1.) La programmation impérative.

1.1.) Les références.

Les références sont des structures de données prédéfinies qui modélisent les cases mémoires de la machine. La propriété caractéristique des références est qu'on peut les lire et les écrire.

Les références sont créées par la construction *ref(val)* où *val* est la valeur initiale :

```
let identifiant = ref expression;
```

Le contenu d'une référence est renvoyé par l'opérateur de déréférencement "*!*" :

```
!identifiant;
```

On change le contenu d'une référence avec l'opérateur d'affectation "*:=*".

```
identifiant := valeur;
```

Les références se comportent comme les variables dans les langages impératifs tels que Pascal. La seule différence est qu'en Caml, il faut explicitement déréférencer l'identificateur à l'aide de l'opérateur "*!*".

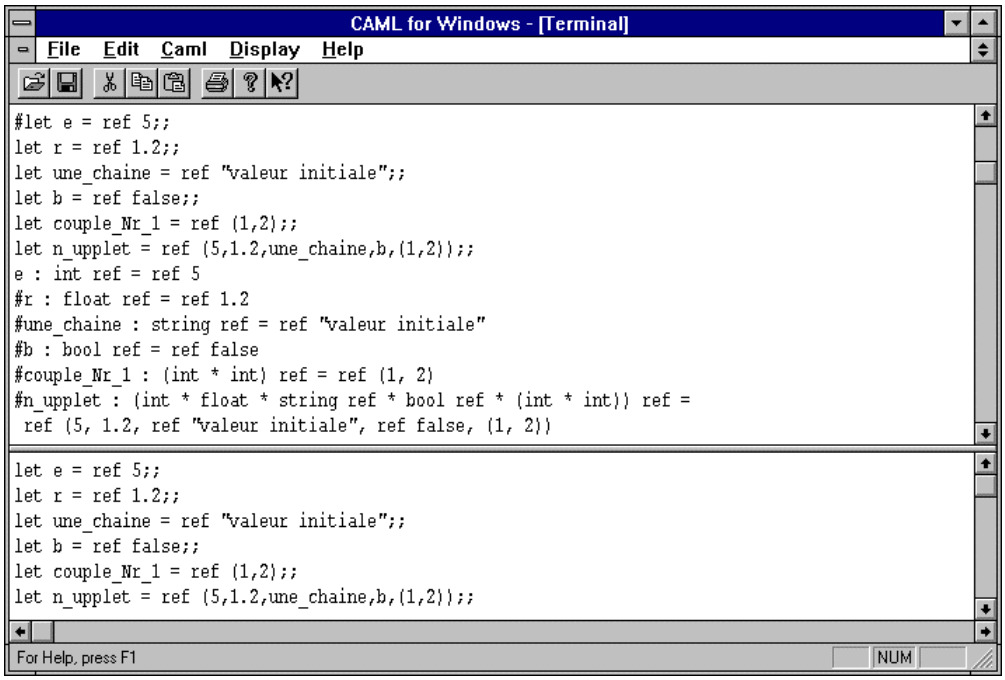


Figure 1 Références.

La construction let (définition) est différente de l'affectation :

`#let x = 1;;` `x : int 1`	`#let x = ref 1;;` `x : int ref = ref 1`
`#let f y = x + y;;` `f : int -> int = <fun>`	`#let f y = !x + y;;` `f : int -> int = <fun>`
`#let x = 2;;` `x : int 2`	`#x:=2` `-: unit()`
`# f 0;;` `-: int = 1`	`#f 0;;` `-: int = 2`

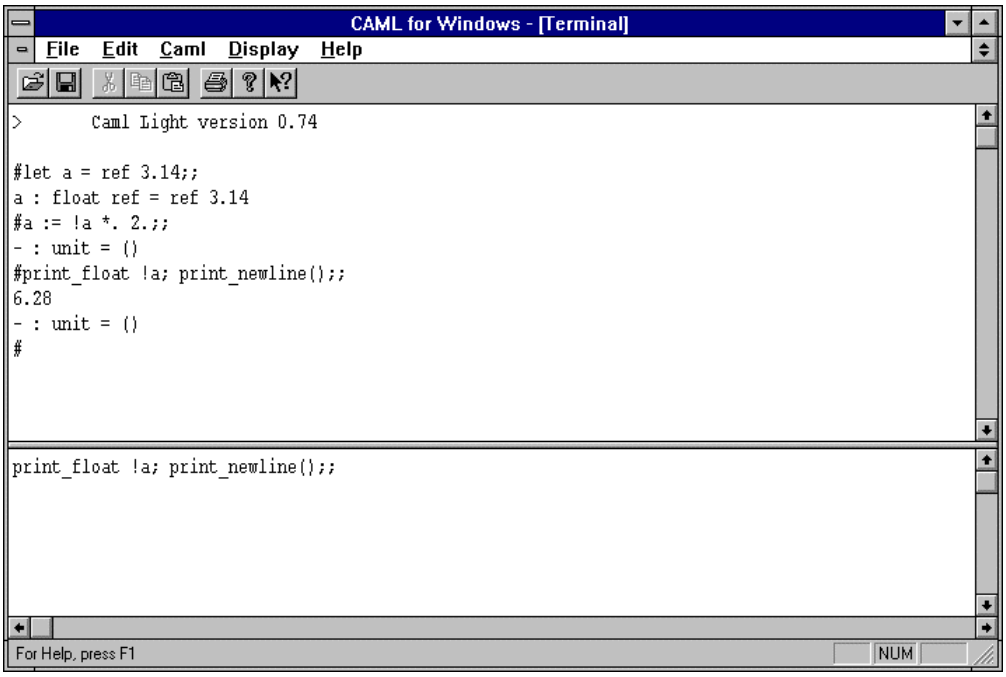


Figure 2 Utilisation des références.

