Yet Another Compiler’s Class Project (YACCP)

14 de junio de 2023

Sergio Tapia Gómez - A00822497

Index

Project Description

Project Scope and purpose

The scope of this project is to develop a program able to simulate a compiler by integrating the previous knowledge acquired along my CompSci formation (Like algorithm design, data structures, etc.) with the different parts and methods that are commonly used for their development as seen during the semester of my Compiler Design class. As for the purpose of this project, I wanted to develop a functional project able to receive a program that follows certain grammar rules and syntax structures that can then read the code, translate it by using the quadruples intermediate interpretation and finally be able to run it by providing the quadruples to a virtual machine that is able to execute them in order to get the desired result. All this with the idea of having a better understanding of how the compilers development process takes place and how certain design ideas or limitations influence what can and can't be done by using the developed language.

Requirements analysis and main Test Cases description

Requirements:

- Variable declaration should be able to support the three atomic types int, float and char
- Functions can support the three atomic types plus a void type which should not return anything
- Variables should be able to be declared as dimensioned variables (arrays and matrices)
- Language should be able to support while and if statement as nonlinear statements
- Language should be able to process and correctly evaluate arithmetic and logical expressions as well as assignations
- User should be only able to declare variables globally at the beginning of the program and inside functions at the beginning of itself
- The structure of the grammar must be followed for the program to be correctly recognized and thus, be parsed
- Data input and output should be supported via the read and print statements respectively
- Function calls can be assigned to variables only if they have a return type
- Recursively calling functions should be supported

Test Cases:

Factorial Calculation

The iterative version of the code was written with a while loop in contrast of the typical for loop implementation, this is due to my language not supporting a for statement

Also as my language only supports if and else, the implementation of the recursive program was made with a nested if-else

The programs have some needed variables declared at the beginning of the program, then the function declaration with more variables declared inside followed by the loops or recursive calls respectively and the return statements needed for them to work, finally the main block runs the function, assigns it to a variable and then prints it.

The recursive code makes function calls until reaching a base case (n == 0 or n == 1) where it will return the accumulated value of the calls, the iterative version just loops until the condition is met and then returns the value

The expected result is the correct value for evaluating the factorial of the number provided. (e.g. Factorial(5) = 120).

Fibonacci Calculation

The iterative version of the code was written with a while loop in contrast of the typical for loop implementation, this is due to my language not supporting a for statement

Also as my language only supports if and else, the implementation of the recursive program was made with a nested if-else

The programs have some needed variables declared at the beginning of the program, then the function declaration with more variables declared inside followed by the loops or recursive calls respectively and the return statements needed for them to work, finally the main block runs the function, assigns it to a variable and then prints it.

The recursive code makes function calls until reaching the base case (n == 0) where it will return the accumulated value of the calls, the iterative version just loops until the condition is met and then returns the value

The expected result is the correct value for evaluating the Fibonacci sequence value of the number provided (e.g. Fibonacci(5) = 5).

General process description

The process for this project was a little shaky from beginning to end since I started with a teammate and ended up doing it alone, not justifying why so many weeks were not productive at all but making sure it's set as a precedent.

Week 1:

Grammar and diagrams first version: Not that much was made during the first week of development, our first version of the grammar and the syntax diagrams were made taking into account the LittleDuck syntax diagrams given to us in Tarea 3.1 and Tarea 3.2

Commitments were to finish the grammar and diagrams and show them to the teacher for feedback.

Week 2:

Grammar and diagrams corrections: Again not a significant advance, we spoke to the teacher for feedback and after receiving some comments on what we should change and add or remove we made the needed changes. This was the second version of both the grammar and the syntax diagrams

Commitments made were to double check for errors and start the lexer and parser during the next week

Week 3:

Lexer and parser development: This journal for the third week was not for the actual third week of the project, more like the fifth week, (it's worth noting that the following weeks will also be behind) we took a two week hiatus and then started developing the lexer and parser as we felt the pressure of the project creeping in. This was the last week were we worked as a team

Commitments made were that we would have both the lexer and parser done by the end of the week

Week 4:

Changing project and grammar/syntax diagrams restructuring: Journal entry for this week is actually for the week 6 of the project, this week my teammate and I mutually

decided that going our separate ways was for the best as we had creative differences and were not a good match in general. Each of us took the grammar, diagrams the lexer and a half made parser we had and started our own project. I revamped my grammar and the syntax diagrams to match a project definition that I liked better. After that I changed the lexer to fit my token definitions and the parser grammar to fit the one I had made.

Commitment was I'd finish a pending structure that I had been assigned and Rodrigo would finish the parser part he had assigned and we'd exchange that as the last part of the joint project and continue each with their own project

Week 5:

Lexer and parser partially completed: Lexer was finished then the parser was still being developed, most rules were already finished, semantic cube and the beginning of some structures was being worked on as I was alone and hadn't really made a significant advance, I had not made the github repository but had a local copy of the things I had been developing

My commitment to myself was to finish this, push into a github repository and start working on the quadruple generation asap

Week 6:

Grammar finished, quadruple generation starting and other structures: During this week I started to really feel the pressure but depression was also tying me down with doing things. I still managed to finish the grammar, add some neural points and started quadruple generation.

QuadrupleGen, QuadActions classes were made, a CustomStack class to manage false bottom was also developed, Semantics cube and lexer were also pushed into github with the rest of the classes aforementioned.

