

Rapport de fin de projet

PATUTA Volodymyr, FAUSSETTE Yann

06/01/20

Bienvenue sur notre rapport de fin de projet :

- Vous trouverez ci-joint le dépôt git contenant notre projet
- Le listing de nos fichiers
- Les commentaires associé aux fonctions et des explications vis à vis de nos choix
- Un essai de notre programme
- Le mode d'emploi
- Le code

En cas de problème, RTFM !

Part I

Listing des fichiers

- menu.rkt → Menu de notre programme
- ressource.rkt → Contient nos fonctions qui créeront les fonctions voulus
- exceptions.rkt → Contient les fonctions non uniformisées

Part II

Commentaires

Le fichier menu.rkt est le corp du projet contenant une forêt de "if" permettant de faire des appels de fonctions de les deux autres fichiers. Nous avons fait le choix de tout centralisé vers deux menus (car nous avons mis les exceptions à part) pour évité à l'utilisateur d'avoir à appelé toutes nos fonctions. Les autres fonctions sont basé sur le modèle vu en cours.

```
1 (define (fct_s nom oper para p) ;FCT somme/produit fcts num
2   (list 'define (list nom para)
3     (puiss)
4     (list 'define (list 'aux para 'c)
5       '(if (= ,para 0)
6         c
7         ,(list 'aux '(- ,para 1) '(',oper ,(list 'puiss p
8   para) c))))
9   (if (equal? oper '+)
10     '(aux ,para 0)
11     '(aux ,para 1))))
```

On remarque tout de même des différences avec le modèle que nous avons adapté à nos besoins en ajoutant notamment des conditions pour la rédaction de la fonction. /newline Le problème majeur rencontré durant la réalisation du projet à été l'uniformisation des calculs mathématique, c'est à dire trouvé les similitudes entre les calculs fait par les fonctions.

Part III

Jeu d'essai

```
menu.rkt - DrRacket
File Edit View Language Racket Insert Scripts Tabs Help
menu.rkt* (define ...)
Check Syntax Debug Macro Stepper Run Stop

1: Untitled 2: menu.rkt

#lang racket
(require "resource.rkt")
(require "exceptions.rkt")

(define (len l)
  (define (le l n)
    (if (pair? l)

Welcome to DrRacket, version 7.2 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (fct)
(sommeliste + l)
(define (sommeliste l)
  (define (puiss p n) (define (puiss_aux p n stock) (if (= p 0) stock (puiss_aux (- p 1) n (* stock n)))) (puiss_aux p n 1))
  (define (ps_aux l c) (if (pair? l) (ps_aux (cdr l) (+ c (puiss 1 (car l)))) c))
  (ps_aux l 0))
> (eval (fct))
(sommeliste + l)
> (sommeliste '(1 2))
1
>

Determine language from source imported from "resource.rkt" 13:2 487.49 MB
1 2 3 4 no IPv6 165.4 GiB | W: (100% at Redmi 8) 192.168.43.4 | CHR 96.50% 00:05:02 0.02 | 2020-01-02 15:23:30 | J: 0%
```

Part IV

Mode d'emploi

Pour l'exécution du programme, on retrouvera deux fonctions :

- fct
- fct_exception

Une fois la fonction exécutée, il faudra rentrer des informations qui serviront à renvoyer les fonctions désirées.

Il s'agit de prototyper votre fonction.

Vous commencerez donc par l'ouverture de parenthèse, le nom de votre fonction, l'opérateur si il y en a un et pour finir des variables et/ou une valeur tel que la puissance. On notera que dans le cas d'un renvoi d'un Booléen l'utilisation du "?".

Si on crée une fonction utilisant des listes, la première variable sera obligatoirement un l ou ll.

Voici quelques prototypage fonctionnant avec notre programme :

1 Fonctions numériques (fct)

```
1 (somme + n)
2 (sommecarre + n 2)
3 (produit * n)
4 (produitcube * n 3)
5 (puissance a b)
6 (soustraction - a b)
7 (division / a b)
```

2 Fonctions avec les listes (fct)

```
1 (doubleton l 2)
2 (nieme l n)
3 (somme + l)
4 (sommecarre + l 2)
5 (sommepositive + l >)
6 (produit * l)
7 (produitcube * l 3)
8 (plus_petit_element < l)
9 (plus_grand_element > l)
10 (plus_grand_que_n > l n)
11 (plus_petit_que_n < l n)
```

3 Exceptions pour les fonctions numériques (fct_excep)

```
1 (pair n ?)
2 (reset / a b r)
3 (pgcd / a b)
4 (ppcm * a b)
5 (fib n)
```

