LOURD Rodolphe GI02
VIRGA Jeremy GI01

Calcul des décimales de PI en LISP

But

Ce projet consiste en le calcul du nombre PI avec le plus de décimales possibles. Pour ce faire, on va utiliser la formule de Machin :

$PI = 4 * (4 * Atan\ (1/5) - Atan\ (1/239))$

avec $Atan\ x = x - (x^3)/3 + (x^5)/5 - \ldots + (-1)^n * (x^{2n+1})/(2n+1)$

En LISP, ces opérations sont facilement réalisables sur des réels, mais la précision est limitée à 15 décimales. D'où l'intérêt de représenter les nombres sous forme de liste, par exemple ((3) 1415) pour 3,1415. Il faut par conséquent définir chaque opération élémentaire utilisée dans la formule de manière à ce qu'elle s'applique sur des listes d'entiers : la somme, la différence, la multiplication, la division. Ensuite des fonctions faisant appel à ces opérations élémentaires seront définies pour arriver au résultat final, comme la puissance.

Par commodité, nous utiliserons des listes inversées ; par exemple 3,1415 sera représenté par (5 1 4 1 3) au cours du calcul, étant donné que le dernier nombre de la liste représente toujours la seule unité non décimale.

Opérations élémentaires

Afin de simplifier et de rendre possibles de telles opérations sur des listes, nous avons imposé que les opérations seraient effectuées sur des listes de même taille, en considérant d'après la formule de Machin que les nombres à manipuler sous forme de listes seraient décimaux sauf pour le facteur multiplicateur 4.

Somme

Définition formelle

$\forall a, b, r \in List\mathbb{N} * List\mathbb{N} * \mathbb{N}$
si a = () et b = () alors somme(a,b,r) = FAUX
si a = () et b ≠ () alors si r ≠ 0 alors somme(a,b,r) = somme(ajout(r,()),b,0)
sinon somme(a,b,r) = b
si a ≠ () et b = () alors si r ≠ 0 alors somme(a,b,r) = somme(a,ajout(r,()),0)
sinon somme(a,b,r) = a
si a ≠ () et b ≠ () alors liste = ajout(((tete(a) + tete(b) + r) mod 10), somme(reste(a),reste(b),(tete(a) + tete(b) + r) div 10))

Définition en LISP

```
(defun somme (a b r)
        (cond
                ((and (eq a ()) (eq b ())) () )
                ((and (eq a ()) (not(eq b ()))) (cond   ((not(eq r 0)) (somme (cons r ()) b
                                                        0))
                                                        (t b)))
                ((and (not(eq a ())) (eq b ())) (cond   ((not(eq r 0)) (somme a (cons r ())
                                                        0))
                                                        (t a)))
                ((and (not(eq a ())) (not(eq b ()))) (setq liste (cons (% (+ (car a) (car b)
                r) 10) (somme (cdr a) (cdr b) (/ (+ (car a) (car b) r) 10)))))))
```

Afin de fonctionner correctement, cette opération doit avoir comme paramètres deux listes d'entiers positifs de taille égale, notamment pour la gestion des retenues.

Différence

Définition formelle

$\forall A, B, R \in List\mathbb{N} * List\mathbb{N} * \mathbb{N}$
si A = () alors difference(A,B,R) = FAUX
si B = () alors si R = 0 alors difference(A,B,R) = A
sinon difference(A,B,R) = difference(A,(1),0)
si tete(B) + R > tete(A) alors difference(A,B,R) = ajout((tete(A) + 10) – (tete(B) + R),difference(reste(A),reste(B),1))
sinon difference(A,B,R) = ajout(tete(A) – (tete(B) + R),difference(reste(A),reste(B),0))

Définition en LISP

```
(defun difference (A B R)
        (cond
                ((eq A ()) ())
                ((eq B ()) (cond ((eq R 0) A)
                                 (t (difference A '(1) 0))))
                ((> (+ (car B) R) (car A)) (cons (- (+ 10 (car A)) (+ (car B) R))
                (difference (cdr A) (cdr B) 1)))
                (t (cons (- (car A) (+ (car B) R)) (difference (cdr A) (cdr B) 0 )))))
```

Tout comme la somme, la différence nécessite deux listes de même longueur pour fonctionner correctement.

Multiplication

Cette opération est composée de deux sous opérations ; multiplatom qui réalise la multiplication de deux atomes, et multiplication qui utilise multiplatom pour décomposer le traitement de listes en atomes.