Commitment was to generate all quadruples asap

Week 7:

Quadruple generation for expressions, variable assignations and nonlinear statements (functions excluded): This was the last week basically for developing the project I asked for more time and received it. I managed to start generating quadruples for expressions, variables and non linear statements by the end of the week even functions were added to the mix by the last day but was not able to make them work correctly

My commitment this week was not only made to me but also to the teacher since I asked for more time in exchange of committing to work harder on the project but also take care of my mental and physical health. I feel like I partially delivered mostly on school but prioritizing my health paid off by the next week

Week 8:

Quadruple generation for functions, function calls and assignation of function calls, plus the development of the Virtual Machine for quadruple execution and the documentation: This was the week I worked the hardest, I feel like I had everything finished but semi functional by the end of the week, all the things generated before this week worked correctly but functions were just not, I ended up developing the Virtual Machine to be able to execute what I had, then I went back to bug fixing functions until they worked first for quadruples and then for the virtual machine, parameters where such a pain since I kept forgetting to save some parameters and that was the root cause of many errors

Commitment was to fix functions executions both normal and recursively since both had problems finding variables while executing in the virtual machine

Week 9:

Fix for functions, documentation and vector development: This two day and a half week (June 12 - 14) was defined basically by me trying to understand why only functions were not working in execution when the methods made for the handling of functions and the quadruples seemed correct until I noticed I was indexing an incorrect quadruple jump to gosub instructions which made all functions skip the first line which caused problems when the first line was an assignations as the variable wouldn't exist by the time it's used again in the code (if used). After figuring that out everything ran smoothly and I was able to add some final touches to the handling of everything I had done until then. Then I finished the documentation and focused on trying to make dimensions work in conjunction with everything I had already working.

My reflection about this experience

This experience for me was such a rollercoaster, the mix of emotions both during the development of the project and outside of it were kind of overwhelming. I usually try to evade all comments about if a class is difficult or not since I tend to put myself under if I know a class “is going to be difficult”, I’m very gullible in that sense, Analysis and Design of Algorithms is one example, this class is another one, I just get in my head and start seeing the class as an uphill battle.

I’m not just blaming the depression because I also think it’s the fact that CompSci is my least favorite part of the career path I chose, but that combination makes it really hard for me to do stuff, add that to some attention deficit and we have a recipe for disaster, this is not just a reflection on how unfair everything was or how everything was just stacked against me, I’m trying to reach my point by setting the scene to how I learned what I learned and why I find it so much more important than just learning about compilers.

As a final detail to the context of what I learned I want to add that I hate asking for help, I hated feeling like a burden to people, but this project helped me so much to let loose of that fear, I asked for helped and I received it, the teacher, my family and even my friends helped me in some form or capacity that was so eye opening for me, I learned that needing help is okay and that people are willing to help if you ask, that was the most important lesson for me. I can also say now that I’m near the finish line of the project that I value way more the hard work that goes behind the design and development of compilers, IDE, languages and all compilers related code. Doing this project made me once and for all decide that the kind of thought and work that needs to be put is just another reason why I’ll really appreciate this area from afar.

Language description

Language name: YACCP (Yet Another Compilers Class Project) is a name that came to me at the beginning of the semester but my teammate was not happy with it, it has a double meaning for me since it’s yet another compiler project that I’m doing as it’s my second time taking this class and I had developed some lines of code for a compiler project on my previous time around this class plus I even started this semester doing a completely different compilers project which makes this the third project (third time makes the charm?) but it’s also a name I chose as a funny thought I had about how the teacher must see it as just another one of the bunch, as Yet Another Compilers Class Project.

Main Characteristics: The language was first thought as a combination of C type code with features brought from languages like R or MATLAB so that we’d have a C type statistical language, unfortunately due to time constraints and some personal issues YACCP was repurposed to a generic type C programming language able of basic stuff like manipulation of expressions, use of nonlinear statements like the while loop or the if

statement, variable declaration plus assignation, function declaration and recursive calls, as well as one dimensioned variables. The structure of the program goes as follows: Program name, followed by global variables declaration, then functions declaration (local variables can be declared at the beginning of functions) and finally the main function which is where the code will begin to run.

Common errors:

Compilation errors:

- While opening a file an error might be thrown if file is not found or it's unable to open the file provided
- Invalid type declaration
- Singleton classes trying to be initialized for a second time
- Type mismatch during condition checking, return quadruple generation, param assignation, variable or function call assignation or expression evaluating
- Variables or functions not declared
- Double declaration of function or variable
- Syntax errors (From missing semicolons and brackets to wrong sequence of keywords or declarations)

Execution errors:

- Division by 0
- Out of bounds
- Access to an empty memory space
- Stack overflow (Memory is fully occupied)
- Unrecognized quadruple operation

Compiler description

Computer equipment, language and special utils used for the project development

Computer equipment: Gigabyte G5 KD with Windows11

Language and special utils: Everything was developed with Python, with the help of PLY as the compiler to parse the code for quadruple generation and VSCode as the development environment tool

Lexical Analysis description

Tokens

Token	Regex
LSB	\[
RSB	\]
LP	\(
RP	\)
LBR	{
RBR	}
COMMA	,
SEMICOLON	;
COLON	:
CONST_INT	\d+
CONST_FLOAT	\d+\.\d+
ID	[a-zA-Z_]\w*
AND	&&
OR	||
GT	>

LT	<
LE	<=
GE	>=
ASSIGN	==
NEQ	!=
EQ	=
PLUS	\+
MINUS	\-
MULT	*
DIV	\/
PERIOD	\.
CONST_STRING	"(\\"|[^\n"])+"
CONST_CHAR	'(\\'|\\n|\\\\\|[^\n'])'

PROGRAM	program
MAIN	main
IF	if
ELSE	else
WHILE	while
FUNC	func

RETURN	return
READ	read
PRINT	print
INT	int
FLOAT	float
CHAR	char
DATAFRAME	dataframe
VOID	void
MEAN	mean
MEDIAN	median
MODE	mode
STD	std
VARIANCE	variance
HIST	hist
PLOT	plot
SCTPLOT	sctplot

