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

18 SEPTEMBER 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

DEDICATION

I dedicate this work:

In memory of my Uncle Numba Emmanuel who was a father to me. Your endless love and care that still gives me inspiration up till date shan't be forgotten.

To God almighty for his endless strength and blessings that have carry me through this project.

To my mother for her everyday love, care and support.

ACKNOWLEDGEMENTS

I thank Jehovah God almighty for His endless grace and blessing that has given me much strength to complete this project.

I take this opportunity to express my profound gratitude and deep regards to my guide (project supervisor) Dr. William S. Shu for his guidance, monitoring and constant encouragement throughout the course of this project.

I would like to thank the following people for their profound help in the development of this project:

My Lecturers:

Dr. Nkweteyim Denis for the deep notions in data structures and algorithms,

Dr. Ali Wacka and Dr. Nyamsi Madeleine for the fundamental notions of software engineering.

My beloved mother, family and friends who kept on encouraging and motivating me to give my best.

My course mates:

Chiatiah Carlson, Mark Ngoran, Suh Edmond, Claude Nkeng, and Layu Romaric who helped me in one way or another with brilliant ideas.

ABSTRACT

Parameter passing mechanisms are the various ways used to pass parameters to procedures or functions. 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 new passing style, remove undesired/unpleasant styles from the structure and also add/remove interpretation for a style on adding/removing the 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.

TABLE OF CONTENT

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 amongst 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, we identify values of these components that are predicted to yield good performance.
- Develop a data structure that holds various parameter passing styles and can be used to showcase these styles. And in addition, we develop functions (operations) for manipulation this data structure. Some of the permissible operations on this data 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. Chapter 3 presents the implementation part of the project. In Chapter 4, I present the results and discussions from the implementation of the analysis and design presented in Chapter 3. It provides results of implementation and explains the algorithm used to implement the main activities. Chapter 5 is the conclusion of the report.

Chapter 2

Analysis and Design

2.1 Requirements 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 and also see all available passing styles in the structure. 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. The choice of a parameter passing mechanism is an important decision of a high level programming language [1]. In parameter passing, the main components (factors) that determine a specific passing style include entity passed, context, evaluation strategy and typing.

2.2.1 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.
- Reference: Refers to a memory location or address of a value.
- Computation: Expression that could be further simplified, possibly non terminating.
- Denotation: Refers to the meaning attributed to expressions or objects from mathematical domains.
- Continuation: Rest of a computation after a given expression has been evaluated.
- Environment: Context in which bindings are found and hence what values and interpretations are found. It maps variables to semantic values (constants or closure) [2].
- Object: Collection of data together with functions to operate on that data [3].

2.2.2 Context

Refers to the environment a function or parameter is called and evaluated. It describes occurrences inside programs where reduction may actually occur [2]. This can be in the body of the called or the calling procedure.

2.2.3 Evaluation Strategy

Evaluation strategy determines when to evaluate the arguments of a function call and what kind of values to pass to the function. It defines the order in which redices must be reduced. 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.

2.2.4 Typing

A Type refers to a collection of values that share some property. Typing permits us to check whether the type of a function's argument matches that of its formal parameter(s). These verifications could be done either at runtime (during program execution) or at compile time (before program execution).

2.2.5 Description of passing styles

Combination of these components (factors) that are known to affect parameter passing as stated in section 2.2 above, 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 the (entity passed) is evaluated in order (from left to right). Any changes to the value inside the procedure is purely local, and therefore, not visible outside [4].
- **Call by reference:** Here, the entity passed is a reference (address in memory) and it is evaluated at the time of procedure call in the context of the calling procedure. The variable passed as an argument can be modified inside the procedure with visible effects outside after the call [4].
- **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. The entity passed are evaluated in order (from left to right). It's a modification of call by value.
- **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. A thunk is created for each argument and each time the argument is needed, the function is called which evaluates and returns the argument.
- **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. It's a modification of call by name where

the thunk is evaluated at most once. The result is stored and used to subsequent evaluations [4].

- **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.

2.2.6 Activities

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:

- Add new passing styles: New passing styles are constructed by selecting a combination of parameter passing factors and a name. A method (function) is used to add the style to the structure.
- Remove passing styles: Removing an existing passing style from the structure was achieved by using a function that takes parameter passing style name and remove the style from the structure if it exist.

For each passing style, the relationship between the various factors affecting the passing style was 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.

