

TC3048.1 Compilers Design

**Final Project.**

**MyStarlight Compiler**

**Team 1**

Tanya Yaretzi González Elizondo A00823408

Text, letter

Description automatically generated

José Alejandro Myrick Asturias A00819666

A picture containing text, clipart

Description automatically generated

June 06, 2022

**Index**

[Description and Technical Documentation 2](#_Toc105195791)

[Project Description 2](#_Toc105195792)

[Purpose and Scope 2](#_Toc105195793)

[Requirements Analysis and description of the main test cases 2](#_Toc105195794)

[Requirements 2](#_Toc105195795)

[Main Test Cases 2](#_Toc105195796)

[Project Follow-up 3](#_Toc105195797)

[Personal Reflections 5](#_Toc105195798)

[Language 5](#_Toc105195799)

[Language Name 5](#_Toc105195800)

[Language Description 5](#_Toc105195801)

[Errors List 5](#_Toc105195802)

[Compilation 5](#_Toc105195803)

[Execution 6](#_Toc105195804)

[Compiler 7](#_Toc105195805)

[Virtual Machine 7](#_Toc105195806)

[Performance Testing 7](#_Toc105195807)

[Code Documentation 7](#_Toc105195808)

[User’s Guide or User’s Manual? 7](#_Toc105195809)

[Quick Reference Manual 7](#_Toc105195810)

[Demo 7](#_Toc105195811)

# Description and Technical Documentation

## Project Description

### Purpose and Scope

The goal of this project is to create, design and implement a declarative object-oriented programming language to apply the knowledge and skills acquired through the Compilers Design course. First and foremost, we define the basics of a programming language, such as tokens, reserved words, single character literals, and the corresponding regular expressions that identify them. Furthermore, we define the syntax diagrams, context free grammar, neural points among other syntactical actions that let us parse and compile the program.

The language must be able to support global variables, local variables, functions, arithmetical, logical, and relational expressions, input/output operations, control flow statements, context management and non-atomic variables such as arrays and two-dimensional matrices. As mentioned before, we are developing an object-oriented language so classes with public attributes will be added as well as object instantiation and single inheritance.

### Requirements Analysis and description of the main test cases

#### Requirements

1. The language must follow the object-oriented paradigm
2. The language will support class inheritance
3. The language shall include int, float, char, and user-defined variables (objects).
4. The language must support arrays and two-dimensional matrices.
5. The language must have conditionals, cycles, and input/output operations (print and read).
6. The language will support parameterized functions, multiple return statements, and recursion.
7. The language performs arithmetical, logical, and relational operations.

#### Main Test Cases

1. Program with global variables, functions with local variables, conditionals, cycles, and input/output operations. Include basic arithmetical, logical, and relational operations.
2. Program with arrays and matrices operations.
3. Program with classes, inheritance and, accessing to object methods.
4. Program with recursive functions.

We aim to ensure with the test cases that MyStarlight Compiler and virtual machine execute expressions, classes, functions, parameters, multiple return statements, structured data i.e., arrays and matrices, classes including inheritance.

### Project Follow-up

During the definition weeks of the project, we worked together up to three times a week in defining our programming language’s scope this being tokens, syntax diagrams, context free grammar rules, and parsing tools for python.

Following this, we have worked collaboratively in the development of neural points embedded actions, as well as the design of semantic cube, memory structure, management of quadruples, structure of the virtual machine among other requirements. Although each of us had some participation generating the code, there were always an equivalent time working in the project either coding or designing the solutions.

Repository: <https://github.com/kcirym10/MyStarlight-Compiler>

Since our commit list is considerable, we will add just some commits for each week and the whole list will be in the next link: <https://github.com/kcirym10/MyStarlight-Compiler/commits/main>

|  |  |  |
| --- | --- | --- |
| Week | Progress | Commits |
| Week 0  May 28, 2022 | * Began with the language proposal | [35a3ef95a0c8f9877f74952ef04ce01f1da1c49b](https://github.com/kcirym10/MyStarlight-Compiler/commit/35a3ef95a0c8f9877f74952ef04ce01f1da1c49b) |
| Week 1  April 4, 2022 | * Regex for token matching * Designed language syntax diagrams * Created DNF rules based in syntax diagrams * Started project development and GitHub repository. | [d494aeb563ac724e10bec524cb1116b7ce4ca69d](https://github.com/kcirym10/MyStarlight-Compiler/commit/d494aeb563ac724e10bec524cb1116b7ce4ca69d) |
| Week 2  April 13, 2022 | * Received the approval of syntax diagrams and DNF rules * Implemented Lexer with complete token matching   Implemented Parser rules | [e90d69fe87cdea83cc798b5f6ec28cbd2ee196a9](https://github.com/kcirym10/MyStarlight-Compiler/commit/e90d69fe87cdea83cc798b5f6ec28cbd2ee196a9) |
| Week 3  April 20, 2022 | * Implemented the neural points * Created the semantic cube * Created VARS table and functions directory | [27991495d67243055eae32240b6668dce49f601b](https://github.com/kcirym10/MyStarlight-Compiler/commit/27991495d67243055eae32240b6668dce49f601b) |
| Week 4  April 24, 2022 | * Modified Symbol Table structure for Functions Directory and Variable tables * Implemented neural points for classes and class derivation * Began implementing expression quadruples class * Implemented generic quadruple processing logic | [995e4b0f52f8b20d98261c691d13b7c17b02ff7e](https://github.com/kcirym10/MyStarlight-Compiler/commit/995e4b0f52f8b20d98261c691d13b7c17b02ff7e) |
| Week 5  May 01, 2022 | * Created a Virtual Memory class and Avail * Constants are saved in global VARS Table with their virtual address * Variables are assigned a memory address * Avail and local addresses reset after exiting local scope * Expressions and assignment quadruples completed * IF-ELSE statement quadruples completed * WHILE statement quadruples completed * Removed parentRef and added address to symbol table records * Fixed bugs in semantic cube * Fixed bug which saved constants by their numeric value which meant floats and ints shared the same address | [12f9f419c17fccb5cf0bd6f5111e5f025afc1e40](https://github.com/kcirym10/MyStarlight-Compiler/commit/12f9f419c17fccb5cf0bd6f5111e5f025afc1e40) |
| Week 6  May 09, 2022 | * Began implementing functions | [82bbac281a11a7c5903f1bef3815e1da3386ad73](https://github.com/kcirym10/MyStarlight-Compiler/commit/82bbac281a11a7c5903f1bef3815e1da3386ad73) |
| Week 7  May 21, 2022 | * Implemented Error-Handling * Created quadruples for function definitions * Created quadruples for function calls * Fixed bugs in VARS Table * Modified records | [a1556bf9b4d4a0f1efbc28c26f9d01a24199641f](https://github.com/kcirym10/MyStarlight-Compiler/commit/a1556bf9b4d4a0f1efbc28c26f9d01a24199641f) |
| Week 8  May 25, 2022 | * Implemented virtual machine up to functions * Compiler now supports recursive functions | [d34e9b378629055ac79861db8248917cb494cc78](https://github.com/kcirym10/MyStarlight-Compiler/commit/d34e9b378629055ac79861db8248917cb494cc78) |