Syntax Analysis description

Grammar

```
program -> PROGRAM ID SEMICOLON var_dec func_dec main
func_dec -> FUNC return_type ID LP param_opt RP func_block func_dec | ε
param_opt -> type_simple ID more_param_opt
more_param_opt -> COMMA param_opt | ε
var_dec -> type_simple ID dim SEMICOLON var_dec | type_complex ID complex_dec
SEMICOLON | ε
complex_dec -> COMMA ID complex_dec | ε
dim -> LSB INT RSB | ε
main -> MAIN LP RP func_block
func_block -> LBR var_dec statute RBR
return_type -> type_simple | VOID
assignation -> var exp_dim_opt ASSIGN exp_or_func_assignation
exp_or_func_assignation -> expression_assignation | func_call
expression_assignation -> exp SEMICOLON
var -> ID exp_dim_opt
complex_var -> ID
exp_dim_opt -> LSB exp RSB | ε
if_statement -> IF LP exp RP block else
else -> ELSE block | ε
while_statement -> WHILE LP exp RP block
read -> READ LP var RP SEMICOLON
constants -> CONST_INT | CONST_FLOAT | CONST_CHAR
func_call -> ID LP opt_args RP SEMICOLON
opt_args -> exp exp_args_more
exp_args_more -> COMMA exp opt_args | ε
statements -> assignation | if_statement | while_statement | read | print |
func_call | return | data_funcs | ε
data_funcs -> mean | mode | median | std | variance | hist | plot | sctplot
return -> RETURN LP exp RP
print -> PRINT LP exp_or_string RP SEMICOLON
exp_or_string -> CONST_STRING | exp | print_rec
print_rec -> COMMA exp_or_string print_rec | ε
block -> LBR statute statute1 RBR
statute : statements | ε
statute1 : statute statute1 | ε
type_simple -> INT | FLOAT | CHAR
type_complex -> DATAFRAME
super_exp -> exp relop exp | exp
relop -> EQ | LE | GT | LT | NEQ | AND | OR
exp -> term PLUS exp | term MINUS exp | term
term -> factor MULT term | factor DIV term
factor -> LP super_exp RP | constants | var | func_call
```

Intermediate Code Generation and Semantic Analysis description

Operation Code and virtual addresses: I ended up working with strings, I know it might not be the best, maybe even harder but I found it less confusing and found some workarounds some stuff like the goto to some functions for example for the jumps to main or the jump to the endfunc I ended up pushing main and endfunc to the left operand portion of the quadruple, this way I can check first if goto is operator and then if either main or endfunc is found in left operand then it knows and executes actions for the correct goto operator, this way I ensured there was no problem while working with strings in the operator.

The operation Codes are:

gotof: Conditional jump to the quad number in result

goto: Jumps to quad number in result can have a main or an endfunc in leftOp, goto just jumps to main or the end of a function respectively

=: Assigns a value to result

All operators (+, -, *, /, &&, etc.): Evaluates the expression

read: Receives a value and assigns it to the variable in result

print: Prints the variable or string found in result

ERA: Starts the local memory for the function in result

GOSUB: Jumps to the start of the function in result to run the function

PARAM: Saves the parameters that are to be assigned left is global variable, result is the parameter of the function called, has to generate one per parameter

RETURN: Saves the value that has to be returned in call stack works with multiple calls since it will keep pushing the results to the stack until finished

ENDFUNC: Marks the end of a void function, exits the function and continues with the call stack

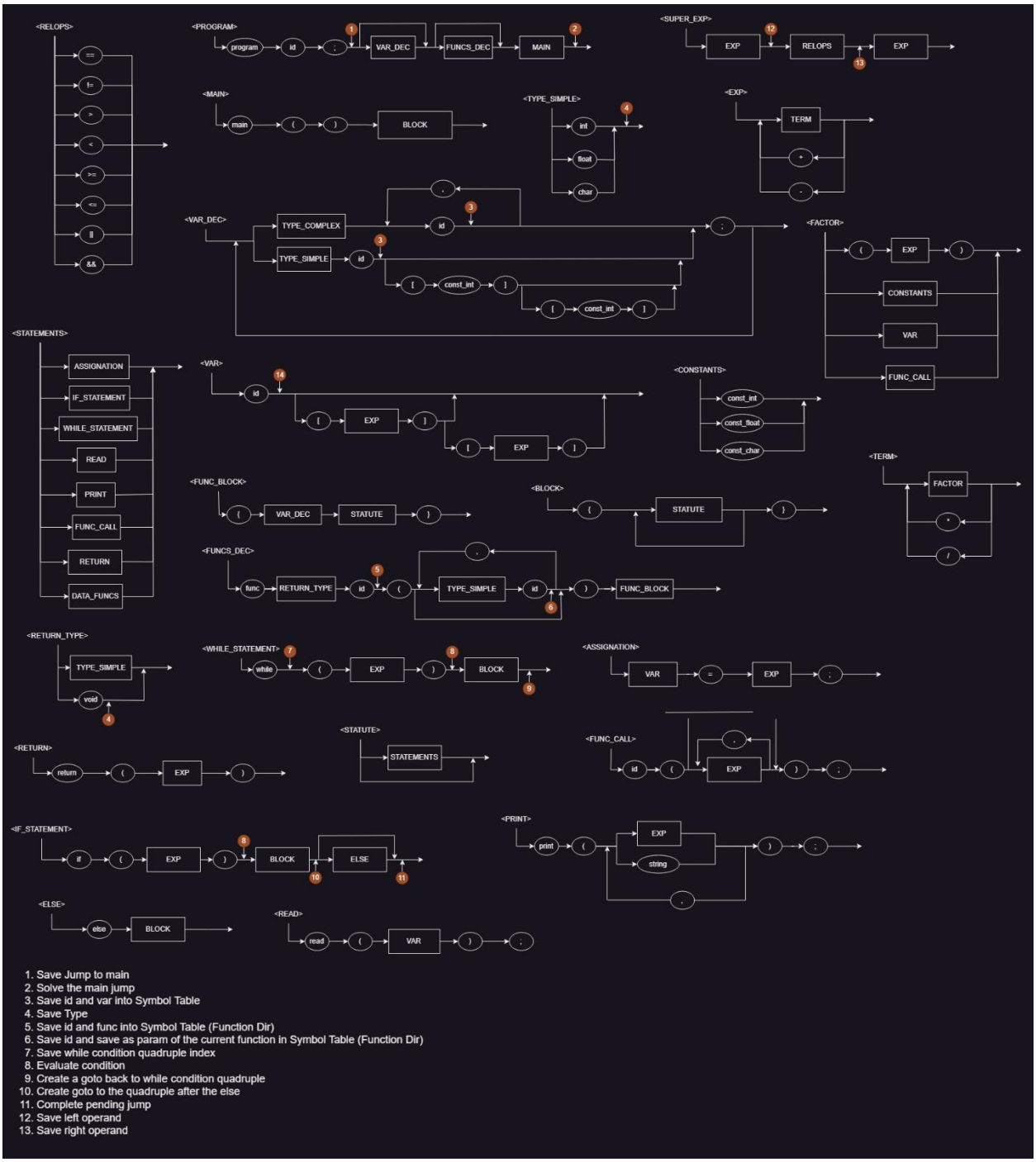
FASSGN: Code used for function call assignations in variables, has to do different stuff than the regular assign since occasionally function calls have to pass a result in another scope, so basically assigns values between scopes when needed

