UNIVERSITY OF BUEA

FACULTY OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

PROJECT REPORT

TITLE: TOWARDS A DATA STRUCTURE FOR PARAMETER PASSING STYLES.

NOUMBA LEONARD

SC17A350

SUPERVISOR: DR WILLIAM S. SHU

31 AUGUST 2020

DECLARATION

I hereby declare that this project report has been written by me Noumba Leonard, that to the best of my knowledge, all borrowed ideas and materials have been duly acknowledged, and that it has not receive any previous academic credit at this or any other institution.

Noumba Leonard

SC17A350

Department of Computer Science

Faculty of Science, University of Buea

CERTIFICATION

This is to certify that this report entitled “TOWARDS A DATA STRUCTURE FOR PARAMETER PASSING STYLES” is the original work of NOUMBA LEONARD with Registration Number SC17A350, student at the Department of Computer Science at the University of Buea. All borrowed ideas and materials have been duly acknowledged by means of references and citations. The report was supervised in accordance with the procedures laid down by the University of Buea. It has been read and approved by:

William S. Shu, PhD CEng CITP MBCS — Date

University of Buea

(Project Supervisor)

Dr. Denis L. Nkweteyim — Date

Head of Department of Computer Science

ABSTRACT

Parameter passing mechanisms are the various ways used to pass parameters to procedures or functions and are widely used in programming. A parameter passing mechanism hugely depends on the nature of its parameters. By nature, we mean how the parameters are used, their values and significance and how they could be combined and passed. Several factors such as context, evaluation strategy, and typing have been exploited and used to describe parameter passing mechanisms. Due to the degree to which parameter passing mechanisms affects computation, they are widely exploited in programming languages. In this project, we construct a structure for parameter passing styles and define permissible operations on this structure. That is, a data structure for major known passing styles and possibly infinitely many user define styles. This structure provide users with the ability to add newly created passing style, remove undesired/unpleasant styles from the structure and also add/remove interpretation on adding/removing a passing style from the structure. We then illustrate usefulness of one of these passing styles in safety systems. Specifically, pass by value is used to prevent changes from being made on an entity before or after it is used.

Chapter 1

Introduction

1.1 Project Motivation

Rapid growth of programming languages and software systems has increased the need for an efficient and more reliable passing mechanism for communication between modules (or functions) of these languages or systems and also across different application domains. With the ever-increasing data to be communicated between different application domains, there is the need for an efficient structure to hold this data and a passing mechanism to safely communicate this data in an efficient way.

My interest in this project is to model and develop a data structure for parameter passing that can serve users to communicate and protect their data between application domains. Also, due to the ever-increasing data to communicate, users can define and add their own new, even more efficient passing style in the structure. With this ability, the ever changing need for efficient parameter passing is met.

1.2 Project Aims

The aim of this project is to develop a data structure (skeletal data structure) that closely examines the various parameter passing mechanism (including novel styles) and takes into account factors that affect parameter passing such as entity passed(e.g. value or computation), evaluation strategy, and execution context as well as typing.

As an objective to this project, I seek to:

- Identity the various parameter passing mechanism and the relationship among them.
- Identify the various basic components involved in parameter passing, group them into basic classes, and combine them to define various parameter passing styles. Also, identify values of these components that are predicted to yield good performance.

- Develop a data structure that holds and can be used to showcase the various parameter passing styles. Also develop function or operations for manipulation this data structure. The operations on this structure include:
  Add a new parameter passing style.
  Remove an existing passing style.
  See various passing styles in the structure.
- Identify the various parameter passing styles with specific application domains and also explore usefulness of novel styles.

1.3 Report Structure

The rest of this report is organized as follows:

Chapter 2 explores the analysis and design of the data structure. It defines the problem statement, the research aims and questions and finally the design algorithms of my program.

In Chapter 3, I present the results and discussions from the implementation of the analysis and design presented in Chapter 2. It provides results of implementation and explains the algorithm used to implement the main activities.

Chapter 4 is the conclusion of the report.

Chapter 2

Analysis and Design

2.1 Requirement of the system

The aim of this project is to develop a skeletal data structure that closely examine the various parameter passing mechanism (including novel styles) and takes into account factors that affect parameter passing such as entity passed(e.g. value or computation), evaluation strategy, and execution context as well as typing. This structure provides users with the ability to add new passing styles, remove an existing passing style. User defined passing styles can then be used by other organisations or users if need be. In order to achieve this, we have to answer questions such as:

Can users get hold of the available factors for parameter passing?

Can users create their own passing style using any number of the available factors affecting parameter passing?

Can users newly created passing style be added in the structure?

Can users remove a passing style they don't desire?

Can users see list of available passing styles?

2.2 Main Entities, activities and data structures

Parameter passing mechanisms are assorted and widely used in programming [1]. The choice of a parameter passing mechanism is an important decision of a high level programming language. In parameter passing, the main components (factors) that determines a specific passing style include entity passed, context, evaluation strategy and typing.

1. Entity passed

Entity passed refers to the parameter that is passed to a method and used to communicate data between modules (or methods).

The various type of possible entity passed include:

- Value: Configuration of states or final result that cannot be simplified further and considered ok [2].
- Reference: Refers to a memory location or address of a value.
- Computation: Expression that could be further simplified, possibly non terminating [2].
- Denotation: Refers to the meaning attributed to expressions or objects from mathematical domains.
- Continuation: Rest of a computation after a given computation is carried out.
- Environment: Context in which bindings are found and hence what values and interpretations are found [2].
- Object: Collection of data together with functions to operate on that data.

