of the sections within the code and the actions to be executed through the program in order to fulfill developer learners objectives. It offers arithmetic operations, logic operands, arrays and matrices, while and for loops, the possibility of creating programs, classes and functions and is an object oriented programming language.

b.3) Common errors in compilation and execution

Compilation Errors	
Error	**Message**
Type-mismatch, variable type is not compatible with valued to be assigned	“Error: assigned value does not match variable type”
Using a variable that has not been previously declared	“Error: variable is not declared yet”
Trying to do an operation of any sort and at least one of the variables has not been declared previously	“Error: variable is not declared for use in expressions yet”
Trying to receive an input and the variable is not declared yet	“Error: input variable is not declared yet”
Using a function that has not been previously declared	“Error: function is not defined yet”
During function call parameter is not valid because it does not exist in its corresponding variable table	“Error: parameters are incorrect”
Using a variable that has not been previously declared as a controller variable in a for loop	“Error: cycle controller variable is not declared yet”
Using a variable that is not int to control for loop cycle	“Error: cycle controller variable is not int type”

Trying to do an operation and the operand types combination is not valid	“Error: type mismatch”
Trying to an operation and the operand is invalid	“Error: operator type incorrect”
Trying to assign values to a variable that has not been previously declared	“Error: variable is not declared for use yet”
Trying to assign values to array that has not been defined	“Error: array is not declared yet”
Trying to assign values to an array and indicating a negative value as the index.	“Error: index size must be equal or greater than 0”
trying to assign values to a matrix and the index is out of limits	“Error: index out of limits”
Displayed when syntax error is detected	“Failed”
Execution Errors	
Error	**Message**
Error during division either because it is by zero or the variable doesn’t have an assigned value yet	“Error: unable to make division, either because it is by 0 or the variable does not have an assigned value yet”
Error during operation with arrays when given index is out of limits	“Error: size out of limits”
Error while printing a variable because it does not have a value assigned yet	“Error: variable to be printed does not have an assigned value yet”

c). COMPILER DESCRIPTION

c.1) Computer equipment, language and special utilities used in the development of the project

The project was developed using the programming language python, with the support of Lex & Yacc to implement the language grammatical and carry out the lexical and syntax analysis. The computer equipment employed during the development were laptops with MacOS operating system.

c.2) Description of Lexical Analysis

Language tokens

Expression	Token
“print”	PRINT
“class”	CLASS
“for”	FOR
“while”	WHILE
“return”	RETURN
‘[a-zA-Z_][a-zA-Z0-9]*’	ID
‘[a-zA-Z_][a-zA-Z0-9]*’	CLASSID
‘[a-zA-Z_][a-zA-Z0-9]*’	OBJID
‘[0-9]+’	CTEINT
‘[0-9]+.[0-9]+’	CTEFLOAT
[a-zA-Z]	CTESTRING
“;”	SEMICOLON
“:”	COLON
“,”	COMMA

“.”	DOT
“+”	PLUS
“-”	MINUS
“*”	MULT
“/”	DIV
“(“	OPARENTHESIS
“)”	CPARENTHESIS
“[“	OBRACKET
“]”	CBRACKET
“{”	OCURLY
“}”	CCURLY
“<”	LESSTHAN
“>”	GREATERTHAN
“!=”	NOTEQUAL
“==”	ISEQUAL
“=”	EQUAL

Reserved Words	
“program”	PROGRAM
“var”	VAR
‘array’	ARRAY
‘matrix’	MATRIX
“assign”	ASSIGN
“fun”	FUN
“return”	RETURN
“int”	INT
“float”	FLOAT
“bool”	BOOLEAN
“mod”	MOD
“void”	VOID
“if”	IF
“else”	ELSE
“elseif”	ELSEIF
“in”	IN
“or”	OR
“and”	AND
“read”	READ

c.3) Syntax analysis description

Formal grammar used to represent syntactic structures

```
<PROGRAM> ::= PROGRAM ID OCURLY <BLOCK> <MAIN> CCURLY

<MAIN> ::= MAIN OCURLY <BLOCK2> CCURLY

<BLOCK> ::= <VAR> <BLOCK>
            | <FUN> <BLOCK>
            | <STATEMENT> <BLOCK>
            | ε

<BLOCK2> ::= <STATEMENT> <BLOCK2>
            | <VAR> <BLOCK2>
            | ε

<STATEMENT> ::= <ASSIGN>
            | <WRITE>
            | <READ>
            | <CONDITION>
            | <FORLOOP>
            | <WHILELOOP>
            | <FUNCALL>
            | <CLASS>

<VAR> ::= VAR INT ID SEMICOLON
            | VAR FLOAT ID SEMICOLON
            | VAR BOOL ID SEMICOLON

<VARARRAY> ::= VAR ARRAY <TYPE> ID EQUAL OBRACKET CTEINT CBRACKET SEMICOLON

<VARMATRIX> ::= VAR MATRIX <TYPE> ID EQUAL OBRACKET CTEINT CBRACKET OBRACKET CTEINT CBRACKET SEMICOLON

<ASSIGN> ::= ASSIGN <ASSIGN2> SEMICOLON
```

<ASSIGN2> ::= ID EQUAL <ASSIGN3>

<ASSIGN3> ::= <ADDOPERAND>
| <EXPRESSION>

<ASSIGNARRAY> ::= ASSIGN ARRAY ID OBRACKET CTEINT CBRACKET EQUAL <ASSIGN3>
| ASSIGN ARRAY ID OBRACKET <EXPRESSION> CBRACKET EQUAL <ASSIGN3>