2.3 Specific solutions/ 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 back-up their data 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.

2.4 Design

This project was developed using the prototyping/exploratory programming approach to problem solving wherein, the development process was broken down into smaller task to ease the management of the data (passing styles) in the data structure. This equally made it appropriate for testing.

2.4.1 Main Structures

To Model parameter passing styles, a record data structure is used to hold the various parameter passing styles which is made up of a passing style name and factors as shown in Figure 2.1:

```
type passing_style is record
        name: string;    factor: factors
end
```

Fig.2.1: Passing style structure

Figure 2.1 shows the structure of a parameter passing style. Used record data structure for holding a passing style because records can hold values of arbitrary types. A simple string variable is used to hold the passing style name and a record data structure is used to hold the factors for the passing style as shown in Figure 2.2:

```
type factors is record
        entity : entity_passed;    context: context_type;
        evaluation : evaluation_strat; typing : typing
end
```

Fig. 2.2 : Passing style factors

Figure 2.2 shows the structure of the passing style factor. A union type of list (e.g. entity_type for entity) is used to hold together all possible values for a factor since a single factors can take more than one value. This is shown in Figure 2.3 for the various factors:

```
type entity_type = Entity_type of factor_instance_list
type context_type = Context_type of factor_instance_list
type evaluation_strat = Evaluation_strat of factor_instance_list
type typing =Correct_type of factor_instance_list
type factor_instance_list = int list
```

Fig.2.3 : various factor types

Figure 2.3 shows the type of the various factors. Each is of type list so as to hold arbitrary number of values for the factors. The passing style name, factors, combination of factors and their values defines a passing style.

A list data structure is used to hold the collection of parameter passing styles including novel styles so as to capture the relationship among them. This data structure is used because it is dynamic, immutable and can add or remove from it easily. This structure is revealed in Figure 2.4:

```
Structure is list
style1 : passing_style; style2 : passing_style;  style3 : passing_style
end
```

Fig.2.4 : Main structure

Figure 2.4 shows the main structure for parameter passing. Its elements are parameter passing styles of type passing_style as indicated in Figure 2.1. Permissible operations are used to perform the intended actions (add new passing styles, remove passing styles, see all passing styles) on this structure.

An association list data structure was used to hold interpretations for the various passing styles. This structure associates a given interpretation with a passing style name as shown in Figure 2.5:

```
Interpretation_structure is list
        style1_interpretation: (style_name: string * interpretation_text: string)
End
```

Fig. 2.5 Interpretation structure

Figure 2.5 shows the structure used to hold passing styles interpretation. New passing styles interpretation are added to the structure at runtime immediately after the style is added to the structure. A function (operation) that takes a passing style name is used to retrieve interpretation for the style.

2.4.2 Design Algorithm

```
Start
 REPEAT
(1) 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) Switch (selected_operation)
      Case (Add passing style):
      Passing_style_list <-- Add_passing_style_and_interpretation Passing_style_list Style
       Break;
          (5) Case (Remove passing style):
    Passing_style_list <-- Remove_passing_style_and_interpretation  Passing_style_list
Style
       Break
(6) Case (See all passing styles):
     Display Passing_style_list
       Break
(7) Case (Select style from structure):
     Display_all_info_about_selected_style  Style
       Break
(8) Case (Exit):
          Exit = true
      Break
UNTILL Exit = true
Stop
```

Fig. 2.6 Design algorithm

Fig 2.6 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/her choice. Some of the main menu items include:

- Add passing style and interpretation
- Remove passing style and interpretation

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 item performs the action under it and gives the user the opportunity of performing another. Figure 2.7 and Appendix 1 reveal some critical tasks carried out in the system, notably, adding new passing styles and removing passing styles respectively.

(1) Start

(2) entity = [a], context = [a], evaluation = [a], typing = [a] (* 'a' = initial value for factors*)

REPEAT

(3) Display factors (entity passed, context, evaluation, typing)

(4) Read selected factor

(5) Read Number N (*number N to initialize selected factor*)

(5) Switch (selected_factor)

Case (entity):

entity <-- add_to_list entity N (*adding N to list that holds values for selected factor (entity in this case)*)

Break

(6) Case (evaluation):

evaluation <-- add_to_list evaluation N

Break

(7) Case (context):

context <-- add_to_list context N

Break

(8) Case (typing):

typing <-- add_to_list typing N

Break

(9) IF (more factors?) then exit = false else exit = true

UNTILL exit = true

(10) Read 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 new style to list of styles)

End

Fig. 2.7 Algorithm to add new passing style.

