Instituto Tecnologico y de Estudios Superiores de Monterrey

Compilers Design

TheOnlyLonely: Project documentation

Wednesday 02/06/2021

jcmartgon

Jesus Carlos Martinez Gonzalez

A01037089

Contents

1 Project description

1.1 Purpose and scope

The purpose of this project is to generate a rather simple programing language, it should include fundamental concepts such as arithmetic expression handling, variables, decision statements, looping statements and functions, as well as simple graphical outputting capabilities.

The project is not entirely autonomous, and it will rely on third-party tools, mainly to figure out the structure of the source code being read during compilation, and to provide to graphical environment upon which the graphical statements will run.

1.2 Requirements and test cases definition

As stated above, the project must include features found in most popular programming languages, such as arithmetic expression handling, sequential statements such as print and return, non-sequential statements such as decision and looping statements, functions, including recursive calls, and graphical outputting statements, as to generate simple geometric figures.

To test the functionality of these features, the following test cases were defined:

Test case #	Description	Test	Expected output
0	Iterative factorial	Factorial of 5	120
1	Recursive factorial	Factorial of 5	120
2	Iterative Fibonacci	9th element of the Fibonacci sequence	34
3	Recursive Fibonacci	9th element of the Fibonacci sequence	34
4	Draw a person	Draw a person using graphical output functions	Something resembling a person

1.3 Software development process

Throughout the development of the project, an agile approach was followed, with specific weekly goals to be met, where every week the corresponding goals would be planned, designed, implemented, and tested before moving on to the next features on the calendar.

To keep track of the progress being done and that one pending, a progress log was kept in which weekly goals were stablished, the pending tasks were noted, and comments were provided whenever significant setbacks took place. The log follows a simple color scheme to denote the urgency of the backlog, with green representing the project being in a healthy state and an intense red representing a critical one.

Week	Goal	Pending	Comments
22/03 - 28/03	Define reserved words and sintactical flow	To define whether to work with graphical output or dimensional variables	Will wait to get a better understanding of which option is best
29/03 - 04/04	Holiday week	To define whether to work with graphical output or dimensional variables	None
05/04 - 11/04	Implement lexical and sintactical analysis	To define whether to work with graphical output or dimensional variables	None
12/04 - 18/04	Implement function's directory and variable's table	To define whether to work with graphical output or dimensional variables. Function's directory and variable's table.	Not clear as to what can be stored during compilation and what during execution
19/04 - 25/04	Implement semantic cube and intermediate code for arithmetic and sequential statements	To define whether to work with graphical output or dimensional variables. Function's directory and variable's table. Intermediate code for arithmetic and sequential statements.	Poor time management
26/04 - 02/05	Implement intermediate code for non-sequential statements	To define whether to work with graphical output or dimensional variables. Intermediate code for arithmetic, sequential and non-sequential statements.	Backlog has been reduced but not enough
03/05 - 09/05	Implement intermediate code for modules	To define whether to work with graphical output or dimensional variables. Intermediate code for non-sequential statements and modules.	Backlog has been reduced but not enough
10/05 - 16/05	Implement memory map and virtual machine for arithmetic and sequential statements	To define whether to work with graphical output or dimensional variables. Intermediate code for modules. Memory map. Virtual machine for arithmetic and sequential statements	Backlog has been reduced but not enough
17/05 - 23/5	Implement intermediate code for arrays and execution of non-sequential statements	Intermediate code for modules and arrays. Virtual machine for arithmetic, sequential and non-sequential statements.	Backlog has been reduced but not enough
24/05 - 30/05	Documentation and complete execution	Intermediate code for arrays. Documentation. Complete execution.	Most likely the project won't be completed.
31/05 - 02/06	Finish	Arrays	

On a personal note, regarding the software development process followed during the project, I believe my time management was the single biggest weakness in the entire project, it is ultimately what prevented me from completing the tasks to be implemented and as is now evident through the log, my organization or lack of thereof seriously jeopardized not only the project itself but the also the outcome of the subject for me. The subject itself was rather enlightening and I would certainly like to complete the project on my free time during the summer, however there is no doubt that the most valuable lesson I am taking from this subject is the importance of being organized.

Even though I have taken subjects dedicated to the craft of task management, it was only during this course that I have truly gotten angry at myself for not addressing a backlog sooner.

2 Language description

2.1 Main features

TheOnlyLonely is a simple programming language which can perform arithmetic operations, store values in variables of types int, float and char. Perform decision statements, including loops, make use of user-defined modules with or without return values and perform some graphical tasks such as drawing lines, circles, arcs and dots.

2.2 Built-in exceptions

Type	Occurrence
TypeError	Raised when there is a type mismatch
NameError	Raised when a variable or function which has not been declared gets called
MemoryError	Raised when there's not enough memory for the function being called
ZeroDivisionError	Raised when the second operand of a division is a 0
RuntimeError	Errors which do not fall under any of the other error categories

3 Compiler description

3.1 Tools used throughout development

TheOnlyLonely was developed on the Python 3.7 programming language and makes use of the PLY lexical and syntactical analysis tool, which seeks to provide extensive input validation, error reporting, and diagnostics. PLY was used both for the lexical analysis as well as for the syntactical.