<ASSIGNMATRIX> ::= ASSIGN MATRIX ID OBRACKET CTEINT CBRACKET OBRACKET CTEINT CBRACKET EQUAL <EXPRESSION>

<WRITE> ::= PRINT OPARENTHESIS <WRITE2> CPARENTHESIS SEMICOLON

<WRITE2> ::= EXPRESSION <WRITE3>
| CTESTRING <WRITE3>

<WRITE3> ::= COMMA <WRITE2>
| ε

<READ> ::= READ ID

<CONDITION> ::= IF <CONDITION2> <END>

<CONDITION2> ::= OPARENTHESIS <EXPRESSION> CPARENTHESIS OCURLY <BLOCK> CCURLY <CONDITION3>

<CONDITION3> ::= ELSEIF <CONDITION2>
| ELSE OCURLY <BLOCK> CCURLY
| ε

<FORLOOP> ::= FOR OPARENTHESIS ID EQUAL <EXPRESSION> COLON <EXPRESSION> CPARENTHESIS OCURLY <STATEMENT> CCURLY

<WHILELOOP> ::= WHILE OPARENTHESIS <G_EXP> CPARENTHESIS OCURLY <BLOCK> CCURLY

```
<FUN> ::= FUN <FUN_ADDFUN> OPARENTHESIS <FUNPARAMS> CPARENTHSIS
OCURLY <BLOCK> CCURLY

<FUN_ADDFUN> ::= <TYPE> ID
                      | VOID ID

<FUNPARAMS> ::= <TYPE> ID <FUNPARAMS2>
                      | ε

<FUNPARAMS2> ::= COMMA <FUNPARAMS>
                      | ε

<FUNCALL> ::= ID OPARENTHESIS <FUNCALL2> CPARENTHESIS SEMICOLON

<FUNCALL2> ::= <PARAMS>
              | ε

<PARAMS> ::= ID <PARAMS2>
              | CTEINT <PARAMS2>
              | CTEFLOAT <PARAMS2>
              | ε

<PARAMS2> ::= COMMA <PARAMS>
              | ε

<EXPRESSION> ::= <T_EXP> <EXPRESSION2>

<EXPRESSION2> ::= OR <EXPRESSION>
              | ε

<T_EXP> ::= <G_EXP> <T_EXP2>

<T_EXP2> ::= AND <T_EXP>
              | ε

<G_EXP> ::= <M_EXP> <G_EXP2>
```

```
<G_EXP2> ::= GREATERTHAN <M_EXP>
             | LESSTHAN <M_EXP>
             | ISEQUAL <M_EXP>
             | NOTEQUAL <M_EXP>

<M_EXP> ::= <T> <M_EXP2>

<M_EXP2> ::= PLUS <M_EXP>
             | MINUS <M_EXP>
             | ε

<T> ::= <F> <T2>

<T2> ::= MULT <T>
       | DIV <T>
       | ε

<F> ::=  OPARENTHESIS <EXPRESSION> CPARENTHESIS
       | <ADDOPERAND>

<ADDOPERAND> ::= <VARVALUE>
                  | ID

<VARVALUE> ::= CTEINT
             | CTEFLOAT
             | CTESTRING

<TYPE> ::= INT
        | FLOAT
        | BOOL

<CLASS> ::= CLASS ID OCURLY <CLASS2> <CLASS3> CCURLY

<CLASS2> ::= <ATTRIBUTE> <CLASS2>
            | ε
```

<CLASS3> ::= <METHOD> <CLASS3>
| ε

<OBJ_CONSTRUCTOR> ::= CLASSID ID OPARENTHESIS <PARAMS> CPARENTHESIS OCURLY <BLOCK> CCURLY

<OBJ_DECLARATION> ::= CLASSID ID OPARENTHESIS <PARAMS> CPARENTHESIS SEMICOLON

OBJ_METHOD_ACCESS ::= OBJID DOT <METHOD> SEMICOLON

OBJ_ATTR_ACCESS ::= OBJID DOT <ATTRIBUTE> SEMICOLON

METHOD ::= <FUN>

ATTRIBUTE ::= <VAR>

RETURN ::= RETURN ID SEMICOLON

c.4) Intermediate code generation and semantic analysis

Operation code

Operation code	Action description
PRINT	Displays expressions and/or strings in the console.
READ	Allows to get input from the user by relating such input to a variable.
GOTO	Indicates the virtual machine to move into a certain quadruple indicated within the operation code.
GOTOF	Indicates the virtual machine to move into a certain quadruple indicated within the operation code if the condition is false.
ENDFUN	Indicates the virtual machine that it has reached the end of a function.
RETURN	Indicates the virtual machine the result

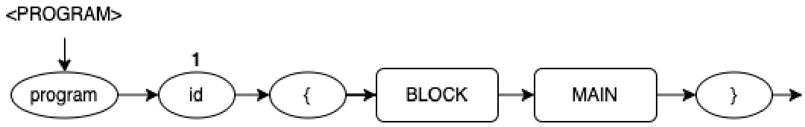
	of a function that has return value.
ERA	Indicates the virtual machine to prepare resources because a function is being called.
PARAM	Indicates the virtual machine to receive the parameter that will be fed to the indicated function.
GOSUB	Indicates the virtual machine to update the instruction pointer to the indicated function.
VER	Verify index for an array or matrix when doing operations with this structures.

Virtual addresses associated with code elements

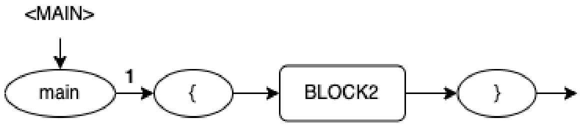
According to the scope and type of the variables we assigned direction ranges, for each category each variable type has 2000 direction spaces which correspond to the ones displayed in the following table.

Scope	Type	First direction	Final direction
Global	Int	1000	2999
Global	Float	3000	4999
Global	Bool	5000	6999
Local	Int	7000	8999
Local	Float	9000	10999
Local	Bool	11000	12999
-	Constants	13000	14999

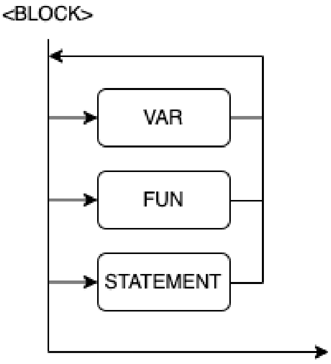
Syntax diagrams with neuralgic points



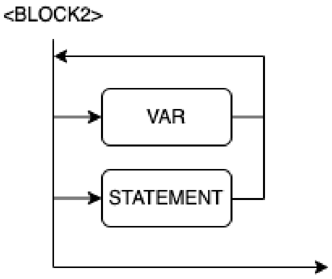
Neuralgic points	Actions description
1	1. Create programDirectory 2. Create scopeList 3. Create quadruple GOTO to go to main