#### Personal Reflections

Alejandro Myrick:

A picture containing text, clipart

Description automatically generated

Tanya González:

Text, letter

Description automatically generated

## Language

### Language Name

We named our programming language “MyStarlight” because of our love of the country night skies when it is brimming with stars.

### Language Description

MyStarlight is an object-oriented language that supports single inheritance. Classes have public attributes and methods. Also, global variables, local variables, arrays, and two-dimensional matrices are supported. The accepted primitive types are ints, floats, chars; strings are only supported inside print statement.

### Errors List

#### Compilation

|  |  |
| --- | --- |
| *Function* | Error |
| *np\_save\_id(p),*  *np\_save\_func\_id(p)* | "Multiple declaration of key: \"{p[-1]}\"" |
| *np\_copy\_class\_record(p)* | "Undefined class derivation" |
| *np\_endfunc(p),*  *np\_func\_gosub(p)* | "Missing return in none-void function" |
| *np\_func\_call(p)* | "Undefined function call id: {p[-2]}" |
| *function\_return(p)* | "Return in void function detected" |
| *np\_push\_var\_operand(p)* | "Key: \"{p[-3]}\" is not defined" |
| *createIfTopIs(operator)* | "Type Mismatch",  “Expression error, posible assignment of void function” |
| *createGotoF()* | "ERROR: Expected bool result" |
| *createParam()* | "Type Mismatch in function call: {argType} and {self.currentSignature[-1][self.sigIndex[-1]][0]}" |
| *createGoSub()* | 'Too little arguments'  'Too many arguments' |
| *semantics(left\_type, right\_type, operator)* | "Invalid operator type: \"{operator}\""  "Invalid right operand type: \"{right\_type}\""  "Invalid left operand type: \"{left\_type}\"" |
| *saveVarRecord(key, value)* | "Multiple declaration of var key: \"{key}\"" |

### Execution

|  |  |
| --- | --- |
| *Quadruple Name* | Error |
| *ERA,*  *(Run Instructions)* | “Stack Overflow” |
| */* | "ERROR division by 0 not supported" |

## Compiler

### Computer equipment, language and special utilities used

MyStarlight compiler was developed using Windows 10 and Python versions 3.10.2 or above. On the other hand, SLY was the lexer and parsing tool chosen because its modern programming style than its successor PLY. It provides full support for empty productions, ambiguous grammars (shift/reduce, reduce/reduce conflicts). For SLY installation we follow the instructions of the original documentation.

<https://sly.readthedocs.io/en/latest/>

### Lexical Analysis

#### List of tokens

|  |  |
| --- | --- |
| Token name | Regular Expression |
| ignore | ' \t' |
| ignore\_newline(t) | r'\n+' |
| CTE\_CHAR | r"'\w'" |
| CTE\_STRING | r'\".\*?\"' |
| CLASS\_ID | r'[A-Z]\w\*' |
| ID | r'[a-z]\w\*' |
| NOT\_EQUAL\_TO | r'\!\=' |
| EQUAL\_TO | r'\=\=' |
| GREATER\_OR\_EQUAL\_TO | r'\>\=' |
| LESS\_OR\_EQUAL\_TO | r'\<\=' |
| CTE\_FLOAT | r'[0-9]+\.[0-9]+' |
| CTE\_INT | r'[0-9]+' |
| PROGRAM | ‘program’ |
| VAR | ‘var’ |
| INT | ‘int’ |
| FLOAT | ‘float’ |
| CHAR | ‘char’ |
| VOID | ‘void’ |
| CLASS | ‘class’ |
| DERIVES | ‘derives’ |
| METHODS | ‘methods’ |
| FUNC | ‘func’ |
| RETURN | ‘return’ |
| PRINT | ‘print’ |
| READ | ‘read’ |
| IF | ‘if’ |
| ELSE | ‘else’ |
| WHILE | ‘while’ |
| MAIN | ‘main’ |

|  |  |  |
| --- | --- | --- |
| Literals | | |
| ';' | ‘:’ | ‘&’ |
| '[' | ‘{‘ | ‘<’ |
| ']' | ‘}’ | ‘>’ |
| ',' | ‘=’ | ‘+’ |
| ‘(‘ | ‘.’ | ‘-‘ |
| ‘)’ | ‘|’ | ‘\*’ |
| ‘/’ |  |  |

### Syntactic Analysis

#### Context Free Grammars

program 🡪 PROGRAM ID ; opt\_vars opt\_classes opt\_funcs main end

opt\_vars 🡪 'vars' | 'eps'

opt\_classes 🡪 ‘classes' | 'eps'

opt\_funcs 🡪 'functions' | 'eps'

vars 🡪 ‘VAR var\_type’

### Intermediate Code Generation and Semantic Analysis

#### Operation code and virtual addresses associated with code elements

The structure of our quadruples is as follows:

*operator left\_operand right\_operand result*

There are some exceptions in the quadruples like print statements which only consists of an operator (print) and a result, left and right operands are left as None.