3.2 Lexical analysis

3.2.1 Construction patterns

Element	REGEX
CT_FLOAT	\-?[0-9]+\.[0-9]+
CT_INT	\-?[0-9]+
CT_CHAR	\'.\'
CT_STRING	\".*\"
ID	[A-Za-z][A-Za-z_0-9]*

3.2.2 Tokens

TOKEN	ELEMENT
PROGRAM	Program
MAIN	main
VARS	vars
INT	int
FLOAT	float

CHAR	char
VOID	void
FUNC	func
RETURN	return
PRINT	print
IF	if
THEN	then
ELSE	else
DO	do
WHILE	while
FROM	from
TO	to
LINE	line
DOT	dot
CIRCLE	circle
ARC	arc
PENUP	penup
PENDOWN	pendown
COLOR	color
SIZE	size
RESET	reset
SETX	setx
SETY	sety
FORWARD	forward
BACKWARD	backward
LEFT	left
RIGHT	right
ROTATEX	rotatex
ROTATEY	rotatey
SEMICOLON	;
COLON	:
COMMA	,
L_PAREN	(
R_PAREN	)
L_BRACKET	{
R_BRACKET	}
L_SBRACKET	[
R_SBRACKET	]
ASSIGN	=
AND	&
OR	|
EQ	==
NE	!=
LTE	<=
GTE	>=
LT	<
GT	>
ADD	+
SUB	-
TIMES	*

DIVIDE	/

3.3 Syntactical analysis

3.3.1 Grammatical rules

- program -> program_decl vars_decl_space funcs_decl_space main
- program_decl -> PROGRAM ID SEMICOLON
- vars_decl_space -> VARS vars_decl vars_decl_list
- varss_decl var_decl vars_list COLON type SEMICOLON
- var_decl -> ID var_dim
- var_dim -> L_SBRACKET CT_INT R_SBRACKET
- vars_list -> COMMA var_decl vars_list empty
- type -> INT | FLOAT | CHAR
- vars_decl_list -> vars_decl vars_decl_list | empty
- funcs_decl_space -> func_decl funcs_decl_space | empty
- func_decl -> func_header vars_decl_space func_body
- func_header -> func_init L_PAREN params_decl R_PAREN SEMICOLON
- func_init -> ret_type FUNC ID
- ret_type -> type | VOID
- params_decl -> param_decl | empty
- param_decl -> param params_list
- param -> ID COLON type
- params_list COMMA param_decl | empty
- func_body -> L_BRACKET stmnt R_BRACKET
- stmnt -> return SEMICOLON | assignment SEMICOLON stmnt | print SEMICOLON stmnt | decision SEMICOLON stmnt | loop SEMICOLON stmnt | call SEMICOLON stmnt | graphis SEMICOLON stmnt | empty
- assignment -> ID var_dim ASSIGN hyper_Exp
- hyper_exp -> super_exp logic
- super_exp -> exp relation
- exp -> term add_sub
- term -> factor times_divide
- factor -> L_PAREN hyper_exp R_PAREN
- atom -> ID | CT_INT | CT_FLOAT | CT_CHAR | call
- times_divide -> times_divide_op term | empty
- times_divide_op -> TIMES | DIVIDE
- add_sub -> add_sub_op exp | empty
- add_sub_op -> ADD | SUB
- relation -> rel_op exp | empty
- rel_op -> GTE | LTE | GT | LT | NE | EQ
- logic -> log_op super_exp | empty
- log_op -> AND | OR
- call -> ID L_PAREN args R_PAREN
- args -> arg | empty
- arg -> hyper_exp arg_list
- arg_list -> COMMA arg | empty

- return -> RETURN L_PAREN hyper_exp R_PAREN
- print -> PRINT L_PAREN to_print R_PAREN
- to_print -> hyper_exp printing_list | CT_STRING printing_list
- printing_list -> COMMA to_print | empty
- decision -> IF L_PAREN hyper_exp R_PAREN THEN L_BRACKET stmnt R_BRACKET else_block
- else_block -> ELSE L_BRACKET stmnt R_BRACKET | empty
- loop -> conditional | non_conditional
- conditional -> WHILE L_PAREN hyper_exp R_PAREN DO L_BRACKET stmnt R_BRACKET
- non_conditional -> FROM assignment TO hyper_exp DO L_BRACKET stmnt R_BRACKET
- graphics -> line | dot | circle | arc | penup | pendown | color | size | reset| left | right
- line -> LINE L_PAREN exp R_PAREN
- dot -> DOT L_PAREN exp R_PAREN
- circle -> CIRCLE L_PAREN exp R_PAREN
- arc -> ARC L_PAREN exp R_PAREN
- penup -> PENUP L_PAREN R_PAREN
- pendown -> PENDOWN L_PAREN R_PAREN
- color -> COLOR L_PAREN CT_STRING R_PAREN
- size -> SIZE L_PAREN exp R_PAREN
- reset -> RESET L_PAREN R_PAREN
- left -> LEFT L_PAREN exp R_PAREN
- right -> RIGHT L_PAREN exp R_PAREN
- main -> main_init func_body
- maint_init -> MAIN L_PAREN R_PAREN
- empty ->

3.4 Semantical analysis

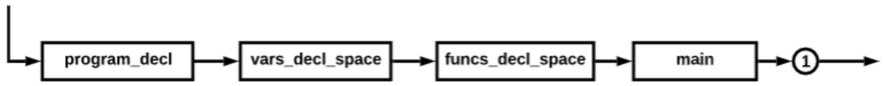
3.4.1 Operations code

Operator	Semantic code
+	0
-	1
*	2
/	3
==	4
!=	5
<	6
<=	7
>	8
>=	9
&	10
|	11
=	12
print	13
goto	14
gotof	15
endunf	16
era	17
param	18
gosub	19

return	20
read	21
endprog	22
line	23
dot	24
circle	25
arc	26
penup	27
pendown	28
color	29
size	30
reset	31
left	32
right	33