Chapter 3

Implementation

The implementation of this project presents the development of a data structure for parameter passing styles where users can create and add new passing styles, remove an existing style from the structure and can access all available passing styles in the structure. Sets of permissible operations are used to carry out these actions on the developed structure. All program snippet 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.

3.1 Implementation of parameter passing styles

Parameter passing styles comprised of a passing style name and factors, alongside their values. A record data structure was used to implement the passing style which holds the passing style name and factors for the style. A simple string variable was used to hold the passing style name. Figure 3.1 highlights using a record data structure to hold the factors for a passing style. Such a record facilitates capturing the relationship amongst styles.

```
type  passing_style = {name: string;    factor: factors }                (* passing style *)
type factors = {entity : entity_passed;    context: context_type;
                        evaluation : evaluation_strat; typing : typing }       (*style factors*)
type entity_type = Entity_type of factor_instance_list
type context_type = Context_type of factor_instance_list
type evaluation_strat = Evaluation_strat of factor_instance_list
type typing =Correct_type of factor_instance_list
type factor_instance_list = int list
```

Figure 3.1: Passing style structure

A method (function) that takes parameter passing factors and a name was used to construct parameter passing styles from a suitable combination of options. This function is shown in Figure 3.2. The system works internally with numbers.

```
(**Method (sanitize) that checks if passing style is in list, adds it if not and returns the list
* @param user_styles, name: list of known passing styles and name of passing styles respectively
* @param entity_type, context_type, evaluation_strat, typing: factors that effect parameter passing
* @param x: record that holds passing style factors with their values
* @param y: passing style constructed* *)
  let sanitize entity_type context_type evaluation_strat typing user_styles name=
    match (entity_type, context_type, evaluation_strat, typing) with
    (Entity_type [1], Context_type [1], Evaluation_strat [1], Correct_type [1]) -> user_styles
    | (Entity_type [1], Context_type [2], Evaluation_strat [3], Correct_type [4]) -> user_styles
    | (Entity_type _, Context_type _, Evaluation_strat _, Correct_type _) -> let new_style =
  let x = {entity = entity_type; context = context_type; evaluation = evaluation_strat; typing = typing}
  in let y = {name = (setName name); factor = x}
  in let () = print_string "\npassing style " in let () = print_string """ in
  let () = print_string name in let () = print_string """
  in  insert_new_style user_styles y
  in new_style
```

Fig 3.2: Method for creating passing styles

Figure 3.2 shows the method used to create parameter passing styles. It takes parameter passing factors and name, constructs a passing style from the combination of the factors (with their values). The record (y) used to hold the parameter passing factors (x) and the name (name) as indicated in Figure 3.2 defines a parameter passing style. This style (y) is then added into the list of styles by the method insert_new_style. Figure 3.3 shows a newly created style.

```
{name = Passing_style "Pass by XXXXX";
   factor = {entity = Entity_type [1; 2]; context = Context_type [3];
         evaluation = Evaluation_strat [1]; typing = Correct_type [1]}}
```

Fig. 3.3 Newly created style

3.2 Implementation of main structure for passing styles

A list data structure was used to implement the main structure for parameter passing. This structure holds the collection of passing styles which are constructed as shown in Figure 3.2. This data structure was used because it gives room for new elements (passing style in this case) to be added and also existing elements can be removed easily. Figure 3.4 shows the main structure for parameter passing styles. It elements are parameter passing styles and permissible operations are used to manipulate this structure.

```
let user_styles = [
{name = Passing_style "Pass by Value";
factor = {entity = Entity_type [value]; context = Context_type [called];
         evaluation = Evaluation_strat [strict];  typing = Correct_type [yes]}};
 {name = Passing_style "Pass by Reference";
   factor = {entity = Entity_type [reference]; context = Context_type [calling];
         evaluation = Evaluation_strat [3]; typing = Correct_type [3]}}]
```

Fig. 3.4 Main structure for parameter passing

3.3 Operations on main structure

The basic operations performed on this structure include: Adding new passing styles, removing an existing passing style.