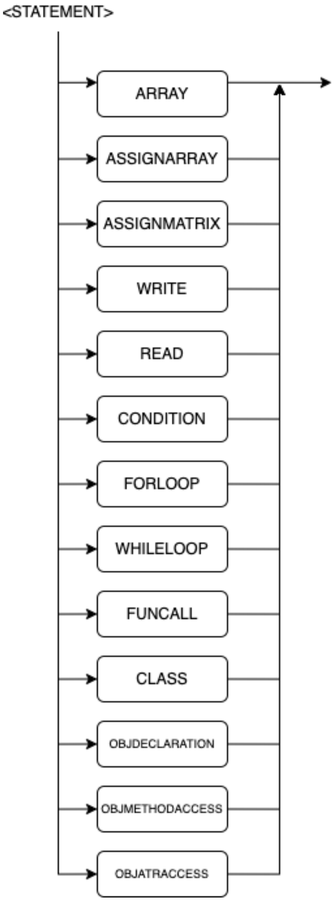
Neuralgic points	Actions description
1	1. Add local scope to scope list 2. Set program directory to main 3. Add quadruple id to first quadruple (GOTO)



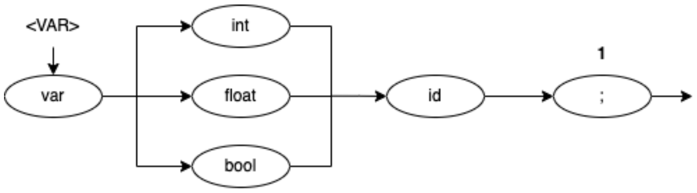
Neuralgic points	Actions description
No Neuralgic Points	



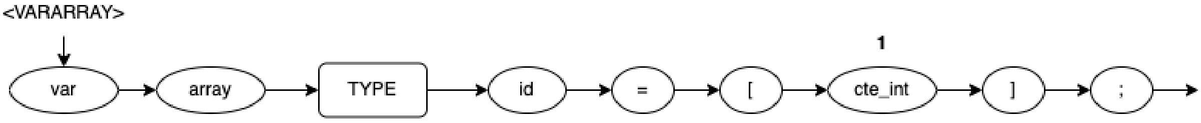
Neuralgic points	Actions description
No Neuralgic Points	



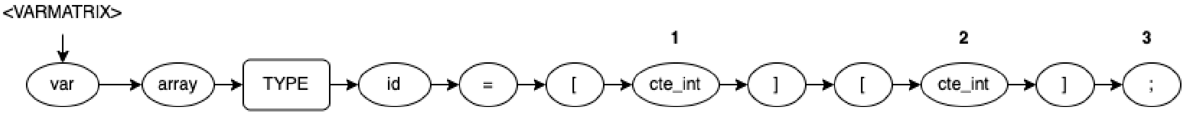
Neuralgic points	Actions description
No Neuralgic Points	



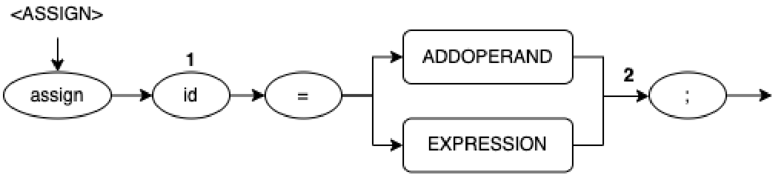
Neuralgic points	Actions description
1	1. Check current scope 2. Reserve memory space 3. Add var to the corresponding var table registry



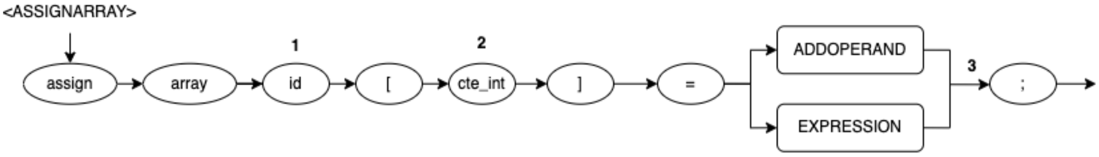
Neuralgic points	Actions description
1	1. Separate space in memory 2. Add array information into corresponding registry of the var table



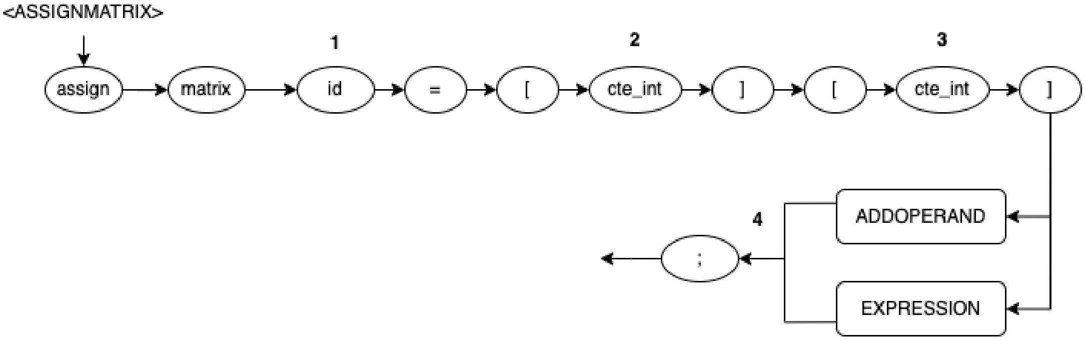
Neuralgic points	Actions description
1	1. Separate space in memory - first dimension
2	1. Separate space in memory - second dimension
3	1. Add matrix information into corresponding registry of the var table



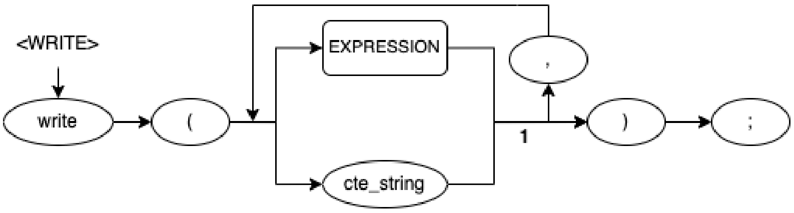
Neuralgic points	Actions description
1	1. Check if variable exist in var table 2. Add id into operand stack 3. Add id type into type stack
2	1. Solve expression 2. Add result or operand into operand stack 3. Add type into type stack 4. Generate assign quadruple