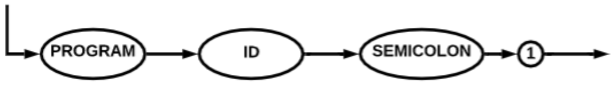
3.4.2 Syntax diagrams

<<program>>



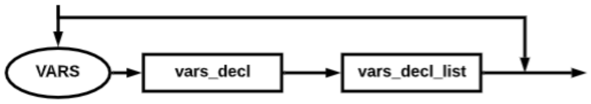
1) endprog quad

<<program_decl>>

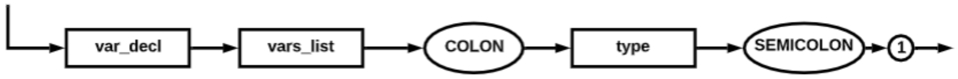


1) goto main quad
adds main func

<<vars_decl_space>>

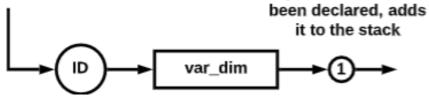


<<vars_decl>>



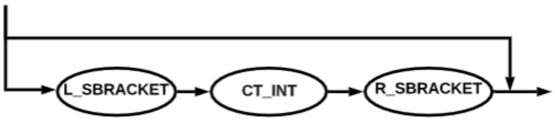
1) adds the vars to
the current function

<<var_decl>>

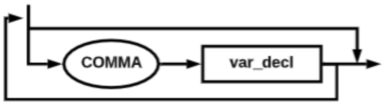


1) if the var hasn't
been declared, adds
it to the stack

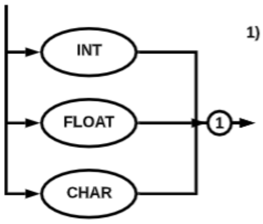
<<var_dim>>



<<vars_list>>

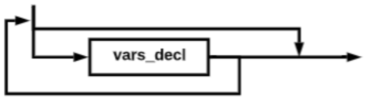


<<type>>

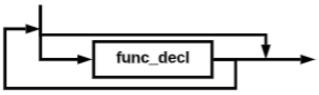


1) adds the type to the stack

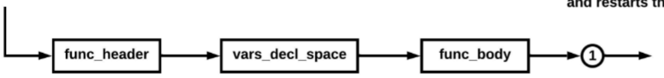
<<vars_decl_list>>



<<funcs_decl_space>>

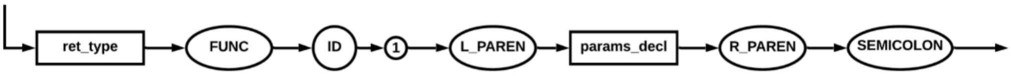


<<func_decl>>



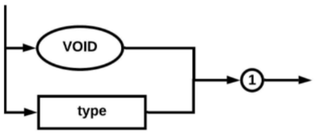
1) pops the function from the stack, adds endfunc quad and restarts the local counter

<<func_header>>



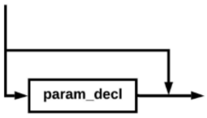
1) if the function hasn't been declared, it gets added

<<ret_type>>



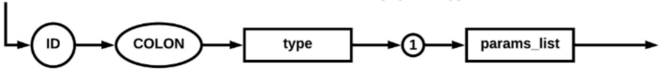
1) adds the type to the stack

<<params_decl>>

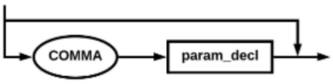


1) adds the variable to the table modifies the function's signature pops the type

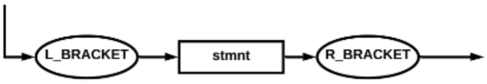
<<param_decl>>



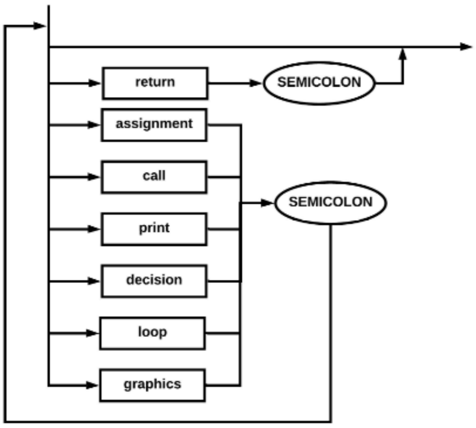
<<params_list>>



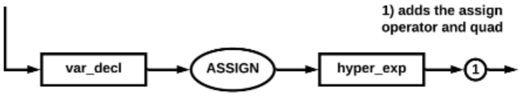
<<func_body>>



<<stmnt>>

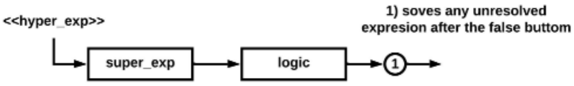


<<assignment>>



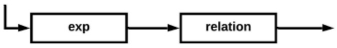
1) adds the assign operator and quad

<<hyper_exp>>

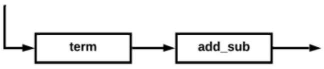


1) soves any unresolved expresion after the false buttom

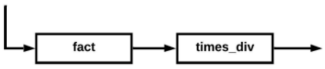
<<super_exp>>



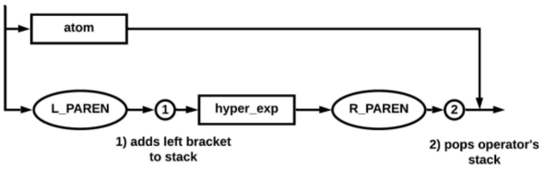
<<exp>>



<<term>>



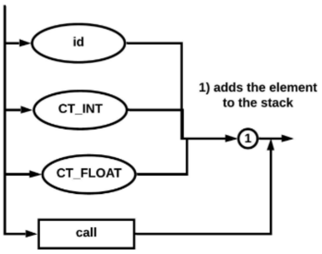
<<factor>>



1) adds left bracket to stack

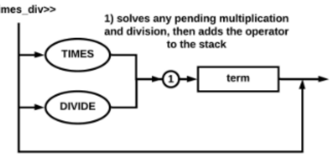
2) pops operator's stack

<<atom>>



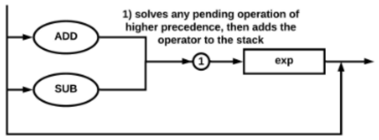
1) adds the element to the stack

<<times_div>>



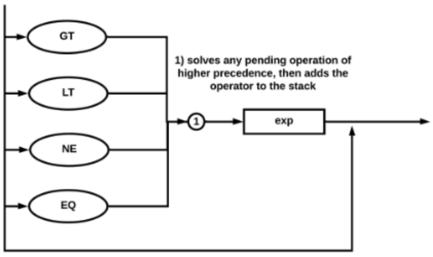
1) solves any pending multiplication and division, then adds the operator to the stack