As to the *virtual addresses* assigned to the different sets of variables I ended up choosing ranges kind of far between scopes just to better differentiate the type of variable, I also chose to not save constants in global scope which is definitely memory space and searches gone to waste but given the time I had and that I had some inconveniences adding those variables as global constants, the idea was scrapped. Just 1000 spaces are

open for each type of variable for locals, global and temporal variables the ranges I chose were

	GLOBAL	LOCAL	TEMP
INT	1000	10000	20000
FLOAT	2000	11000	21000
BOOL	3000	12000	22000
CHAR	4000	13000	23000

Syntax Diagrams with neural points and descriptions



Link to diagrams: https://app.diagrams.net/#G1k5JhhAK7HTrnhMf8aY6Lh27WAo5kLOVH

Semantic Considerations Table

INT	INT	+	INT
INT	INT	-	INT
INT	INT	/	INT
INT	INT	*	INT
INT	INT	&&	ERROR
INT	INT	||	ERROR
INT	INT	>	BOOL
INT	INT	<	BOOL
INT	INT	>=	BOOL
INT	INT	<=	BOOL
INT	INT	==	BOOL
INT	INT	!=	BOOL

INT	FLOAT	+	FLOAT
INT	FLOAT	-	FLOAT
INT	FLOAT	/	FLOAT
INT	FLOAT	*	FLOAT
INT	FLOAT	&&	ERROR
INT	FLOAT	||	ERROR
INT	FLOAT	>	BOOL
INT	FLOAT	<	BOOL
INT	FLOAT	>=	BOOL
INT	FLOAT	<=	BOOL
INT	FLOAT	==	BOOL
INT	FLOAT	!=	BOOL

FLOAT	FLOAT	+	FLOAT
FLOAT	FLOAT	-	FLOAT
FLOAT	FLOAT	/	FLOAT
FLOAT	FLOAT	*	FLOAT
FLOAT	FLOAT	&&	ERROR
FLOAT	FLOAT	||	ERROR
FLOAT	FLOAT	>	BOOL
FLOAT	FLOAT	<	BOOL
FLOAT	FLOAT	>=	BOOL
FLOAT	FLOAT	<=	BOOL
FLOAT	FLOAT	==	BOOL
FLOAT	FLOAT	!=	BOOL

Char and bool type have error with everything except themselves and comparison operators (e.g. C == C or B && B)

Memory Administration during compilation description

Memory
get()
getGlobalMemory(self)
getConsts(self)
addConstant(self, constant, constantType)
localMemoryStacks(self)
activeMemory(self)

ScopeManager
init(self)
funcs(self)
vars(self)
scopes(self)
parent(self)
setParent(self, parent)
func(self, funcName)
var(self, varName)
addFunc(self, newName, funcType)
addVar(self, newName, varType, isConst)
getVarFromId(self, varId)
getFuncFromId(self, funcId)

FunctionDirectory
exists(funcId)
isNewFunction(funcId)
get()
setReturnQuad(funcId, returningQuad)
setReturnQuadVal(funcId, returnTo)
setReturnVarId(funcId, returnVarId)
getReturnVarId(funcId)
saveFuncToDir(funcId, startingQuadPosition)
setFuncStartingQuad(funcId, startingQuadPosition)
getFunc(funcId)
getParams(funcId)
saveParams(st)
saveLocalVars(st)
saveTempVar(st, varName, varType)
getLocalVar(funcId)

CustomStack
init(self)
stack(self)
push(self)
pop(self)
top(self)
pushOp(st, op)
popOp(st, op)
isEmpty(self)
popOp(self)

SymbolTable
init(self)
get(arg)
currentScopeName(self)
currentScope(self)
scopeStack(self)
setCurrType(self, newType)
currentType(self)
setCurrId(self, newId)
currentId(self)
lastSavedFunc(self)
setLastSavedFunc(self, savedFunc)
saveVar(self)
saveTempVar(self, name, varType)
saveFunc(self)
saveParameter(self)
pushNewScope(self)