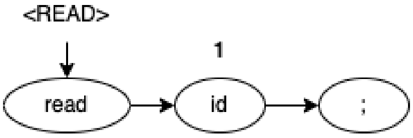
Neuralgic points	Actions description
1	1. Check if id exists in var table
2	1. Check if range is correct (VER quadruple)
3	1. Generate assign quadruple



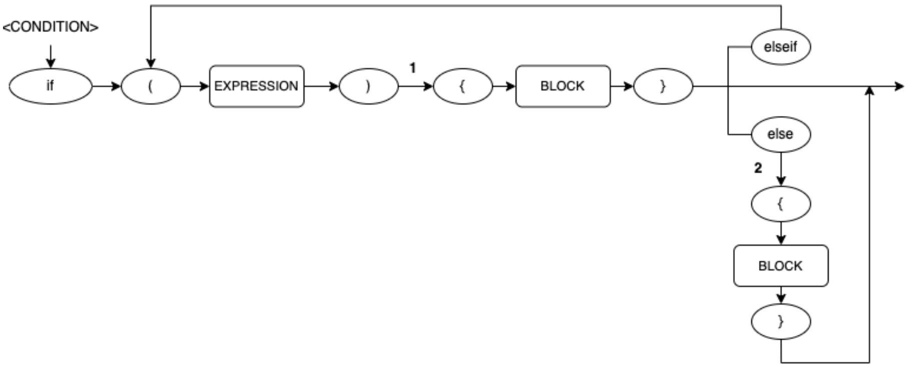
Neuralgic points	Actions description
1	1. Check if id exists in var table
2	1. Get first coordinate
3	1. Get second coordinate 2. Check if range is correct (VER quadruple)
4	1. Generate assign quadruple



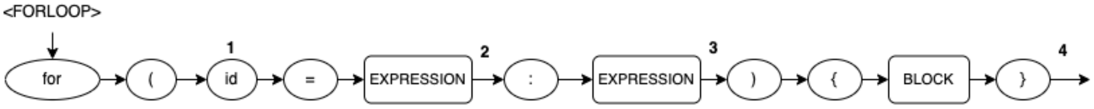
Neuralgic points	Actions description
1	1. Generate print quadruple



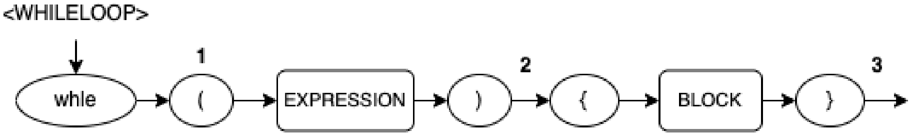
Neuralgic points	Actions description
1	1. Generate read quadruple



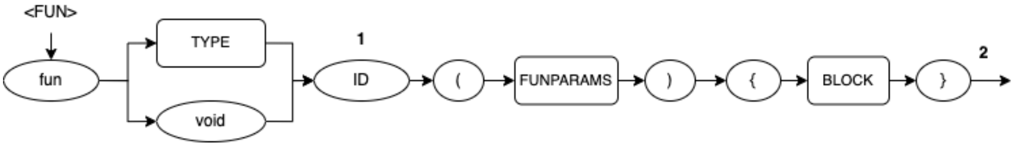
Neuralgic points	Actions description
1	1. Generate GOTOF quadruple
2	1. Generate GOTO quadruple



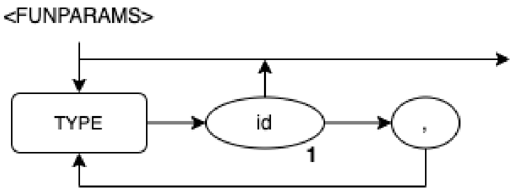
Neuralgic points	Actions description
1	1. Check if var exists in var directory 2. Check if var type is int 3. Add var to operand stack
2	1. Get expression final value 2. Create control variable assign quadruple
3	1. Create final variable assign quadruple 2. Create cycle condition quadruple
4	1. Update control variable 2. Create goto quadruple to return to cycle condition quadruple



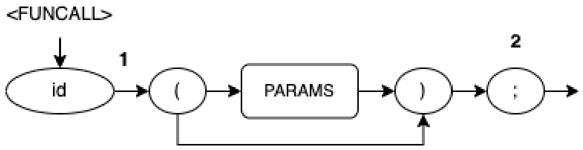
Neuralgic points	Actions description
1	1. Add quadruple id to jump stack
2	1. Generate GOTOF quadruple
3	2. Generate GOTO quadruple to return to cycle condition using previous saved quadruple id from NP 1



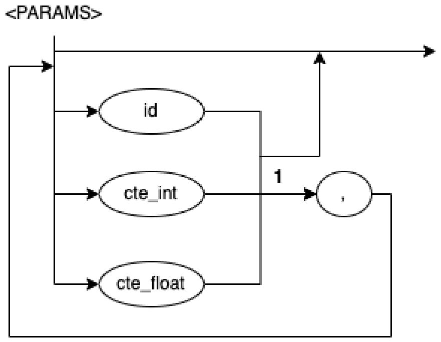
Neuralgic points	Actions description
1	1. Add local to scope list 2. Add id to Program directory 3. Save function id
2	1. Create ENDFUN quadruple



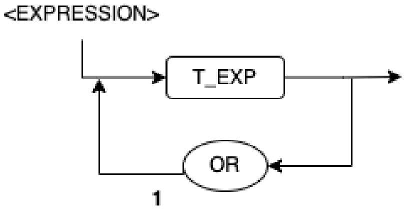
Neuralgic points	Actions description
1	1. Add var to corresponding var table registry