4 Exceptions pour les fonctions avec des listes (fct_excep)

```
1 (autant l n) #la variable n ne peut pas etre change par un autre
    caractere
2 (medianboolean l n ?)
3 (element_median l)
4 (concat l l1)
5 (miroir l m) #la variable m ne peut pas etre change par un autre
    caractere
```

Part V

Code

5 menu.rkt

```
1 #lang racket
2 (require "ressource.rkt")
3 (require "exceptions.rkt")
4
5 (define (len l)
6   (define (le l n)
7     (if (pair? l)
8         (le (cdr l) (+ n 1))
9         n))
10  (le l 0))
11
12 (define (fct)
13   (let ((l (read)))
14     (let ((n (len l)))
15       (if (or (< n 3) (> n 4))
16           (write "RTFM!")
17           (if (or (equal? (caddr l) 'l) (equal? (caddr l) 'l1) (
18               equal? (cadr l) 'l))
19               (if (pair? (caddr l))
20                   (if (and (equal? (cadr l) '+) (number? (caddr l)
21                       (make_ls (car l) (cadr l) (caddr l) (caddr l) 1
22                           ) 0)
23                       (if (and (equal? (cadr l) '*') (number? (
24                           caddr l)))
25                           (make_ls (car l) (cadr l) (caddr l) (
26                           caddr l) '=)
27                           (if (and (equal? (cadr l) '+) (equal? (
28                           caddr l) '>))
29                               (make_ls (car l) (cadr l) (caddr l) 1
30                                   (caddr l))
31                               (if (or (equal? (cadr l) '>) (equal?
32                                   (cadr l) '<))
33                                   (make_ptgr (car l) (cadr l) (
34                                   caddr l) (caddr l))
35                                   (write "RTFM!"))))))
36               (if (equal? (cadr l) '+)
37                   (make_ls (car l) (cadr l) (caddr l) 1 0)
38                   (if (equal? (cadr l) '*')
39                       (make_ls (car l) (cadr l) (caddr l) 1 '=)
40                       (if (or (equal? (cadr l) '<) (equal? (
41                           cadr l) '>))
42                           (make_ptgr (car l) (cadr l) (caddr l)
43                               0)
44                           (if (and (or (equal? (cadr l) 'l) (
45                               equal? (cadr l) 'l1)) (number? (caddr l)))
46                               (make_ton (car l) (cadr l) (caddr
47                                   l))
48                               (make_nieme (car l) (cadr l) (
```

```

caddr 1)))))
37         (if (pair? (cddddr 1))
38             (if (number? (cddddr 1))
39                 (fct_s (car 1) (cadr 1) (caddr 1) (cddddr 1))
40                 (if (or (equal? (cadr 1) '+) (equal? (cadr 1)
'-))
41                     (make_add_ss (car 1) (cadr 1) (caddr 1) (
caddr 1))
42                     (if (or (equal? (cadr 1) '*') (equal? (cadr
1) '/))
43                         (make_mult_div (car 1) (cadr 1) (caddr
1) (cddddr 1))
44                         (if (equal? (cadr 1) '^')
45                             (puiss)
46                             (display "RTFM!")))))
47                 (if (or (equal? (cadr 1) '*') (equal? (cadr 1) '+))
48                     (fct_s (car 1) (cadr 1) (caddr 1) 1)
49                     (display "RTFM!")))))
50
51 (define (fct_excep)
52     (let ((l (read)))
53         (let ((n (len l)))
54             (if (= n 5)
55                 (make_reste (car 1) (cadr 1) (caddr 1))
56                 (if (= n 4)
57                     (if (equal? (cadr 1) '/')
58                         (make_pgcd (car 1) (cadr 1) (caddr 1))
59                         (if (equal? (cadr 1) '*')
60                             (make_ppcm (car 1) (cadr 1) (caddr 1))
61                             (if (equal? (caddr 1) '?')
62                                 (make_mediantf (car 1) (cadr 1) (caddr 1)
63                                     (display "RTFM!")))))
64                     (if (= n 3)
65                         (if (equal? (caddr 1) '?')
66                             (make_pair (car 1) (cadr 1))
67                             (if (and (equal? (cadr 1) 'l) (equal? (caddr
1) 'n))
68                                 (make_autant (car 1) (cadr 1) (caddr 1))
69                                 (if (equal? (cadr 1) 'm)
70                                     (make_mirroir (car 1) (cadr 1))
71                                     (make_concat (car 1) (cadr 1) (caddr
1)))))
72                     (if (= n 2)
73                         (if (equal? (cadr 1) 'l)
74                             (make_median (car 1) (cadr 1))
75                             (make_fib (car 1) (cadr 1)))
76                         (display "RTFM!"))))))))

