

This is a brief document which explains certain aspects of the implementation document "paramstyle12.ml".

Used a list data structure to hold the various passing styles and used a record to represent specific passing styles which consist of passing style name and factors. Also, used a record to hold the various factors of the passing styles and used a list to hold instance/instances for factors that effects parameter passing.

Determining or defining a passing style depends on the values of these factors that are assumed to effect parameter passing style. When the code is first run, users are allowed to select (give) the factors type they are aware of by giving a number that correspond to the factor as indicated by a help menu that is first displayed. After selecting the factor, users are allowed (asked) to initialize the selected factor. After initializing the selected factor, users are asked if they wish to add more factors for the passing style. If these selected factors and their values corresponds to factors and values of known passing styles then the passing style is return alongside it properties (name and factors) and the effects induced by these values. If values do not corresponds to values for known passing style then a new passing style is created with an input name for the passing style and is added to the list of passing styles. This is illustrated below.

Available factors for passing styles

- 1 - Entity Passed
- 2 - Context
- 3 - Evaluation Strategy
- 4 - Typing

This table represents information the user faces concerning factors of parameter passing styles

Factors	*****Instance values	*****Meaning
---------	----------------------	--------------

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 Strategy	1	Strict
	2	Lazy
	3	Non
	4	Manual
Typing	1	Yes
	2	No

Fig.1. Factor numbers and meaning of instances keyed in.

```
Enter number for the selected factor type for the passing style:
1
Initialise selected factor: 2
More factors? (y/n):
```

Fig.2. Selecting factor and giving an instance of the selected factor

If users goes for more factors (entering yes or y) then the users are allow to give another factor for the passing style and initializing the selected factor. This can be seen below.

```
More factors? (y/n): y
Enter number for the selected factor type for the passing style:
3
Initialise selected factor: 9
More factors? (y/n): n
```

Fig.3.

Also a specific factor can be given more than one instance by selecting the factor more than one time and giving an instance (value) for all the selected case of the factor. The initialized instances for the factor are held in the instance list for that factor (e.g. as in ent below). For factors not of interest to the user, a default value would be assigned to this factor (the number 5 in this case). This can be seen as indicated below.

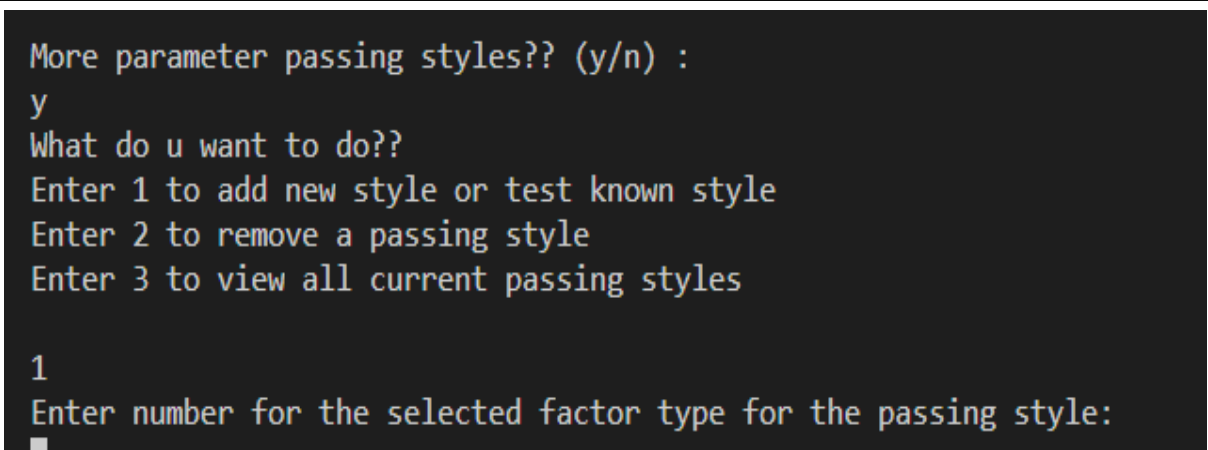
```
Enter number for the selected factor type for the passing style:
1
Initialise selected factor: 22
More factors? (y/n): y
Enter number for the selected factor type for the passing style:
1
Initialise selected factor: 32
More factors? (y/n): n
val record : time = {ent = [22; 32]; context = [5]; eval = [5]; typ = [5]}
Enter name for your passing style:
```

Fig.4. Selecting factor 1 (entity_type) more than one time.