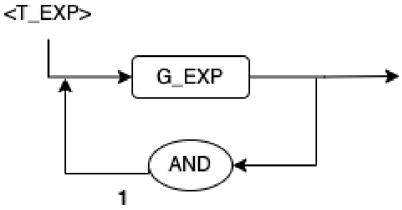
Neuralgic points	Actions description
1	1. Create ERA quadruple
2	1. Check if function has been previously declared 2. Create GOSUB quadruple



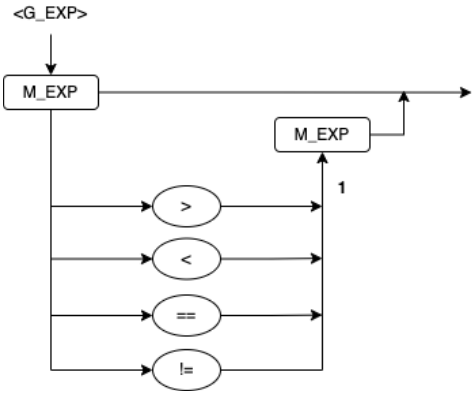
Neuralgic points	Actions description
1	1. Create PARAM quadruple



Neuralgic points	Actions description
1	1. Check if conditions are met to create a new quadruple: ○ Previous operator has greater or equal hierarchy ○ Enough elements in operand stack 2. Add operator to operator stack

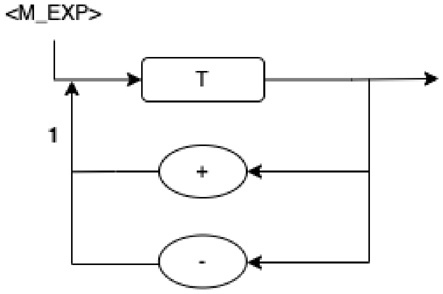


Neuralgic points	Actions description
1	1. Check if conditions are met to create a new quadruple: ○ Previous operator has greater or equal hierarchy ○ Enough elements in operand stack 2. Add operator to operator stack

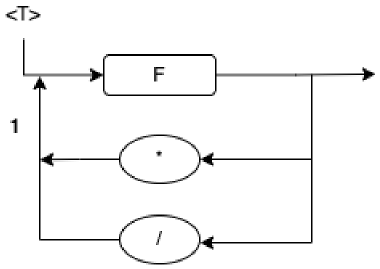


Neuralgic points	Actions description

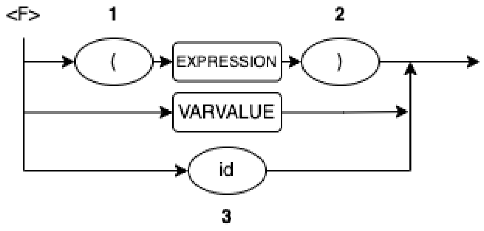
1	1. Check if conditions are met to create a new quadruple: ○ Previous operator has greater or equal hierarchy ○ Enough elements in operand stack 2. Add operator to operator stack



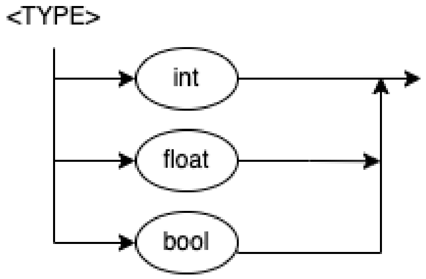
Neuralgic points	Actions description
1	1. Check if conditions are met to create a new quadruple: ○ Previous operator has greater or equal hierarchy ○ Enough elements in operand stack 2. Add operator to operator stack



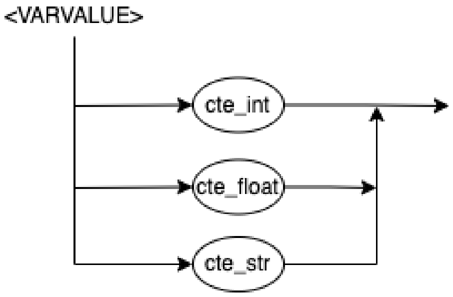
Neuralgic points	Actions description
1	1. Check if conditions are met to create a new quadruple: ○ Previous operator has greater or equal hierarchy ○ Enough elements in operand stack 2. Add operator to operator stack



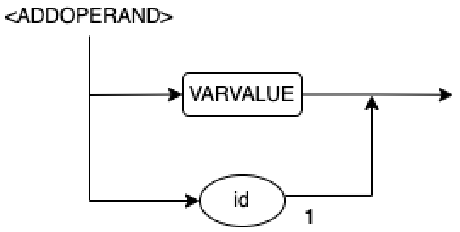
Neuralgic points	Actions description
1	1. Add fake bottom to operand stack
2	1. Pop fake bottom from operand stack
3	1. Check if ID exists in var table 2. Add element to operand stack 3. Add element type to type stack



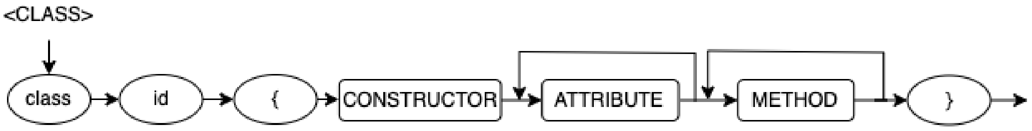
Neuralgic points	Actions description
No Neuralgic Points	



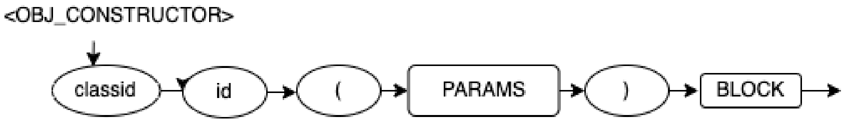
Neuralgic points	Actions description
No Neuralgic Points	



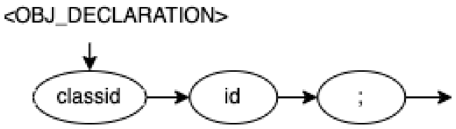
Neuralgic points	Actions description
1	1. Check if id exists in corresponding var table



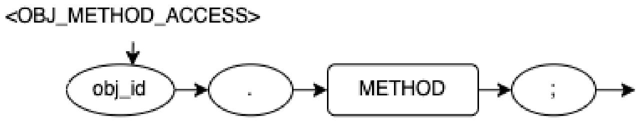
Neuralgic points	Actions description
No Neuralgic Points defined yet	