For the management of the virtual memory, we decide to divide it as global segment, local segment, temporary segment, and constant segment.

|  |  |  |
| --- | --- | --- |
| Segment | Type | Address |
| Global Segment | int | 0 - 1999 |
| float | 2000 - 3999 |
| char | 4000 - 5999 |
| Local Segment | int | 6000 - 7999 |
| float | 8000 – 9999 |
| char | 10000 – 11999 |
| Temporary Segment | int | 12000 – 13999 |
| float | 14000 – 15999 |
| bool | 16000 – 17999 |
| Constant Segment | int | 18000 – 19999 |
| float | 20000 – 21999 |
| char | 22000 - |

Local and Temporary segments only lives as long as the function declaration is active.

#### Syntactic Diagrams with Neural Points

Diagramas de Sintaxis con las acciones correspondientes marcadas sobre ellos (puntos neurálgicos)

-Breve descripción de cada una de las acciones semánticas y de generación de código (no más de 2 líneas).

#### Semantic Cube

For the management of the semantic cube enumerators were used to take advantage of its properties where if the key is not found, it returns a "None" value in order to save mapping all possible combinations (including errors).

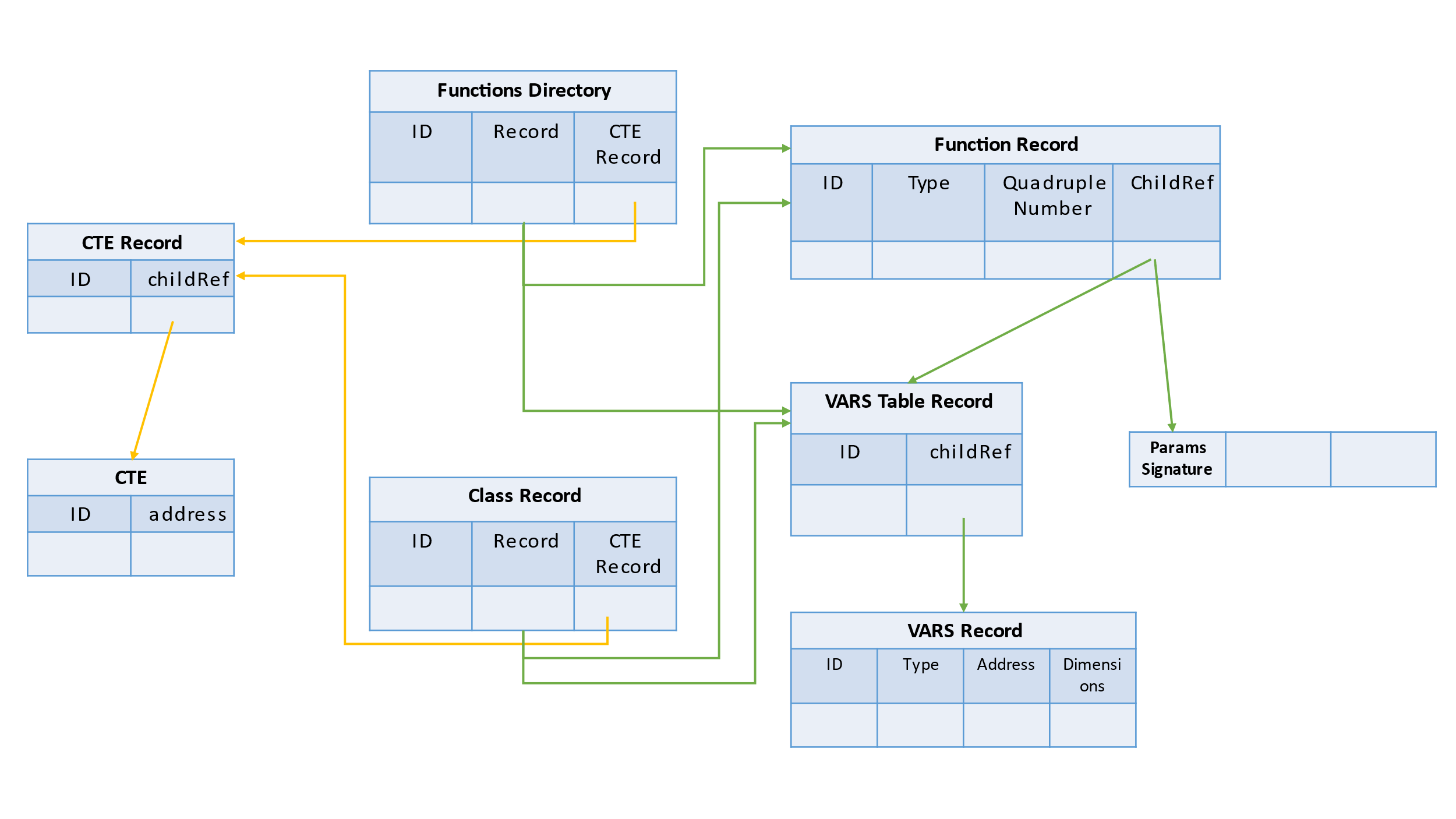
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Left operand | Right operand | + | - | \* | / | = | > | < | >= | <= | == | != | & | | |
| int | **int** | int | int | int | int | int | bool | bool | bool | bool | bool | bool | none | none |
| int | **float** | float | float | float | float | none | bool | bool | bool | bool | bool | bool | none | none |
| float | **float** | float | float | float | float | float | bool | bool | bool | bool | bool | bool | none | none |
| float | **int** | float | float | float | float | float | bool | bool | bool | bool | bool | bool | none | none |
| char | **char** | none | none | none | none | char | none | none | none | none | none | none | none | none |
| bool | **bool** | none | none | none | none | none | none | none | none | none | none | none | bool | bool |

### Memory Management

Especificación gráfica de CADA estructura de datos usada y su JUSTIFICACIÓN. (Dir.Func., Tablas de Var's,

Cuádruplos, Pilas, etc..