QuadActions
saveMainQuad(qg)
solveMainQuad(qg)
operationsActions(st, qg)
normalAssignationActions(st, qg)
functionAssignation(st, qg, params)
paramAssignQuads(st, qg)
gosubJump(st, qg)
setReturn(st, qg)
saveFuncCallOp(st)
createGotoFQuadIf(st, qg)
createGotoQuadIf(qg)
updatePendingJumpIf(qg)
whileStatementActions(st, qg)
createGotoQuadWhile(st, qg)
updatePendingJumpWhile(qg)
updatePendingJumpWhile(qg)

QuadrupleGen
get(arg)
init(self)
quadruples(self)
pendingJumps(self)
tempCounter(self)
generateTemp(self)
generateQuadruple(self, operator, operand1, opeand2, result, addQuad)
printQuadruples(self)
addPendingJump(self, jump)

MemoryChunk
getVars(self, addressType)
findAddress(self, varId, recursiveLookup)
initAddress(self, varId, addressType, scope)
setValIt(self, resId, valueId)
print(self)
getVal(self, varId)
getAddressValue(self, address)
assignAddressValue(self, value, address)

The structures used for this project are CustomStack, FunctionDirectory, Memory, QuadActions, QuadrupleGen, ScopeManager, SymbolTable, Vars and Functions

CustomStack is self-explanatory, this class was designed to handle the false bottoms during expressions, works as a stack just the push and pop move and retrieve stacks used for expressions whenever they find parenthesis in the expression. Push saves the current stack in a stack of stacks then creates a new stack for the expression between parenthesis, Pop retrieves the last stack when a closing parenthesis is found this way it "restores" the previous state to the parenthesis

FunctionDirectory isn't actually a class but more of a global dictionary, since it's global it acts as a Singleton the key is the function id and then inside it has the starting quadruple index, the index of the quad where it ends, the variables that it must return, a list of params represented as a tuple of (id, type) and the local variables that the function holds, also represented as a tuple of (id, type). I thought an actual dictionary would be easier for searching than attributes of a class and that made me choose that over defining a function directory class. Also I already had many classes interacting between themselves and I felt like it was way too much happening

The **Memory** class does as it may be obvious the memory handling, we have two classes inside, Memory which is the super class that contains MemoryChunks which contain the variables addresses and values. During compilation it just prepares the spaces for global variables and constants and assigns a memory value to those variables but no value is stored as during compilation no values are still being managed

QuadActions isn't a class by itself but it sure does the heavy lifting during quadruple generation as here you can find all the actions that take place to generate most of the quadruples, for the exception of some that were moved to lexer_parser

QuadrupleGen is the class that makes the magic around the quadruples generation, this class is in charge of receiving the instruction then creating and storing the quadruples as well as storing pending jumps. This class also generates all temporal variables used during quadruple generation, increasing the value of the temp each time a temp is created

ScopeManager is the class that as its name indicates, manages all scope shenanigans, as it works as a pseudo symbol table since this class also stores what functions or variables are stored in what scope, when a variable is declared or is called this class takes makes sure it's not already declared or that it indeed exists respectively

SymbolTable was one of the first classes I built, it's a big class that basically works as middle man between the scope, function directory and virtual machine as this function stores different variables and stacks (All operand, operator and their types stacks are stored here) and that are needed by other classes or methods, holds from types to functions parameters, the point of this class at the beginning was to have it work together with the function directory as the place to store variables but then that shifted after the scopes class seemed like a better fit for me to store variables. It's still a vital part of the

code working as many information goes through this class reason enough for me to not break apart the class since would mean many relocation of methods and variables.

Both **Vars** and **Functions** are found in the same file and are classes that define the attributes of the classes, not much more to say, they are needed for all the variable and functions processing

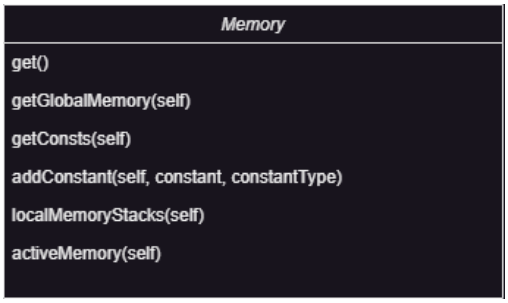
Virtual Machine description

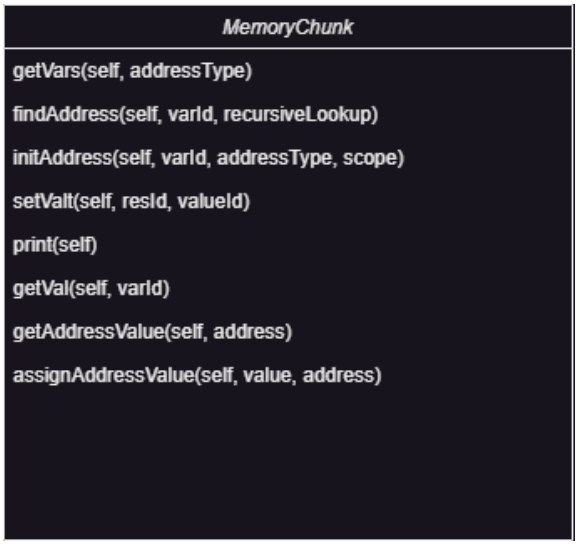
Computer equipment, language and special utils used for the project development