1.2.) Les vecteurs.

Les vecteurs (aussi appelés tableaux) sont des suites finies et modifiables de valeurs d'un même type. Puisque les éléments du tableau sont tous de même nature, on qualifie les tableaux de suites homogènes de valeurs.

`#let v = [|5;4;1;12;2|];;` `v : int vect = [|5; 4; 1; 12; 2|]`	Définition d'un vecteur.
`#v.(3);;` `- : int = 12`	Accès à un élément du vecteur.
`#v.(0)<-3;;` `- : unit = ()`	Modification d'un élément du vecteur.

Soit en EBNF :

```
tableau ::= [|expression (;expression)*|]
Accès dans un tableau ::= tableau.(indice)
Modification d'un élément ::= tableau.(indice) <- expression
```

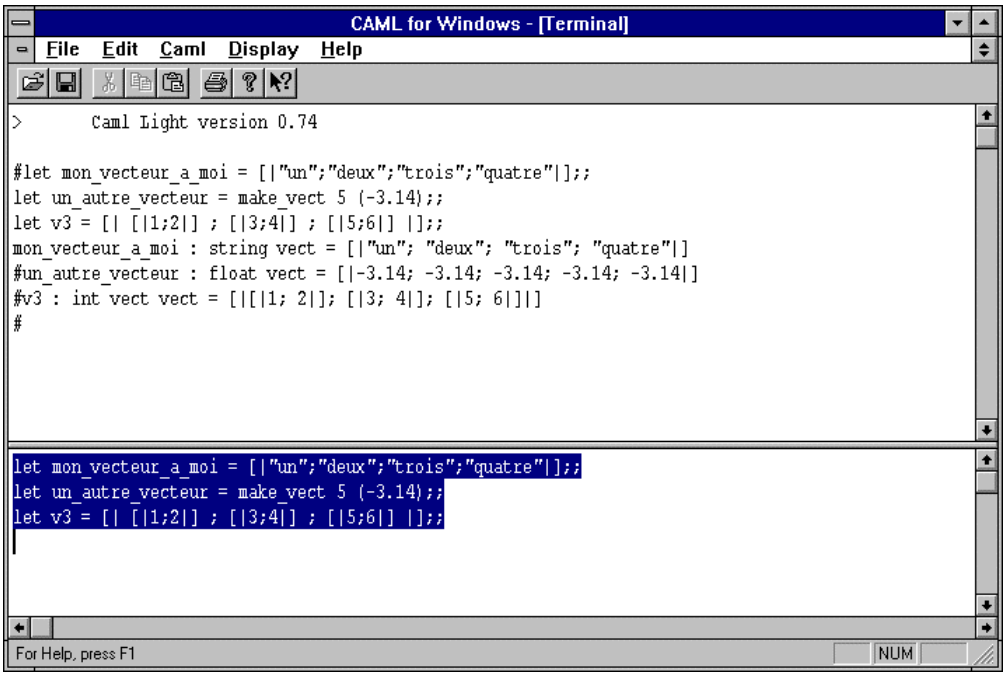


Figure 3 Définitions de vecteurs.

`vect_length : 'a vect -> int`	Renvoie le nombre d'éléments d'un vecteur.
`value vect_item : 'a vect -> int -> 'a`	"vect_item v n" renvoie l'élément de rang n du vecteur v. Equivalent à `v.(n)`
`value vect_assign : 'a vect -> int -> 'a -> unit`	"vect_assign v n x" modifie le vecteur v en remplaçant l'élémént de rang n par x. Equivalent à `v.(n)<-x`.
`value make_vect : int -> 'a -> 'a vect`	"make_vect n x" renvoie un vecteur de longueur n initialisé avec x.
`value make_matrix : int -> int -> 'a -> 'a vect vect`	"make_matrix dimx dimy e" renvoie un tableau de 2 dimensions (dimx et dimy) initialisé avec e.
`value concat_vect : 'a vect -> 'a vect -> 'a vect`	"concat_vect v1 v2" renvoie un nouveau vecteur égal à la concaténation de v1 et v2.
`value sub_vect : 'a vect -> int -> int -> 'a vect`	"sub_vect v start len" renvoie un nouveau vecteur contenant les éléments du vecteur v de rang compris entre start et start+len-1
`value copy_vect : 'a vect -> 'a vect`	"copy_vect v" renvoie une copie du vecteur v.
`value fill_vect : 'a vect -> int -> int -> 'a -> unit`	"fill_vect v ofs len x" modifie les éléments du vecteur v de rang compris entre ofs et ofs+len-1 en leur donnant la valeur x.