2. Context

Refers to the environment a function or parameter is called and evaluated. This can be in the body of the called or the calling procedure.

3. Evaluation strategy

Evaluation strategy determines when to evaluate the arguments of a function call and what kind of value to pass to the function. It can be classified into strict and lazy evaluation. In strict evaluation, the arguments of a function are completely evaluated before the function is applied.
In lazy evaluation, arguments to a function are not evaluated except when they are used in the evaluation of the function body.

4. Typing

Type refers to a collection of values that share some property [4]. Typing permits us to check whether the type of a function's argument matches that of its formal parameter. This verification can be done either at runtime or compile time.

Combination of these components (factors) that affect parameter passing, their types, and values, affects the computation carried out and an instance of its use is broadly considered as a specific passing style.

This is illustrated below for known parameter passing styles:

- Call by value: In call by value, the entity passed is a value and it is evaluated in the context of the calling procedure at the time of procedure call and they (entity passed) are evaluated in order.
- Call by reference: Here, the entity passed is a reference (address in memory) and it is evaluated it the time of procedure call in the context of the calling procedure.
- Call by copy-restore: Here, the entity passed is a value and it is evaluated at the time of procedure call in the context of the called procedure and the entity passed are evaluated in order.
- Call by name: In call by name, the entity passed is a value and it is evaluated in order at the time they are used (or needed) in the context enriched by the computation so far [2].
- Call by need: Here, the entity passed is a value (or computation) and it is evaluated in order, in the context of the called procedure at the time they are needed. When the entity is evaluated, the result is stored for subsequent uses.
- Call by sharing: In call by sharing, the entity passed is an object and it is evaluated in order, in the context of the calling procedure at the time of procedure call.

Activities refers to the possible operations that can be carried out on the developed data structure for parameter passing and also on its elements which are parameter passing styles in this case. The main activities here include

Constructing and adding a new parameter passing style to the structure.

Retrieve an existing passing style and it properties.

Remove an existing passing style from the structure.

Add possible interpretations over the novel styles.

Retrieve an interpretation for a passing style.

Display list of available passing styles.

For each passing style, the relationship between the various factors affecting that passing style is captured by using a record data structure to hold these factors. An enumerated type was used to hold the different values (possible instances) of these factors. A function is then used for assigning values to these factors. Adding new passing styles to the structure is achieved using a function that takes parameter passing style factors and constructs a passing style from the combination of these factors and add to the structure. This combination of factors defines the behaviour of a passing style. A record data structure is used to hold the combination of these factors with their values as records can hold values or variables of arbitrary type.

2.3. Analysis

A hierarchical parameter passing (HPP) diagram is used to visually reveal the workflow of the system through which we develop the intended structure and breakdown the problem into smaller parts for better understanding. It depicts the various tasks that can be performed by whichever user of the system. The term user also include but is not limited to a programmer or computer scientist.

2.3.1 HPP Diagram

The HPP diagram in Figure 2.1 is used below to demonstrate a hierarchical structure of the program design.