When users go for no more factors, if selected factor(s) and initialized value(s) correspond to that of known passing style (e.g. 1, 1, 1, 1 for all four factors), then the passing style name is displayed alongside its properties (name and factors with their values) after which it has displayed the list of passing style.

For values of selected factor(s) that does not correspond to that of known passing style (e.g. 8, 8, for any two factors), a name is demanded for the passing style and a new passing style is constructed with those values for the selected factors and add the passing style to the list of passing styles.

After adding/displaying the passing style to/from list of styles, a select menu is then displayed asking the user if they want more passing style. If users go for yes or y then a submenu is displayed giving them options (operations) from which they can select from and it's immediately initiated once it is selected. If the users go for no, the final list of passing styles is displayed which contain the new passing style added if users defines their own new style. This can be seen as indicated below.



```
More parameter passing styles?? (y/n) :  
y  
What do u want to do??  
Enter 1 to add new style or test known style  
Enter 2 to remove a passing style  
Enter 3 to view all current passing styles  
  
1  
Enter number for the selected factor type for the passing style:
```

Fig.5

For values of factors (all four factors) that corresponds to a known passing style, the passing style is displayed alongside with its properties (name and factors). For example, values that corresponds to call by value (1, for all four selected factors), the passing style is displayed alongside its properties (name and factors). This can be seen below.

```

passing style is: Pass by Value
val test : style_factors =
  {name = Passing_style "Pass by Value";
   factor =
    {fac1 = Entity_type [1]; fac2 = Context_type [1];
     fac3 = Evaluation_strat [1]; fac4 = Correct_type [1]}}
Factors that effects it and with their value are:

val fac : factors =
  {fac1 = Entity_type [1]; fac2 = Context_type [1];

```

Fig.5. Known passing style

For the effects induced by values of factors for this passing style, used a record to hold the effects that the specific values of factors induces and is used to showcase this effects. It ask for a number which is used to initialize a variable `init_var`, which 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, we see that the value entered by user for the variable `init_var` does not change as both values (`init_var` and `final_var`) are the same after performing a computation and not different form the value the user entered. This can be seen as shown below.

Effects Induced by specific values of factors is:

Enter value for a variable t: 5

After computation, variable is now:

```
val effect : by_val = {init_var = 5; eval = 8; final_var = 5}
```

Fig.6

For adding/defining a new passing style (when values of selected factor(s) does not correspond to that of known passing style (e.g. 8, 8, for any two factors), new style is constructed and added to list of styles, and users then asked if they want more passing styles. If users go for no, the list of passing style is displayed which contain the newly added style. This can be seen below.

```
Enter number for the selected factor type for the passing style:
1
Initialise selected factor: 8
More factors? (y/n): y
Enter number for the selected factor type for the passing style:
2
Initialise selected factor: 8
More factors? (y/n): n
val record : time = {ent = [8]; context = [8]; eval = [5]; typ = [5]}
Enter name for your passing style: Leonardo style
```

