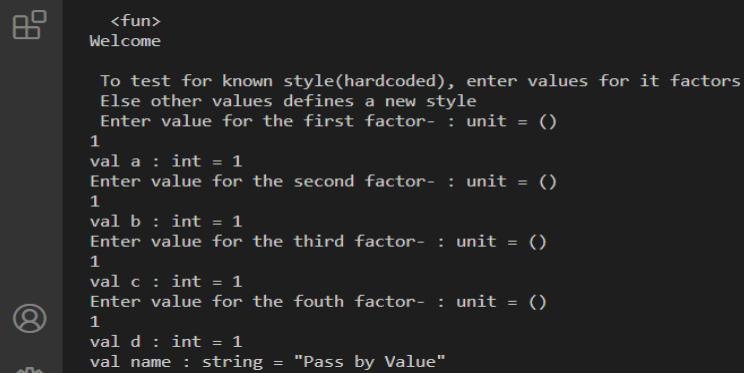


This is a brief document which explains certain aspects of the implementation document "paramstyle8.ml".

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.

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, values of these factors (Entity type, Context, Evaluation and typing) are demanded. If these values corresponds to 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.

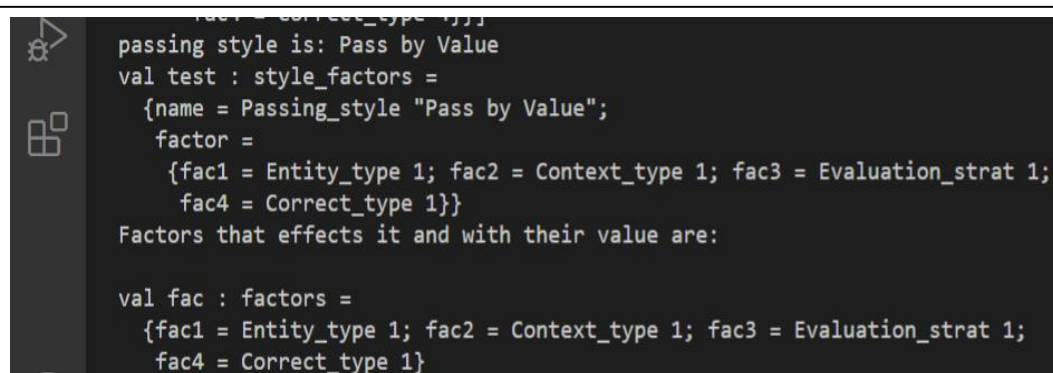


```
<fun>
Welcome

To test for known style(hardcoded), enter values for it factors
Else other values defines a new style
Enter value for the first factor- : unit = ()
1
val a : int = 1
Enter value for the second factor- : unit = ()
1
val b : int = 1
Enter value for the third factor- : unit = ()
1
val c : int = 1
Enter value for the fourth factor- : unit = ()
1
val d : int = 1
val name : string = "Pass by Value"
```

Fig.1

For values of known passing style (1, 1, 1, 1), this will display the name of the passing style, properties (name and factors) after which it has display the list of passing styles. This can be seen in Fig.2 below.

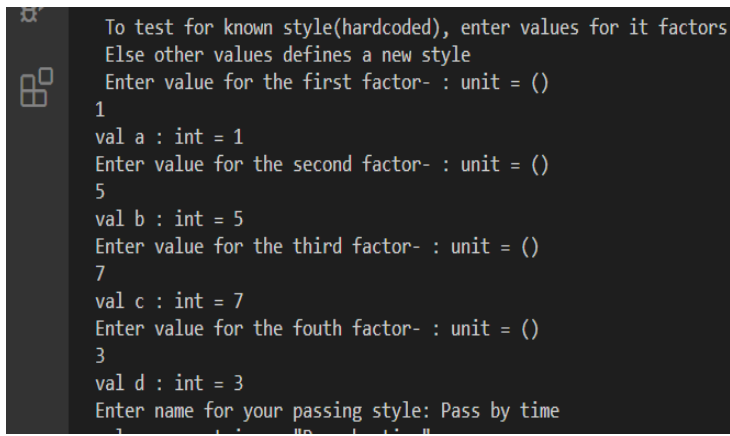


```
fact = correct_type 1}}
passing style is: Pass by Value
val test : style_factors =
  {name = Passing_style "Pass by Value";
   factor =
    {fac1 = Entity_type 1; fac2 = Context_type 1; fac3 = Evaluation_strat 1;
     fac4 = Correct_type 1}}
Factors that effects it and with their value are:

val fac : factors =
  {fac1 = Entity_type 1; fac2 = Context_type 1; fac3 = Evaluation_strat 1;
   fac4 = Correct_type 1}
```