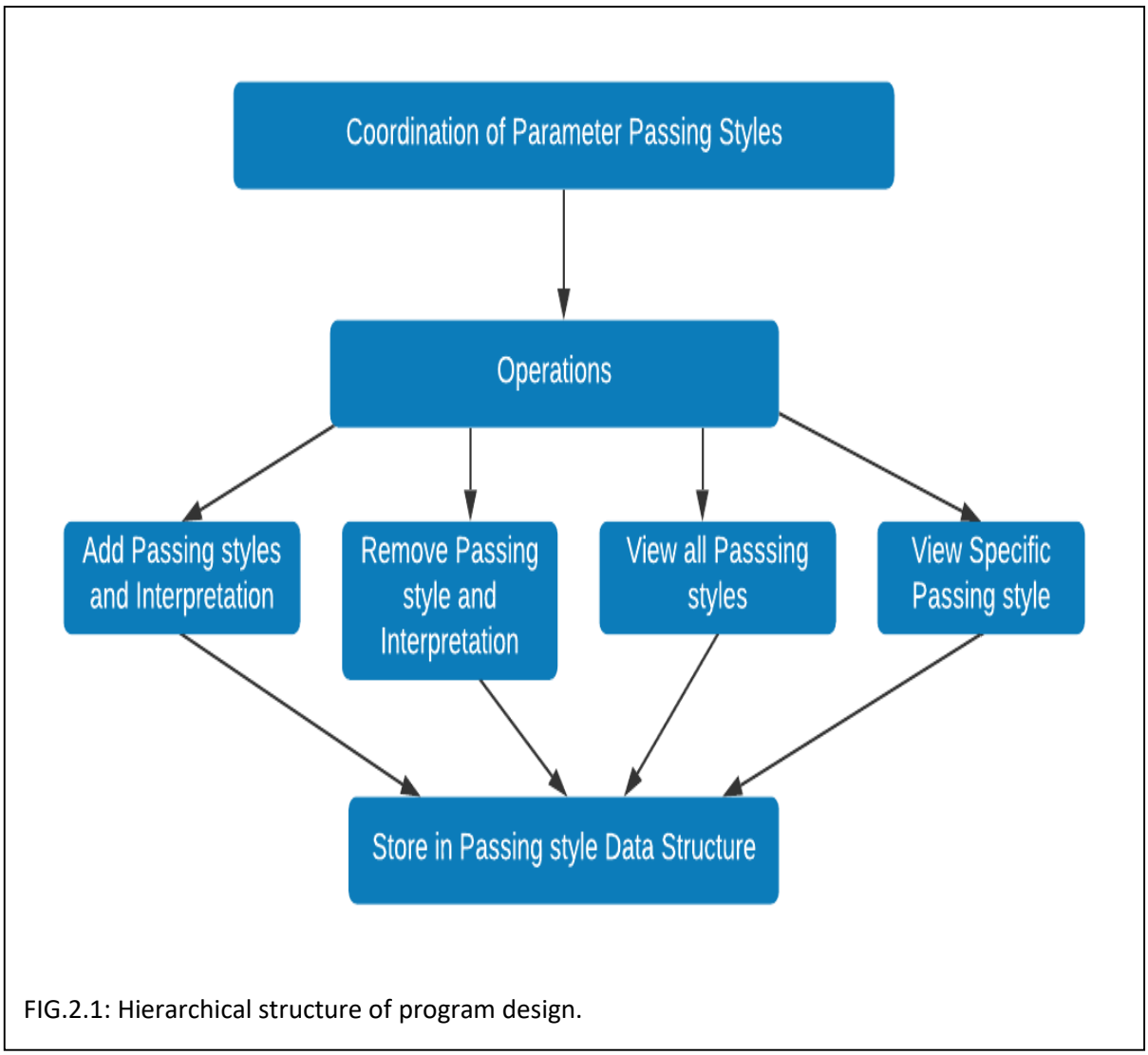

FIG.2.1: Hierarchical structure of program design.

2.3.2 Description of HPP Diagram

The HPP diagram is made up of 4 hierarchical levels organized in a top-down design starting with Coordination of parameter passing styles at the highest level. The diagram shows input and output procedures and various store locations. Input procedures are denoted using lines going into an operation and output processes are denoted using lines going out of an operation.

Level 1:

The activity at this level is *Coordination of parameter passing styles*. This is the highest level of the diagram which is then broken down into smaller tasks. It is the main project activity, for the system as a whole.

Level 2:

The main task here is let user's *select specific task* to perform. At this level, the system pulls all data from the data structure at the end of each operation and updates the passing style data structure which can later be used for other operations.

Level 3:

The tasks at this level include carrying out task for the selected operation. The various operations at this level includes: *Create and adding a new passing style, Remove a passing style, View a specific passing style, and view all available passing styles.* A user will get to carry out any of these task (operations) if at level 2, they chose any of the above mentioned operation.

Level 4:

The task at this level is *Store to passing style data structure.* This is the task done by the system every time the user carries out an operation in level 3.

2.3.3 Specific solution/ problem instances

Parameter passing styles are used to solve varieties of problems in programming and other related (or non-related) fields. To begin with,

In systems or programs where there is no side effects on the variables, entities, or program used, this can be achieved by using the *call by value* passing mechanism which prevents the modification of the values of the entity or variable used in the program. This improves program safety and security.

Also, parameter passing can be used to provide direct communication across different application domains. Application domains can talk to each other by passing objects via marshalling by value, and also by reference through a proxy.

In a system or a program where clients wants to back up their data in the system to some remote server, the semantics of pass by copy-restore is used to provide a copy of the data (referred to as a parameter), copies the data to the server and then restores the changes to the original data in place. In addition, in system where users can download, each time a download request is made, the copy-restore semantic is used to make a copy of the downloadable which is then copied to the user system.

2.4 Design

This project was developed using the prototyping/exploratory programming approach to problem solving. In which case, Developing a Data Structure for parameter passing was broken down into smaller task to ease the management of the data (passing styles) in the structure and test. A stepwise refinement of these tasks is given in Appendix 1.

A list data structure is used to hold the various parameter passing styles so as to capture the relationship among them. This data structure is used because it is dynamic, immutable and has arbitrary-length and also, operations for adding and removing from them can easily be performed. To Model a passing style (which is made up of name and factors) in the structure, a simple string variable is used to hold the passing style name and a record is used to hold the various factors with their values. A list data structure is used to hold all the possible values for a factor. A record is used to hold both the passing style name and the factors of that passing style alongside it values. The passing style name, factors, combination of factors and their values defines a passing style. An associative list data structure is used to store and retrieve interpretation for a passing style. This data structure associates an interpretation with a passing style (name).

2.4.1 Design Algorithm

(1) Start

(2) Display menu (Add passing style and interpretation, Remove passing and interpretation,
See all passing styles, Select style from passing style structure)

(3) Read selected_operation f

(4) IF (f = Add passing style) THEN

Passing_style_list <-- Add_passing_style_and_interpretation Passing_style_list Style

GOTO step 2

ENDIF

(5) IF (f = Remove passing style) THEN

Passing_style_list <-- Remove_passing_style_and_interpretation Passing_style_list Style

GOTO step 2

ENDIF

(6) IF f = (See all passing styles) THEN

Display Passing_style_list

GOTO step 2

ENDIF

(7) IF (f = Select style from passing style structure) THEN

Display_all_info_about_selected_style Style

GOTO step 2

ENDIF

(8) Stop

Fig. 2.1 Design algorithm

Fig 2.1 shows an algorithm on how the system for developing the data structure for parameter passing works. There are a number of actions that could be performed on the system. As the system starts, these actions are displayed as a menu so the user makes his choice.