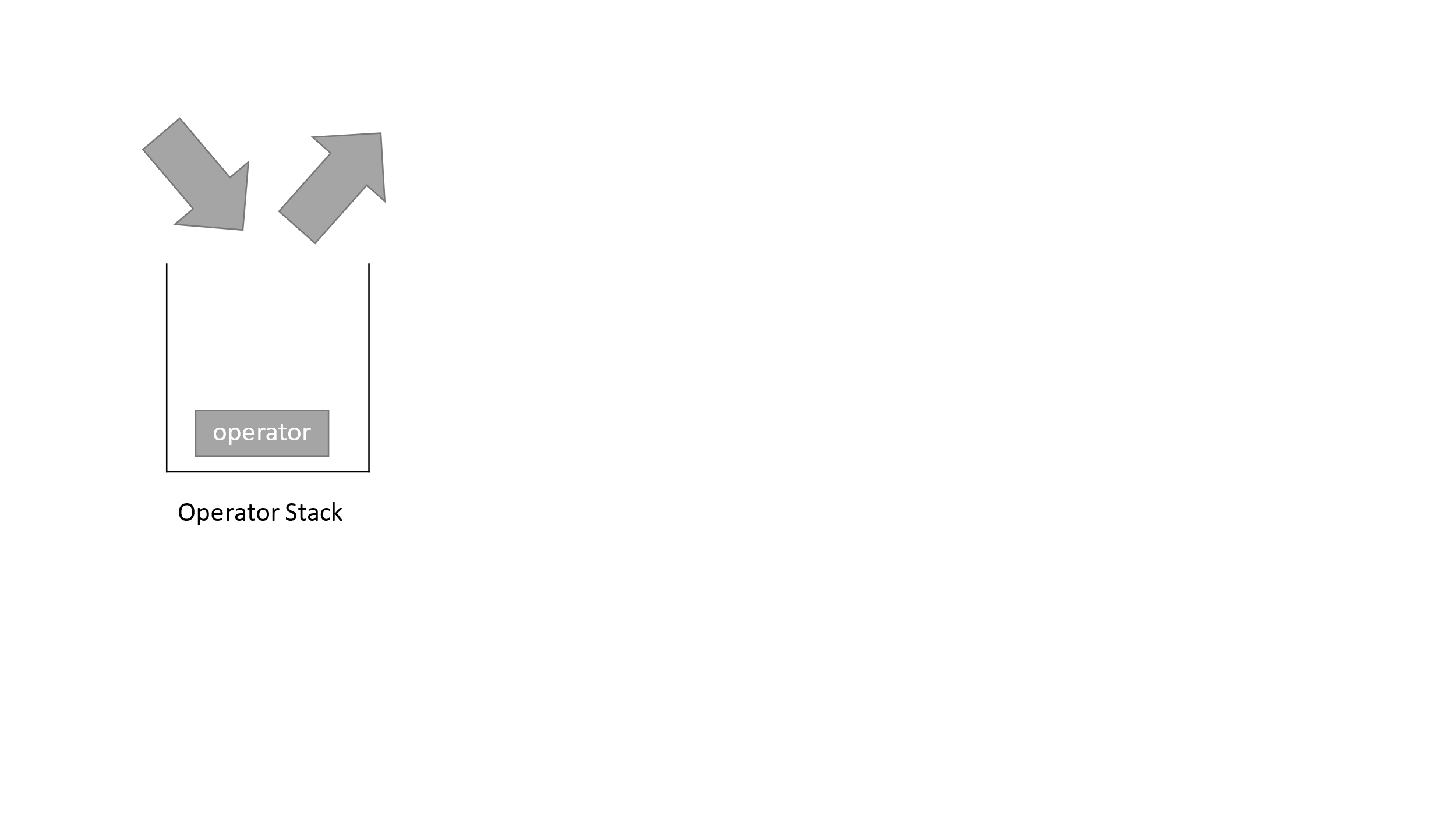
#### Structure of Symbol Table



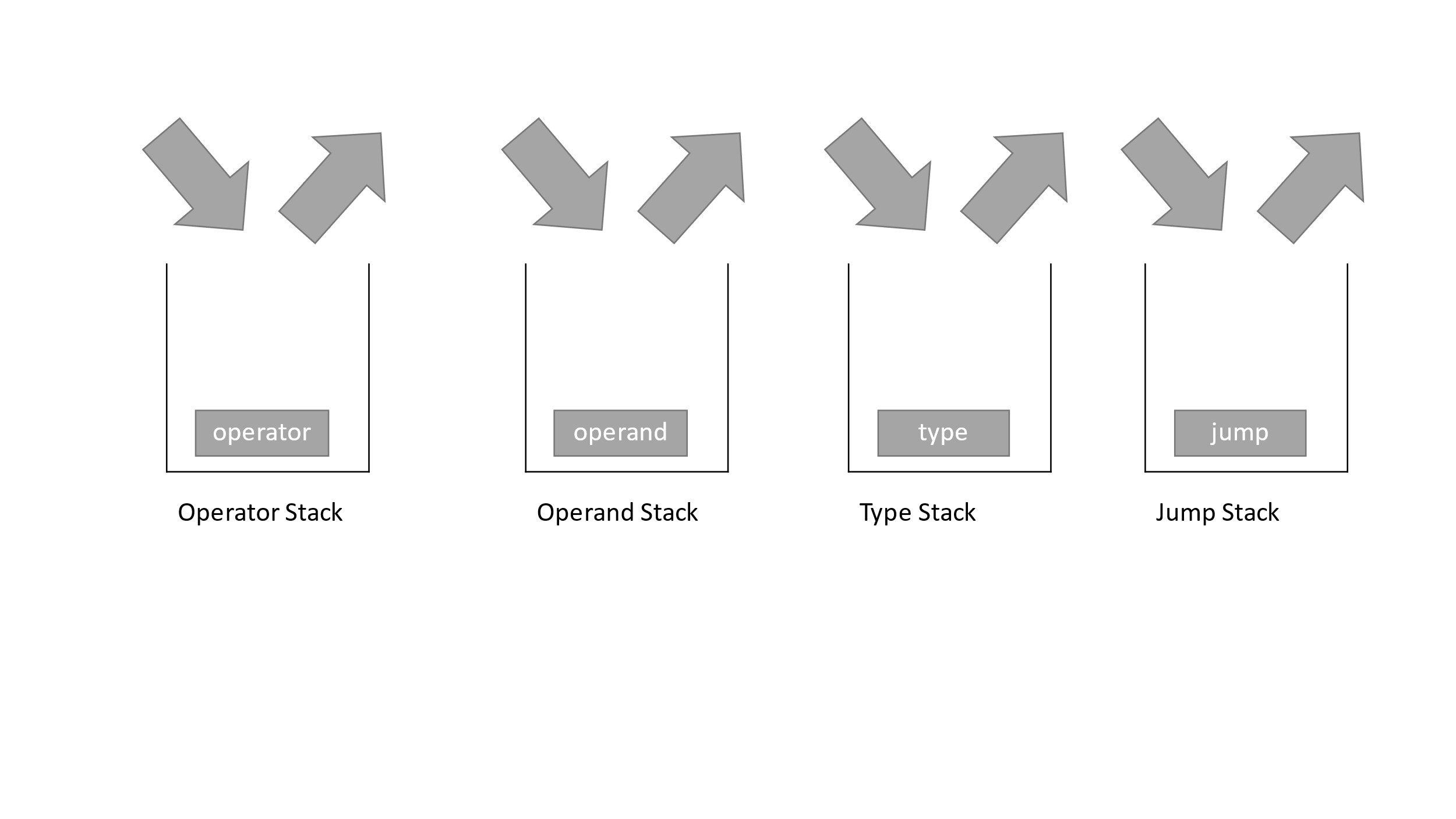
For the structure of the directory function and vars table we create two classes, Symbol Table Manager and Symbol Table. The Symbol Table Manager manages the active symbol/variable table. It makes use of a stack to manage the top/active table and is the working interface between the translation process. The Symbol Table contains its own scope's reference declarations as well as an optional variables table. The scope's references are those of function names, the program name or any other function-like declaration.

We decide to do it this way in order to reuse the Symbol table structure instead of creating a structure for every table needed. Symbol Manager is a stack of dictionaries.