3.3.1 Adding new passing styles

A method that takes a parameter passing style constructed in Figure 3.2 was used to insert (add) new passing styles into the main structure shown in Figure 3.3. This method is shown in Figure 3.5:

```
(** Inserts a new passing style in the initial list of passing style (known passing styles)
* @param user_styles: the list from which to add new passing style
* @param y : passing style to be added to list* *)
  let rec insert_new_style user_styles y =
     match user_styles with
     [] -> let () = print_string “successfully added.\n\n" in  [y]
    |h :: t ->if h = y then let () = print_string " already exists.\n”
 in  user_styles else h :: insert_new_style t y
```

Fig. 3.5: Add new style to structure

Figure 3.5 shows the method used to add the newly constructed passing styles into the main structure that is made up of a collection of passing styles. This method is being used in the “sanitize” method to add new passing styles in the structure immediately after they are created as shown in Figure 3.2. Parameter passing style factors are constructed by selecting factors of interest alongside their values from each class of factors. Implementation of the method to remove passing styles from the structure is given in Appendix 3.

Chapter 4

Results and Discussion

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 program snippets 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 4.1 below:

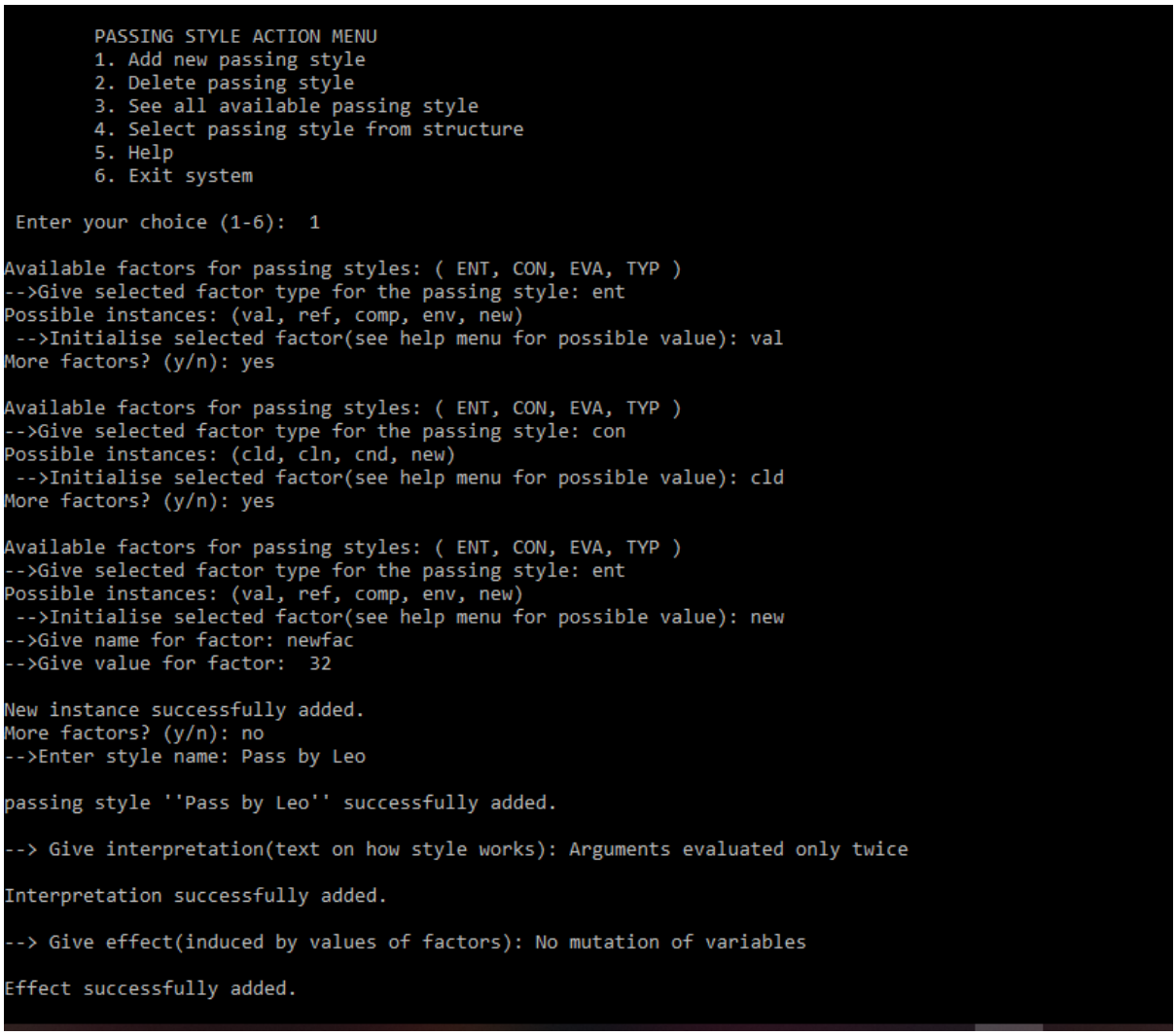

Fig. 4.1 Add new passing style