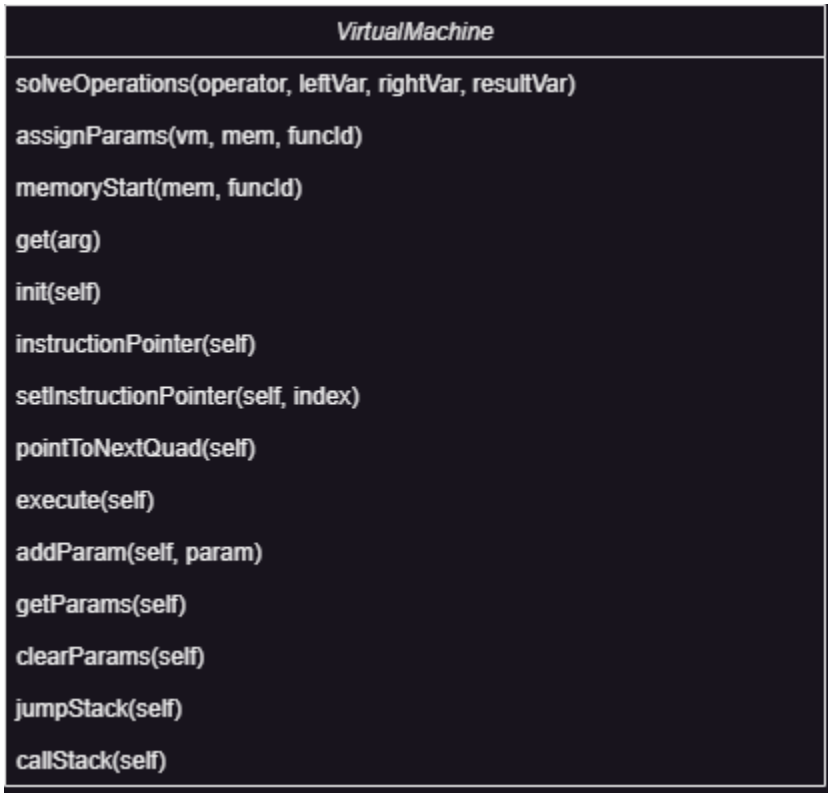
Computer equipment: Gigabyte G5 KD with Windows11

Language and special utils: Virtual Machine was also developed with Python, with the help of PLY as the compiler to parse the code for quadruple generation and VSCode as the development environment tool

Memory Administration during execution description







FunctionDirectory as mentioned previously this class stores function so it's involvement during execution is a must, with the methods from this class Memory is able to find the functions or the parameters when handling any quadruple within functions or that reference functions

The **Memory** class, during compilation, together with its sister class MemoryChunk and the FunctionsDirectory finds the variable, function or parameter and then hands the information to the VM

And finally **VirtualMachine** is the class that does all the execution, works together with Memory to assign, retrieve and manipulate values in the memory. Memory provides all the information needed to complete the execution

The values were previously exposed but as a reminder:

	GLOBAL	LOCAL	TEMP
INT	1000	10000	20000
FLOAT	2000	11000	21000
BOOL	3000	12000	22000
CHAR	4000	13000	23000

This were the ranges I chose arbitrarily, I gave 1000 spaces to each type of memory for each scope and treated them as if they were the real memory, each address stores only one value and is assigned to one variable of the type

Proof of functionality

For this part I'll test both Fibonacci implementations and both Factorial implementations, the value I chose to add to every test will be 5 and 8 for fibonnacci, and 3 and 6 for factorial after running both files 2 times results should be:

Fibonacci(5) = 5

Fibonacci(8) = 21

Factorial(3) = 6

Factorial(6) = 720

The codes that will be running are the following:

Factorial Recursive:

```
program factorialRec;
int fact;

func int factorialRec(int n){
    int temp;
    int result;
    if(n == 0) {
        return(1);
    }
    else {
        if(n == 1){
            return(1);
        }
        else {
            temp = n - 1;
            result = factorialRec(temp);
            return(n * result);
        }
    }
}

main(){
    fact = factorialRec(3);
    print(fact);
}
```

Factorial Iterative:

```
program factorialIter;
int result;
int fact;

func int factorial(int n){
    result = 1;
    while(n > 1){
        result = result * n;
        n = n - 1;
    }
    return(result);
}

main(){
    fact = factorial(6);
    print(fact);
}
```

Fibonacci Recursive:

```
program fibonacci;

int result;

func int fibonacciRec(int n) {
    int temp1;
    int temp2;

    if (n == 0) {
        return(0);
    }
    else {
        if (n == 1) {
            return(1);
        }
        else {
            temp1 = fibonacciRec(n - 1);
            temp2 = fibonacciRec(n - 2);
            return (temp1 + temp2);
        }
    }
}

main() {
    result = fibonacciRec(8);
    print(result);
};
```

Fibonacci Iterative:

```
program fibonacci;
    int result;


func int fibIterativo(int n){
    int num1;
    int num2;
    int num3;
    int i;

    if(n <= 0) {
        return(0);
    }
    else {
        if(n == 1){
            return(1);
        }
        else{
            num1 = 0;
            num2 = 1;
            i = 2;

            while(i <= n){
                num3 = num1 + num2;
                num1 = num2;
                num2 = num3;
                i = i + 1;
            }
            return(num2);
        }
    }
}


main(){
    result = fibIterativo(5);
    print(result);
};
```