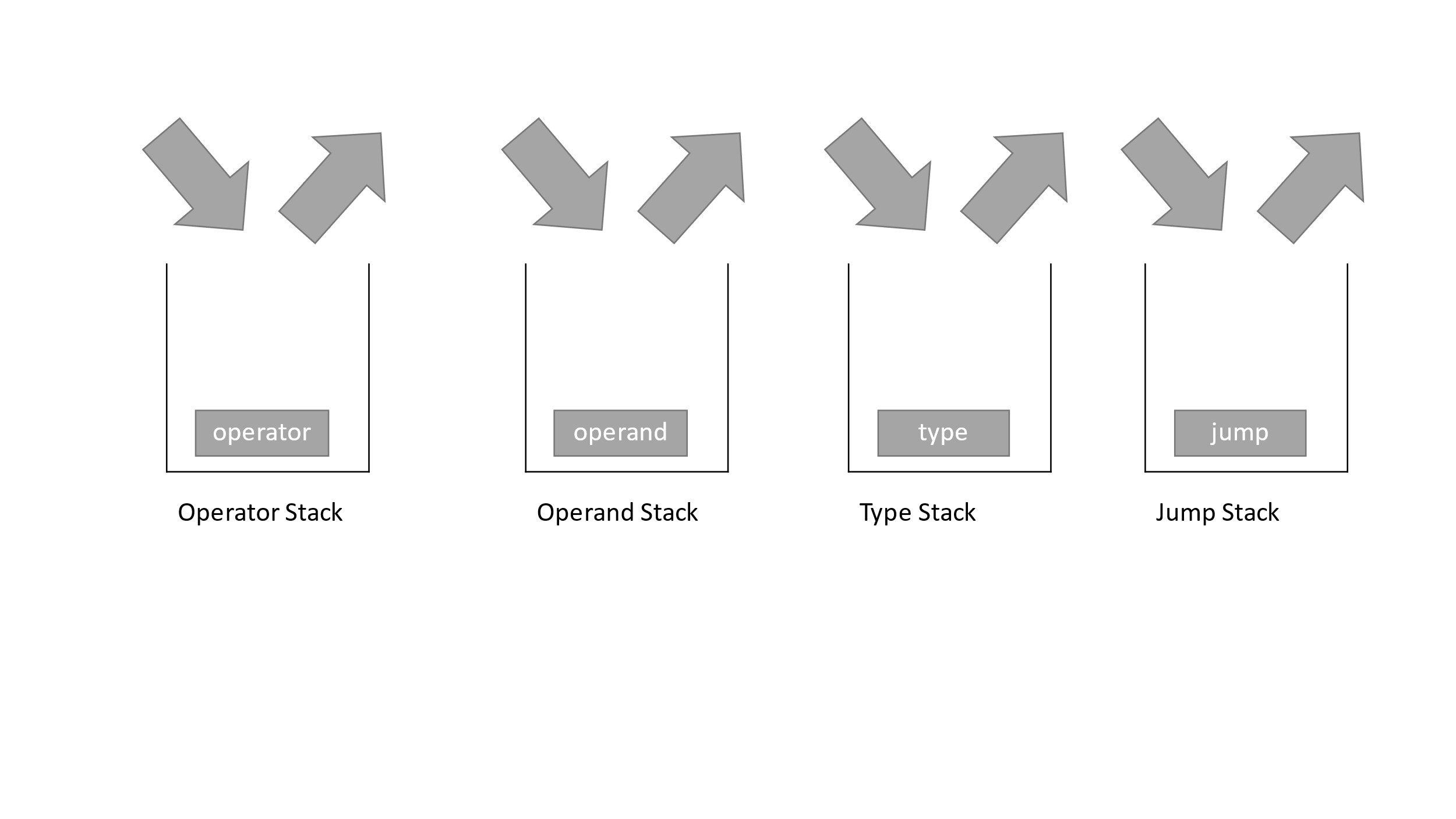
#### Quadruples



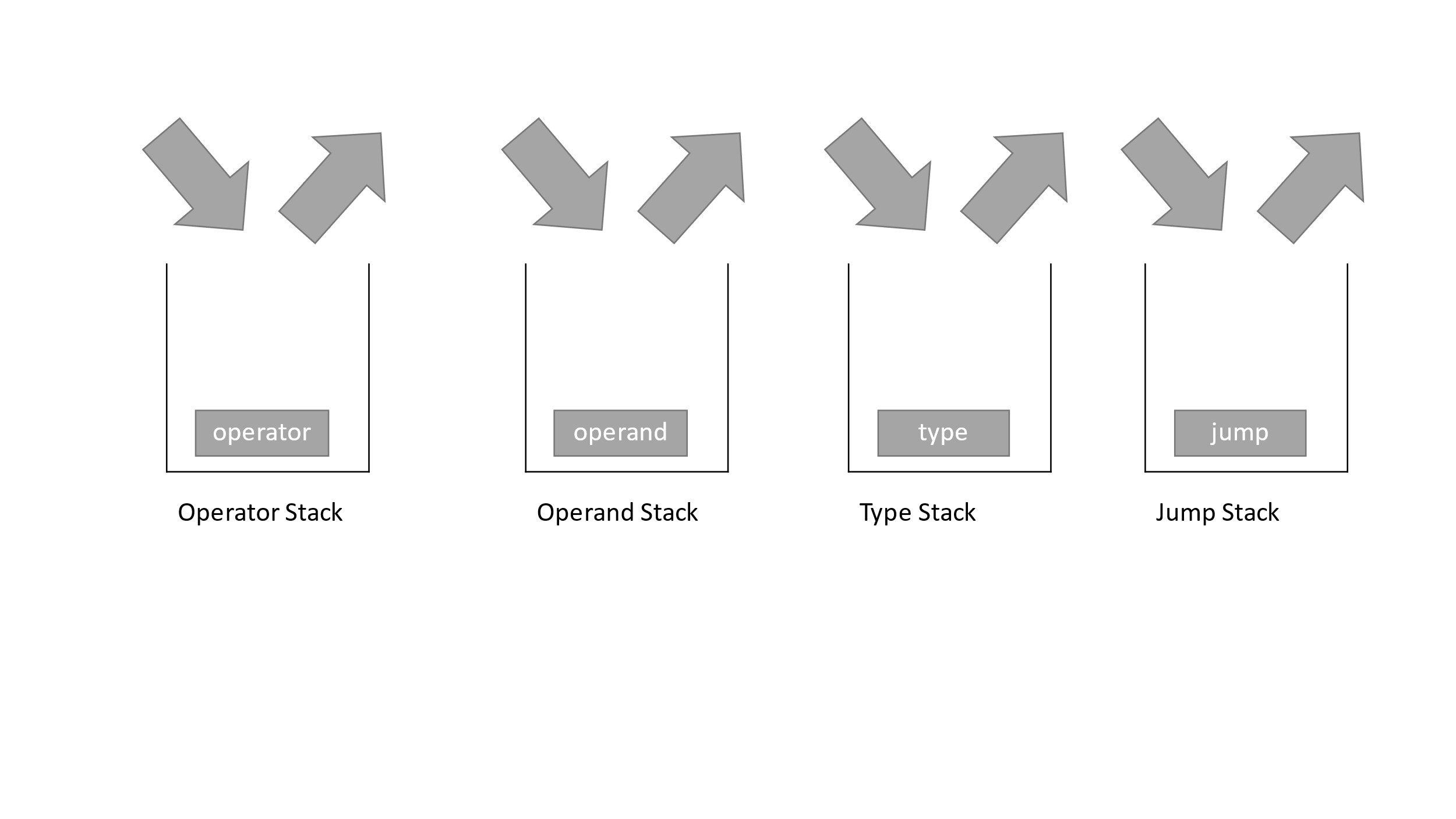
The operator stack helps us to keep track about the operator’s hierarchy, we use stacks to take advantage of its properties (LIFO) that allow us to have this hierarchy.



The operand stack helps us to keep track about the operands order, we use stacks to take advantage of its properties (LIFO) that allow us to keep left and right operands order.