1.3.) Les chaînes de caractères.

Les chaînes se comportent essentiellement comme des tableaux de caractères. Le premier élément d'une chaîne a la position 0, le dernier a la position (longueur de la chaine - 1).

Pour accéder à un élément dans une chaîne on utilisera :

```
expr  ::=  . expr .[ expr ]
```

Ceci est équivalent à `nth_char expr expr`. Par exemple, pour accéder au cinquième caractère de la chaîne "*il était un petit navire*" on pourra écrire :

```
#let s = "il etait un petit navire";;
s : string = "il etait un petit navire"
#s.[5];;
- : char = `a`
#nth_char s 5;;
- : char = `a`
```

Si on tente d'adresser à un rang qui n'existe pas, le système renvoie l'exception "Invalid_argument"

```
#nth_char s 50;;
Exception non rattrapée: Invalid_argument "nth_char"
```

Pour modifier un caractère dans une chaîne :

```
expr .[ expr ] <- expr
```

ce qui est équivalent à "`set_nth_char expr1 expr2 expr3`".

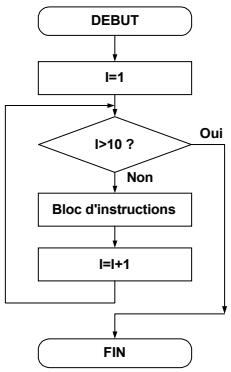
```
#s.[5]<-`o`;;
- : unit = ()
#set_nth_char s 5 `o`;;
- : unit = ()
```

`value string_length : string -> int`	Renvoie le nombre de caractères de la chaîne.
`value prefix ^ : string -> string -> string`	s1^s2 renvoie une nouvelle chaîne formée de la concaténation de s1 et s2.
`value concat : string list -> string`	Renvoie une nouvelle chaîne égale à la concaténation de toutes les chaînes contenues dans list.
`value sub_string : string -> int -> int -> string`	"sub_string s start len" retourne une nouvelle chaîne le longueur len contenant les caractères de rang start à start+len-1 de la chaîne s.
`value create_string : int -> string`	Création d'une chaîne de longueur n. Les caractères contenus dans la chaîne sont arbitraires.
`value make_string : int -> char -> string`	make_string n c permet la création d'une chaîne de longueur n et contenant le caractère c répété n fois.
`value fill_string : string -> int -> int -> char -> unit`	fill_string s start len c modifie s en replaçant tous les caractères compris entre start et start+len+1 par c.
`value replace_string : string -> string -> int -> unit`	replace_string dest src start copie tous les caractères de la chaîne src dans la chaîne dest en partant de start.

1.4.) Boucles et alternative.

1.4.1.) BOUCLES "POUR".

```
...
for i = 1 to 10 do
      Bloc d'instructions
done;
...
```



<boucle_pour> ::=
for <identificateur> **=** <expression> **to** <expression> **done**
| **for** <identificateur> **=** <expression> **downto** <expression> **done**

Ecrire une chaîne de caractère passée en paramètre à l'envers (méthode itérative) :

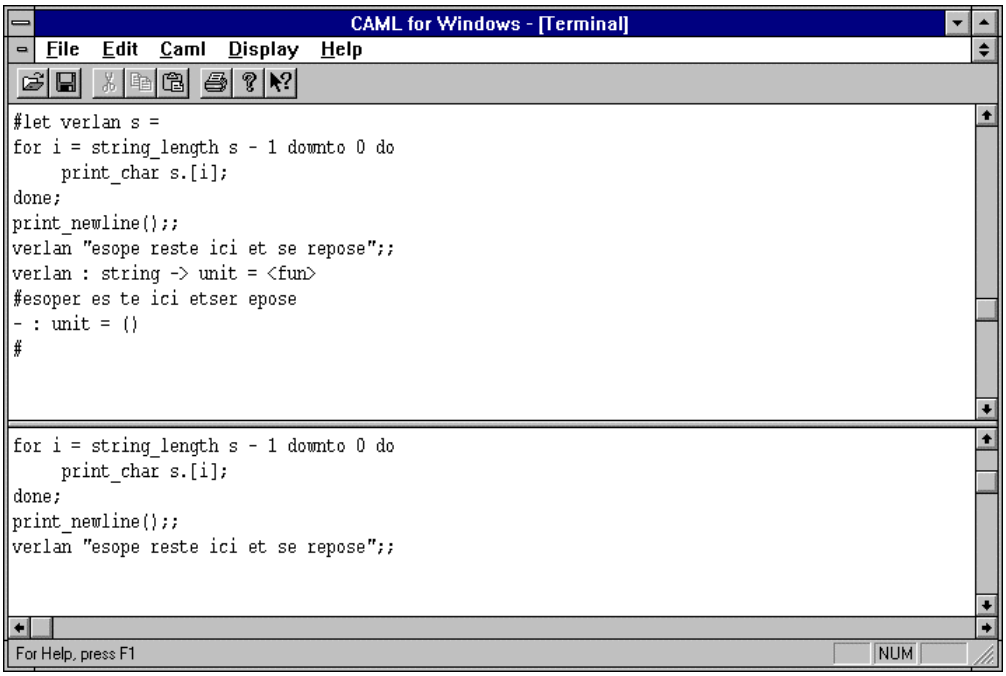


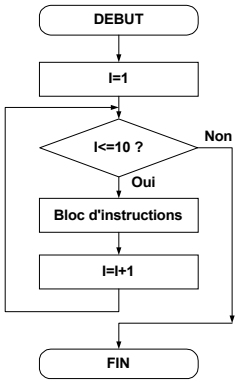
Figure 4 Boucle ""Pour".

```
let verlan s =
  for i = string_length(s)-1 downto 0 do
  begin
    print_char(s.[i]);
  end
  done;
  print_newline();;
```