Définition formelle

$\forall A, b, r \in List\mathrm{N} * \mathrm{N} * \mathrm{N}$

*si A ≠ () alors multiplatom(A,b,r) = ajout((tete(A) * b + r) mod 10, multiplatom(reste(A),b,(tete(A) * b + r) div 10)*

$\forall A, B \in List\mathrm{N} * List\mathrm{N} * \mathrm{N}$

si B ≠ () alors multiplication(A,B) = somme(multiplatom(A,tete(B),0),ajout (0,multiplication(A,reste(B))),0)

Définition en LISP

```
(defun multiplatom (A b r)
        (cond ((not(eq A ())) (cons (% (+ (* (car A) b) r) 10) (multiplatom (cdr A) b (/
        (+ (* (car A) b) r) 10))))))

(defun multiplication (A B) (if (not(null B)) (somme (multiplatom A (car B) 0) (cons 0
(multiplication A (cdr B))) 0)))
```

La multiplication nécessite également deux listes de même longueur pour fonctionner.

Division

Cette opération fait appel à plusieurs fonctions : multiplication, extraire et inferieur. Inferieur et extraire seront définies par la suite.

Définition formelle

$\forall a,b,q,n \in List\mathrm{N} * List\mathrm{N} * \mathrm{N} * \mathrm{N}$

si a = () et b = () alors divise(a,b,q,n) = FAUX
si a = () et b ≠ () alors divise(a,b,q,n) = FAUX
si a ≠ () et b = () alors divise(a,b,q,n) = FAUX
si a ≠ () et b ≠ () alors
multi = multiplication(q,b)
si longueur(q) = n + 2 alors divise(a,b,q,n) = reste(q)
si inverse(multi) = inverse(extraire(a,longueur(multi) – 1)) alors divise(a,b,q,n)= q
si inverse(multi) < inverse(extraire(a,longueur(multi) – 1)) alors
q = ajout(tete(q) + 1,reste(q))
si tete(q) = 10 alors q = ajout((),ajout(tete(reste(q)) + 1,reste(reste(q))))
divise(a,b,q,n)
sinon q = ajout(tete(q) – 1,reste(q))
si tete(q) = -1 alors q = ajout(9,ajout(tete(reste(q)) – 1,reste(reste(q))))
q = ajout((),q))
divise(a,b,q,n)

Définition en LISP

```
(defun divise(a b q n)
  (cond
        ((and (eq a ()) (eq b ())) ())
        ((and (eq a ()) (not (eq b ()))) ())
        ((and (not (eq a ())) (eq b ())) ())
        ((and (not (eq a ())) (not (eq b ()))) (setq multi (multiplication q b))
              (cond
              ((eq (length q) (+ n 2)) (cdr q))
              ((egal (reverse multi) (reverse (extraire a (- (length multi) 1)))) q)
              ((inferieur (reverse multi) (reverse (extraire a (- (length multi) 1))))
                    (setq q (cons (+ (car q) 1) (cdr q)))
                    (if (eq (car q) 10) (setq q (cons 0 (cons (+ (car (cdr q)) 1) (cdr
                    (cdr q))))) (setq q q))
                    (divise a b q n))
              (t    (setq q (cons (- (car q) 1) (cdr q)))
                    (if (eq (car q) -1) (setq q (cons 9 (cons (- (car (cdr q)) 1) (cdr
                    (cdr q))))) (setq q q))
                    (setq q (cons 0 q))
                    (divise a b q n))))))
```

Cette opération est la plus complexe de notre calcul ; il nous a fallu l'optimiser pour réduire les temps de calcul trop longs. On trouve les mêmes limitations que pour les autres opérations ; la taille des listes doit être la même ; ici n sert à définir le nombre de décimales souhaitées.

Inferieur

Cette opération est nécessaire pour la division notamment car il faut que l'on puisse comparer des listes pour effectuer des divisions entières successives pour trouver le quotient puis les décimales du nombre calculé.

Définition formelle

$\forall a, b \in List\mathbb{N} * List\mathbb{N}$
si a = () et b = () alors inferieur(a,b) = VRAI
si a = () et b ≠ () alors inferieur(a,b) = FAUX
si a ≠ () et b = () alors inferieur(a,b) = VRAI
si a ≠ () et b ≠ () alors
si tete(a) = tete(b) alors inferieur(a,b) = inferieur(reste(a),reste(b))
si tete(a) < tete(b) alors inferieur(a,b) = VRAI
sinon inferieur(a,b) = FAUX

Définition en LISP

```
(defun inferieur (a b)
    (cond
        ((and (eq a ()) (eq b ())) t)
        ((and (eq a ()) (not(eq b ()))) ())
        ((and (not(eq a ())) (eq b ())) t)
        ((and (not(eq a ())) (not(eq b ())))
            (cond ((= (car a) (car b)) (inferieur (cdr a) (cdr b)))
                ((< (car a) (car b)) t)
                (t ())))))
```

Lorsque inferieur(a,b) renvoie la valeur VRAI, cela signifie que a est inférieur à b.