Fig.7. Defining new passing style

```
val styles : style_factors list =
  [{name = Passing_style "Pass by Value";
    factor =
      {fac1 = Entity_type [1]; fac2 = Context_type [1];
       fac3 = Evaluation_strat [1]; fac4 = Correct_type [1]}};
  {name = Passing_style "Pass by Reference";
    factor =
      {fac1 = Entity_type [1]; fac2 = Context_type [2];
       fac3 = Evaluation_strat [3]; fac4 = Correct_type [3]}};
  {name = Passing_style "Pass by Name";
    factor =
      {fac1 = Entity_type [2]; fac2 = Context_type [2];
       fac3 = Evaluation_strat [2]; fac4 = Correct_type [2]}};
  {name = Passing_style "Pass by Copy_restore";
    factor =
      {fac1 = Entity_type [3]; fac2 = Context_type [3];
       fac3 = Evaluation_strat [3]; fac4 = Correct_type [3]}};
  {name = Passing_style "Pass by Need";
    factor =
      {fac1 = Entity_type [1]; fac2 = Context_type [2];
       fac3 = Evaluation_strat [3]; fac4 = Correct_type [4]}};
  {name = Passing_style "Leonardo_style";
    factor =
      {fac1 = Entity_type [8]; fac2 = Context_type [8];
       fac3 = Evaluation_strat [5]; fac4 = Correct_type [5]}}
```

Fig.8. List containing newly added passing style.

Also for newly defined passing style, the style is displayed with its properties (name and factors) after it has been defined. This can be seen as shown below.

```
fac3 = Evaluation_strat [5]; fac4 = Correct_type [5]]]]  
passing style is :Leonardo_style  
val test : style_factors =  
  {name = Passing_style "Leonardo_style";  
   factor =  
     {fac1 = Entity_type [8]; fac2 = Context_type [8];  
      fac3 = Evaluation_strat [5]; fac4 = Correct_type [5]}}
```

Fig.9. newly defined style.

After this process, if users want to still carry out operations like add or remove a passing style, the method is invoked by typing the command (calling the method) “usermind styles” where usermind is the method invoked and styles refers to the current list of passing styles and selecting yes for more passing styles then select the operation they wish to carry out.

The method responsible for users selecting factors and initializing them can be seen below on the figure (Fig.10.) below, though not complete because of screen size issue.

Also the method used to ask users if they want more passing style (as in Fig.5) can be seen on the figure (Fig.11) below.

The methods “insert_new_passing_style” and “removal” carry out operations for inserting and removing a passing style from the list of passing styles respectively.

```

let rec factor_initializer record = let () = print_endline "Enter number for the selected factor type for the passing
in let t = read_int ()
in if t = 1 then
  let () = print_string "Initialise selected factor: " in let y = read_int() in
  let () = print_string "More factors? (y/n): " in let opt = read_line() in
  if opt.[0] = 'y' then
    factor_initializer ({record with ent = insert_instance (record.ent) y })
  else
    preserve_default {record with ent = insert_instance (record.ent) y }
  else
    if t = 2 then
      let () = print_string "Initialise selected factor: " in let y = read_int() in
      let () = print_string "More factors? (y/n): " in let opt = read_line() in
      if opt.[0] = 'y' then
        factor_initializer ({record with context = insert_instance (record.context) y })
      else
        preserve_default {record with context = insert_instance (record.context) y }
    else
      if t = 3 then
        let () = print_string "Initialise selected factor: " in let y = read_int() in
        let () = print_string "More factors? (y/n): " in let opt = read_line() in
        if opt.[0] = 'y' then
          factor_initializer ({record with eval = insert_instance (record.eval) y })
        else
          preserve_default {record with eval = insert_instance (record.eval) y }

```

Fig.10

```

297 (**Method that gives users options on what next they may wish to do*)
298 let rec usermind styles = let () = print_endline "Have you had enough???\n Enter 1 for yes\n Enter 2 for no" in let op:
299 if opinion = 2 then
300   (let operation = let () = print_endline "What do u want to do?? \n Enter 1 to add new style or test known style\n \n
301   in let opinion2 = read_int() in
302     (if opinion2 = 1 then
303       usermind (insert_new_passing_style styles)
304     else if opinion2 = 2 then
305       usermind (removal styles)
306     else if opinion2 = 3 then
307       styles
308     else styles ) in operation )
309 else styles
310
311
312 let styles = usermind styles

```

Fig.11