1.4.2.) BOUCLES "TANT QUE".

```
...
let i=ref 0 in
begin
      i:=1;
      while i <= 10 do

            Bloc d'instructions
      done;
end;
...
```



<boucle_tantque> ::=
while <expression> **do** <expression> **done**

Ecrire les chiffres de 0 à 9 en utilisant une boucle "tant que".

```
let boucle () =
  let i = ref 9 in
  while !i >= 0 do
    print_int !i;
    i := !i - 1;
  done;;;
```

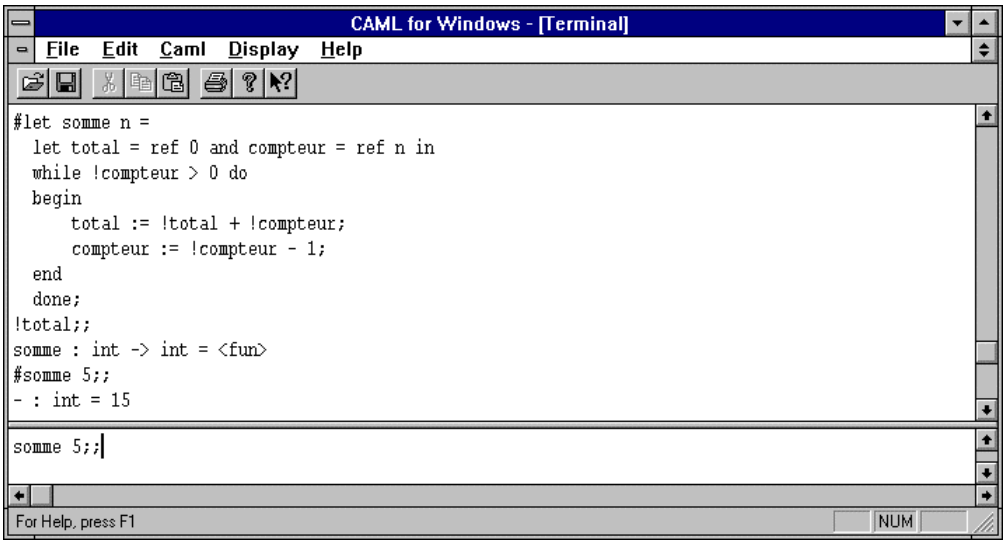


Figure 5 : Boucle "Tant que".

2.) Les tris élémentaires.

2.1.) Tri par insertion.

Cette méthode de tri insère (au i-ième passage) le i-ème élément T[i] à la bonne place parmi T[1] T[2]...T[i-1]. On peut dire qu'après l'étape i, tous les éléments entre la première et la i-ème position sont triés.

Voici l'algorithme :

pour i:=2 à n faire
déplacer T[i] vers le début du tableau jusqu'à la position j<=i telle que T[j] < T[k] pour j<=k<i et ou bien T[j]>=T[j-1] ou bien j=1.

Appliqué au vecteur `[|4;2;0;5;3|]`, on aura :

4	2	0	5	3	Vecteur de départ
2	4	0	5	3	Les cellules 0 à 2 sont triées
0	2	4	5	3	Les cellules 0 à 3 sont triées
0	2	4	5	3	Les cellules 0 à 4 sont triées
0	2	3	4	5	Les cellules 0 à 5 sont triées

```
#let tri vecteur =
let j = ref 0 in
for i=1 to (vect_length(vecteur)-1) do
begin
 j := i - 1;
 while (!j >= 0) && (vecteur.(!j) > vecteur.(!j+1)) do
 begin
 echange !j (!j+1);
      j := !j - 1;
 end;
 done;
end
done;
vecteur;
where echange i j =
let a = vecteur.(i) in
begin
 vecteur.(i)<-vecteur.(j);
 vecteur.(j)<-a
end;;
tri : 'a vect -> 'a vect = <fun>
```

2.2.) Tri par insertion en double sens.

Pour diminuer le nombre de déplacements d'éléments, le vecteur à trier est considéré comme cyclique (sa fin rejoint son début) et on n'opère les déplacements que sur des moitiés de vecteur trié. Pour ce faire, on compare l'élément à insérer à l'élément qui se trouve au milieu de la séquence triée. S'il est plus grand, on le place dans la partie droite sinon on le place dans la partie gauche. Si on le place dans la partie gauche, on conserve l'élément à gauche du plus petit des éléments triés, pour le placer à l'étape suivante et on décale vers la gauche les éléments triés pour insérer l'élément. Si on le place dans la partie droite, on décale vers la droite les éléments triés, en commençant par le plus grand des triés, pour insérer l'élément à placer.