<<add_sub>>



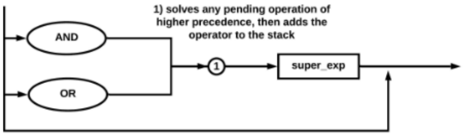
1) solves any pending operation of higher precedence, then adds the operator to the stack

<<relation>>



1) solves any pending operation of higher precedence, then adds the operator to the stack

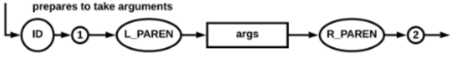
<<logic>>



1) solves any pending operation of higher precedence, then adds the operator to the stack

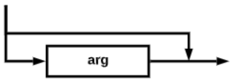
<<call>>

1) era quad prepares to take arguments



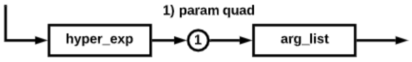
2) stores the value of the global variable corresponding to the function being called into a temporary on the current scope

<<args>>

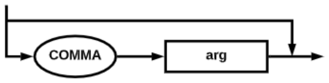


<<arg>>

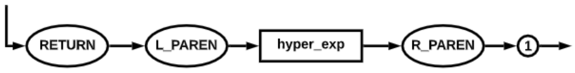
1) param quad



<<arg_list>>

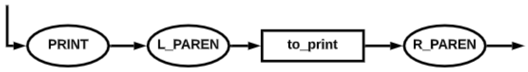


<<return>>

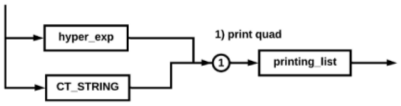


1) stores the solution to the hyper expression into the global variable corresponding to the returning function

<<print>>

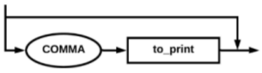


<<to_print>>

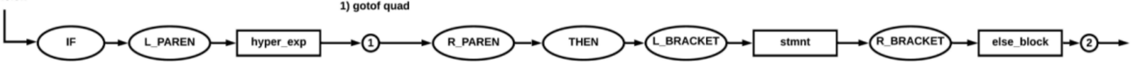


1) print quad

<<printing_list>>



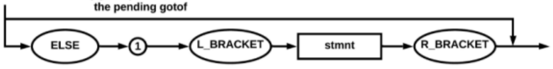
<<decision>>



1) gotof quad

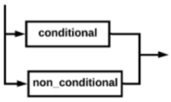
2) fills the pending goto/gotof

<<else_block>>

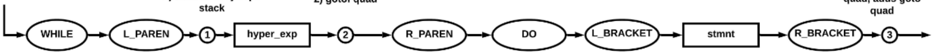


1) goto quad and fills the pending gotof

<<loop>>



<<conditonal>>

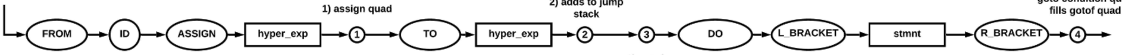


1) adds to the jumps stack

2) gotof quad

1) fills pending gotof quad, adds goto quad

<<non_conditonal>>



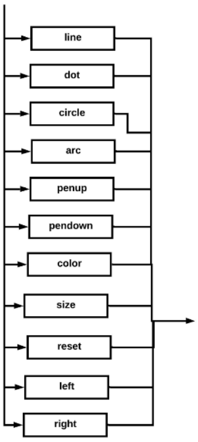
1) assign quad

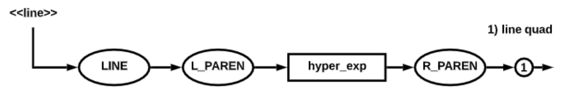
2) adds to jump stack

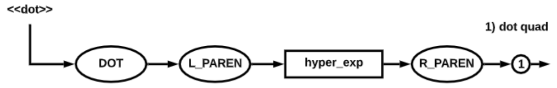
3) gotof

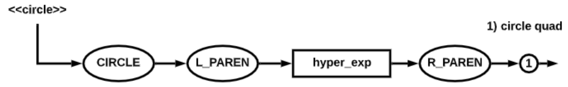
4) iterator++ goto condition quad fills gotof quad

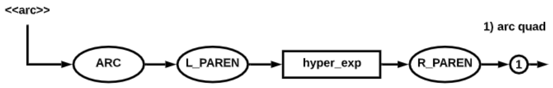
<<graphics>>

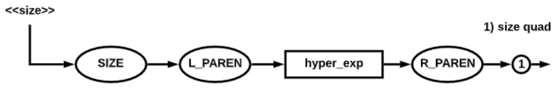


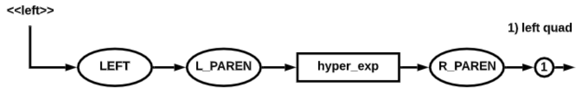


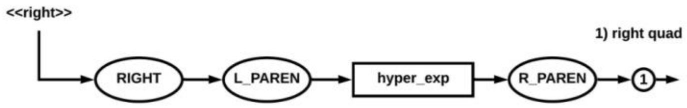


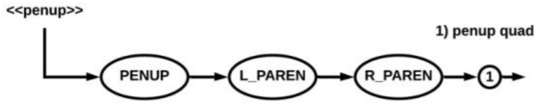


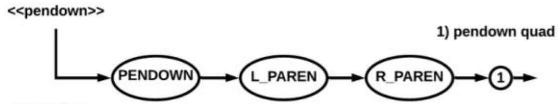


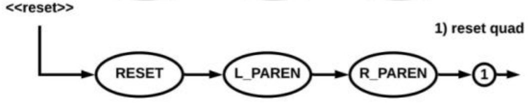


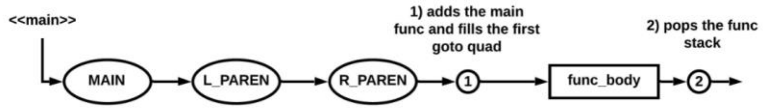












3.4.3 Semantic considerations

The following semantic considerations were made, any possible combination not on the table is not supported by the language.