FibRecursive(5) result:

```
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project> py yaccp.py fibonacciRecursive.txt
0: ['goto', 'main', '', 25]
1: ['==', 'n', 0, 't0']
2: ['gotof', 't0', '', 6]
3: ['RETURN', '', '', 0]
4: ['goto', 'ENDFUNC', '', '']
5: ['goto', '', '', 24]
6: ['==', 'n', 1, 't1']
7: ['gotof', 't1', '', 11]
8: ['RETURN', '', '', 1]
9: ['goto', 'ENDFUNC', '', '']
10: ['goto', '', '', 24]
11: ['ERA', '', '', 'fibonacciRec']
12: ['-', 'n', 1, 't2']
13: ['PARAM', 't2', '', 'n']
14: ['GOSUB', '', '', 'fibonacciRec']
15: ['FASSGN', 'temp1', '', 'fibonacciRec']
16: ['ERA', '', '', 'fibonacciRec']
17: ['-', 'n', 2, 't3']
18: ['PARAM', 't3', '', 'n']
19: ['GOSUB', '', '', 'fibonacciRec']
20: ['FASSGN', 'temp2', '', 'fibonacciRec']
21: ['+', 'temp1', 'temp2', 't4']
22: ['RETURN', '', '', 't4']
23: ['goto', 'ENDFUNC', '', '']
24: ['ENDFUNC', '', '', '']
25: ['ERA', '', '', 'fibonacciRec']
26: ['PARAM', 5, '', 'n']
27: ['GOSUB', '', '', 'fibonacciRec']
28: ['FASSGN', 'result', '', 'fibonacciRec']
29: ['print', '', '', ('result',)]
Jumping into compilation...
The result is
5
Compilation went smoothly! Thanks for trying YACCP
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project>
```

FibIterative(5) result:

```
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project> py yaccp.py fibonacciIter.txt
0: ['goto', 'main', '', 26]
1: ['<=', 'n', 0, 't0']
2: ['gotof', 't0', '', 6]
3: ['RETURN', '', '', 0]
4: ['goto', 'ENDFUNC', '', '']
5: ['goto', '', '', 25]
6: ['==', 'n', 1, 't1']
7: ['gotof', 't1', '', 11]
8: ['RETURN', '', '', 1]
9: ['goto', 'ENDFUNC', '', '']
10: ['goto', '', '', 25]
11: ['=', 0, '', 'num1']
12: ['=', 1, '', 'num2']
13: ['=', 2, '', 'i']
14: ['<=', 'i', 'n', 't2']
15: ['gotof', 't2', '', 23]
16: ['+', 'num1', 'num2', 't3']
17: ['=', 't3', '', 'num3']
18: ['=', 'num2', '', 'num1']
19: ['=', 'num3', '', 'num2']
20: ['+', 'i', 1, 't4']
21: ['=', 't4', '', 'i']
22: ['goto', '', '', 14]
23: ['RETURN', '', '', 'num2']
24: ['goto', 'ENDFUNC', '', '']
25: ['ENDFUNC', '', '', '']
26: ['ERA', '', '', 'fibIterativo']
27: ['PARAM', 5, '', 'n']
28: ['GOSUB', '', '', 'fibIterativo']
29: ['FASSGN', 'result', '', 'fibIterativo']
30: ['print', '', '', ('result',)]
Jumping into compilation...
The result is
5
Compilation went smoothly! Thanks for trying YACCP
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project>
```

FibRecursive(8) result:

```
Compilation went smoothly! Thanks for trying YACCP
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project> py yaccp.py fibonacciRecursive.txt
0: ['goto', 'main', '', 25]
1: ['==', 'n', 0, 't0']
2: ['gotof', 't0', '', 6]
3: ['RETURN', '', '', 0]
4: ['goto', 'ENDFUNC', '', '']
5: ['goto', '', '', 24]
6: ['==', 'n', 1, 't1']
7: ['gotof', 't1', '', 11]
8: ['RETURN', '', '', 1]
9: ['goto', 'ENDFUNC', '', '']
10: ['goto', '', '', 24]
11: ['ERA', '', '', 'fibonacciRec']
12: ['-', 'n', 1, 't2']
13: ['PARAM', 't2', '', 'n']
14: ['GOSUB', '', '', 'fibonacciRec']
15: ['FASSGN', 'temp1', '', 'fibonacciRec']
16: ['ERA', '', '', 'fibonacciRec']
17: ['-', 'n', 2, 't3']
18: ['PARAM', 't3', '', 'n']
19: ['GOSUB', '', '', 'fibonacciRec']
20: ['FASSGN', 'temp2', '', 'fibonacciRec']
21: ['+', 'temp1', 'temp2', 't4']
22: ['RETURN', '', '', 't4']
23: ['goto', 'ENDFUNC', '', '']
24: ['ENDFUNC', '', '', '']
25: ['ERA', '', '', 'fibonacciRec']
26: ['PARAM', 8, '', 'n']
27: ['GOSUB', '', '', 'fibonacciRec']
28: ['FASSGN', 'result', '', 'fibonacciRec']
29: ['print', '', '', ('result',)]
Jumping into compilation...
The result is
21
Compilation went smoothly! Thanks for trying YACCP
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project>
```

FibIterative(8) result:

```
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project> py yaccp.py fibonacciIter.txt
0: ['goto', 'main', '', 26]
1: ['<=', 'n', 0, 't0']
2: ['gotof', 't0', '', 6]
3: ['RETURN', '', '', 0]
4: ['goto', 'ENDFUNC', '', '']
5: ['goto', '', '', 25]
6: ['==', 'n', 1, 't1']
7: ['gotof', 't1', '', 11]
8: ['RETURN', '', '', 1]
9: ['goto', 'ENDFUNC', '', '']
10: ['goto', '', '', 25]
11: ['=', 0, '', 'num1']
12: ['=', 1, '', 'num2']
13: ['=', 2, '', 'i']
14: ['<=', 'i', 'n', 't2']
15: ['gotof', 't2', '', 23]
16: ['+', 'num1', 'num2', 't3']
17: ['=', 't3', '', 'num3']
18: ['=', 'num2', '', 'num1']
19: ['=', 'num3', '', 'num2']
20: ['+', 'i', 1, 't4']
21: ['=', 't4', '', 'i']
22: ['goto', '', '', 14]
23: ['RETURN', '', '', 'num2']
24: ['goto', 'ENDFUNC', '', '']
25: ['ENDFUNC', '', '', '']
26: ['ERA', '', '', 'fibIterativo']
27: ['PARAM', 8, '', 'n']
28: ['GOSUB', '', '', 'fibIterativo']
29: ['FASSGN', 'result', '', 'fibIterativo']
30: ['print', '', '', ('result',)]
Jumping into compilation...
The result is
21
Compilation went smoothly! Thanks for trying YACCP
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project>
```