Exemple: Soit à trier:

2	0	4	1	3	9

La partie déjà triée est écrite en rouge. 0 est inférieur à 2 et doit donc être placé dans la partie gauche. Il se place donc à gauche du premier élément, à savoir en dernière position puisque le vecteur est considéré comme cyclique.

2	0	4	1	3	0

et on garde le 9 qui a été écrasé et doit donc être placé à cette étape. Comme 9 est supérieur à l'élément milieu (entre 0 et 2), il doit être placé dans la partie droite. Il est supérieur à 2 et donc aucun déplacement n'est nécessaire avant de le placer.

2	9	4	1	3	0

Il faut maintenant placer l'élément suivant, à savoir 4. L'élément qui se trouve au milieu de la partie triée est 2 et il est inférieur à l'élément à placer. Il faut donc insérer le 4 dans la partie droite. On recule le 9 et on insère le 4.

2	4	9	1	3	0

Il faut placer l'élément suivant, à savoir le 1. Il est plus petit que l'élément qui est au milieu de la partie triée. Il faut donc l'insérer dans la partie gauche. On conserve l'élément à gauche du plus petit des triés, à savoir 3. On déplace le 0 vers la gauche et on place le 1. Nous conservons le 3 qui doit être placé à l'étape suivante.

2	4	9	1	0	1

Comme 3 est plus grand que l'élément qui se trouve au milieu de la partie triée, on doit le placer dans la partie droite. On déplace, vers la droite; le 9, le 4 et on place le 3.

2	3	4	9	0	1

Le vecteur est trié mais il faut opérer plusieurs décalages pour amener le plus petit élément en première position:

1	2	3	4	9	0

et après le deuxième décalage:

0	1	2	3	4	9

2.3.) Tri par sélection-échange.

Le principe du tri par sélection-échange d'un vecteur est d'aller chercher le plus petit élément du vecteur pour le mettre en premier, puis de repartir du second, d'aller chercher le plus petit élément pour le mettre en second etc.

Au i-ème passage, on sélectionne l'enregistrement ayant la plus petite clé parmi les positions i..n et on l'échange avec T[i].

Appliqué au vecteur `[|4;2;0;5;3|]`, on aura :

4	2	0	5	3	Vecteur de départ
0	2	4	5	3	Le plus petit élément est à sa place
0	2	4	5	3	Les 2 plus petits éléments sont à leur place
0	2	3	5	4	Les 3 plus petits éléments sont à leur place
0	2	3	4	5	Les 4 plus petits éléments sont à leur place

```
#let tri vecteur =
for i=0 to (vect_length(vecteur)-2) do
 for j=(i+1) to (vect_length(vecteur)-1) do
 if vecteur.(i) > vecteur.(j) then echange vecteur i j
 else ();
 done;
done;
vecteur;
where echange vecteur i j =
let a = vecteur.(i) in
begin
 vecteur.(i)<-vecteur.(j);
 vecteur.(j)<-a
end;;
tri : 'a vect -> 'a vect = <fun>
```

2.4.) Tri bulle.

Le principe du tri bulle (*bubble sort*) est de comparer deux à deux les éléments e1 et e2 consécutifs d'un tableau et d'effecteur une permutation si e1 > e2. On continue de trier jusqu'à ce qu'il n'y ait plus de permutation.

Appliqué au vecteur `[|4;2;0;5;3|]`, on aura :

4	2	0	5	3	Vecteur de départ
4	0	2	3	5	Fin du premier passage.
0	2	3	4	5	Fin du deuxième et dernier passage.

```
#let tri_bulle vecteur =
let permut = ref true in
begin
 while !permut do
 begin
      permut:=false;
      for i=0 to (vect_length(vecteur)-2) do
           if vecteur.(i)>vecteur.(i+1) then
           begin
                echange vecteur i (i+1);
                permut:=true;
           end
           else ();
      done;
 end
 done;
end;
vecteur;
where echange vecteur i j =
let a = vecteur.(i) in
begin
 vecteur.(i)<-vecteur.(j);
 vecteur.(j)<-a
end;;
tri_bulle : 'a vect -> 'a vect = <fun>
```

2.5.) Tri shell.

Le tri Shell consiste à trier séparément des sous-suites de la table, formées par les éléments répartis de h en h. Empiriquement, Shell propose la suite d'incréments vérifiant $h_1 = 1$, $h_{n+1} = 3h_n+1$ en réalisant les tris du plus grand incrément possible vers le plus petit.

```
#let tri vecteur h = let i = ref 0 and j = ref 0 in
for k=0 to (h-1) do
begin
 i:=k+h;
 while (!i < vect_length(vecteur)) do
 let x = vecteur.(!i) in
 begin
      j:=!i - h;
      while (!j>=0) && vecteur.(!j)>x do
           vecteur.(!j+h) <- vecteur.(!j);
           j:=!j - h;
           vecteur.(!j + h) <- x;
      done;
      i:=!i + h;
 end
 done;
end
done;;
#let  tri_shell  vecteur  =  let  h  =  ref  1  and  l  =
vect_length(vecteur)+1 in
begin
 while !h<(l/9) do h:=3 * !h + 1;done;
 while (!h>0) do
 begin
      tri vecteur !h;
      h:=!h / 3;
 end
done;
end;;
tri : 'a vect -> int -> unit = <fun>
tri_shell : 'a vect -> unit = <fun>
```

