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 dépot 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

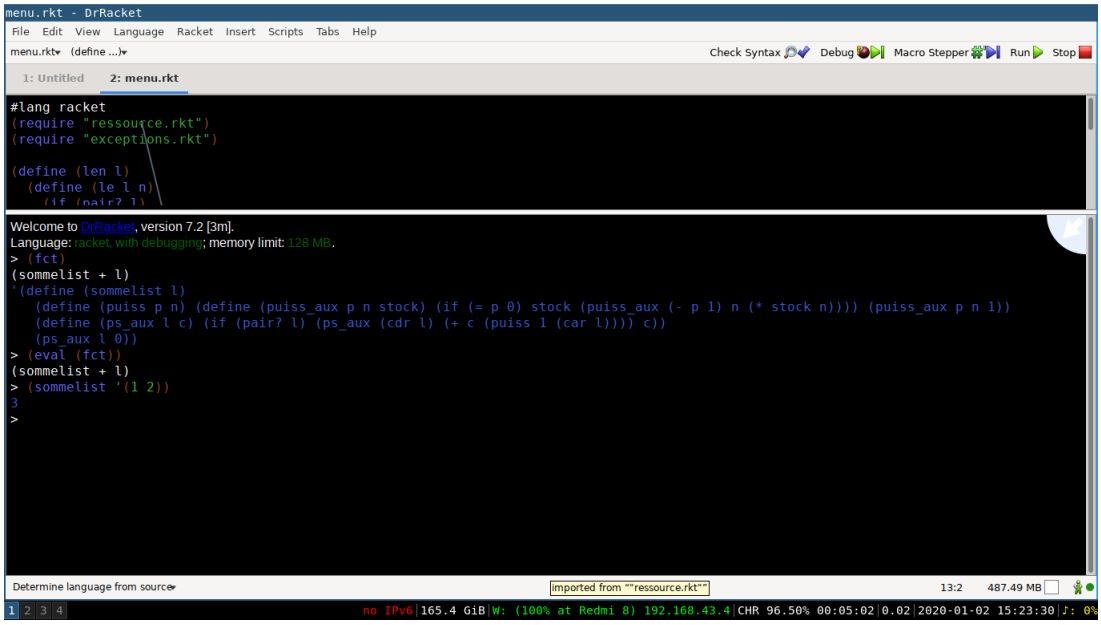
Part II
Commentaires

Le fichier menu.rkt est le corp du projet contenant une forêt de "if" permetant 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.

```
(define (fct_s nom oper para p) ;FCT somme/produit fcts num
  (list 'define (list nom para)
        (puiss)
        (list 'define (list `aux para `c)
              `(if (= ,para 0)
                   c
                   ,(list `aux `(- ,para 1) `(,oper ,(list `puiss p
    para) c))))
  (if (equal? oper '+)
        `(aux ,para 0)
  `(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 calcules mathématique, c'est à dire trouvé les similitudes entre les calcules fait par les fonctions.

Part III
Jeu d'essai



Part IV
Mode d'emploi

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

- fct
- fct_exception

Une fois la fonction executée, 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 créé une fonction utilisant des listes, la première variable sera obligatoirement un l ou l1.
Voici quelques prototypage fonctionnant avec notre programme :

1 Fonctions numériques (fct)

```
(somme + n)
(sommecarre + n 2)
(produit * n)
(produitcube * n 3)
(puissance    a b)
(soustraction - a b)
(division / a b)
```

2 Fonctions avec les listes (fct)

```
(doubleton l 2)
(nieme l n)
(somme + l)
(sommecarre + l 2)
(sommepositive + l >)
(produit * l)
(produitcube * l 3)
(plus_petit_element < l)
(plus_grand_element > l)
(plus_grand_que_n > l n)
(plus_petit que_n < l n)
```

3 Exceptions pour les fonctions numériques (fct_excep)

```
(pair n ?)
(reset / a b r)
(pgcd / a b)
(ppcm * a b)
(fib n)
```

4 Exceptions pour les fonctions avec des listes (fct_excep)