Figure 4.1 demonstrate how new parameter passing styles are created and added to the structure alongside its interpretation and effects. It presents a list of factors known to affect parameter passing and then provides a field to input the selected factor of interest. After selecting a factor of interest, a list of instances known for the selected factor is presented and then provides a field to input an instance for the factor. A new instance for a factor is created by entering "new" when initialising a selected factor, where the new instance name is entered and it meaning (a value) as shown in Figure 4.1.

Once a factor is selected and initialised, the user can decide to add more factors of interest. Users can provide more than one instances for a factor by selecting the factor more than once and providing different instances for each selection as shown in Figure 4.1. Default values are provided to factors not of interest to the user when creating a passing style.

After initialising a selected factor, you get the option to select more factors of interest and if user go for no more factor, a field to input the name for the passing style is presented. The passing style is then created with values for the selected factors of interest and inputted name. The newly created style is then added to the structure and it's indicated by a success message.

After adding the style, input fields are provided for the interpretation and effects of the newly created passing style followed by success messages after adding each (interpretation and effect) as shown in Figure 4.1. Figure 4.2 reveals that the newly created passing style was indeed added to the structure.

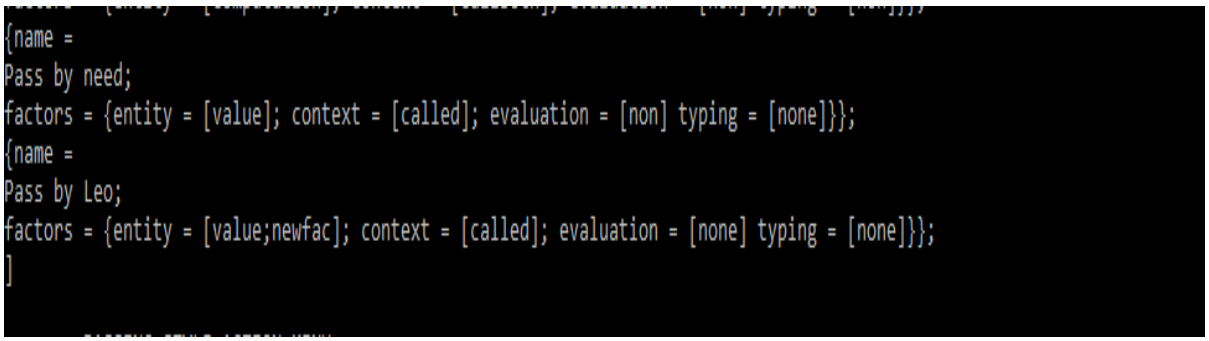

Fig.4.2 New style

Figure 4.2 shows the presence of the newly added style in the structure alongside passing styles known to the system. It also shows the newly defined instance “newfac” for the factor “entity”.

Figure 4.3 shows all possible instances that users can enter for the various factors that affect parameter passing.