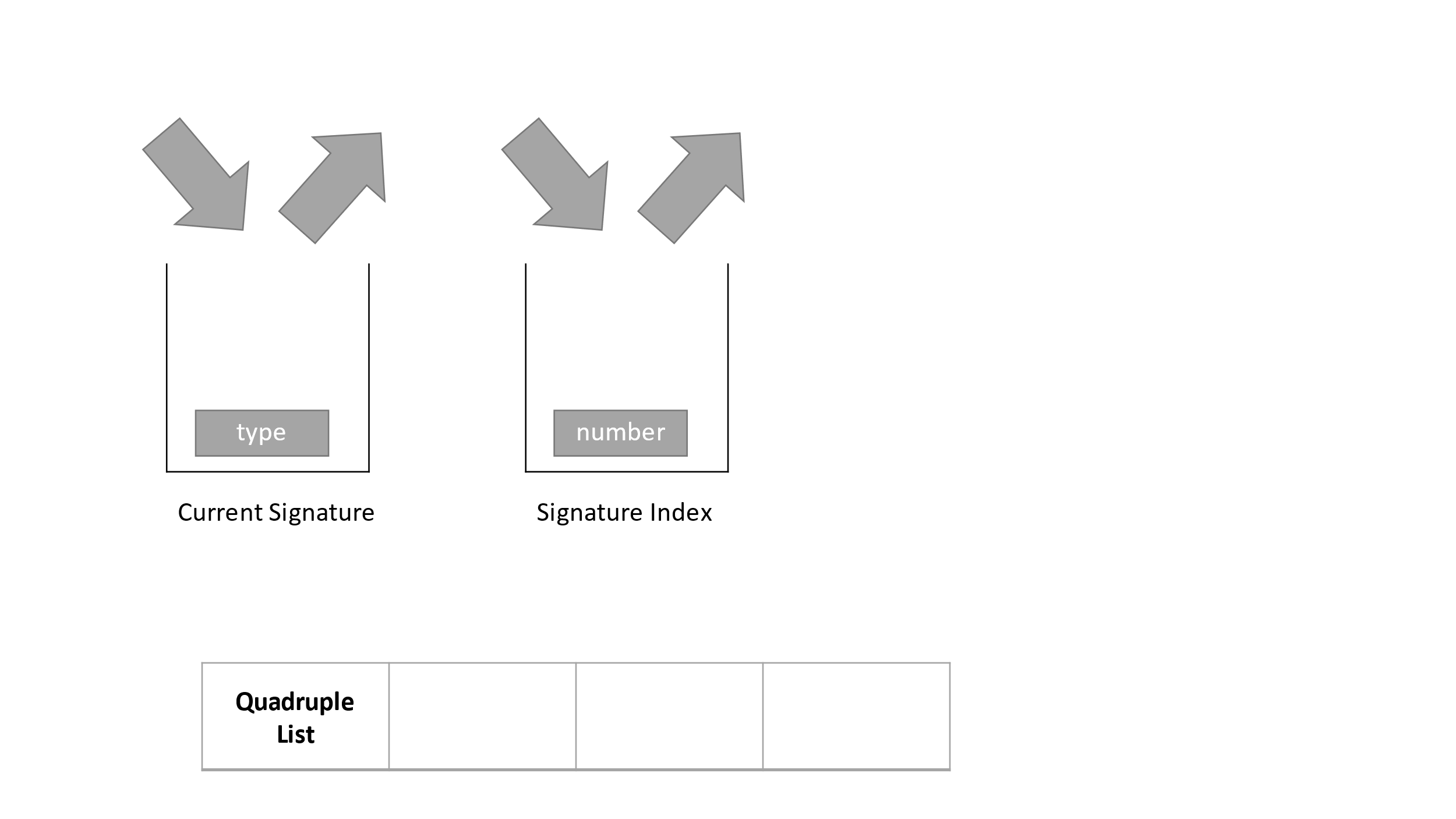
The type stack helps us to keep track about the type of the operands, we use stacks to take advantage of its properties (LIFO) that allow us to get the correct type for each operand.



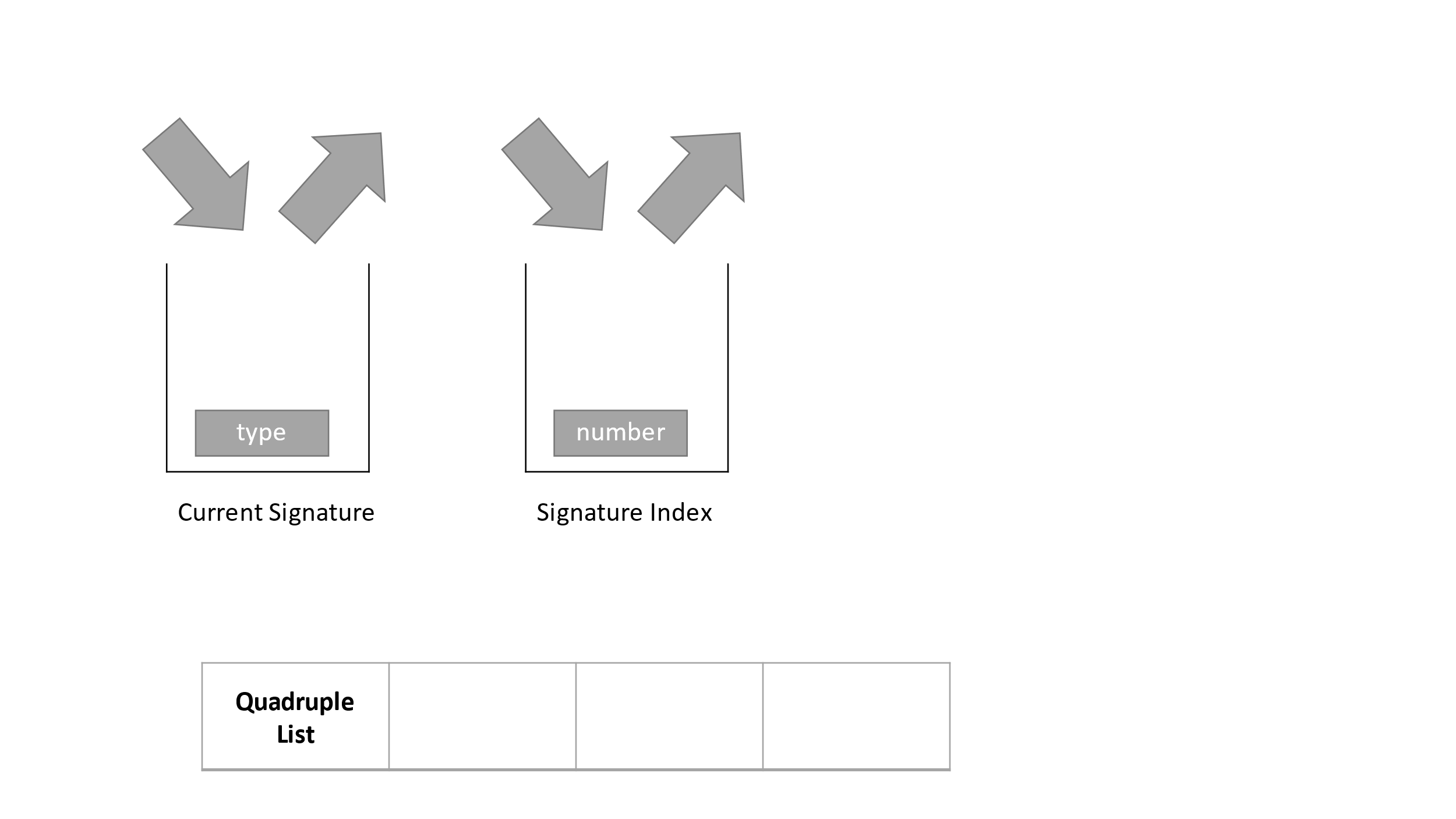
The jump stack helps us to keep track about the jumps the program must do, we use stacks to take advantage of its properties (LIFO) that allow us to get the correct quadruple number where the program must jump (go to’s).