Egal

Tout comme la fonction inferieur, la fonction egal est utilisée pour comparer deux listes de nombres ; elle renvoie VRAI lorsqu'elles sont égales.

Définition formelle

$\forall a, b \in List\mathbb{N} * List\mathbb{N}$
res = VRAI
si a = () et b = () alors egal(a,b) = FAUX
si tete(a) = tete(b) et reste(a) ≠ () et reste(b) ≠ () alors
egal(a,b) = egal (reste(a),reste(b))
si tete(a) = tete(b) et reste(a) = () et reste(b) ≠ () alors
tant que b ≠ () faire
si tete(b) ≠ 0 alors res = FAUX
b = reste(b)
fait
si tete(a) = tete(b) et reste(a) = () et reste(b) = () alors egal(a,b) = VRAI
si tete(a) ≠ tete(b) alors egal(a,b) = FAUX

Définition en LISP

```
(defun egal(a b)
    (setq res t)
    (cond
        ((and (eq a ()) (eq b ())) ())
        ((and (eq (car a) (car b)) (not(eq (cdr a) ())) (not(eq (cdr b) ())))) (egal (cdr a)
        (cdr b)))
        ((and (eq (car a) (car b)) (eq (cdr a) ()) (not (eq (cdr b) ())))
                (while (not (eq b ()))
                        (if (not (eq (car b) 0)) (setq res ())
                        (setq res res))
                        (setq b (cdr b))))
         ((and (eq (car a) (car b)) (not (eq (cdr a) ())) (eq (cdr b) ()))
                (while (not (eq a ()))
                        (if (not (eq (car a) 0)) (setq res ())
                        (setq res res))
                        (setq a (cdr a))))
        ((and (eq (car a) (car b)) (eq (cdr a) ()) (eq (cdr b) ())) t)
        ((not (eq (car a) (car b))) ())))
```

Extraire

Cette fonction sert à limiter ou agrandir le nombre d'éléments d'une liste, pour dans notre cas rendre les listes de nombres de même taille avant chaque opération élémentaire.

Définition formelle

$\forall a, n \in List\mathrm{N} * \mathrm{N}$

tant que longueur(a) > n + 1 faire
  a = reste(a)
fait
tant que longueur(a) < n + 1 faire
  a = ajout((),a)
fait
extraire(a,n)=a

Définition en LISP

```
(defun extraire (a n)
  (while (> (length a) (+ n 1))
        (setq a (cdr a)))
  (while (< (length a) (+ n 1))
        (setq a (cons 0 a)))
  a)
```

Ici a est la liste à réduire/agrandir ; n est le nombre de décimales voulues.

Puissance

Cette fonction utilise des multiplications successives pour calculer la puissance nième d'un nombre, en l'occurrence une liste.

Définition formelle

$\forall a, n \in List\mathrm{N} * \mathrm{N}$
si n = 0 et a ≠ () alors puissance(a,n) = 1
si a = () alors puissance(a,n) = FAUX
si n ≠ 0 et a ≠ () alors
b = 1
tant que n > 0 faire
b = multiplication(b,a)
n = n – 1
fait
puissance(a,n) = b

Définition en LISP

```
(defun puissance (a n)
  (cond
        ((and (eq n 0) (not(eq a ()))) '(1))
        ((eq a ()) ())
        ((and (not(eq n 0)) (not(eq a ())))
              (setq b '(1))
              (while (> n 0)
                    (setq b (multiplication b a))
                    (setq n (- n 1)))
              b)))
```

Opérations finales

Ces opérations se servent des fonctions précédemment définies afin de progresser jusqu'au résultat final ; la valeur de pi avec un nombre de décimales entré par l'utilisateur.

Atan

Cette fonction calcule atan(x) à l'aide de la formule de Taylor ; l'ordre est donné par i ; le nombre de décimales par n.