Fonctionnement avec

5	8	3	2	5	9	0	6	4	7

n=3

5	8	3	**2**	5	9	**0**	6	4	**7**		0	8	3	2	5	9	5	6	4	7
0	**8**	3	2	**5**	9	5	**6**	4	7		0	5	3	2	6	9	5	8	4	7
0	5	**3**	2	6	**9**	5	8	**4**	7		0	5	3	2	6	4	5	8	9	7

n=2

0	5	**3**	2	**6**	9	**5**	8	**4**	7		0	5	3	2	4	9	5	8	6	7
0	**5**	3	**2**	4	**9**	5	**8**	6	**7**		0	2	3	5	4	7	5	8	6	9

n=1

0	**5**	**3**	**2**	**4**	**9**	**5**	**8**	**6**	**7**		0	2	3	4	5	5	6	7	8	9

2.6.) Tri Shuttle

Le tri Shuttle (shuttle=navette) fonctionne comme le tri bulle mais effectue un retour arrière dès qu'une permutation a été effectuée.

Appliqué au vecteur `[|4;2;0;5;3|]`, on aura :

4	2	0	5	3	Vecteur de départ
2	4	0	5	3	Après le premier retour arrière.
0	2	4	5	3	Après le deuxième retour arrière
0	2	4	3	5	Après le troisième et dernier retour arrière

```
#let echange vecteur i j =
let a = vecteur.(i) in
begin
 vecteur.(i)<-vecteur.(j);
 vecteur.(j)<-a
end;;
echange : 'a vect -> int -> int -> unit = <fun>
#let tri_shuttle vecteur =
let test = ref true and j = ref 1 in
for i=0 to (vect_length(vecteur)-2) do
 j:=i;
 test:=(vecteur.(!j)>vecteur.(!j+1));
 while !test do
       echange vecteur !j (!j+1);
       j:=!j-1;
       if (!j<0) then test:=false
       else
       test:=vecteur.(!j)>vecteur.(!j+1);
 done;
done;;
tri_shuttle : 'a vect -> unit = <fun>
```

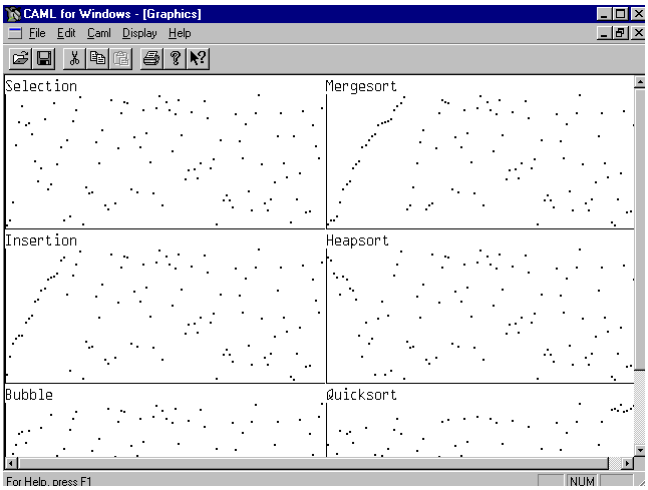


Figure 6 Illustration de différentes méthodes de tri[1].

[1] Le programme contenant cette illustration se trouve, avec d'autres exemples Caml, dans le répertoire `.../Caml/Examples/Showsort/Loadall.ml`.

3.) Notions fondamentales à retenir.

3.1.) Langage Caml.

3.1.1.) REFERENCES.

Les références sont des structures de données prédéfinies qui modélisent les cases mémoires de la machine. La propriété caractéristique des références est qu'on peut les lire et les écrire. Les références se comportent comme les variables dans les langages impératifs tels que Pascal.

Définition:	`let x = ref 0 in ...`
Accès:	l'opérateur **`!`** retourne le contenu de la référence argument (`! reference`).
Affectation:	l'opérateur **`:=`** modifie le contenu de la référence en partie gauche (`reference := valeur`)

3.1.2.) VECTEURS.

Les vecteurs (aussi appelés tableaux) sont des suites finies et modifiables de valeurs d'un même type. Puisque les éléments du tableau sont tous de même nature, on qualifie les tableaux de suites homogènes de valeurs.

Définition:	`[| 1; 2; 3 |]` ou `make_vect` *`nombre_d'éléments valeur_initiale`*
Accès:	`v.(2)` ou `vect_item v 2`
Affectation:	`v.(2) <-` *`nouvelle_valeur`* ou `vect_assign v 2` *`nouvelle_valeur`*
Fonctions :	`vect_length, concat_vect, sub_vect`

3.1.3.) CHAINES DE CARACTERES.

Les chaînes se comportent essentiellement comme des tableaux de caractères. Le premier élément d'une chaîne a la position 0, le dernier a la position (longueur de la chaine - 1).