Some of the main menu items include:

- Add passing style and interpretation
- Remove passing style and interpretation
- View all passing styles
- Select a passing style

Depending on the user's choice, a given action is carried out in the system. Except for the exit menu item (not listed above) which exits the app, every other menu performs the action under it and gives the user the opportunity of performing another action. Algorithm for critical tasks like adding new passing styles in the system can be seen on the figure below.

```
(1) Start
(2) Display factors (entity passed, context, evaluation, typing)
(3) entity = [a], context = [a], evaluation = [a], typing = [a]  (* 'a' represents initial value for factors*)
(4) Read selected factor fac
(5) IF (fac = entity) THEN
      Read Number N            (*number N to initialize selected factor*)
      entity <-- add_to_list entity N (*adding N to list that holds values for selected factor (entity in this case)*)
      GOTO step 9  ENDIF
(6) IF (fac = evaluation) THEN
      Read Number N
      evaluation <--  add_to_list  evaluation  N
      GOTO step 9 ENDIF
(7) IF (fac = context) THEN
      Read Number N
      context <--  add_to_list context N
      GOTO step 9  ENDIF
(8) IF (fac = typing) THEN
      Read Number N
      typing <--   add_to_list typing  N
     GOTO step 9  ENDIF
(9) IF (more factors?) THEN
    GOTO step 4 ELSE GOTO step 10  ENDIF
(10) Read name name                                  (*name for the passing style*)
(11) factors <-- preserve_default {entity ; context; evaluation; typing}     (preserve default values for factors not of interest to the user but removes default values for selected factors)
(12) Passing_style <-- {name; factors}                                       (*creating the passing style*)
(13) List_of_styles <-- List_of_styles:: Passing_style                       (*adding newly created style to list of styles if not in list*)
```

Fig. 2.2 Algorithm to add new passing style.

Chapter 3

Investigations and Discussions

In this section, some of the results obtained after implementing all the main activities is presented. This program makes use of a Command Line Interface (CLI) to serve as a medium of communication between the user and the application. This means that the system will only recognize text-like commands when a user is "talking" to it. All codes used or referenced in this project report are of the ocaml programming language version 4.10.0, on a 64 bit computer with an MS Windows 10 operating system.

Results obtained after a user adds a passing style is shown in figure 3.1 below:

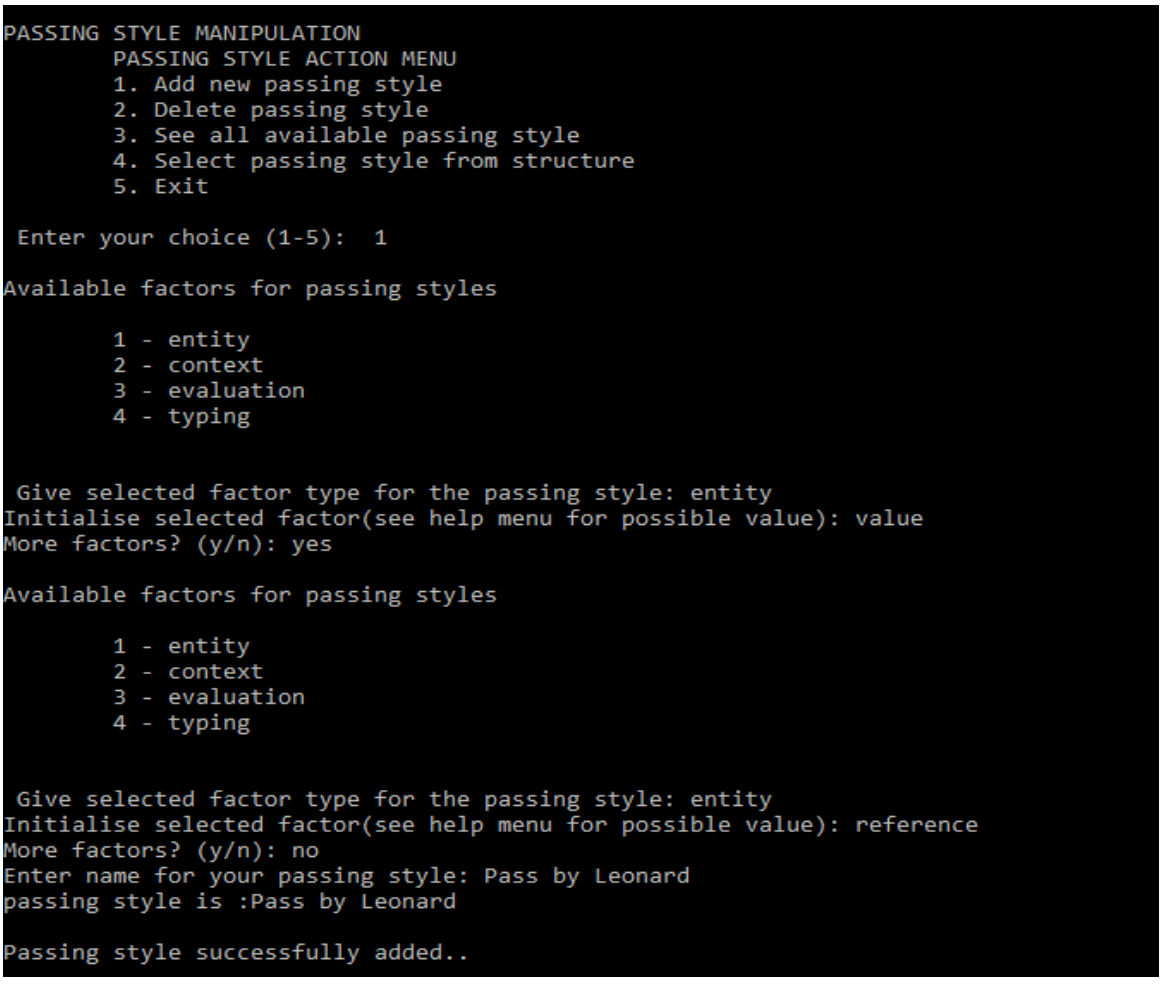

Fig.3.1: Add new passing style

Figure 3.1 demonstrate how new parameter passing styles are created and added to the structure. List of factors known to affect parameter passing is displayed. Users enter value(s) for the factor(s) of interest and name for the new style as shown in figure 3.1 above. Once a factor is selected and initialised, the user can decide to add more factors of interest. User can select and initialise a factor of interest more than one time. A default value (5) is given to factors not of interest to the user. The demo showing that the newly created passing style was indeed added to the structure is seen in figure 3.2.

Results obtained after a user view all styles found in the structure is shown in figure 3.2 below:

```
PASSING STYLE MANIPULATION
        PASSING STYLE ACTION MENU
        1. Add new passing style
        2. Delete passing style
        3. See all available passing style
        4. Select passing style from structure
        5. Exit

 Enter your choice (1-5):  3
[{name =
Pass by Value;
factors = {entity = [1 ]; context = [1 ]; evaluation = [1 ] typing = [1 ]}};
{name =
Pass by Reference;
factors = {entity = [1 ]; context = [2 ]; evaluation = [3 ] typing = [3 ]}};
{name =
Pass by Name;
factors = {entity = [2 ]; context = [2 ]; evaluation = [2 ] typing = [2 ]}};
{name =
Pass by Copy_restore;
factors = {entity = [3 ]; context = [3 ]; evaluation = [3 ] typing = [3 ]}};
{name =
Pass by Need;
factors = {entity = [1 ]; context = [2 ]; evaluation = [3 ] typing = [4 ]}};
{name =
Pass by Leonard;
factors = {entity = [1 2 ]; context = [5 ]; evaluation = [5 ] typing = [5 ]}};
]
```

Fig.3.2: View newly added style in structure

Figure 3.2 shows the presence of the newly added style in the structure alongside passing styles known to the system. It also shows the two instances entered for the selected factor (entity) for the passing style Pass by Leonard.

After a user has added a passing style to the structure, it follows that they add an interpretation for the newly added style.

Results obtained after a user add an interpretation for a newly added style is shown in figure 3.3 below:

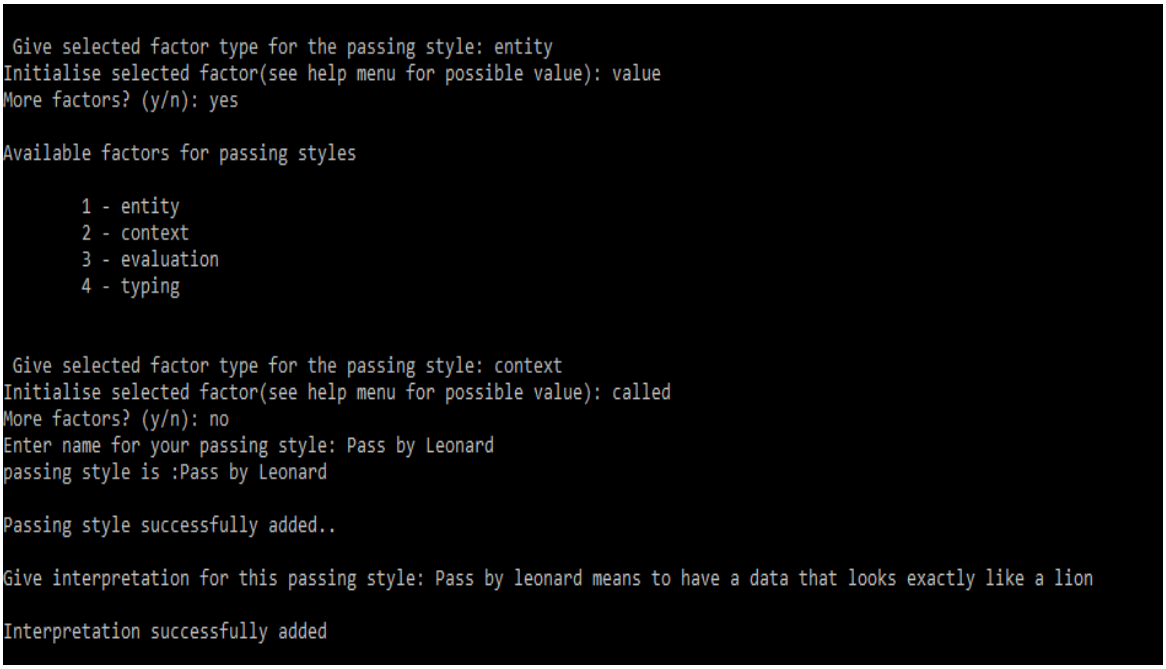

Fig.3.3: Add interpretation for a new passing style

Figure 3.3 demonstrate how an interpretation for a new passing style is added immediately after the style is created and added to the structure.

The demonstration showing that the interpretation for the newly added style was indeed added is seen in figure 3.4. The user selects the operation "*Select passing style from structure"* from the "passing style action menu". It asked for the name of the passing style. The name is then used to get the interpretation of the passing style.