Définition formelle

$\forall x, n \in List\mathrm{N} * \mathrm{N}$

```
result = 0
i = 0
tant que i < 15 faire
  si tete(puissance((-1),i)) > 0 alors
        aux = puissance(x,2* i + 1)
        aux2 = divise(aux,extraire((2* i + 1),longueur(aux) - 1),(1),n)
        si longueur(result) > longueur(aux2) alors
              result = somme(result,extraire(aux2,longueur(result) – 1),0)
        sinon   result = somme(extraire(result,longueur(aux2) – 1),aux2,0)
  sinon   aux = puissance(x,2* i + 1)
        aux2 = divise(aux,extraire((2* i + 1),longueur(aux) – 1),(1),n)
        si longueur(result) > longueur(aux2) alors
              result = difference(result,extraire(aux2,longueur(result) – 1),0)
        sinon   result = difference(extraire(result,longueur(aux2) – 1),aux2,0)
  i = i + 1
fait
atan(x,n) = result
```

Définition en LISP

```
(defun atan(x n)
    (setq result '(0))
    (setq i 0)
    (while (< i 15)
        (cond
        ((> (car (puissance '(-1) i)) 0)
              (setq aux (puissance x (+ (* 2 i) 1)))
              (setq aux2 (divise aux (extraire (list (+ (* 2 i) 1)) (- (length aux) 1)) '(1)
n))
              (cond
              ((> (length result) (length aux2))
                    (setq result (somme result (extraire aux2 (- (length result) 1))
0)))
              (t (setq result (somme (extraire result (- (length aux2) 1)) aux2 0)))))
        (t
              (setq aux (puissance x (+ (* 2 i) 1)))
```

```
                (setq aux2 (divise aux (extraire (list (+ (* 2 i) 1)) (- (length aux) 1)) '(1)
n))
                (cond
                ((> (length result) (length aux2))
                        (setq result (difference result (extraire aux2 (- (length result) 1))
0)))
                (t (setq result (difference (extraire result (- (length aux2) 1)) aux2
0))))))
        (setq i (+ i 1))) result)
```

Afin d'avoir une bon résultat sur les décimales de pi par la suite, il est nécessaire de conduire le calcul de l'arctan suffisamment loin ; après divers essais, nous avons une bonne précision pour n décimales avec un calcul d'arctan à l'ordre n + 5.

1/239

La fonction divise ne prenant en compte que les décimaux inférieurs à 1, il faut définir une fonction simple qui fera le calcul de 1/239 à la précision n que l'on souhaite.

Définition formelle

$\forall n \in {}^*\mathbb{N}$
boucle = (0 0 4 1 8 4 1)
list = (0)
tant que longueur(list) < n + 1 faire
  list = ajout(tete(boucle),list)
  si reste(boucle) = () alors boucle = (0 0 4 1 8 4 1)
  sinon boucle = reste(boucle)
fait
1/239(n) = list

Définition en LISP

```
(defun 1/239(n)
  (setq boucle '(0 0 4 1 8 4 1))
  (setq list '(0))
  (while (< (length list) (+ n 1))
        (setq list (cons (car boucle) list))
        (if (eq (cdr boucle) ()) (setq boucle '(0 0 4 1 8 4 1)) (setq boucle (cdr boucle))))
  list)
```

Pour réaliser cette fonction, nous nous sommes aperçus que les décimales de 1/239 suivaient une période 0041841 indéfiniment ; il était donc facile de reproduire cette période autant de fois que nous voulions de décimales. Pour 1/5, nous n'avons pas besoin de fonction vu que le résultat est fini et vaut (2 0).

Machin

Cette fonction est une composée des précédentes ; elle effectue le calcul final de pi avec une précision de n.

Définition formelle

$\forall n \in$ *N

p = inverse(extraire(multiplication(difference(multiplication(atan((2
0),n+5),(4)),atan(1/239(n),n + 5),0),(4)),n)
ajout(ajout(tete p,()),reste(p))

Définition en LISP

```
(defun machin(n)
        (setq p (reverse (extraire (multiplication (difference (multiplication (atan '(2 0)
        (+ n 5)) '(4)) (atan (1/239 n) (+ n 5)) 0) '(4)) n)))
        (cons (cons (car p) ()) (cdr p)))
```

Le résultat est finalement renvoyé sous la forme ((3) 1 4 1 5).

Conclusion

Dans ce mini projet le plus difficile fut tout d'abord d'apprendre à coder du LISP du fait de sa syntaxe pour le moins « originale » ; beaucoup d'erreurs venaient de parenthèses mal placées ou oubliées. De plus il fallait gérer tous les cas dans chaque fonction ; dès qu'un cas n'était pas traité, il fallait le rajouter. Enfin, malgré quelques optimisations du code sur la fin, le calcul reste relativement lent ; le calcul à plus de 15 ou 20 décimales est possible, mais il faut un certain temps pour arriver au résultat qui nécessite un agrandissement de la pile !