Définition:	`"chaine"` ou `make_string` *`longueur caractere_de_remplissage`*
Accès:	`s.[2]` ou `nth_char s 2`
Affectation:	`s.[2] <- 'x'` ou `set_nth_char s 2 'x'`
Fonctions :	`string_length, sub_string`

3.1.4.) BOUCLES.

```
for i = 0 to 10 do print_int i done
for i = 10 downto 0 do print_int i done
while !j > 0 do j := !j - 1 done
```

3.2.) Algorithmique.

3.2.1.) ALGORITHME DU TRI PAR INSERTION.

On souhaite trier un vecteur *T* de *n* éléments :

pour i:=2 à n faire
déplacer T[i] vers le début du tableau jusqu'à la position j<=i telle que
T[j] < T[k] pour j<=k<i et ou bien T[j]>=T[j-1] ou bien j=1.

3.2.2.) ALGORITHME DU TRI PAR SELECTION-ECHANGE.

Le principe du tri par sélection-échange d'un vecteur est d'aller chercher le plus petit élément du vecteur pour le mettre en premier, puis de repartir du second, d'aller chercher le plus petit élément pour le mettre en second etc.

3.2.3.) ALGORITHME DU TRI BULLE.

Le principe du tri bulle est de comparer deux à deux les éléments e1 et e2 consécutifs d'un tableau et d'effecteur une permutation si e1 > e2. On continue de trier jusqu'à ce qu'il n'y ait plus de permutation.

4.) Exercices.

EXERCICE A :

Cherchez les 5 nombres entiers positifs compris entre 1 et 500 qui sont égaux à la somme des cubes de leurs chiffres.

```
#let cube x = x*x*x;;
let f () = let s = ref "000" and c = ref 0 in
for i = 1 to 500 do
     s:=string_of_int i;
     c:=0;
     for j = 0 to ((string_length !s)-1) do
          c:= !c + cube ((int_of_char !s.[j]) - 48);
     done;
     if !c = i then
     begin
          print_string "Nombre ";
          print_int i;
          print_newline();
     end
     else ();
done;;
cube : int -> int = <fun>
f : unit -> unit = <fun>
```

EXERCICE B :

Ecrire le début du nombre de Champernowne : 0,123456789101112... construit avec les entiers successifs. Se limiter à 100 décimales.

```
let f () = let s = ref (create_string 110) and i = ref 0 in
s := "0,";
while (string_length !s)<102 do
     i:=!i+1;
     s:=!s ^ string_of_int !i;
done;
print_string !s;
print_newline();;

f : unit -> unit = <fun>
```

EXERCICE C :

Trouver le plus petit entier N dont le chiffre des unités est six (6) et tel que, quand on efface celui-ci et que celui-ci est placé à gauche du nombre sans modifier les autres chiffres, on obtienne le quadruple de N.

```
let f () = let s = ref "000000" and s1 = ref "000000" and i = ref 15
and trouve = ref false in
while not !trouve do
     i := !i + 6
     s := string_of_int !i;
     if !s.[(string_length !s)-1]=`6` then
     (
          s1 := "6" ^ sub_string !s 0 ((string_length !s) - 1);
          if (!i * 4) = (int_of_string !s1) then trouve:=true else
();
     )
     else ();
done;
print_string "Nombre trouve : ";
print_int !i;
print_newline();;
f : unit -> unit = <fun>
#f();;
Nombre trouve : 153846
- : unit = ()
```

EXERCICE D :

Que fait la fonction suivante :

```
#let mystere x y =
let a = ref x and b = ref y in
while !a >= !b do
     if (!a mod 2)=0 then a:=!a / 2
     else a := !a - !b;
done;
if !a=0 then "Oui" else "Non";;
mystere : int -> int -> string = <fun>
```

EXERCICE E :

Ecrire une fonction qui code une chaîne de caractère en utilisant le principe suivant :

- Remplacer chaque caractère de la chaîne entre A et Y par son suivant dans l'ordre alphabétique.
- Remplacer le caractère Z par A.

```
#let change = function
| c when (c >=`A`) && (c <= `Y`) -> let i = int_of_char c in
char_of_int (i+1)
| `Y` -> `A`
| c when (c >=`a`) && (c <= `y`) -> let i = int_of_char c in
char_of_int (i+1)
| `y` -> `y`
| autre -> autre;;
change : char -> char = <fun>
#let code s =
for i=0 to (string_length s -1) do
     set_nth_char s i (change s.[i]);
done;
s;;
code : string -> string = <fun>
#code "A la fin tu es las de ce monde ancien";;
- : string = "B mb gjo uv ft mbt ef df npoef bodjfo"
```

EXERCICE F :

Placez les entiers de 1 à 9 dans un carré de trois lignes et trois colonnes tel que la somme de chaque ligne est égale à la somme de chaque colonne et à la somme des diagonales. Par exemple :