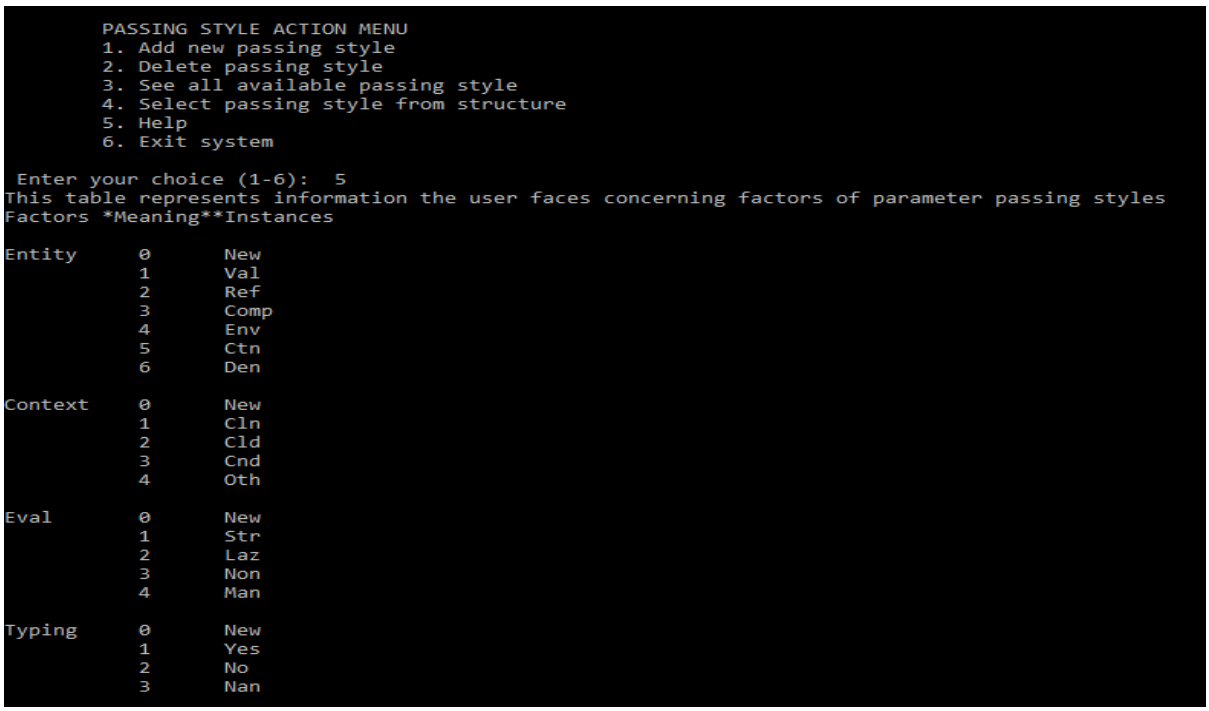

Fig. 4.3 Help menu

Figure 4.3 shows the help menu from which users can view possible instances for the various factors known to affect parameter passing.

Results obtained after a user deletes a passing style from the structure is shown in figure 4.3:

```
       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. Help
       6. Exit system

 Enter your choice (1-6):  2
 Available styles: (PBV;PBR;PBN;PBCR;PBNE)
-->Enter name for the style: pbr
Passing style ''Pass by reference'' deleted successfully.
No interpretation for the passing style
```

Fig.4.4 Delete a passing style

Figure 4.4 above shows how a passing style can be deleted (removed from structure). The user selects the passing style he/she wants to remove by writing out its correct name. The name is then used to delete that passing style from the list and a success message displayed upon a successful deletion. Names of available passing styles users can select from are displayed before the input field (like “pbr” for pass by reference) as shown in Figure 4.4 above. Figure 4.5 shows that the passing style was indeed deleted from the structure.

```
       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. Help
       6. Exit system

 Enter your choice (1-6):  4
 Available styles: (PBV;PBN;PBCR;PBNE)
-->Give passing style name: pbr
No such passing style, try again.
```

Fig. 4.5 : Style deleted

Figure 4.6 shows the results obtained after a user selects a passing style from structure.

