

T.P. 7 – Corrigé

Calculatrice (partie 4)

Étape 1

```

Itoa      ; Sauvegarde les registres.
          movem.l d0/a0,-(a7)

          ; Si D0.W est positif ou nul, saute à \positive.
          tst.w   d0
          bpl     \positive

\negative ; Sinon écrit le '-' dans la chaîne
          ; (et fait pointer A0.L sur le caractère suivant).
          move.b  #'-',(a0)+

          ; Détermine l'opposé de D0.W.
          neg.w   d0

\positive ; Lance la conversion.
          jsr     Uitoa

\quit     ; Restaure les registres puis sortie.
          movem.l (a7)+,d0/a0
          rts

```

Étape 2

```

          ; =====
          ; Initialisation des vecteurs
          ; =====

          org      $0

vector_000 dc.l     $ffb500
vector_001 dc.l     Main

          ; =====
          ; Programme principal
          ; =====

          org      $500

Main      ; Affichage du message de saisie de l'expression.
          ; (L'affichage s'effectue en haut à gauche de la fenêtre.)
          movea.l  #sInput,a0
          clr.b    d1
          clr.b    d2
          jsr      Print

          ; Saisie de l'expression par l'utilisateur.
          ; (La chaîne saisie est placée à l'adresse sBuffer.)
          ; (Elle s'affichera deux lignes sous le premier message.)
          movea.l  #sBuffer,a0
          addq.b   #2,d2
          move.l   #60000,d3
          move.l   #8000,d4

```

```

jsr      GetInput

; Suppression des espaces de la chaîne.
jsr      RemoveSpace

; Affichage du message de résultat (avec saut de deux lignes).
movea.l  #sResult,a0
addq.b   #2,d2
jsr      Print

; Saut de deux lignes pour l'affichage du résultat.
addq.b   #2,d2

; Calcul du résultat (dans D0.L)
; et saut à \error si une erreur est détectée.
movea.l  #sBuffer,a0
jsr      GetExpr
bne      \error

\noError  ; Aucune erreur n'a été détectée.
          ; Conversion du résultat en chaîne ASCII.
          ; (La conversion se fait dans le buffer car A0 = sBuffer.)
jsr      Itoa

          ; Affichage du résultat et sortie.
jsr      Print
bra      \quit

\error    ; Une erreur a été détectée.
          ; Affichage du message d'erreur.
movea.l  #sError,a0
jsr      Print

\quit     ; Point d'arrêt du programme principal.
illegal

; =====
; Sous-programmes
; =====

; ...
; ...
; (Tous les sous-programmes réalisés.)
; ...
; ...

GetInput  incbin  "GetInput.bin"
PrintChar incbin  "PrintChar.bin"

; =====
; Données
; =====

sInput    dc.b    "Veuillez saisir une expression :",0
sResult   dc.b    "Resultat :",0
sError    dc.b    "Erreur",0
sBuffer   ds.b    60

```

Étape 3

Découpez **NextOp** en deux sous-programmes : **NextOp1** et **NextOp2**.

```

NextOp1      ; Si le caractère est nul (fin de chaîne),
              ; il n'y a pas d'opérateur dans la chaîne.
              ; A0 pointe sur le caractère nul. On quitte.
tst.b    (a0)
beq      \quit

              ; Compare le caractère aux opérateurs '+' et '-'.
              ; Si le caractère est un opérateur, on peut quitter.
              ; (A0 contient l'adresse de l'opérateur.)
cmpi.b   #'+',(a0)
beq      \quit

cmpi.b   #'-',(a0)
beq      \quit

              ; Passage au caractère suivant.
addq.l   #1,a0
bra      NextOp1

\quit        ; Sortie.
rts

```

```

NextOp2      ; Si le caractère est nul (fin de chaîne),
              ; il n'y a pas d'opérateur dans la chaîne.
              ; A0 pointe sur le caractère nul. On quitte.
tst.b    (a0)
beq      \quit

              ; Compare le caractère aux opérateurs '*' et '/'.
              ; Si le caractère est un opérateur, on peut quitter.
              ; (A0 contient l'adresse de l'opérateur.)
cmpi.b   #'*',(a0)
beq      \quit

cmpi.b   #'/',(a0)
beq      \quit

              ; Passage au caractère suivant.
addq.l   #1,a0
bra      NextOp2

\quit        ; Sortie.
rts

