UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

PML

Final Report

By:
Coralis Camacho Rodríguez (802-11-1065)
Mikael J. Del Valle Rodríguez (802-12-2035)
Eric J. Barbosa López (802-12-0625)
Christopher I. Maysonet Delgado (802-12-4125)

Dr. Wilson Rivera Gallego
wilson.riveragallego@upr.edu
ICOM 4036 – Programming Languages

Introduction

In this project, we propose a helpful tool to help in the project management specifically in the planning and design steps. For this, a programming language will be implemented in which users can program their own brainstorming, schedule and generate graphs. Such a language will include in its implementation a basics forms to personalize this tasks like different colors and different icons. The goal of “PM Language” is that a team member be able to implement their own tools to help in the planning and design steps, depending of the project needs.

Language Tutorial

To execute the program, we must first go to **PML directory** and run the **Parser.py** script. Then, we can follow this code fragment to see all features of PML in action:

```
#Creating new project
new_project Demo of PML

#Managing Project Members
add_member Coralis 1
add_member Mikael 1
add_member Eric 1
add_member Christopher 1
add_member Wilson 1
view_members
delete_member 5 1
view_members

#Creating Brainstorming
create_brainstorm Days of week 1
add_idea Monday 1
add_idea Friday 1
add_idea Saturday 1
add_idea Sunday 1
view_brainstorm 1

#Managing Project Tasks
add_task Edit video for Demo 08-05-2018 15-05-2018 1
add_task Add music to video 10-05-2018 17-05-2018 1
add_task Create project website 20-05-2018 24-05-2018 1
add_task Update documentation 22-05-2018 28-05-2018 1
assign_task 1 1 1
assign_task 2 2 1
assign_task 3 3 1
complete_task 3 1
view_tasks 1
list_week 1
list_today 1
edit_task 2 10-05-2018 20-05-2018 1
list_today 1
```

```
list_overdue 1

#Generating Gantt Chart
view_schedule 1

# Generating project Text file
generate_project 1

#Generating Graphs
generate_pie (apple,grape,orange) (50,25,25)
generate_bar (apple,grape,orange) (50,25,25)
generate_line [x] [y] (1,2,3) (3,2,1)
```

Language Reference Manual

DATA TYPES:

- **NAME** - string without spaces
- **DATE** - date format is dd-mm-yyyy. e.g. 11–09-2001 represent September 9, 2001
- **NUMBER** - integer number
- **PHRASE** - string containing spaces
- **NUMBERLIST** - a list of integer numbers separated by commas. e.g. (1,2,3)
- **NAMELIST** - list of strings separated by commas. e.g. (ab,bc,cd)
- **LBRK** - [
- **RBRK** -]

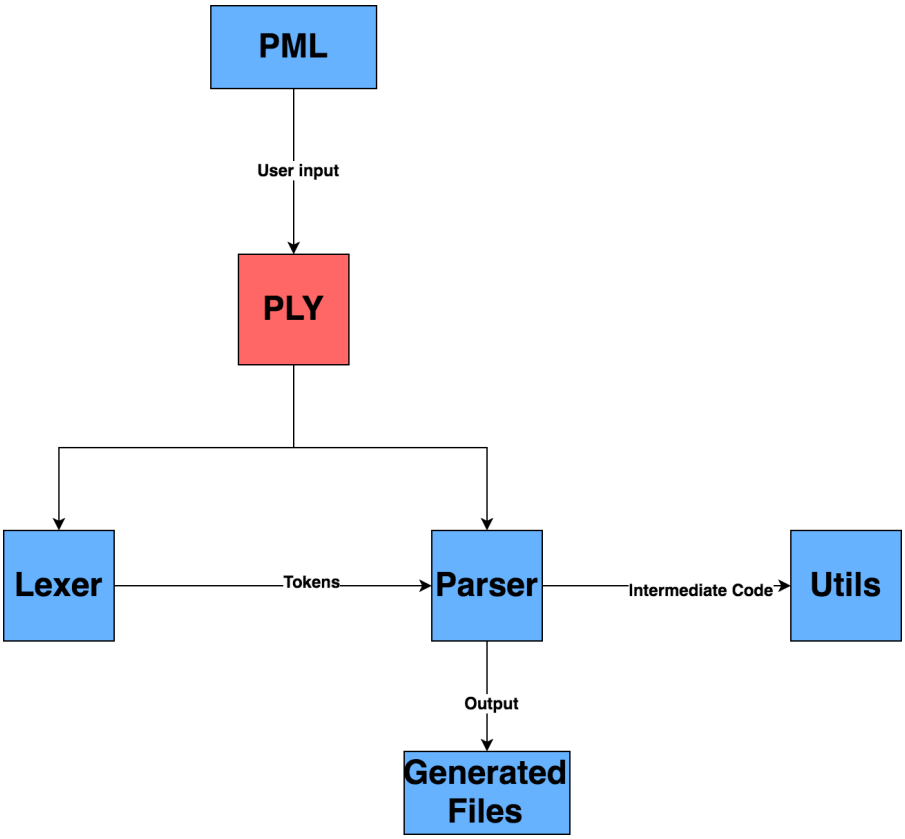
COMMANDS:

- **new_project <PHRASE>** - Creates a new project. PHRASE represent the project name
- **add_member <PHRASE> <NUMBER>** - Add member to a project. PHRASE represent the user name and NUMBER the project ID
- **delete_member <NUMBER1> <NUMBER2>** - Delete member from a project. NUMBER1 represent member ID and NUMBER2 is the project ID
- **view_members <NUMBER>** - Display list of all members of a project. NUMBER represent the project ID
- **create_brainstorm <PHRASE> <NUMBER>** - Add brainstorm with main topic to a project. PHRASE represents the main topic and NUMBER is the project ID
- **add_idea <PHRASE> <NUMBER>** - Add idea to created brainstorm. PHRASE represent the idea to add, and NUMBER is the project ID
- **view_brainstorm <NUMBER>** - Generate the brainstorm diagram for a project. NUMBER represent the project ID
- **add_task <PHRASE> <DATE1> <DATE2> <NUMBER>** - Add task to a project. PHRASE is the task description, DATE1 start date, DATE2 end date, and NUMBER is the project ID

- **delete_task <NUMBER1> <NUMBER2>** - Delete task from a project. NUMBER1 represent the task ID and NUMBER2 the project ID
- **completed_task <NUMBER1> <NUMBER2>** - Mark a task as completed. NUMBER1 represent the task ID and NUMBER2 the project ID
- **edit_**task <NUMBER1> <DATE1> <DATE2> <NUMBER2> - Edit dates of a task. NUMBER1 is the task ID, DATE1 start date, DATE2 end date, and NUMBER2 the project ID.
- **assign_task <NUMBER1> <NUMBER2> <NUMBER3>** - Assign task to a member. NUMBER1 represent the task ID, NUMBER2 the user ID, and NUMBER3 is the project ID
- **view_tasks <NUMBER>** - Display list of all tasks in a project. NUMBER represent the project ID
- **view_schedule <NUMBER>** - Generate Gantt chart for a project - NUMBER represent the project ID
- **list_today <NUMBER>** - Display list of all tasks due in current day. NUMBER represent the project ID
- **list_week <NUMBER>** - Display list of all tasks due in current week. NUMBER represent the project ID
- **list_overdue <NUMBER>** - Display list of all overdue tasks. NUMBER represent the project ID
- **generate_project <NUMBER>** - Generate text file containing a project information. NUMBER represent the project ID
- **generate_pie <NAMELIST> <NUMBERLIST>** - Generate a pie chart. NAMELIST represent labels and NUMBERLIST is the values
- **generate_bar <NAMELIST> <NUMBERLIST>** - Generate a bar chart. NAMELIST represent labels and NUMBERLIST is the values
- **generate_line <LBRK><PHRASE1><RBRK> <LBRK><PHRASE2><RBRK> <NUMBERLIST1> <NUMBERLIST2>** - Generate a line chart. PHRASE1 represent label for x-axis, and PHRASE2 is the label for y-axis. In the same way, NUMBERLIST1 is the values for x, and NUMBERLIST2 represent values for y.

Language Development

TRANSLATOR ARQUITECTURE:



DEVELOPMENT TOOLS:

- **Python 3.6**
- **PLY** – Python implementation of Lex-Yacc
- **Graphviz** – For generating the brainstorming diagram
- **Pickle** – For data persistence (saving and loading project data)
- **Plotly** – For generating Gantt Chart
- **Matplotlib Pyplot** – For generating Graphs (Line, Bar, Pie)

TEST METHODOLOGY:

To test the programming language, we first test that all the Lexer tokens were defined correctly by giving some input data and debugging using the console output. Then, we test that all libraries for generating diagrams worked as expected. For this, we generated the diagrams with dummy data. Then we continue the development by integrating the libraries to the intermediate code to be called from the Parser. In the final testing steps, we run the Language Tutorial code to ensure that the program worked correctly.

Conclusion

Creating PML we were able to understand how a programming language is developed and how its internal components integrate together. Using PLY library, we learned how the internal components work. At first, using PLY was a little confusing but after some practice the development of the programming language was very straightforward. The biggest challenge while developing PML was managing dates in python and defining the required regular expressions in Lexer. Also, the testing was time consuming because we wanted to ensure the expected behavior of all commands. We were happy with the final result and gained some useful experience while developing the programming language.