2	9	4
7	5	3
6	1	8

```
#let carre () =
for i1 = 1 to 9 do
  for i2 = 1 to 9 do
    if i2 <> i1 then
    for i3=1 to 9 do
       if (i3<>i1) && (i3<>i2) then
       for i4=1 to 9 do
          if (i4<>i1) && (i4<>i2) && (i4<>i3) then
          for i5 = 1 to 9 do
           if (i5<>i1) && (i5<>i2) && (i5<>i3) && (i5<>i4) then
             for i6 = 1 to 9 do
                if (i6<>i1) && (i6<>i2) && (i6<>i3) && (i6<>i4)
                && (i6<>i5) then
                for i7=1 to 9 do
                   if (i7<>i1) && (i7<>i2) && (i7<>i3) && (i7<>i4)
                   && (i7<>i5) && (i7<>i6) then
                   for i8=1 to 9 do
                   if (i8<>i1) && (i8<>i2) && (i8<>i3) &&
                   (i8<>i4)&& (i8<>i5) && (i8<>i6) && (8<>i7) then
                   for i9=1 to 9 do
                            if (i9<>i1) && (i9<>i2) && (i9<>i3) &&
                     (i9<>i4)  && (i9<>i5) && (i9<>i6) && (i9<>i7)
                      && (i9<>i8) then
                           if ((i1+i2+i3)=(i4+i5+i6)) &&
                           ((i4+i5+i6)=(i7+i8+i9)) &&
                           ((i1+i2+i3)=(i1+i4+i7)) &&
                           ((i1+i4+i7)=(i2+i5+i8)) &&
                           ((i2+i5+i8)=(i3+i6+i9)) &&
                           ((i1+i2+i3)=(i1+i5+i9)) &&
                           ((i1+i2+i3)=(i3+i5+i7)) then
                           begin
                                   print_newline();
                                   print_int i1;print_string " ";
                           print_int i2;print_string " ";
                           print_int i3;print_newline();
                           print_int i4;print_string " ";
                           print_int i5;print_string " ";
                           print_int i6;print_newline();
                           print_int i7;print_string " ";
                           print_int i8;print_string " ";
                           print_int i9;print_newline();
                           end;
                      done;
                   done;
                done;
             done;
          done;
       done;
    done;
  done;
done;;
carre : unit -> unit = <fun>
```

4.1.) Exercices.

EXERCICE 4.1

La procédure suivante, écrite en Pascal, calcule le jour et le mois de Pâques (le premier dimanche qui suit la pleine lune après l'équinoxe de printemps) pour une année donnée.

```
PROCEDURE PAQUES (Yr : INT);

VAR G, C, X, Z, D, E, N : INT;
    mois,jour : INT;

BEGIN
     G := Yr MOD 19 + 1;
     C := Yr DIV 100 + 1;
     X := (3 * C) DIV 4 - 12;
     Z := (8 * C + 5) DIV 25 - 5;
     D := (5 * Yr) DIV 4 - X - 10;
     E := (11 * G + 20 + Z - X) MOD 30;
     IF E < 0 THEN E := E + 30;
     IF ((E = 25) AND (G > 11)) OR (E = 24) THEN E := E + 1;
     N := 44 - E;
     IF N < 21 THEN N := N + 30;
     N := N + 7 - ((D + N) MOD 7);
     IF N > 31 THEN
     BEGIN
          mois := 4;
          jour := N - 31
     END
     ELSE
     BEGIN
          mois := 3;
          jour := N
     END;
     WRITELN("En ",Yr," Pâques se fêtera le ",jour," / ",mois);
END;
```

L'instruction Pascal "x **DIV** y" calcule la division entière de "x" par "y". Elle correspond à l'instruction Caml "x / y". L'instruction "**:=**" est l'instruction d'affectation ; elle correspond à l'instruction "=" en Caml.

Réécrivez cette fonction en Caml.

EXERCICE 4.2

On considère 2 suite (U) et (V) définies (pour i > 1) à partir de

$U_1=1$, $U_2=2$ $U_i=U_{i-1} + U_{i-2}$
$V_i = U_i / U_{i-1}$

La suite V tend vers une limite appelée nombre d'or. On supposera que le n-ième terme de la suite (V), soit V_n, donne une valeur approchée du nombre d'or avec une précision E, dès que

$$|V_n-V_{n-1}| < E.$$

Ecrire un programme Caml donnant la valeur du nombre d'or avec une précision E.

EXERCICE 4.3

Ecrivez une fonction récursive qui reçoit la chaîne *s* et un caractère *c* en paramètres et qui renvoie true si s contient c et false sinon.

EXERCICE 4.4

Le vecteur t est un ensemble de dates (int*int*int) qui contient, par exemple, les éléments suivants :

```
#let t = [|(1854,10,20); (1768,09,04); (1844,03,30); (1802,02,26);
(1799,05,20); (1905,06,21);(1524,09,01)|];;
```

Evidemment, (1854,10,20) correspond au 20 octobre 1854.

En s'inspirant de l'algorithme du tri bulle, écrivez une fonction qui tri le vecteur t par ordre chronologique croissant.

EXERCICE 4.5

Ecrire un programme permettant de calculer la racine carrée d'un nombre entier a avec une précision E en utilisant l'algorithme d'Héron d'Alexandrie :

$$U_0 = r$$
$$U_1 = \frac{1}{2}\left(U_0 + \frac{a}{U_0}\right)$$
$$U_{n+1} = \frac{1}{2}\left(U_n + \frac{a}{U_n}\right)$$

Avec r = le plus grand des entiers p tels que $p^2 <= a$.

On dira qu'on a atteint la précision E dés que |Un-Un-1| < E.