```

Découpez **GetNum** en deux sous-programmes : **GetNum1** et **GetNum2**.

```

GetNum1      ; Sauvegarde les registres.
              movem.l d1/a1-a2,-(a7)

              ; Mémoire le début de la chaîne dans A1.
              movea.l a0,a1

              ; Cherche le prochain opérateur ('*' ou '/') ou le caractère nul
              ; (c'est-à-dire le caractère qui suit le nombre),
              ; et mémorise sa position dans A2.
              jsr      NextOp2
              movea.l a0,a2

              ; Sauvegarde l'opérateur ou le caractère nul dans D1.
              move.b  (a2),d1

              ; Place un caractère nul juste après le nombre.
              clr.b   (a2)

              ; Lance la conversion
              ; (avec l'adresse de départ comme paramètre dans A0).
              movea.l a1,a0
              jsr      Convert

              ; Si la conversion est valide,
              ; D0 contient la valeur numérique du nombre ASCII.
              ; On quitte sans erreur.
              beq      \true

\false       ; Sortie avec erreur.
              ; D0 n'a pas été modifié.
              ; A0 contient l'adresse de départ de la chaîne.
              ; Il ne reste plus qu'à restaurer le caractère sauvegardé dans D1.
              move.b  d1,(a2)

              ; Et renvoyer Z = 0.
              andi.b  #%11111011,ccr
              bra      \quit

\true        ; Sortie sans erreur.
              ; On commence par restaurer le caractère sauvegardé dans D1.
              move.b  d1,(a2)

              ; On place l'adresse située après le nombre dans A0.
              movea.l a2,a0

              ; Et enfin, on renvoie Z = 1.
              ori.b   #%00000100,ccr

\quit        ; Restaure les registres.
              movem.l (a7)+,d1/a1-a2
              rts

```

```

GetNum2      ; Sauvegarde les registres.
             movem.l d1/a1-a2,-(a7)

             ; Mémoire le début de la chaîne dans A1.
             movea.l a0,a1

             ; Cherche le prochain opérateur ('+' ou '-') ou le caractère nul
             ; (c'est-à-dire le caractère qui suit le terme),
             ; et mémorise sa position dans A2.
             jsr      NextOp1
             movea.l a0,a2

             ; Sauvegarde l'opérateur ou le caractère nul dans D1.
             move.b  (a2),d1

             ; Place un caractère nul juste après le nombre.
             clr.b   (a2)

             ; Lance la conversion d'un terme
             ; (avec l'adresse de départ comme paramètre dans A0).
             movea.l a1,a0
             jsr      GetExpr1

             ; Si la conversion est valide,
             ; D0 contient la valeur numérique du nombre ASCII.
             ; On quitte sans erreur.
             beq      \true

\false       ; Sortie avec erreur.
             ; D0 n'a pas été modifié.
             ; A0 contient l'adresse de départ de la chaîne.
             ; Il ne reste plus qu'à restaurer le caractère sauvegardé dans D1.
             move.b  d1,(a2)

             ; Et renvoyer Z = 0.
             andi.b  #%11111011,ccr
             bra      \quit

\true        ; Sortie sans erreur.
             ; On commence par restaurer le caractère sauvegardé dans D1.
             move.b  d1,(a2)

             ; On place l'adresse située après le nombre dans A0.
             movea.l a2,a0

             ; Et enfin, on renvoie Z = 1.
             ori.b   #%00000100,ccr

\quit        ; Restaure les registres.
             movem.l (a7)+,d1/a1-a2
             rts