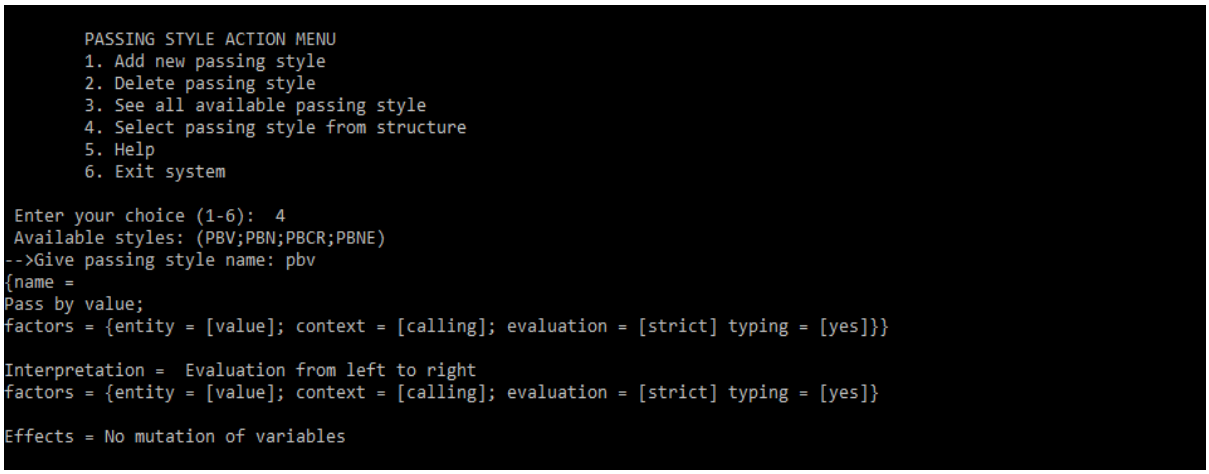

Fig. 4.6 Result after user selects passing style

Figure 4.6 shows the result after a user selects a passing style from the passing style structure. The user selects the passing style by writing out it name and the name is used to display the properties of the style if it exist in the structure. By properties, we mean the passing style, factors that effects it, it interpretation and effects induced by values of it factors as shown in Figure 4.6.

Figure 4.7 shows the results obtained after a user view all styles found in the structure.

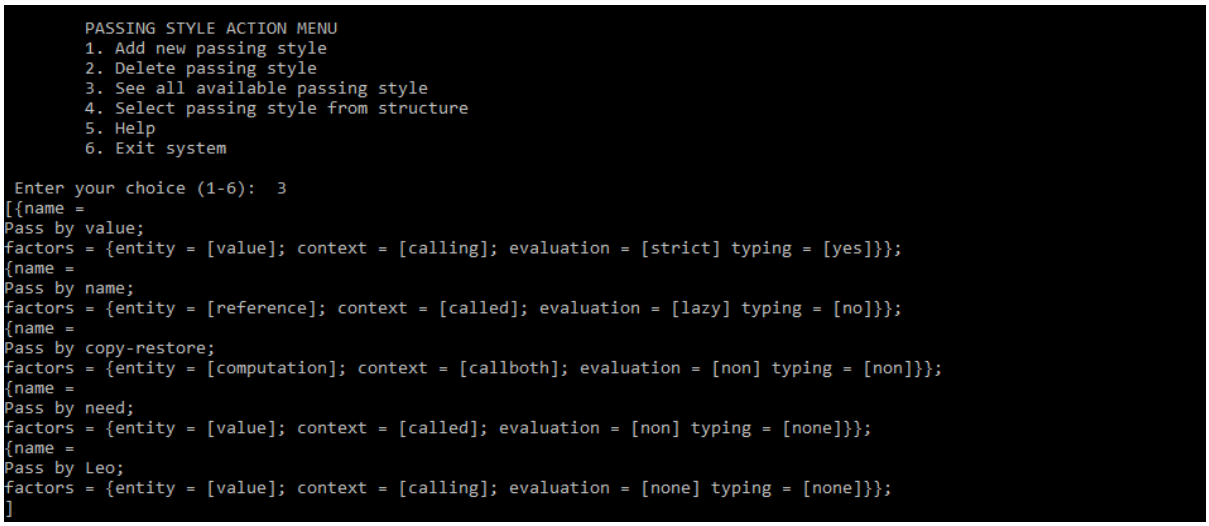

Fig. 4.7 All available styles

Figure 4.7 shows all available passing styles in the structure. It displays the passing styles in the structure after whatsoever operation that have been carried out on the structure.

All of the above Figures show the results obtained when we ran the program. There are still some minor changes to be done to the code so that it runs with a better level of perfection. This project develops a data structure for parameter passing styles where operations to add, remove passing styles are defined on the structure as shown in Figure 4.1 to Figure 4.5. The developed data structure for parameter passing can be used to hold future parameter passing styles in a way that the parameter passing styles can easily be used and manipulated in the structure. The developed structure can also be integrated into programming languages. This can provide programming languages with the ability to implement the semantic of if not all but a greater amount of parameter passing styles.

The main challenge I faced was identifying suitable data structures to use for holding the main entities involved in parameter passing and also implementing the flexibility of users to select (give) their own factor(s) of interest, and initialising the selected factor(s) at least one time when creating a passing style.

As a scope of improvement on this project, the semantics of a programming language can be used to implement the effects induced by the various passing styles including novel styles. This can be done by say generating code fragments that implements these effects induced by values of factors for the various passing styles instead of giving text description as done in this project.