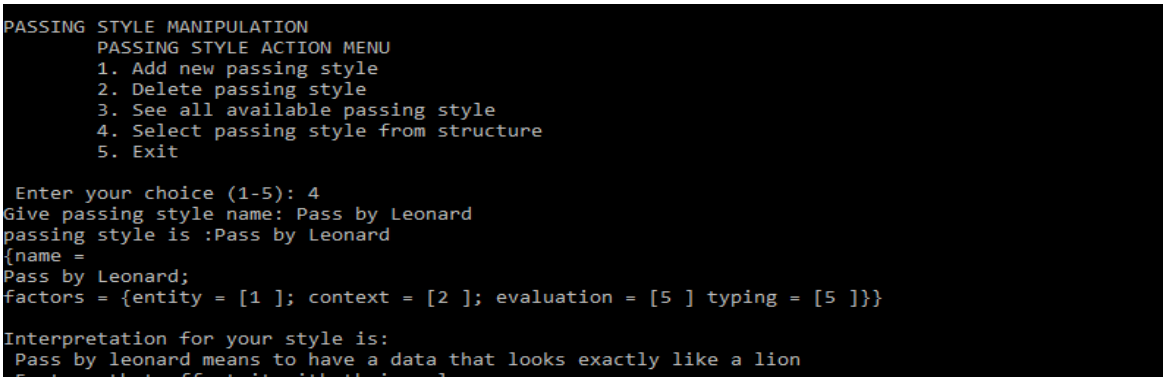

Fig.3.4: View interpretation of a style

Possible instances for the various factors that can be entered by the user, and the meaning (values) of these instance (e.g. value for entity) is shown in figure 3.5 below:

```
# user_facing_information();;
This table represents information the user faces concerning factors of parameter passing styles
  Factors ******************  Meaning ***********************Instances

  Entity passed                 1                           Value
                                2                           Reference
                                3                           Computation
                                4                           Environment
                                5                           Continuation
                                6                           Denotation

  Context                       1                           Calling method
                                2                           Called method
                                3                           Calling and called
                                4                           Other

  Evaluation                    1                           Strict
  Strategy
                                2                           Lazy
                                3                           Non
                                4                           Manual

  Typing                        1                           Yes
                                2                           No
                                3                           Non
```

Fig.3.5 User facing information

Figure 3.5 shows the possible instances users can enter for a selected factor of interest when carrying out a task. Users can view it by selecting the help menu. On Instantiating a selected factor of interest, if user enter an instance that is not known to the system (assumed to be newly defined instance), they are asked to give meaning (value) for the entered instance. The value is then used to instantiate the selected factor as shown in fig 3.6 below:

```
 Enter your choice (1-5):  1

Available factors for passing styles

        1 - entity
        2 - context
        3 - evaluation
        4 - typing


 Give selected factor type for the passing style: entity
Initialise selected factor(see help menu for possible value): new_instance
Give value for the factor:  10
More factors? (y/n): no
Enter name for your passing style: Pass by new_instance
passing style is :Pass by new_instance

Passing style successfully added..
```

Fig.3.6: New instance for factor(s) of interest

Figure 3.7 shows that the new style was added with the new instance value for the selected factor of interest (entity in this case).

```
 Enter your choice (1-5):  3
[{name =
Pass by Value;
factors = {entity = [1 ]; context = [1 ]; evaluation = [1 ] typing = [1 ]}};
{name =
Pass by Reference;
factors = {entity = [1 ]; context = [2 ]; evaluation = [3 ] typing = [3 ]}};
{name =
Pass by Name;
factors = {entity = [2 ]; context = [2 ]; evaluation = [2 ] typing = [2 ]}};
{name =
Pass by Copy_restore;
factors = {entity = [3 ]; context = [3 ]; evaluation = [3 ] typing = [3 ]}};
{name =
Pass by Need;
factors = {entity = [1 ]; context = [2 ]; evaluation = [3 ] typing = [4 ]}};
{name =
Pass by new_instance;
factors = {entity = [10 ]; context = [5 ]; evaluation = [5 ] typing = [5 ]}};
]
```

Fig.3.7: Style with new instance for factor of interest

Results obtained after a user deletes a passing style from the structure is shown in figure 3.8 below:

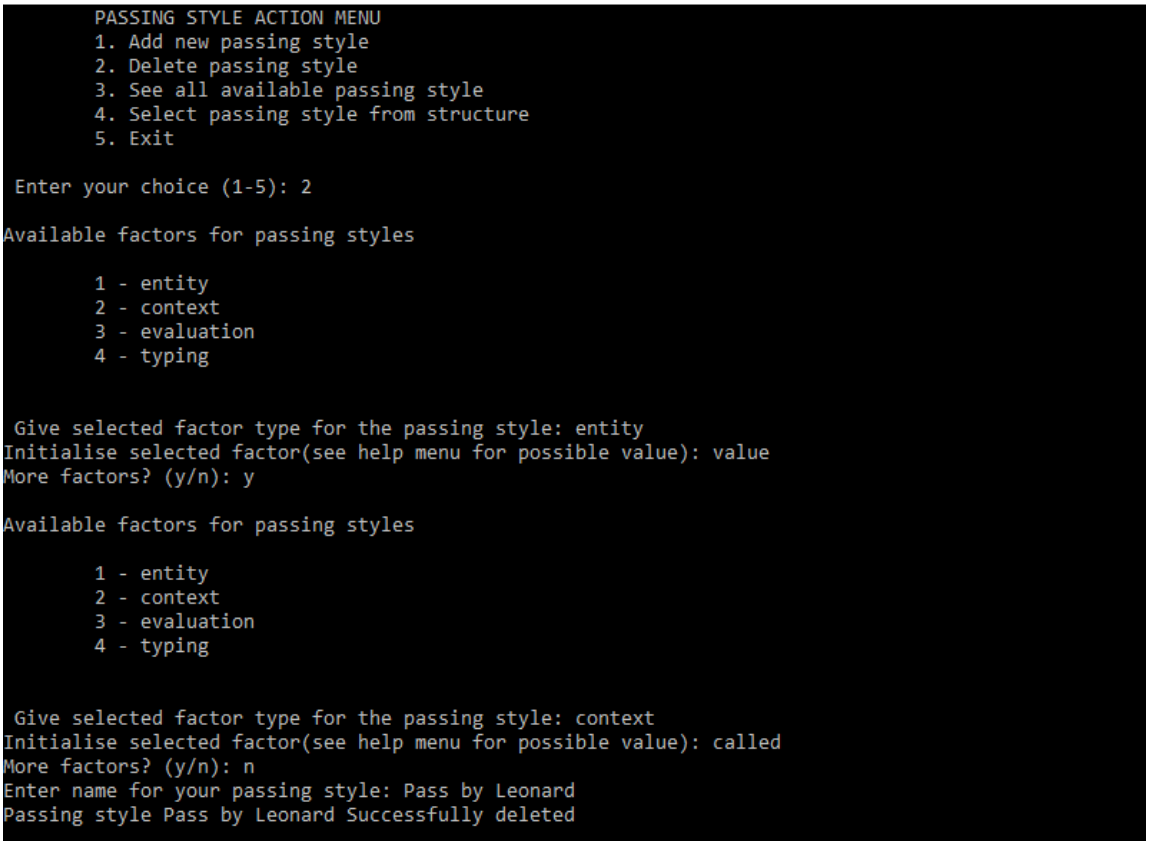

Fig.3.8 Deleting a passing style

Figure 3.8 shows the process of deleting a passing style from the structure. User enters values for known factors of interest and name for the passing style. The passing style is then deleted from the structure as shown in figure 3.8 above. This is required as two passing styles may have the same name but different values for factors of interest. Deleting a passing style also delete its interpretation. The demo showing that the passing style was indeed deleted from the structure is seen in figure 3.9 below.

Results obtained after a user view all styles found in the structure is shown in figure 3.9 below

```
PASSING STYLE MANIPULATION
        PASSING STYLE ACTION MENU
        1. Add new passing style
        2. Delete passing style
        3. See all available passing style
        4. Select passing style from structure
        5. Exit

 Enter your choice (1-5): 3
[{name =
Pass by Value;
factors = {entity = [1 ]; context = [1 ]; evaluation = [1 ] typing = [1 ]}};
{name =
Pass by Reference;
factors = {entity = [1 ]; context = [2 ]; evaluation = [3 ] typing = [3 ]}};
{name =
Pass by Name;
factors = {entity = [2 ]; context = [2 ]; evaluation = [2 ] typing = [2 ]}};
{name =
Pass by Copy_restore;
factors = {entity = [3 ]; context = [3 ]; evaluation = [3 ] typing = [3 ]}};
{name =
Pass by Need;
factors = {entity = [1 ]; context = [2 ]; evaluation = [3 ] typing = [4 ]}};
]
```

Figure.3.9 Result after deletion of passing style

Figure 3.9 shows the result of the structure after deletion of passing style.

Results obtained after a user selects a passing style from structure is shown in figure 3.10 below:

```
PASSING STYLE MANIPULATION
        PASSING STYLE ACTION MENU
        1. Add new passing style
        2. Delete passing style
        3. See all available passing style
        4. Select passing style from structure
        5. Exit

 Enter your choice (1-5):  4
Give passing style name: pass by value
{name =
pass by value;
factors = {entity = [1 ]; context = [1 ]; evaluation = [1 ] typing = [1 ]}}

Interpretation for your style is:
 Here, entity passed evaluated from left to right
 Factors that effect it with their values are:
factors = {entity = [1 ]; context = [1 ]; evaluation = [1 ] typing = [1 ]}
```

Fig.3.10: Results after selecting passing style from structure

Figure 3.10 show the result after a user selects a passing style from the passing styles data structure. The passing style name is being asked. Any style added in the add new passing style can be viewed as shown in figure 3.10 above. If your selection doesn't exist, the program will display an error message informing the user that his/her selection doesn't exist in the structure and then asks the user to enter another passing style name. If your selection does exist, the passing style is displayed, alongside it interpretation and factors that affects the style as shown in figure 3.10 above.