Type 1	Operator	Type 2	Resulting type
Int	+	Int	Int
Int	-	Int	Int
Int	*	Int	Int
Int	/	Int	Int
Int	==	Int	Bool
Int	!=	Int	Bool
Int	<	Int	Bool
Int	<=	Int	Bool
Int	>	Int	Bool

Int	>=	Int	Bool
Int	&	Int	Bool
Int	|	Int	Bool
Int	=	Int	int
int	==	Float	Bool
Int	!=	Float	Bool
Int	<	Float	Bool
Int	<=	Float	Bool
Int	>	Float	Bool
Int	>=	Float	Bool
Int	&	Float	Bool
Int	|	Float	Bool
Float	==	Int	Bool
Float	!=	Int	Bool
Float	<	Int	Bool
Float	<=	Int	Bool
Float	>	Int	Bool
Float	>=	Int	Bool
Float	&	Int	Bool
Float	|	Int	Bool
Float	+	Float	Float
Float	-	Float	Float
Float	*	Float	Float
Float	/	Float	Float
Float	==	Float	Bool
Float	!=	Float	Bool
Float	<	Float	Bool
Float	<=	Float	Bool
Float	>	Float	Bool
Float	>=	Float	Bool
Float	&	Float	Bool
Float	|	Float	Bool
Float	=	Float	Float
Char	==	Char	Bool
Char	!=	Char	Bool
Char	<	Char	Bool
Char	<=	Char	Bool
Char	>	Char	Bool
Char	>=	Char	Bool
Char	=	Char	char

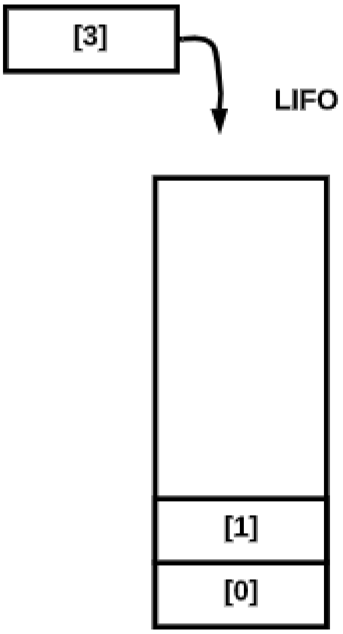
3.5 Memory management during compilation

During the compilation process, memory is mostly an illusion, since in compilation no variable values are known, no values are stored and hence the addresses corresponding to the variables are merely scope-specific and type-specific counters.

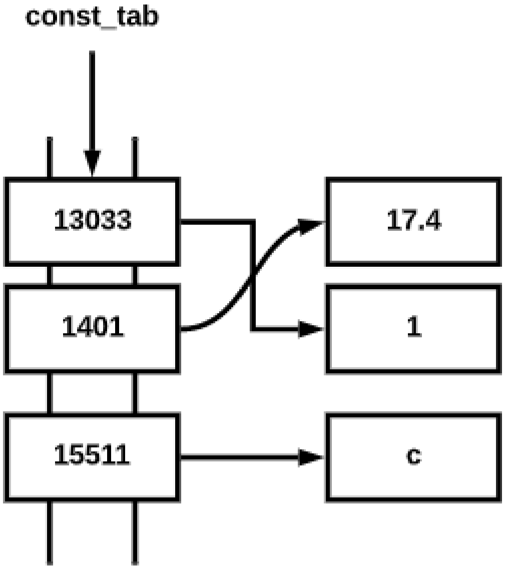
3.5.1 Data structures used during compilation

The following data structures are used during the compilation process:

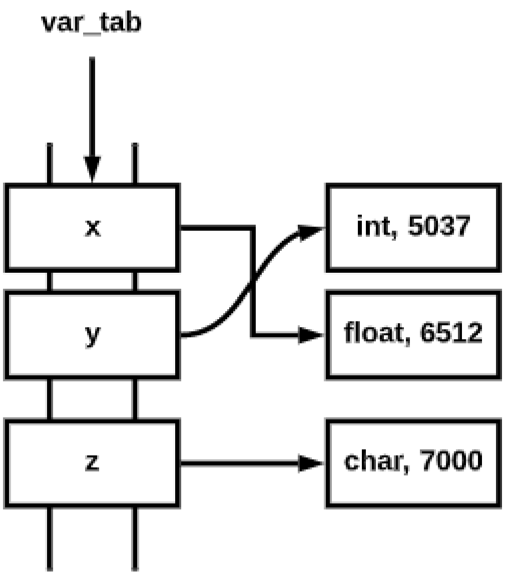
- Stacks: Several stacks are used in the form of python lists, lists are used in this context since the most common operations to be performed on them are pops, in python, popping a list has a 0(1) time complexity.



- Constants table: A very simple python dictionary, with the addresses (counters) being the keys for the values, although lists would have gotten the job done, in Python, dictionaries outperform lists when it comes to indexing.