Chapter 5

Conclusion

This project's sorts to develop a data structure for parameter passing that provides users with the tools to create, add, remove passing styles from the developed data structure, add an interpretation for a style, and to explore the various factors from which the behaviour of a style is defined. Through this work, we demonstrated how simple data structures could be exploited to construct a skeletal data structure which supports almost all parameter passing style type (known and novel styles alike). To ensure this structure is as flexible and efficient as possible, it effectively factors out entity, context, evaluation and typing which have been identified over the years as factors known to affect passing styles. The analysis section identified permissible operations that can be carried out on the developed structure for parameter passing. The main 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.

REFERENCES

[1] M. F. Eric Crank, Parameter passing and the lambda calculus, 1892.

[2] L. Moreau, Introduction to continuation, June 1994.

[3] J. Hickey, Introduction to Objective Caml, 2008.

[4] C. Verela, Typing, Parameter Passing, Lazy Evaluation, November 25, 2014.

APPENDIX

1. **Algorithm to remove style from structure**

```
start
Read style_name   (*name of style to be deleted)
exit = false, temp = []   (*temporal list *)
REPEAT
Read head_of_list        (*first passing style in list*)
if style_name = head_of_list.name
 then
        list1 <--  remove_hd   hd  list   (*removing style if found*)
        list <-- Add list1 temp
        exit = true
 else
        temp <-- Add  hd  temp   (*add head elements that does not match to temp*)
        list <-- tail
endif
UNTILL exit = true
stop
```

Fig.A : Algorithm to remove style

Figure A shows the algorithm for removing a passing style from the developed structure (list of styles).

2. Implementation of the design algorithm

```
(**Method that gives users options on they may wish to do
* @param user_styles : list of passing style to work with.
* @param interpretation_list: list holding interpretation for styles.
* *)
let rec usermind user_styles interpretation_list record effect_list=
  let () = display_menu ()
  in let () = print_string " Enter your choice (1-6): "
  in let opinion  = validate_input() in
  match (getOpinion opinion) with
 |"Add_style"-> let update = (insert_new_passing_style user_styles interpretation_list
effect_list    record)
 in let record = {update.record with ent = [5]; context = [5]; eval = [5]; typ = [5] }
                in  usermind update.styles update.interp record  update.effect
 |"Del_style"->let update = (delete_passing_style user_styles interpretation_list record
effect_list)
                    in   usermind  update.styles update.interp update.record update.effect
 |"Show_styles"->let update = (display_styles user_styles interpretation_list record
effect_list)
                in  usermind  update.styles update.interp update.record update.effect
 |"Specific_style"->let update = (view_specific_passing_style user_styles interpretation_list
record effect_list)
               in  usermind  update.styles update.interp update.record update.effect
 |"Help"->let update = (user_facing_information user_styles interpretation_list record
effect_list) in
                usermind  update.styles update.interp update.record update.effect
```

Fig.B : Implementation of design algorithm

Figure B shows the implementation of the design algorithm in Figure 2.6. It provides users with a number of actions from which they can select from. Depending on the user's choice a specific action is carried out except for the exit menu item which exits the system.

3. Method used to remove passing style

```
(**Method to Remove a passing style from list of passing style
* @param styles: list of passing style
* *)
  let rec delete_passing_style styles interpretation_list record effect=
  let () = print_string " Available styles: " in
  let () = print_styles_name styles in    (*shows available styles in the list of styles*)
  let () = print_string "\n-->Enter name for the style: " in  (*name of style to delete*)
  let name = read_line() in
  let rec removal styles interpretation_list=
  match styles with
  [] -> let () = print_string "Style don't exist." in []
  |hd :: tl -> if String.lowercase_ascii (name) = String.lowercase_ascii (getName hd.name)
  then let () = print_string "Passing style " in
  let () = print_string """ in
  let () = print_string (ful_name_style name)
  in  let () = print_string """ in
  remove_passing_style styles hd else
  hd :: removal tl interpretation_list
  in let styles = removal styles interpretation_list
  in let interp = delete_interpretation name interpretation_list
  in
  { styles = styles ; interp = interp; record = record ; effect = effect}
```

Fig.C : Implementation of deletion method

Figure C shows the implementation of the method used to remove a passing style from the list of styles.