FactorialRecursive(3) result:

```
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project> py yaccp.py factorialRec.txt
0: ['goto', 'main', '', 21]
1: ['==', 'n', 0, 't0']
2: ['gotof', 't0', '', 6]
3: ['RETURN', '', '', 1]
4: ['goto', 'ENDFUNC', '', '']
5: ['goto', '', '', 20]
6: ['==', 'n', 1, 't1']
7: ['gotof', 't1', '', 11]
8: ['RETURN', '', '', 1]
9: ['goto', 'ENDFUNC', '', '']
10: ['goto', '', '', 20]
11: ['-', 'n', 1, 't2']
12: ['=', 't2', '', 'temp']
13: ['ERA', '', '', 'factorialRec']
14: ['PARAM', 'temp', '', 'n']
15: ['GOSUB', '', '', 'factorialRec']
16: ['FASSGN', 'result', '', 'factorialRec']
17: ['*', 'n', 'result', 't3']
18: ['RETURN', '', '', 't3']
19: ['goto', 'ENDFUNC', '', '']
20: ['ENDFUNC', '', '', '']
21: ['ERA', '', '', 'factorialRec']
22: ['PARAM', 3, '', 'n']
23: ['GOSUB', '', '', 'factorialRec']
24: ['FASSGN', 'fact', '', 'factorialRec']
25: ['print', '', '', ('fact',)]
Jumping into compilation...
The result is
6
Compilation went smoothly! Thanks for trying YACCP
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project>
```

FactorialIterative(3) result:

```
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project> py yaccp.py factorialIter.txt
0: ['goto', 'main', '', 12]
1: ['=', 1, '', 'result']
2: ['>', 'n', 1, 't0']
3: ['gotof', 't0', '', 9]
4: ['*', 'result', 'n', 't1']
5: ['=', 't1', '', 'result']
6: ['-', 'n', 1, 't2']
7: ['=', 't2', '', 'n']
8: ['goto', '', '', 2]
9: ['RETURN', '', '', 'result']
10: ['goto', 'ENDFUNC', '', '']
11: ['ENDFUNC', '', '', '']
12: ['ERA', '', '', 'factorial']
13: ['PARAM', 3, '', 'n']
14: ['GOSUB', '', '', 'factorial']
15: ['FASSGN', 'fact', '', 'factorial']
16: ['print', '', '', ('fact',)]
Jumping into compilation...
The result is
6
Compilation went smoothly! Thanks for trying YACCP
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project>
```

FactorialRecursive(6) result:

```
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project> py yaccp.py factorialRec.txt
0: ['goto', 'main', '', 21]
1: ['==', 'n', 0, 't0']
2: ['gotof', 't0', '', 6]
3: ['RETURN', '', '', 1]
4: ['goto', 'ENDFUNC', '', '']
5: ['goto', '', '', 20]
6: ['==', 'n', 1, 't1']
7: ['gotof', 't1', '', 11]
8: ['RETURN', '', '', 1]
9: ['goto', 'ENDFUNC', '', '']
10: ['goto', '', '', 20]
11: ['-', 'n', 1, 't2']
12: ['=', 't2', '', 'temp']
13: ['ERA', '', '', 'factorialRec']
14: ['PARAM', 'temp', '', 'n']
15: ['GOSUB', '', '', 'factorialRec']
16: ['FASSGN', 'result', '', 'factorialRec']
17: ['*', 'n', 'result', 't3']
18: ['RETURN', '', '', 't3']
19: ['goto', 'ENDFUNC', '', '']
20: ['ENDFUNC', '', '', '']
21: ['ERA', '', '', 'factorialRec']
22: ['PARAM', 6, '', 'n']
23: ['GOSUB', '', '', 'factorialRec']
24: ['FASSGN', 'fact', '', 'factorialRec']
25: ['print', '', '', ('fact',)]
Jumping into compilation...
The result is
720
Compilation went smoothly! Thanks for trying YACCP
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project>
```

FactorialIterative(6) result:

```
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project> py yaccp.py factorialIter.txt
0: ['goto', 'main', '', 12]
1: ['=', 1, '', 'result']
2: ['>', 'n', 1, 't0']
3: ['gotof', 't0', '', 9]
4: ['*', 'result', 'n', 't1']
5: ['=', 't1', '', 'result']
6: ['-', 'n', 1, 't2']
7: ['=', 't2', '', 'n']
8: ['goto', '', '', 2]
9: ['RETURN', '', '', 'result']
10: ['goto', 'ENDFUNC', '', '']
11: ['ENDFUNC', '', '', '']
12: ['ERA', '', '', 'factorial']
13: ['PARAM', 6, '', 'n']
14: ['GOSUB', '', '', 'factorial']
15: ['FASSGN', 'fact', '', 'factorial']
16: ['print', '', '', ('fact',)]
Jumping into compilation...
The result is
720
Compilation went smoothly! Thanks for trying YACCP
PS D:\ITESM\11º Semestre\Compiladores\Proyecto Final\Compilers-Class-Project>
```