When a user selects a passing style from the structure, the effects induced by values of factors of the passing style (known styles) is shown in figure 3.11 below:

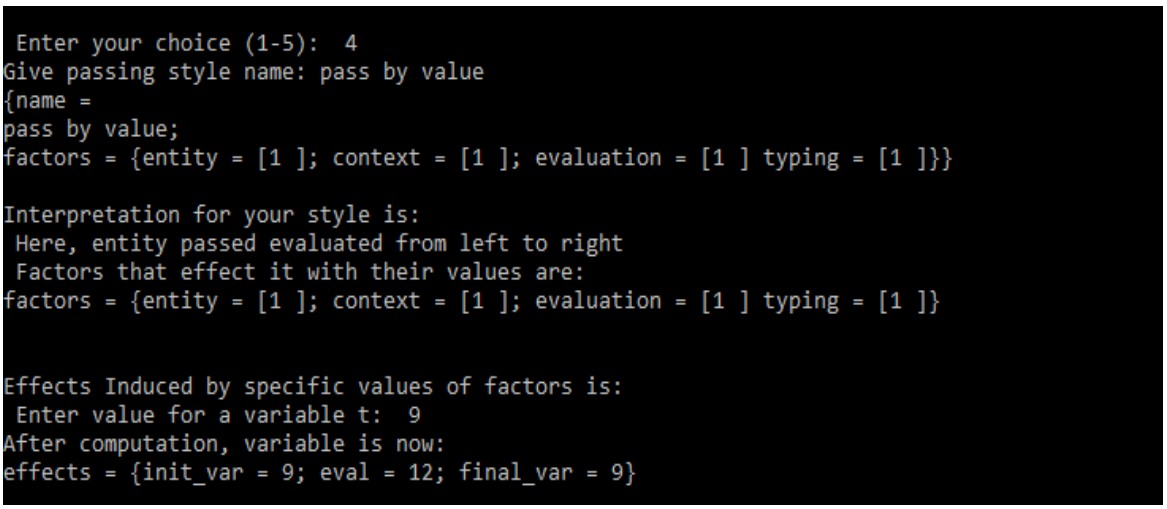

Fig.3.11 Effects of pass by value

Figure 3.11 shows the effects induced by values of factors for pass by value. Init_var is the initial value of the variable, eval is the result of a computation and final_var is the final value of the variable after the computation. For call by value, it is seen that the value entered by the user for the variable init_var does not change as both values (init_var and final_var) stays the same after performing the computation. This style is used in security systems to prevent the modification of data communicated.

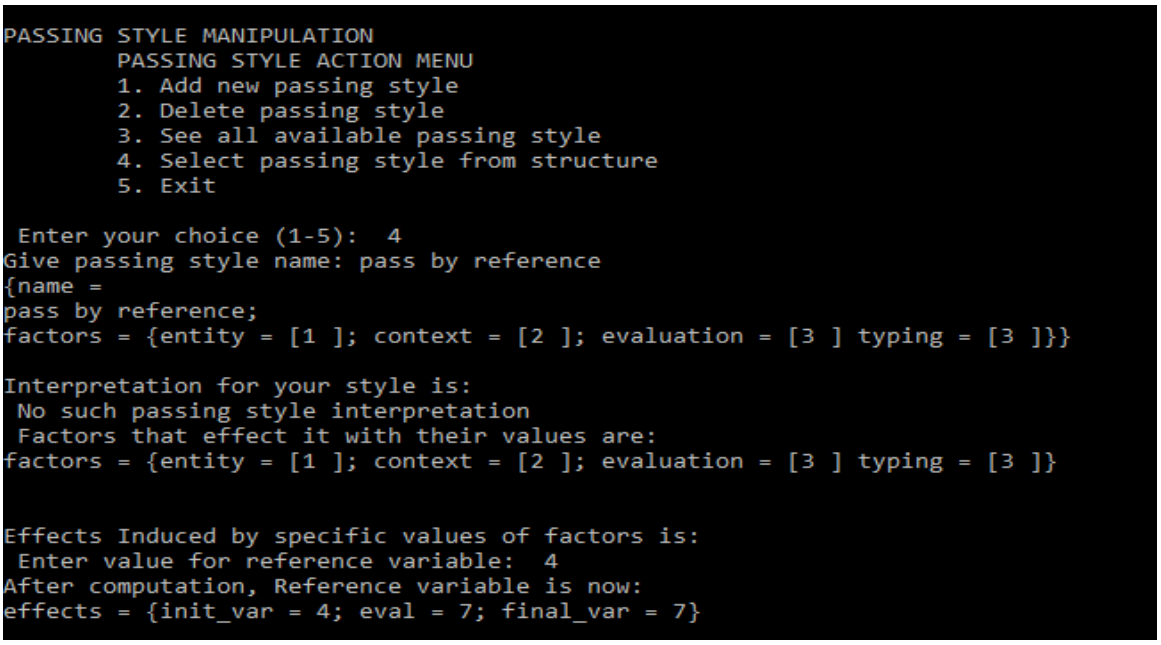

Fig.3.12: effects of pass by reference

Figure 3.12 shows the effects induced by values of factors for pass by reference. It is seen that, the value of the variable init_var has been modified. After performing the computation, the value of init_var has been changed and it's reflected by final_var. This mutation of variables is necessary in applications that need feedback results from a modification end. Pass by reference is fast because a copy of the variable is not made.

All of the above figures show the results I obtained when I ran my program. There are still some minor changes to be done to the code so that it runs with a better level of perfection.

The main challenge I faced was in implementation of the section under initialising selected factor when adding new style. I have not completed all of its minor actions, but the major actions have been developed as shown from figure 3.1 to figure 3.6.

Chapter 4

Conclusion

This project's goal was to develop a data structure for parameter passing that provides users with the tools to create and add/remove new passing styles to/from structure, add an interpretation for a style, and to explore the various factors from which the behaviour of a style is defined.

My objective for this report was to develop a data structure for parameter passing and standardized methods and procedures used for efficiently managing the data in the structure. This has been accomplished