|  |  |  |  |
| --- | --- | --- | --- |
| **Quadruple List** |  |  |  |

We create a quadruple list to save the generated quadruples, we decide to use a list instead of a stack because we want to access to any part of the list.



Current Signature is a stack that stores the parameters type.



Signature Index is a stack that stores the index of the current parameter of a function.

…

## Virtual Machine

### Computer equipment, language and special utilities used

As mentioned before, MyStarlight compiler was developed using Windows 10 and Python versions 3.10.2 or above. No special utilities were installed.

### Description of Memory Management in execution

o Especificación gráfica de CADA estructura de datos usada para manejo de scopes y su JUSTIFICACIÓN. (Memoria Local, global, etc..)

* Asociación hecha entre las direcciones virtuales (compliación) y las reales (ejecución).

Son las mismas solo que manejamos scopes por medio de stacks, es decir en un nuevo stack metemos el segmento de memoria (o contexto actual) que maneja las mismas direcciones que compilación haciéndolas reutilizables.

## Performance Testing

### Fibonacci

#### Recursive

#### Iterative

### Factorial

#### Recursive

program factorial;

func int recursiveFactorial(int n){

if(n > 1){

return(n \* recursiveFactorial(n-1));

}

else{

return(1);

}

}

main()

var

int numberUser;

{

read(numberUser);

print(recursiveFactorial(numberUser));

}

#### Iterative

### Sort

### Find

### Matrix Multiplication

Others (Pruebas específicas del funcionamiento de su tipo de proyecto. Hay que hacer lucir "a su hija" en su presentación en Sociedad)

## Code Documentation

# User’s Guide

## Quick Reference Manual

MyStarlight is an object-oriented programming language which enclosed all the concepts seen in Design of Compilers course. Our compiler works like Java, it has a compiler and a virtual machine.

### Getting Started

#### Requirements

* [Python 3.10.2 or above](https://www.python.org/downloads/)
* [SLY](https://github.com/dabeaz/sly)

#### Installation

1. Clone or download our [GitHub repository](https://github.com/kcirym10/MyStarlight-Compiler).
2. Install SLY using command:

**pip install sly**

#### Running the compiler with virtual machine

In order to compile the code, run the following command where name of file is your test program:

**python myStarlight.py <nameofFile.txt>**

*Example: python MyStarlight.py test.txt*

*Note: file must be in src/compilers folder.*

On a successful compilation (no errors found), the compiler creates a new file on the working directory (src) named ‘out.obejota’ containing code that the virtual machine will execute.

*Extra information, the ‘out.obejota’ contains information about the constants value with virtual addresses and the list of quadruples.*

On a failed compilation, the compiler will display a syntaxis error or a list of errors, and the virtual machine code will not be generated nor executed (‘out.obejota’ will be empty).

#### Running specific modules

If you want to run a specific module (aka parser.py) you must follow the next command:

**python -m compiler.<pythonModule>**

*Example: python -m compiler.parser*

*Note: Remember that you must have an if \_\_name\_\_ == '\_\_main\_\_': in the particular python file you want to execute.*

### Language Reference

#### Program Name

This MUST be the first line of your code. It is the name that your program will be identified.

**program name;**  *The name of the program must start with a lowercase letter.*

**Example:** program patito;

#### Primitive types

The most basic data types available in MyStarlight are:

* int
* float
* char

Strings are only supported in print statements, and Booleans exits only in logical operations.

#### Declaration of variables

Variables must be declared after the program name or after the function signature declaration with the next format:

**var**

**<primitiveType> <varName>;**

You can declare several variables within the same primitive type and type of variable (simple or complex) separated with commas:

**var**

**<primitiveType> <varName1>, <varName2>, <varName3> ;**

*Note: Complex variables are arrays and matrices.*

**Example:**

var

int a, b;

int d[10], m[1,2];

float l;

char c;

#### Statements

Display in console:

**print(<variable | constant | function | string | char>);**

*Note: strings are only allowed on print statements*

Read from console:

**read(<variable>);**

Assign

**<variable> = <variable | constant | function>**

Conditional:

**if(<expression>) {**

**…**

**} else {**

**…**

**}**

*Note: Else clause is optional.*

Cycles:

**while(<expression>) {**

**…**

**}**

Call Functions:

**<funcName>(<params>);**

Return:

**return( <variable> | <constant> | <function>)**

#### Functions

Functions must be declared before main() and after global variables (if exists). The syntaxis of functions is:

**func <primitiveType> <name> ( <params> ) {**

**<*statements>;***

**…**

**}**

Parameter declaration is <primitiveType> <name>. For all non-void functions, a return statement must be included.

**Example:**

func void dos(int a, int b, float g)

var

int i;

{

i = b;

while (i > 0){

a = a + b \* i + b;

print(a);

i = i - 1;

}

}

#### Expressions

#### Classes

##### Simple Class

##### Single Inheritance

#### Full Examples

## Demo