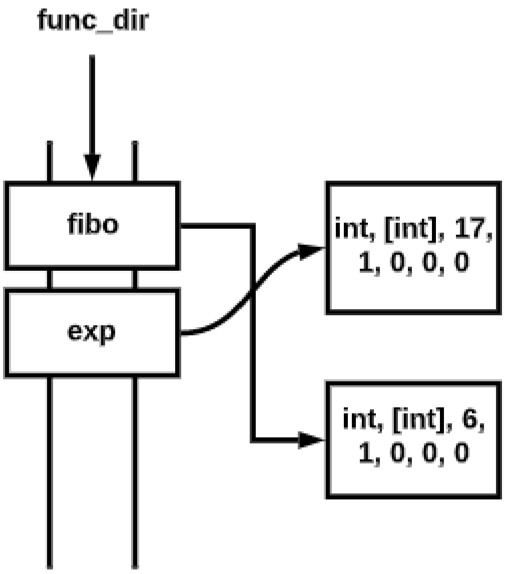


- Variables table: Also a Python dictionary, picked for the same reasons as before.



- ○

- Functions directory: Yet another Python dictionary



4 Virtual machine description

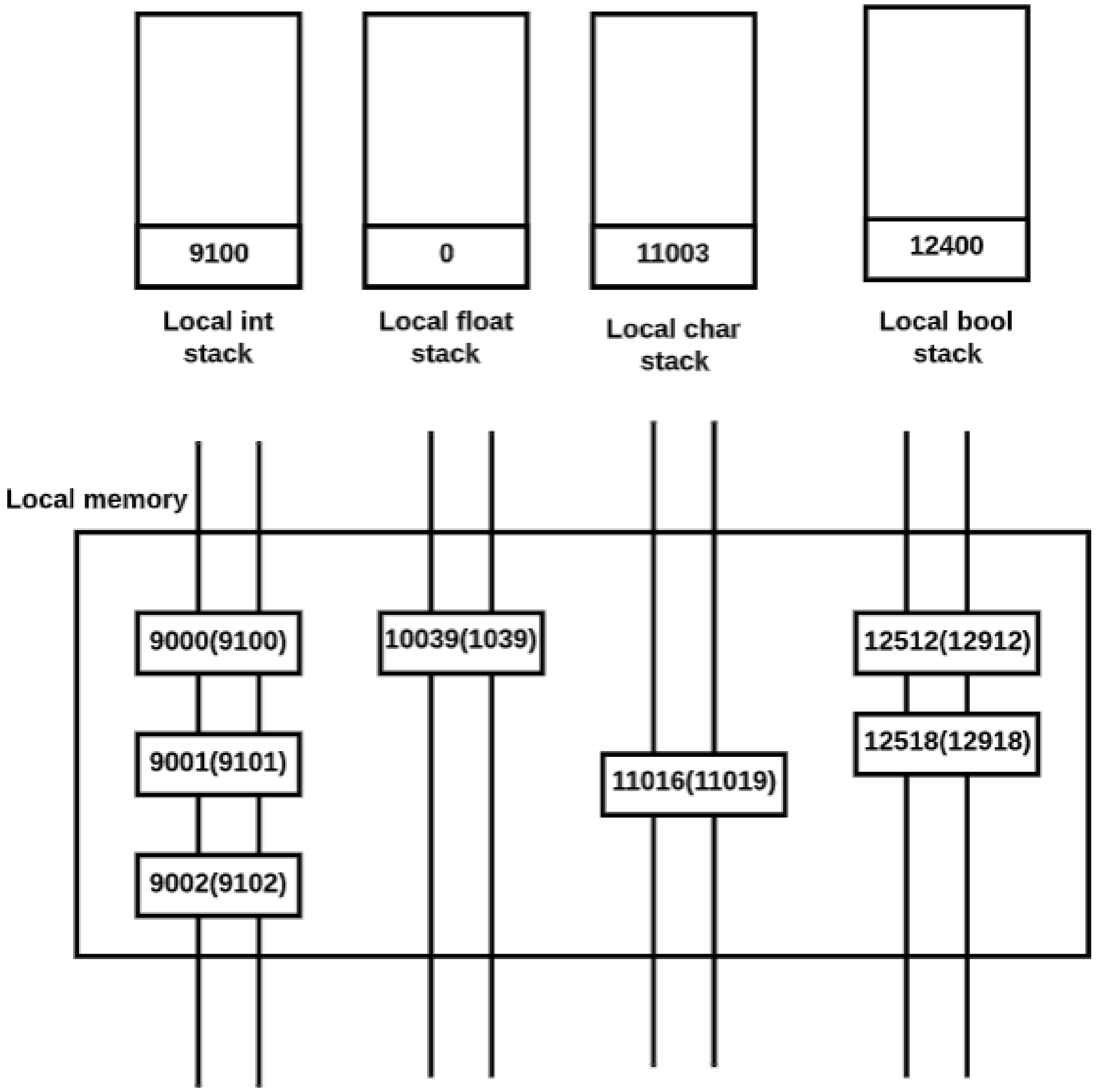
4.1 Tools used throughout development

The only tool used during execution which is not present in compilation is the Turtle Graphics library, which is a built-in Python library that provides graphical output capabilities, such as a drawing board and instructions.

4.2 Memory management during execution

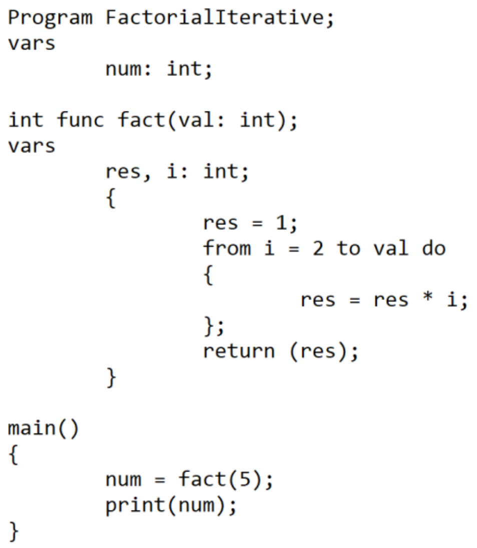
Memory on execution is made up of 12 memory strips implemented through 3 memories, each of 4 strips. The memories are the following: global memory, local memory, constant memory; and each one has an integer, float, char, and Boolean strip, which are dictionaries. The way scopes are implemented is through a series of scope offsets handled through , every time a gosub operation is performed, the amount of elements on each local strip gets added to the stacks as the current scope-offset, every time an endfunc operation is executed, memory from the outgoing scope gets released.