```

6 ressource.rkt

```
1 #lang racket
2 (provide make_ls)
3 (provide make_ptgr)
4 (provide make_add_ss)
5 (provide fct_s)
6 (provide make_mult_div)
7 (provide make_ton)
8 (provide make_nieme)
9 (provide puiss)
10
11 (define (puiss)
12   (list 'define '(puiss p n)
13         (list 'define '(puiss_aux p n stock)
14               '(if (= p 0)
15                     stock
16                     (puiss_aux (- p 1) n (* stock n))))
17         '(puiss_aux p n 1)))
18 (define (make_nbele)
19   (list 'define '(nbele l)
20         (list 'define '(nb_aux l n)
21               '(if (pair? l)
22                     (nb_aux (cdr l) (+ n 1))
23                     n))
24         '(nb_aux l 0)))
25
26 (define (fct_s nom oper para p) ;FCT somme/produit fcts num
27   (list 'define (list nom para)
28         (puiss)
29         (list 'define (list 'aux para 'c)
30               '(if (= ,para 0)
31                     c
32                     ,(list 'aux '(- ,para 1) '(',oper ,(list 'puiss p
33                       para) c))))
34         (if (equal? oper '+)
35             '(aux ,para 0)
36             '(aux ,para 1))))
37
38 (define (make_ls nom oper param p sym) ;FCT somme produit listes
39   (list 'define (list nom param)
40         (puiss)
41         (list 'define (list 'ps_aux param 'c)
42               (list 'if (list 'pair? param)
43                     (if (equal? sym '0)
44                         '(ps_aux (cdr ,param) (,oper c (puiss ,p (
45                           car ,param))))
46                         (make_sp oper param sym p))
47                     'c))
48         (fin oper param)))
49 (define (fin oper param) ;pour somme produit (fin de la fct) listes
50   (if (equal? oper '*)
51       (list 'ps_aux param 1)
52       (list 'ps_aux param 0)))
53 (define (make_sp oper param sym p) ;pour produit / somme non nul
54   listes
55   (list 'if '(',sym 0 (car ,param))
```



```

53      '(ps_aux (cdr ,param) c)
54      '(ps_aux (cdr ,param) (,oper (puiss ,p (car ,param)) c))))
55
56 (define (make_p param1 oper) ;pour ppt normal etc
57   (list 'if (list oper 'c '(car ,param1))
58     (list 'ptgr_aux '(cdr ,param1) 'c)
59     (list 'ptgr_aux '(cdr ,param1) '(car ,param1))))
60 (define (make_p1 param1 param2 oper) ;pour ppt que n etc
61   (list 'if (list oper '(car ,param1) param2)
62     (list 'ptgr_aux '(cdr ,param1) param2 '(+ c 1))
63     (list 'ptgr_aux '(cdr ,param1) param2 'c)))
64 (define (par nom param1 param2) ;pour ppt (soit ppt soit ppt que n)
65   (if (number? param2)
66     ',(list nom param1)
67     ',(list nom param1 param2)))
68 (define (par2 nom param1 param2) ;pour ppt (soit ppt soit ppt que n
69   )
70   (if (number? param2)
71     ',(list 'ptgr_aux param1 'c)
72     ',(list 'ptgr_aux param1 param2 'c)))
73 (define (make_ptgr nom oper param1 param2) ;FCT ppt plgr
74   (list 'define (par nom param1 param2)
75     (list 'define (par2 nom param1 param2)
76       (list 'if (list 'pair? param1)
77         (if (number? param2)
78           (make_p param1 oper)
79           (make_p1 param1 param2 oper))
80         'c))
81     (if (number? param2)
82       (list 'ptgr_aux param1 '(car ,param1))
83       (list 'ptgr_aux param1 param2 0))))
84 (define (make_add_ss nom oper para1 para2) ;FCT addition/
85   soustraction +1 -1
86   (list 'define (list nom para1 para2)
87     '(if (= ,para2 0) ,para1
88       ,(list nom '(',oper ,para1 1) '(- ,para2 1))))
89 (define (make_mult_div nom oper param1 param2 born sym) ;FCT
90   multiplication / division + et - num
91   (let ((inv (if (equal? oper '*)
92     1
93     param2)))
94     (inv2 (if (equal? oper '*)
95       param2
96       1)))
97     (list 'define (list nom param1 param2)
98       (list 'define (list 'aux param1 param2 'c)
99         '(if (,sym ,param1 ,born)
100           c
101           ,(list 'aux '(- ,param1 ,inv) param2 '(+ c ,inv2)
102             )))
103         '(aux ,param1 ,param2 0))))
104 (define (make_ton nom param_list nb) ;FCT singl/double/triple TON
105   listes
106   (list 'define (list nom param_list)