```

Découpez **GetExpr** en deux sous-programmes : **GetExpr1** et **GetExpr2**.

```

GetExpr1      ; Sauvegarde les registres.
              movem.l d1-d2/a0,-(a7)

              ; Conversion du premier nombre de l'expression (dans D0).
              ; Si erreur, on renvoie false.
              jsr      GetNum1
              bne      \false

              ; Le premier nombre est chargé dans D1.
              ; (D1 contiendra le résultat des opérations successives.)
              move.l   d0,d1

\loop         ; L'opérateur ou le caractère nul est copié dans D2.
              ; S'il s'agit du caractère nul, on renvoie true (pas d'erreur).
              move.b   (a0)+,d2
              beq       \true

              ; Conversion du prochain nombre (dans D0).
              ; Si erreur, on renvoie false.
              jsr      GetNum1
              bne      \false

              ; Détermine le type de l'opération (*, /).
              cmp.b    #'/',d2
              beq       \divide

\multiply     ; Effectue l'opération puis passe au nombre suivant.
              muls.w   d0,d1
              bra       \loop

\divide       ; Renvoie une erreur si une division par zéro est détectée.
              tst.w    d0
              beq       \false

              ; Le résultat entier de la division est sur 16 bits. Il faut
              ; réaliser une extension de signe pour l'avoir sur 32 bits.
              divs.w   d0,d1
              ext.l     d1
              bra       \loop

\false        ; Sortie avec erreur (Z = 0).
              andi.b   #%11111011,ccr
              bra       \quit

\true         ; Sortie sans erreur (Z = 1).
              ; (Avec la copie du résultat dans D0.)
              move.l   d1,d0
              ori.b     #%00000100,ccr

\quit         ; Restaure les registres puis sortie.
              movem.l (a7)+,d1-d2/a0
              rts

```

```

GetExpr2      ; Sauvegarde les registres.
               movem.l d1-d2/a0,-(a7)

               ; Conversion du premier terme de l'expression (dans D0).
               ; Si erreur, on renvoie false.
               jsr      GetNum2
               bne      \false

               ; Le premier terme est chargé dans D1.
               ; (D1 contiendra le résultat des opérations successives.)
               move.l   d0,d1

\loop          ; L'opérateur ou le caractère nul est copié dans D2.
               ; S'il s'agit du caractère nul, on renvoie true (pas d'erreur).
               move.b   (a0)+,d2
               beq       \true

               ; Conversion du prochain terme (dans D0).
               ; Si erreur, on renvoie false.
               jsr      GetNum2
               bne      \false

               ; Détermine le type de l'opération (+, -).
               cmp.b     #'-',d2
               beq        \subtract

\add           ; Effectue l'opération puis passe au nombre suivant.
               add.l     d0,d1
               bra        \loop

\subtract      sub.l     d0,d1
               bra        \loop

\false         ; Sortie avec erreur (Z = 0).
               andi.b    #%11111011,ccr
               bra        \quit

\true          ; Sortie sans erreur (Z = 1).
               ; (Avec la copie du résultat dans D0.)
               move.l    d1,d0
               ori.b     #%00000100,ccr

\quit          ; Restaure les registres puis sortie.
               movem.l  (a7)+,d1-d2/a0
               rts

```

Pour finir, modifiez le programme principal.

```

Main      ; Affichage du message de saisie de l'expression.
          ; (L'affichage s'effectue en haut à gauche de la fenêtre.)
          movea.l #sInput,a0
          clr.b   d1
          clr.b   d2
          jsr     Print

          ; Saisie de l'expression par l'utilisateur.
          ; (La chaîne saisie est placée à l'adresse sBuffer.)
          ; (Elle s'affichera deux lignes sous le premier message.)
          movea.l #sBuffer,a0
          addq.b  #2,d2
          move.l  #60000,d3
          move.l  #8000,d4
          jsr     GetInput

          ; Suppression des espaces de la chaîne.
          jsr     RemoveSpace

          ; Affichage du message de résultat (avec saut de deux lignes).
          movea.l #sResult,a0
          addq.b  #2,d2
          jsr     Print

          ; Saut de deux lignes pour l'affichage du résultat.
          addq.b  #2,d2

          ; Calcul du résultat (dans D0.L)
          ; et saut à \error si une erreur est détectée.
          movea.l #sBuffer,a0
          jsr     GetExpr2
          bne     \error

\noError  ; Aucune erreur n'a été détectée.
          ; Conversion du résultat en chaîne ASCII.
          ; (La conversion se fait dans le buffer car A0 = sBuffer.)
          jsr     Itoa

          ; Affichage du résultat et sortie.
          jsr     Print
          bra     \quit

\error    ; Une erreur a été détectée.
          ; Affichage du message d'erreur.
          movea.l #sError,a0
          jsr     Print

\quit     ; Point d'arrêt du programme principal.
          illegal

```