# 

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

June 06, 2022

**Index**

[**Description and Technical Documentation**](#_h1hm8xc7huzu) **2**

[Project Description](#_7ris5i5n51l) 2

[Language](#_854glbq6qq0b) 2

[Compiler](#_1bjy4qvtzadh) 2

[Virtual Machine](#_uog7fj1bn0ge) 2

[Performance Testing](#_whmff066hwv) 2

[Code Documentation](#_ogg3v5lh7khl) 2

[**User’s Guide**](#_fsfiygvirene) **2**

[Quick Reference Manual](#_clrfl3hexqss) 2

[Demo](#_zi2zpekswz40) 2

# 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

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

|  |  |
| --- | --- |
| Week | Progress |
| Week 0 | * Began with the language proposal |
| Week 1 | * Regex for token matching * Designed language syntax diagrams * Created DNF rules based in syntax diagrams * Started project development and GitHub repository. |
| Week 2 | * Received the approval of syntax diagrams and DNF rules * Implemented Lexer with complete token matching   Implemented Parser rules |
| Week 3 | * Implemented the neural points * Created the semantic cube * Created VARS table and functions directory |
| Week 4 | * 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 |
| Week 5 | * 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 |
| Week 6 | * Began implementing functions |
| Week 7 | * Implemented Error-Handling * Created quadruples for function definitions * Created quadruples for function calls * Fixed bugs in VARS Table * Modified records |

#### Personal Reflections

Alejandro Myrick:

Tanya González:

## Language

## Compiler

## 

## Virtual Machine

## 

## Performance Testing

## 

## Code Documentation

# User’s Guide or User’s Manual?

## Quick Reference Manual

## 

## Demo