```
(autant l n) #la variable n ne peut pas etre change par un autre
    charactere
(medianboolean l n ?)
(element_median l)
(concat l l1)
(mirroir l m) #la variable m ne peut pas etre change par un autre
    charactere
```

Part V
Code

5 menu.rkt

```
#lang racket
(require "ressource.rkt")
(require "exceptions.rkt")

(define (len l)
  (define (le l n)
    (if (pair? l)
        (le (cdr l) (+ n 1))
        n))
  (le l 0))

(define (fct)
  (let ((l (read)))
    (let ((n (len l)))
      (if (or (< n 3) (> n 4))
          (write "RTFM!")
          (if (or (equal? (caddr l) 'l) (equal? (caddr l) '11) (
    equal? (cadr l) 'l))
              (if (pair? (cdddr l))
                  (if (and (equal? (cadr l) '+) (number? (cadddr l)
    ))
                      (make_ls (car l) (cadr l) (caddr l) (cadddr l
    ) 0)
                      (if (and (equal? (cadr l) '*) (number? (
    cadddr l)))
                          (make_ls (car l) (cadr l) (caddr l) (
    cadddr l) '=)
                          (if (and (equal? (cadr l) '+) (equal? (
    cadddr l) '>))
                              (make_ls (car l) (cadr l) (caddr l) 1
     (cadddr l))
                              (if (or (equal? (cadr l) '>) (equal?
    (cadr l) '<))
                                  (make_ptgr (car l) (cadr l) (
    caddr l) (cadddr l))
                                  (write "RTFM!")))))
                  (if (equal? (cadr l) '+)
                      (make_ls (car l) (cadr l) (caddr l) 1 0)
                      (if (equal? (cadr l) '*)
                          (make_ls (car l) (cadr l) (caddr l) 1 '=)
                          (if (or (equal? (cadr l) '<) (equal? (
    cadr l) '>))
                              (make_ptgr (car l) (cadr l) (caddr l)
     0)
                              (if (and (or (equal? (cadr l) 'l) (
    equal? (cadr l) 'l1)) (number? (caddr l)))
                                  (make_ton (car l) (cadr l) (caddr
     l))
                                  (make_nieme (car l) (cadr l) (
```

```
    caddr l)))))))
            (if (pair? (cdddr l))
                (if (number? (cdddr l))
                    (fct_s (car l) (cadr l) (caddr l) (cadddr l))
                    (if (or (equal? (cadr l) '+) (equal? (cadr l)
    '-))
                        (make_add_ss (car l) (cadr l) (caddr l) (
    cadddr l))
                        (if (or (equal? (cadr l) '*) (equal? (cadr
     l) '/))
                            (make_mult_div (car l) (cadr l) (caddr
     l) (cadddr l))
                            (if (equal? (cadr l) '^)
                                (puiss)
                                (display "RTFM!")))))
                (if (or (equal? (cadr l) '*) (equal? (cadr l) '+))
                    (fct_s (car l) (cadr l) (caddr l) 1)
                    (display "RTFM!")))))))

(define (fct_excep)
   (let ((l (read)))
    (let ((n (len l)))
      (if (= n 5)
          (make_reste (car l) (cadr l) (caddr l))
          (if (= n 4)
              (if (equal? (cadr l) '/)
                  (make_pgcd (car l) (cadr l) (caddr l))
                  (if (equal? (cadr l) '*)
                      (make_ppcm (car l) (cadr l) (caddr l))
                      (if (equal? (cadddr l) '?)
                          (make_mediantf (car l) (cadr l) (caddr l)
    )
                          (display "RTFM!"))))
              (if (= n 3)
                  (if (equal? (caddr l) '?)
                      (make_pair (car l) (cadr l))
                      (if (and (equal? (cadr l) 'l) (equal? (caddr
    l) 'n))
                          (make_autant (car l) (cadr l) (caddr l))
                          (if (equal? (cadr l) 'm)
                              (make_mirroir (car l) (cadr l))
                              (make_concat (car l) (cadr l) (caddr
    l)))))
                  (if (= n 2)
                      (if (equal? (cadr l) 'l)
                          (make_median (car l) (cadr l))
                          (make_fib (car l) (cadr l)))
                      (display "RTFM!")))))))))
```

6 ressource.rkt