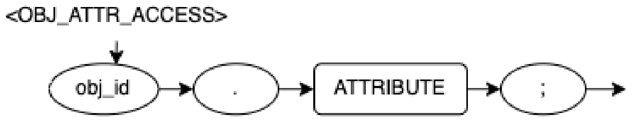
Neuralgic points	Actions description
No Neuralgic Points defined yet	



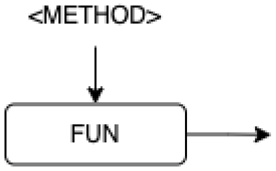
Neuralgic points	Actions description
No Neuralgic Points defined yet	



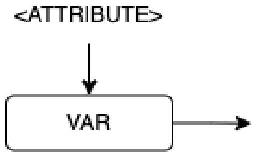
Neuralgic points	Actions description
No Neuralgic Points defined yet	



Neuralgic points	Actions description
No Neuralgic Points defined yet	



Neuralgic points	Actions description
No Neuralgic Points defined yet	



Neuralgic points	Actions description
No Neuralgic Points defined yet	

Semantic consideration table

The semantic cube in the actual code is a dictionary with dictionaries because this allows us to search through it in optimal time. It is made from strings because our typeStack contains only string elements.

op1	**op2**	*	/	+	-	>	<	<=	>=	==	**!=**	**and**	**or**	**!**
int	**int**	int	int	int	int	bool	bool	bool	bool	bool	bool	error	error	error
int	**float**	float	float	float	float	error	error	error	error	error	error	error	error	error
int	**bool**	error	error	error	error	error	error	error	error	error	error	error	error	error
float	**int**	float	float	float	float	error	error	error	error	error	error	error	error	error
float	**float**	float	float	float	float	error	error	error	error	error	error	error	error	error
float	**bool**	error	error	error	error	error	error	error	error	error	error	error	error	error
bool	**int**	error	error	error	error	error	error	error	error	error	error	error	error	error
bool	**float**	error	error	error	error	error	error	error	error	error	error	error	error	error
bool	**bool**	error	error	error	error	error	error	error	error	error	error	bool	bool	bool

d). VIRTUAL MACHINE DESCRIPTION

d.1) Memory Management

Jade's virtual machine has 5 components:

directory
quadruples
eraData
assignedVars
pointerglobal

After the compilation process is over, we send the final VarTable in the constructor of the Virtual Machine, where it is assigned to its "directory" variable. During compilation the VarTable stores the id, type, size, scope, and direction of each of the variables and since it provides all the information we need from each variable, we decided to keep and use this structure during execution. Similarly, "quadruples" is the list of quadruples (intermediate code) generated by compilation and is received in the constructor; they do nothing by themselves but once the virtual machine receives them, each one of them is executed and together form the program; we decided to use a list as data structure because each quadruple needs to be executed right after the previous one but we also needed the possibility to jump from one quadruple to another depending on the "goto"s, so a queue or a stack would not have allowed us to do that.

Finally, "eraData" is a list made up of 2-cell arrays that helps to manage functions and its parameters, the first cell contains the name of the function and the second contains the type of the parameter; during execution the second cell is modified so that it actually stores the value of the parameter and can be used.

On the other hand, “assignedVars” and “pointerGlobal” are not received in the constructor of the virtual machine. The purpose of assignedVars is to store the addresses and the values of the used elements for that reason, everytime that an “assign” quadruple is read, the virtual machine adds or updates elements; we chose to have a dictionary as data structure in this case because they allow us to search through it in an optimal time.

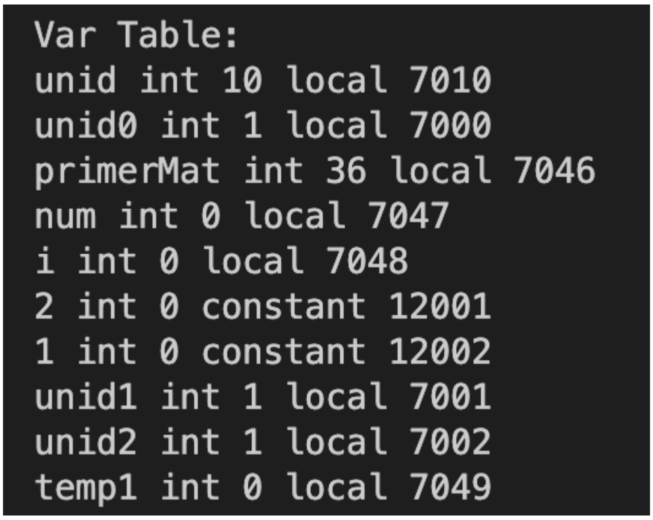
The “pointerglobal” variable starts in 0 when the VM is first created, it is increased by 1 when a quadruple is executed. This variable is very useful because when a quadruple’s operator is “goto”, “gotoF” or “GOSUB”, then the whe send a pointer that will execute quadruples until it is the same as the pointer global and that way the execution continues after that change in context.

<u>Association between addresses in compilation and execution</u>

We decided to assign memory addresses to all variables and constants during compilation and include them in the VarTable. The virtual machine receives the VarTable during execution and creates a dictionary that adds/updates a new element everytime an assign (“=”) quadruple is executed. The keys of the dictionary are the memory addresses and the pairs are the actual values that are stored in that address.

In the other operations that the virtual machine executes, the virtual machine looks in the vartable for the address, and then looks for that address in its dictionary; this process ensures that all variables have been assigned a value before trying to being used.