Fig.2

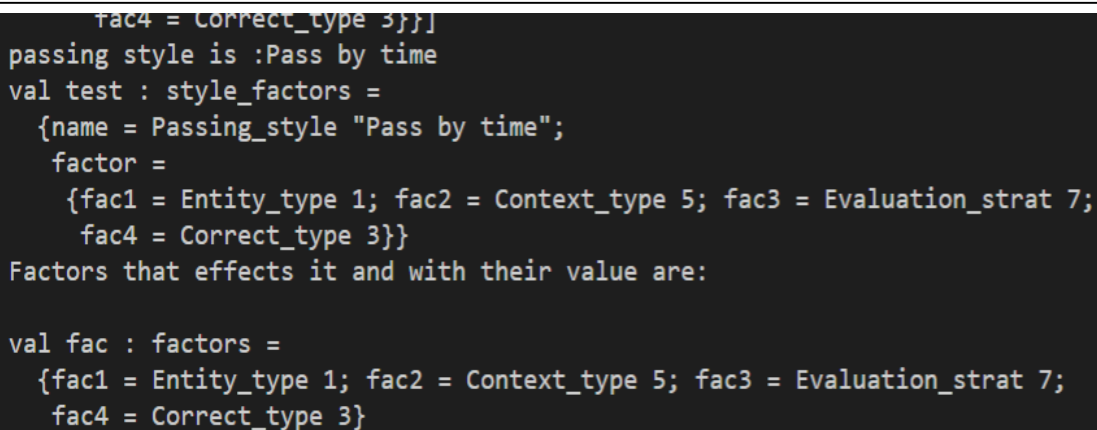
For values of factors that doesn't correspond to known passing styles (1, 5, 7, 3), this will ask for a name to give to the passing style with values of factors (1, 5, 7, 3), constructs the passing style then adds it to the list of passing style. This can be seen in Fig.3 below.



```
To test for known style(hardcoded), enter values for it factors
Else other values defines a new style
Enter value for the first factor- : unit = ()
1
val a : int = 1
Enter value for the second factor- : unit = ()
5
val b : int = 5
Enter value for the third factor- : unit = ()
7
val c : int = 7
Enter value for the fourth factor- : unit = ()
3
val d : int = 3
Enter name for your passing style: Pass by time
```

Fig.3

The new passing style is then added to the list of passing style and the style is displayed (its name and factors) after displaying the list of passing style which has the new style defined. This can be seen as indicated in the figure below.



```
fac4 = Correct_type 3}}]
passing style is :Pass by time
val test : style_factors =
{name = Passing_style "Pass by time";
 factor =
 {fac1 = Entity_type 1; fac2 = Context_type 5; fac3 = Evaluation_strat 7;
  fac4 = Correct_type 3}}
Factors that effects it and with their value are:

val fac : factors =
{fac1 = Entity_type 1; fac2 = Context_type 5; fac3 = Evaluation_strat 7;
 fac4 = Correct_type 3}
```