```
#lang racket
(provide make_ls)
(provide make_ptgr)
(provide make_add_ss)
(provide fct_s)
(provide make_mult_div)
(provide make_ton)
(provide make_nieme)
(provide puiss)

(define (puiss)
  (list 'define '(puiss p n)
               (list 'define '(puiss_aux p n stock)
                     '(if (= p 0)
                          stock
                          (puiss_aux (- p 1) n (* stock n))))
               '(puiss_aux p n 1)))
(define (make_nbele)
  (list 'define '(nbele l)
        (list 'define '(nb_aux l n)
              '(if (pair? l)
                   (nb_aux (cdr l) (+ n 1))
                   n))
        '(nb_aux l 0)))

(define (fct_s nom oper para p) ;FCT somme/produit fcts num
  (list 'define (list nom para)
        (puiss)
        (list 'define (list 'aux para 'c)
              '(if (= ,para 0)
                   c
                   ,(list 'aux '(- ,para 1) '(,oper ,(list 'puiss p
    para) c))))
  (if (equal? oper '+)
        '(aux ,para 0)
  '(aux ,para 1))))

(define (make_ls nom oper param p sym) ;FCT somme produit listes
  (list 'define (list nom param)
        (puiss)
        (list 'define (list 'ps_aux param 'c)
              (list 'if (list 'pair? param)
                    (if (equal? sym '0)
                        '(ps_aux (cdr ,param) (,oper c (puiss ,p (
   car ,param))))
                        (make_sp oper param sym p))
                    'c))
        (fin oper param)))
(define (fin oper param) ;pour somme produit (fin de la fct) listes
  (if (equal? oper '*)
      (list 'ps_aux param 1)
      (list 'ps_aux param 0)))
(define (make_sp oper param sym p) ;pour produit / somme non nul
    listes
  (list 'if '(,sym 0 (car ,param))
```

```
        `(ps_aux (cdr ,param) c)
        `(ps_aux (cdr ,param) (,oper (puiss ,p (car ,param)) c))))

(define (make_p param1 oper) ;pour ppt normal etc
  (list 'if (list oper `c `(car ,param1))
        (list 'ptgr_aux `(cdr ,param1) `c)
        (list 'ptgr_aux `(cdr ,param1) `(car ,param1))))
(define (make_p1 param1 param2 oper) ;pour ppt que n etc
  (list 'if (list oper `(car ,param1) param2)
        (list 'ptgr_aux `(cdr ,param1) param2 `(+ c 1))
        (list 'ptgr_aux `(cdr ,param1) param2 `c)))
(define (par nom param1 param2) ;pour ppt (soit ppt soit ppt que n)
  (if (number? param2)
      `,(list nom param1)
      `,(list nom param1 param2)))
(define (par2 nom param1 param2) ;pour ppt (soit ppt soit ppt que n
    )
  (if (number? param2)
      `,(list `ptgr_aux param1 `c)
      `,(list `ptgr_aux param1 param2 `c)))
(define (make_ptgr nom oper param1 param2)  ;FCT ppt plgr
  (list 'define (par nom param1 param2)
        (list 'define (par2 nom param1 param2)
              (list 'if (list 'pair? param1)
                    (if (number? param2)
                        (make_p param1 oper)
                        (make_p1 param1 param2 oper))
                    `c))
        (if (number? param2)
            (list 'ptgr_aux param1 `(car ,param1))
            (list 'ptgr_aux param1 param2 0))))

(define (make_add_ss nom oper para1 para2) ;FCT addition/
    soustraction +1 -1
  (list 'define (list nom para1 para2)
        `(if (= ,para2 0) ,para1
             ,(list nom `(,oper ,para1 1) `(- ,para2 1)))))

(define (make_mult_div nom oper param1 param2 born sym) ;FCT
    multiplication / division + et - num
  (let ((inv (if (equal? oper '*)
                 1
                 param2))
        (inv2 (if (equal? oper '*)
                  param2
                  1)))
    (list 'define (list nom param1 param2)
    (list 'define (list `aux param1 param2 `c)
             `(if (,sym ,param1 ,born)
                  c
                  ,(list `aux `(- ,param1 ,inv) param2 `(+ c ,inv2)
    )))
          `(aux ,param1 ,param2 0))))