4.2.1 Data structures used to achieve the execution memory



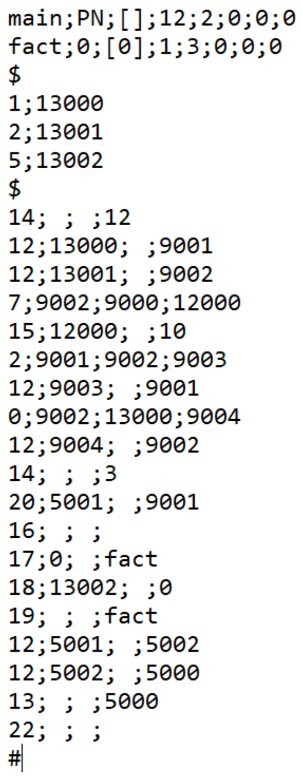
5 Tests performed

- Iterative factorial of 5:
    - Expected output: 120
    - Source code:



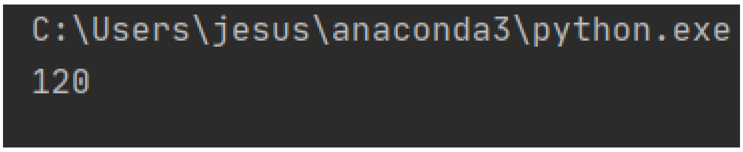
```
Program FactorialIterative;
vars
        num: int;

int func fact(val: int);
vars
        res, i: int;
        {
                res = 1;
                from i = 2 to val do
                {
                        res = res * i;
                };
                return (res);
        }

main()
{
        num = fact(5);
        print(num);
}
```

- Object code:



```
main;PN;[];12;2;0;0;0
fact;0;[0];1;3;0;0;0
$
1;13000
2;13001
5;13002
$
14; ; ;12
12;13000; ;9001
12;13001; ;9002
7;9002;9000;12000
15;12000; ;10
2;9001;9002;9003
12;9003; ;9001
0;9002;13000;9004
12;9004; ;9002
14; ; ;3
20;5001; ;9001
16; ; ;
17;0; ;fact
18;13002; ;0
19; ; ;fact
12;5001; ;5002
12;5002; ;5000
13; ; ;5000
22; ; ;
#
```

- Output:



```
C:\Users\jesus\anaconda3\python.exe
120
```

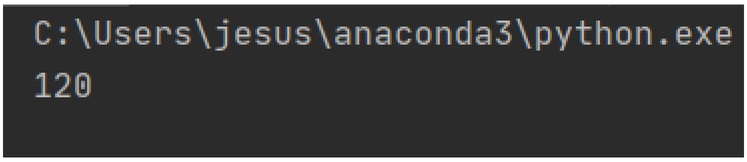
- Recursive factorial of 5:
    - Expected output: 120
    - Source code:

```
Program FactorialRecursive;
vars
        a: int;

int func fact(val: int);
        {
                if ( val == 0) then
                {
                        return (1);
                }
                else
                {
                        return (fact(val - 1) * val);
                };
        }

main()
{
        a = 5;
        print(fact(a));
}
```

- Object code:

```
main;PN;[];13;2;0;0;0
fact;0;[0];1;1;0;0;0
$
0;13000
1;13001
5;13002
$
14; ; ;13
4;9000;13000;12000
15;12000; ;5
20;5001; ;13001
14; ; ;12
17;0; ;fact
1;9000;13001;9001
18;9001; ;0
19; ; ;fact
12;5001; ;9002
2;9002;9000;9003
20;5001; ;9003
16; ; ;
12;13002; ;5000
17;0; ;fact
18;5000; ;0
19; ; ;fact
12;5001; ;5002
13; ; ;5002
22; ; ;
#
```

- Output:



```
C:\Users\jesus\anaconda3\python.exe
120
```

- Iterative Fibonacci for the 9th element:

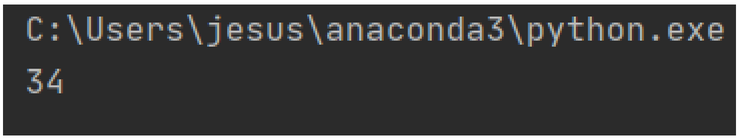
- Expected output: 34
- Source code:

```
Program FaibonacciIterative;
vars
        num: int;

int func fibo(val: int);
vars
        a, b, c, i: int;
        {
                a = 0;
                b = 1;
                if (val == 0) then
                {
                        return (a);
                };
                from i = 2 to val do
                {
                        c = a + b;
                |       a = b;
                        b = c;
                };
                return (b);
        }

main()
{
        num = 9;
        print(fibo(num));
}
```

- Object code:

