Université Paris 6
Licence d'informatique

Module de Programmation
ANNÉE 2000-2001

Travaux Dirigés n° 1

Exercice 1 Évaluez les expressions ci-dessous. On supposera qu'à chaque question correspond une nouvelle session CAML interactive.

1 *(Manipulation d'expressions)*

```
4 < 3;;
"debut " ^ "fin";;
4.0 * 6;;
string_of_int 34;;
4.0 >= 45.6;;
4.0 >= 45;;
```

2 *(Déclarations de booléens)*

```
let x = 4 < 3;;
let x = x = true;;
```

3 *(Déclarations consécutives)*

```
let x = false;;
let y = (x = true);;
let z = if x = true then false else true;;
let x = 3;;
let x = false;;
let x = x + 2;;
```

4 *(Déclarations simultanées)*

```
let nb1 = 4 and nb2 = nb1;;
let nb1 = 4;;
let nb2 = nb1;;
let y = 1;;
let y = 3 and z = y;;
```

5 *(Portée statique)*

```
let a = 3 and b = 1;;
let f x = a * x + b;;
f 1;;
let a = 0 and b = 0;;
f 1;;
```

6 *(Priorité des opérateurs et de l'application)*

```
let g x y = x + y;;
g 4 g 5 6;;
g 4 5 * 6;;
sub_string "abcdef" 0 3 ^ "gh";;
let h s1 s2 = s1^s2;;
h sub_string "abcdef" 0 3 "gh";;
```

Un opérateur tel que `*` ou `^` est moins prioritaire que l'application d'une fonction. L'application de tête est plus prioritaire qu'une application dans un des arguments.

Moralité : si vous avez le moindre doute, parenthèsez, vous serez sûr d'avoir écrit ce que vous aviez en tête et votre code gagnera en lisibilité.

Exercice 2 *(Équations booléennes)*

1 *(« ou » logique)* Montrer que les expressions :

- `if` a `then true else` $F(a)$
- `if` a `then` a `else` $F(a)$
- a `||` $F(a)$

ont les mêmes valeurs pour tout booléen a et toute fonction booléenne F.

2 *(« et » logique)* Montrer que les expressions :

- `if` a `then` $F(a)$ `else false`
- `if` a `then` $F(a)$ `else` a
- a `&&` $F(a)$

ont les mêmes valeurs pour tout booléen a et toute fonction booléenne F.

3 Qu'en concluez vous pour l'écriture de vos programmes ?

Exercice 3 Écrivez une fonction prenant en argument deux nombres entiers strictement positifs x et y, et renvoyant le PGCD de ces deux nombres. Pour calculer le PGCD, on utilisera les règles suivantes :

- si $x = y$ alors $\text{PGCD}(x, y) = x$;
- si $x > y$ alors $\text{PGCD}(x, y) = \text{PGCD}(x - y, y)$;
- si $y > x$ alors $\text{PGCD}(x, y) = \text{PGCD}(x, y - x)$.

Exercice 4 On veut créer une fonction générant, pour un entier $k > 0$, la suite de k lignes suivantes :
ligne 1 : 1
ligne 2 : 1 1
ligne 3 : 2 1
ligne 4 : 1 2 1 1
ligne 5 : 1 1 1 2 2 1
ligne 6 : 3 1 2 2 1 1
En fait, on obtient la ligne k en lisant ce qui est écrit sur la ligne $k - 1$. Par exemple, la ligne 5 s'obtient en lisant la ligne 4 : sur cette dernière on voit 1 « 1 », 1 « 2 » et 2 « 1 ».

1 Écrivez une fonction `calc_préfixe` qui prend en argument une chaîne de caractères `str` ayant la forme des lignes mentionnées ci-dessus, et qui renvoie 0 si `str` est vide, ou le nombre de chiffres identiques au début de la chaîne `str`. Par exemple `calc_préfixe "1 1 1 2 2 1 "` renvoie la valeur 3 car la chaîne de caractères commence par trois « 1 ».

2 Écrivez une fonction `génère_ligne` qui prend en argument une chaîne de caractères correspondant à l'une des lignes décrites ci-dessus, et qui renvoie la ligne suivante. Pour simplifier, on supposera qu'après chaque nombre, il y a un espace. On admettra en outre que tous les nombres affichés sont compris entre 1 et 9.

3 Écrivez une fonction `génère` qui prend en argument un entier $k > 0$, et qui renvoie une chaîne de caractères contenant les k premières lignes de la suite. La chaîne de caractères permettant d'aller à la ligne suivante est « $\backslash n$ ».

4 Écrivez une fonction `écrit_suite` qui prend en argument un entier $k > 0$, et qui affiche la chaîne de caractères contenant les k premières lignes de la suite. La fonction permettant d'afficher une chaîne de caractères s'appelle `print_string`.