(define (make_ton nom param_list nb) ;FCT singl/double/triple TON
    listes
  (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

```
#lang racket
(require "ressource.rkt")
(provide make_reste)
(provide make_pgcd)
(provide make_ppcm)
(provide make_fib)
(provide make_pair)
(provide make_concat)
(provide make_mirroir)
(provide make_autant)
(provide make_mediantf)
(provide make_median)
(provide make_racine)


;make reste
(define (make_reste nom param1 param2)
  (list 'define (list nom param1 param2)
        `(if (< ,param1 0)
             (+ ,param1 ,param2)
             (,nom (- ,param1 ,param2) ,param2))))
;make pgcd
(define (make_pgcd nom param1 param2)
  (list 'define (list nom param1 param2)
        `(if (= ,param1 ,param2) ,param1
             (if (> ,param1 ,param2)
                 (,nom (- ,param1 ,param2) ,param2)
                 (,nom ,param1 (- ,param2 ,param1))))))
;make ppcm
(define (make_ppcm nom param1 param2)
  (list 'define (list nom param1 param2)
        (make_pgcd 'pgcd param1 param2)
        `(/ (* ,param1 ,param2) (pgcd ,param1 ,param2))))
;pair / impair
(define (make_pair nom param)
  (list 'define (list nom param)
        `(if (= ,param 0) #t
             (if (> ,param 0)
                 (,nom (- ,param 2))
                 #f))))
;racine !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
(define (make_racine nom param) (display ":'("))
;fib
(define (make_fib nom param)
  (list 'define (list nom param)
        (list 'define `(fib_aux ,param z x)
              `(if (= ,param 0)
                   (if (< z x) x z)
                   (if (< z x)
                       (fib_aux (- ,param 1) (+ z x) x)
                       (fib_aux (- ,param 1) z (+ x z)))))
        `(fib_aux ,param 1 0)))
;median #t #f
```

```
(define (make_mediantf nom param_list param)
  (list 'define (list nom param_list param)
        (make_ptgr 'ppt '< 'z 'x)
        (make_ptgr 'plgr '> 'z 'x)
        (make_autant 'autant 'l 'n)
        (list 'define `(medi_aux p g ,param_list ,param)
              `(if (pair? ,param_list)
                   (if (autant ,param_list ,param)
                       #t
                       (if (= p (+ g 1))
                           #t
                           (if (= (+ p 1) g)
                               #t
                               #f)))
                   #f))
        `(medi_aux (ppt ,param_list ,param) (plgr ,param_list ,param) ,param_list ,param)))
;element median
(define (make_median nom param_list)
  (list 'define (list nom param_list)
        (make_mediantf 'median? param_list 'n)
        (make_nieme 'nieme param_list 'n)
        (list 'define `(med_aux ,param_list n)
              `(if (pair? ,param_list)
                   (if (median? ,param_list n)
                       n
                       (med_aux ,param_list (nieme ,param_list (+ 1 n))))
                   n))
        `(med_aux ,param_list 1)))
;concat
(define (make_concat nom param1 param2)
  (list 'define (list nom param1 param2)
        (make_mirroir 'mirroir param1)
        (list 'define `(conc_aux ,param1 ,param2 l3)
              `(if (pair? ,param1)
                   (conc_aux (cdr ,param1) ,param2 (cons (car ,param1) l3))
                   (if (pair? ,param2)
                       (conc_aux ,param1 (cdr ,param2) (cons (car ,param2) l3))
                       l3)))
        `(mirroir (conc_aux ,param1 ,param2 '()))))
;mirroir
(define (make_mirroir nom param)
  (list 'define (list nom param)
        (list 'define `(mir_aux ,param i1)
              `(if (pair? ,param)
                   (mir_aux (cdr ,param) (cons (car ,param) i1))
                   i1))
        `(mir_aux ,param '())))
;autant
(define (make_autant nom param_list param)
  (list 'define (list nom param_list param)
        (make_ptgr 'ppt '< 'z 'x)
        (make_ptgr 'plgr '> 'z 'x)
        `(if (pair? ,param_list)
```

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