Fig.4

All the above mentioned is produced by the piece of code below

```
277
278 print_string "Welcome\n\n To test for known style(hardcoded), enter values for it factors\n Else other values defines
279 let a = read_int();;
280 print_string "Enter value for the second factor"
281 let b = read_int() ;;
282 print_string "Enter value for the third factor"
283 let c = read_int() ;;
284 print_string "Enter value for the fourth factor"
285 let d = read_int();;
286 let name = get_name (Entity_type a) (Context_type b) (Evaluation_strat c) (Correct_type d) ;; (**Variable "name" holds
287
288 let styles = sanitize (Entity_type a) (Context_type b) (Evaluation_strat c) (Correct_type d) user_styles name (**Holdi
289 let test = paramstyle (Entity_type a) (Context_type b) (Evaluation_strat c) (Correct_type d) styles name (**Returns t
290 let fac = print_endline ("Factors that effects it and with their value are:\n ") ; f test.factor (**Displaying a recor
291
```

Fig.5

For the known passing styles, effects induced by specific values of factors are displayed as indicated below. This case is used to showcase (explain) the effects of call by value.

```
Effects Induced by specific values of factors is:
Enter value for a variable t: 3
After computation, Reference var is now:
- : by_val = {init_var = 3; eval = 6; final_var = 3}
val usermind : style_factors list -> style_factors list = <fun>
Have you had enough???
```

Fig.6

As seen in the figure above, `by_val` is a record type that holds the effect that the specific values of factors induced and is used to showcase this effects. It ask for a number which is used to initialize a variable (`v`), `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, we see that the value of the variable (`v`) used as argument in the method `f` is not modified after the computation. The piece of code that produces this effect can be seen in the figure below.

```

131 Context_type 1, Evaluation_strat 1, Correct_type 1) -> let no_mut = (**For call by value*)
132 | let () = print_string "Enter value for a variable t: " in
133 | let t = read_int() in
134 | let r = ref t in
135 | let v = r in print_endline "After computation, Reference var is now:";
136 | (let f = fun r -> r := !v+3; !r
137 | in let b = {init_var = t; eval = f (v); final_var = !v }
138 | in let ft = fun x -> x in ft b | )
139 | in no_mut

```

Fig.7

This effect is only for call by value. Other effects can be seen in code.

After this effect induced by specific values of factors has been displayed, a select menu is displayed asking if satisfaction has been achieved. If user goes for no (by selecting 2), a menu is displayed giving options on what they may wish to do next. If yes, then the list containing all passing styles (be it user defined or known styles) is displayed.

```

val usermind : style_factors list -> style_factors list = <fun>
Have you had enough???
Enter 1 for yes
Enter 2 for no
2
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

```

Fig.8

If option 2 is selected in the first menu, then a list of operations will be displayed for one to select from, and when selected, the operation is immediately initiated. This can be seen as shown below.

Selecting 1 for add new passing style, the values of the factors known to effect the passing style are entered and also a name for the passing style. The style is then added to the list of passing styles and the menu asking if satisfaction reached is displayed again. To see the list of styles, the selected option this time should be 1. This can be seen as shown on the figure below.

```
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 value for the first factor: 6
Enter value for the second factor: 6
Enter value for the third factor: 6
Enter value for the fourth factor: 7
Enter name for your passing style: Naldo
passing style is :Naldo
Have you had enough???
Enter 1 for yes
Enter 2 for no
1
```

Fig.9

Selecting or entering 1 displays the list of passing styles included the recently add style or the result of the previous operation carried out if the option was entered. The list of passing style displayed when 1 is entered can be seen on the below. It includes the properties of the recently added passing style.

```
Have you had enough???  
Enter 1 for yes  
Enter 2 for no  
1  
- : style_factors list =  
[{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 "Naldo";  
  factor =  
    {fac1 = Entity_type 6; fac2 = Context_type 6; fac3 = Evaluation_strat 6;  
    fac4 = Correct_type 7}}]
```

Activate Windows
Go to Settings to activate Windows

Fig.10

The piece of code used to perform the above relating to Fig 8, 9, and 10 can be seen on the figure below.

```
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
313  tt
```

Fig.11

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

After this process, if users wants 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.