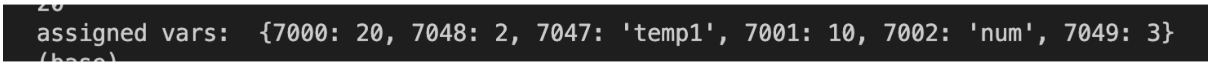
Example of VarTable:



```
Var Table:
unid int 10 local 7010
unid0 int 1 local 7000
primerMat int 36 local 7046
num int 0 local 7047
i int 0 local 7048
2 int 0 constant 12001
1 int 0 constant 12002
unid1 int 1 local 7001
unid2 int 1 local 7002
temp1 int 0 local 7049
```

In this example the array “unid” was declared and the user specified that it will contain 10 elements, for that reason, the Memory returns the direction of the last cell that is assigned for the array. Similarly with the matrix, but in that case the number of rows is multiplied by the number of columns in order to obtain the total size that needs to be assigned. Then, each of the elements of the arrays and/or matrixes is stored with the id of the array plus its index and the VarTable stores the address of that unique element based on the total size and index given.

Example of Assigned Vars:



```
20
assigned vars:  {7000: 20, 7048: 2, 7047: 'temp1', 7001: 10, 7002: 'num', 7049: 3}
(base)
```

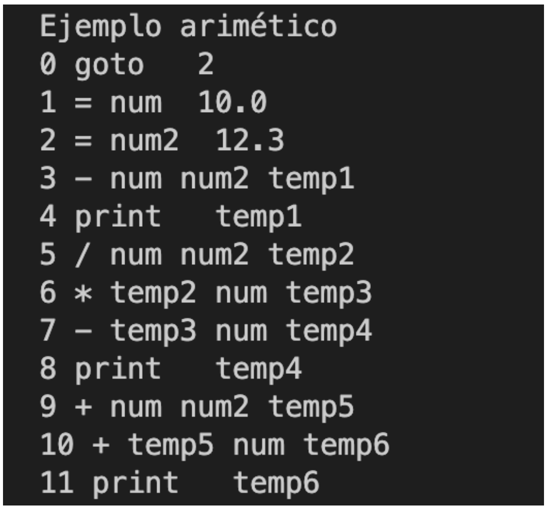
In this example we can see that the address 7000 (belonging to the element 0 of the array “unid”) holds the value “20”. Another good example is that the address 7047 (belonging to the variable “num”) has the value “temp1”, so we check the Var Table and find that the variable “temp1” has the memory address 7049 assigned, so we can go back to Assigned Vars and look for that address, as we can see, it holds the value “3”, which means that num = 3;

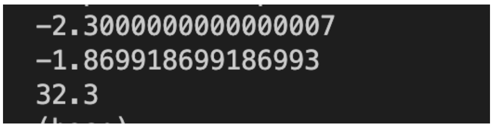
e). TESTS

e.1) Proof of results

Test 1 - Arithmetic example

Code	Generated quadruples

program test1 {
var float num;
var float num2;
assign num = 10.0;
main {
assign num2 = 12.3;
print(num - num2);
print(num / num2 * num - num);
print(num + num2 + num);
}
}
	
	Ejemplo arimético
	0 goto 2
	1 = num 10.0
	2 = num2 12.3
	3 - num num2 temp1
	4 print temp1
	5 / num num2 temp2
	6 * temp2 num temp3
	7 - temp3 num temp4
	8 print temp4
	9 + num num2 temp5
	10 + temp5 num temp6
	11 print temp6

Expected Result	**Result**
-2.3000000000000007 -1.869918699186993 32.3	 -2.3000000000000007 -1.869918699186993 32.3

Test 2 - Function example

<table>
<tr><th>Code</th><th>Generated quadruples</th></tr>
<tr><td>

```
program test2 {
    var int num3;

    fun void imprimeuno(int a){
        assign num3 = 3490;
        print(num3);
        print(a);
    }

    main {
        var int ejemplo;
        assign ejemplo = 20;
        print(ejemplo);
        imprimeuno(5);
    }
}
```

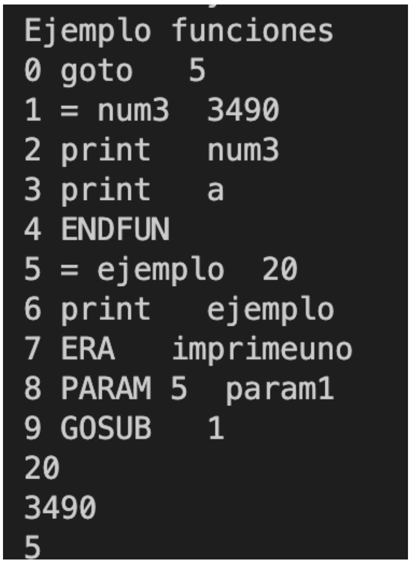
</td><td>

```
0 goto   5
1 = num3  3490
2 print   num3
3 print   a
4 ENDFUN
5 = ejemplo  20
6 print   ejemplo
7 ERA   imprimeuno
8 PARAM 5  param1
9 GOSUB   1
```

</td></tr>
<tr><th>Expected Result</th><th>Result</th></tr>
<tr><td>

```
20
3490
5
```

</td><td>



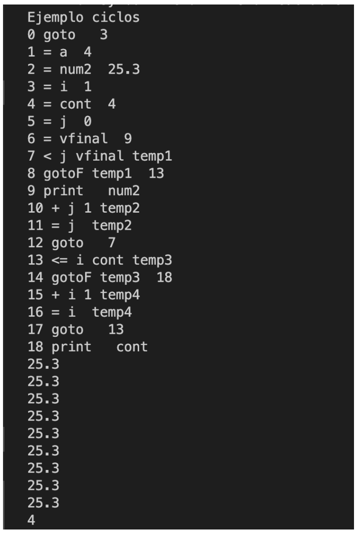
```
Ejemplo funciones
0 goto   5
1 = num3   3490
2 print   num3
3 print   a
4 ENDFUN
5 = ejemplo   20
6 print   ejemplo
7 ERA   imprimeuno
8 PARAM 5   param1
9 GOSUB   1
20
3490
5
```

</td></tr>
</table>

Test 3 - For loop and while loop example

Code	Generated quadruples

program test3 {
    var int num;
    var int a;
    assign a = 4;
    var float num2;
    assign num2 = 25.3;
 	
    main {
        var int i;
        assign i = 1;
        var int cont;
        assign cont = 4;
        var int j;
        for(j = 0 : 9){
            print(num2);}
        while(i <= cont;){
            assign i = i + 1;}
        print(cont);}
}
	0 goto 3
	1 = a 4
	2 = num2 25.3
	3 = i 1
	4 = cont 4
	5 = j 0
	6 = vfinal 9
	7 < j vfinal temp1
	8 gotoF temp1 13
	9 print num2
	10 + j 1 temp2
	11 = j temp2
	12 goto 7
	13 <= i cont temp3
	14 gotoF temp3 18
	15 + i 1 temp4
	16 = i temp4
	17 goto 13
	18 print cont