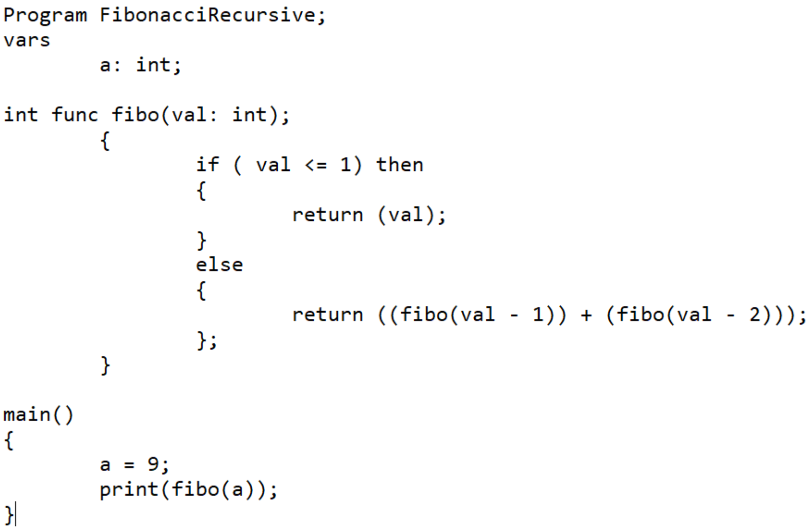
```
main;PN;[];18;2;0;0;0
fibo;0;[0];1;5;0;0;0
$
0;13000
1;13001
2;13002
9;13003
$
14; ; ;18
12;13000; ;9001
12;13001; ;9002
4;9000;13000;12000
15;12000; ;6
20;5001; ;9001
12;13002; ;9004
7;9004;9000;12001
15;12001; ;16
0;9001;9002;9005
12;9005; ;9003
12;9002; ;9001
12;9003; ;9002
0;9004;13001;9006
12;9006; ;9004
14; ; ;7
20;5001; ;9002
16; ; ;
12;13003; ;5000
17;0; ;fibo
18;5000; ;0
19; ; ;fibo
12;5001; ;5002
13; ; ;5002
22; ; ;
#
```

- Output:



```
C:\Users\jesus\anaconda3\python.exe
34
```

- Recursive Fibonacci for the 9th element:
    - Expected output: 34
    - Source code:

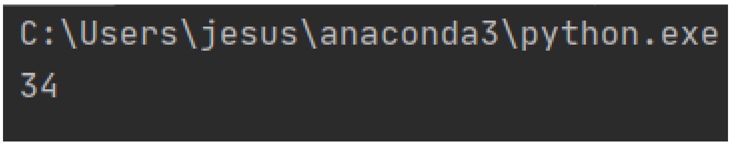


```
Program FibonacciRecursive;
vars
        a: int;

int func fibo(val: int);
        {
                if ( val <= 1) then
                {
                        return (val);
                }
                else
                {
                        return ((fibo(val - 1)) + (fibo(val - 2)));
                };
        }

main()
{
        a = 9;
        print(fibo(a));
}
```

- Object code:

```
main;PN;[];18;2;0;0;0
fibo;0;[0];1;1;0;0;0
$
1;13000
2;13001
9;13002
$
14; ; ;18
7;9000;13000;12000
15;12000; ;5
20;5001; ;9000
14; ; ;17
17;0; ;fibo
1;9000;13000;9001
18;9001; ;0
19; ; ;fibo
12;5001; ;9002
17;0; ;fibo
1;9000;13001;9003
18;9003; ;0
19; ; ;fibo
12;5001; ;9004
0;9002;9004;9005
20;5001; ;9005
16; ; ;
12;13002; ;5000
17;0; ;fibo
18;5000; ;0
19; ; ;fibo
12;5001; ;5002
13; ; ;5002
22; ; ;
#
```

- Output:

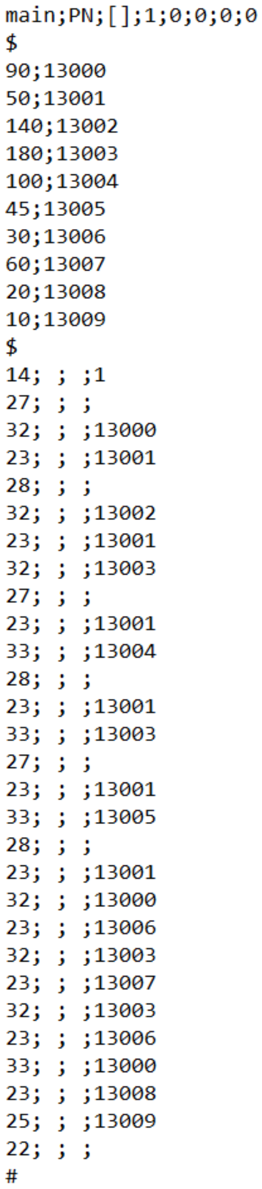


```
C:\Users\jesus\anaconda3\python.exe
34
```

- Draw a person:
    - Expected output: Something resembling a person
    - Source code:

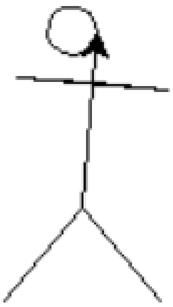
```
Program Person;

main()
{
        penup();
        left(90);
        line(50);
        pendown();
        left(140);
        line(50);
        left(180);
        penup();
        line(50);
        right(100);
        pendown();
        line(50);
        right(180);
        penup();
        line(50);
        right(45);
        pendown();
        line(50);
        left(90);
        line(30);
        left(180);
        line(60);
        left(180);
        line(30);
        right(90);
        line(20);
        circle(10);
}
```

- Object code:



```
main;PN;[];1;0;0;0;0
$
90;13000
50;13001
140;13002
180;13003
100;13004
45;13005
30;13006
60;13007
20;13008
10;13009
$
14; ; ;1
27; ; ;
32; ; ;13000
23; ; ;13001
28; ; ;
32; ; ;13002
23; ; ;13001
32; ; ;13003
27; ; ;
23; ; ;13001
33; ; ;13004
28; ; ;
23; ; ;13001
33; ; ;13003
27; ; ;
23; ; ;13001
33; ; ;13005
28; ; ;
23; ; ;13001
32; ; ;13000
23; ; ;13006
32; ; ;13003
23; ; ;13007
32; ; ;13003
23; ; ;13006
33; ; ;13000
23; ; ;13008
25; ; ;13009
22; ; ;
#
```

- Output:



6 References

- David M. Beazley. (2018). PLY (Python Lex-Yacc). 02/06/2021, de N.A Sitio web: https://www.dabeaz.com/ply/ply.html
- N.A. (2021). Turtle graphics. 02/06/2021, de The Python Software Foundation Sitio web: https://docs.python.org/3/library/turtle.html