```

```

105      (make_nbele)
106      '(if (= (nbele ,param_list) ,nb)
107            #t
108            #f)))
109
110 (define (make_nieme nom param_list nb) ;FCT nieme element de la
      liste
111   (list 'define (list nom param_list nb)
112         '(if (pair? l)
113               (if (= ,nb 1)
114                     (car ,param_list)
115                     (,nom (cdr ,param_list) (- ,nb 1)))
116               '()))))

```

7 exceptions.rkt

```
1 #lang racket
2 (require "ressource.rkt")
3 (provide make_reste)
4 (provide make_pgcd)
5 (provide make_ppcm)
6 (provide make_fib)
7 (provide make_pair)
8 (provide make_concat)
9 (provide make_mirroi)
10 (provide make_autant)
11 (provide make_mediantf)
12 (provide make_median)
13 (provide make_racine)
14
15
16 ;make reste
17 (define (make_reste nom param1 param2)
18   (list 'define (list nom param1 param2)
19     '(if (< ,param1 0)
20       (+ ,param1 ,param2)
21       (nom (- ,param1 ,param2) ,param2))))
22 ;make pgcd
23 (define (make_pgcd nom param1 param2)
24   (list 'define (list nom param1 param2)
25     '(if (= ,param1 ,param2) ,param1
26         (if (> ,param1 ,param2)
27             (nom (- ,param1 ,param2) ,param2)
28             (nom ,param1 (- ,param2 ,param1))))))
29 ;make ppcm
30 (define (make_ppcm nom param1 param2)
31   (list 'define (list nom param1 param2)
32     (make_pgcd 'pgcd param1 param2)
33     '(/ (* ,param1 ,param2) (pgcd ,param1 ,param2))))
34 ;pair / impair
35 (define (make_pair nom param)
36   (list 'define (list nom param)
37     '(if (= ,param 0) #t
38         (if (> ,param 0)
39             (nom (- ,param 2))
40             #f))))
41 ;racine
42   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
43
44 (define (make_racine nom param) (display ":'("))
45 ;fib
46 (define (make_fib nom param)
47   (list 'define (list nom param)
48     (list 'define '(fib_aux ,param z x)
49       '(if (= ,param 0)
50           (if (< z x) x z)
51           (if (< z x)
52               (fib_aux (- ,param 1) (+ z x) x)
53               (fib_aux (- ,param 1) z (+ x z))))))
54     '(fib_aux ,param 1 0)))
55 ;median #t #f
```

```

54 (define (make_mediantf nom param_list param)
55   (list 'define (list nom param_list param)
56     (make_ptgr 'ppt '< 'z 'x)
57     (make_ptgr 'plgr '> 'z 'x)
58     (make_autant 'autant 'l 'n)
59     (list 'define '(medi_aux p g ,param_list ,param)
60       '(if (pair? ,param_list)
61         (if (autant ,param_list ,param)
62           #t
63           (if (= p (+ g 1))
64             #t
65             (if (= (+ p 1) g)
66               #t
67               #f)))
68         #f)))
69     '(medi_aux (ppt ,param_list ,param) (plgr ,param_list ,
70       param) ,param_list ,param)))
71 ;element median
72 (define (make_median nom param_list)
73   (list 'define (list nom param_list)
74     (make_mediantf 'median? param_list 'n)
75     (make_nieme 'nieme param_list 'n)
76     (list 'define '(med_aux ,param_list n)
77       '(if (pair? ,param_list)
78         (if (median? ,param_list n)
79           n
80           (med_aux ,param_list (nieme ,param_list (+ 1
81             n))))
82         '(med_aux ,param_list 1)))
83 ;concat
84 (define (make_concat nom param1 param2)
85   (list 'define (list nom param1 param2)
86     (make_mirroir 'mirroir param1)
87     (list 'define '(conc_aux ,param1 ,param2 13)
88       '(if (pair? ,param1)
89         (conc_aux (cdr ,param1) ,param2 (cons (car ,
90           param1) 13))
91         (if (pair? ,param2)
92           (conc_aux ,param1 (cdr ,param2) (cons (car ,
93             param2) 13))
94           13)))
95     '(mirroir (conc_aux ,param1 ,param2 '()))))
96 ;mirroir
97 (define (make_mirroir nom param)
98   (list 'define (list nom param)
99     (list 'define '(mir_aux ,param i1)
100       '(if (pair? ,param)
101         (mir_aux (cdr ,param) (cons (car ,param) i1))
102         i1))
103     '(mir_aux ,param '()))
104 ;autant
105 (define (make_autant nom param_list param)
106   (list 'define (list nom param_list param)
107     (make_ptgr 'ppt '< 'z 'x)
108     (make_ptgr 'plgr '> 'z 'x)
109     '(if (pair? ,param_list)

```

```
107      (if (= (plgr ,param_list ,param) (ppt ,param_list ,
      param))
108          #t
109          #f)
110      #t)))
```