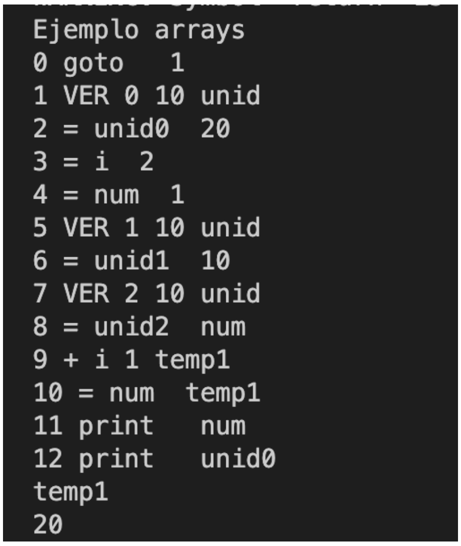
Expected Result	**Result**
25.3 25.3 25.3 25.3 25.3 25.3 25.3 25.3 25.3 4	 Ejemplo ciclos 0 goto 3 1 = a 4 2 = num2 25.3 3 = i 1 4 = cont 4 5 = j 0 6 = vfinal 9 7 < j vfinal temp1 8 gotoF temp1 13 9 print num2 10 + j 1 temp2 11 = j temp2 12 goto 7 13 <= i cont temp3 14 gotoF temp3 18 15 + i 1 temp4 16 = i temp4 17 goto 13 18 print cont 25.3 25.3 25.3 25.3 25.3 25.3 25.3 25.3 25.3 4

Test 4 - Array example

Code	Generated quadruples

`program test4{`
`    main{`
`        var array int unid = [10];`
`        assign array unid [0] = 20;`
`        var matrix int primerMat =`
`[12][3];`
`        var int num;`
`        var int i;`
`        assign i = 2;`
`        assign num = 1;`
`        assign array unid[1] = 10;`
`        assign array unid[2] = num;`
`        assign num = i + 1;`
`        print(num);`
`        print(unid[0]);`
`    }`
`}`
	0 goto 1
	1 VER 0 10 unid
	2 = unid0 20
	3 = i 2
	4 = num 1
	5 VER 1 10 unid
	6 = unid1 10
	7 VER 2 10 unid
	8 = unid2 num
	9 + i 1 temp1
	10 = num temp1
	11 print num
	12 print unid0

Expected Result	**Result**

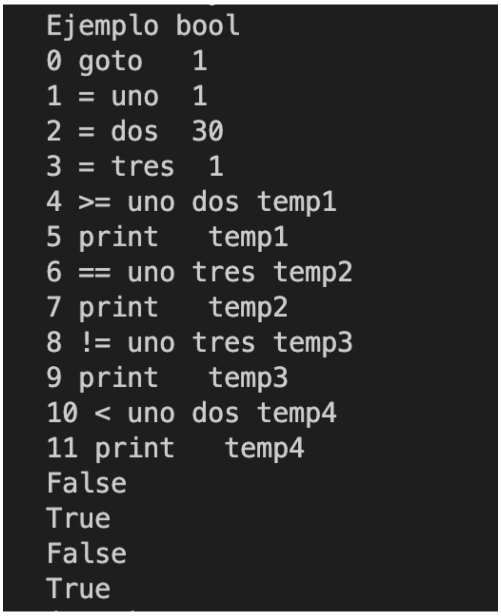
temp1
20
	
	`Ejemplo arrays`
	`0 goto   1`
	`1 VER 0 10 unid`
	`2 = unid0  20`
	`3 = i  2`
	`4 = num  1`
	`5 VER 1 10 unid`
	`6 = unid1  10`
	`7 VER 2 10 unid`
	`8 = unid2  num`
	`9 + i 1 temp1`
	`10 = num  temp1`
	`11 print   num`
	`12 print   unid0`
	`temp1`
	`20`

Test 5 - Bool operations example

Code	Generated quadruples

`program test5 {`
`    var int uno;`
`    var int dos;`
`    main{`
`        assign uno = 1;`
`        assign dos = 30;`
`        var int tres;`
`        assign tres = 1;`
`        print(uno >= dos);`
`        print(uno == tres);`
`        print(uno != tres);`
`        print(uno < dos);`
`    }`
`}`
	0 goto 1
	1 = uno 1
	2 = dos 30
	3 = tres 1
	4 >= uno dos temp1
	5 print temp1
	6 == uno tres temp2
	7 print temp2
	8 != uno tres temp3
	9 print temp3
	10 < uno dos temp4
	11 print temp4

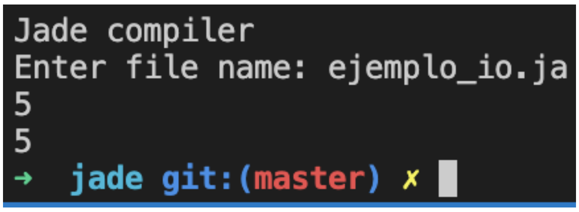
Expected Result	**Result**

False
True
False
True
	
	`Ejemplo bool`
	`0 goto   1`
	`1 = uno  1`
	`2 = dos  30`
	`3 = tres  1`
	`4 >= uno dos temp1`
	`5 print   temp1`
	`6 == uno tres temp2`
	`7 print   temp2`
	`8 != uno tres temp3`
	`9 print   temp3`
	`10 < uno dos temp4`
	`11 print   temp4`
	`False`
	`True`
	`False`
	`True`

Test 5 - Input and Output example

Code	**Generated quadruples**

program inputoutput{
 	
main{
var int num;
read num;
 	
print(num);
}
}
	0 goto 1
	1 READ num
	2 print num

Expected Result	Result
5	 Jade compiler Enter file name: ejemplo_io.ja 5 5 ➜ jade git:(master) ✗