Premier cycle Jeudi 22 Juin 2006

# I3 - Algorithmique et Langage C

Durée : *3h00*
Documents autorisés : **AUCUN** (calculatrice comprise)

**Remarques** :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.

## 1 Multiplication égyptienne (3 points)

### 1.1 Méthode

Nous avons vu dans le dernier partiel que les égyptiens de l'antiquité savaient :
- additionner deux entiers strictement positifs,
- soustraire 1 à un entier strictement positif,
- multiplier par 1 et 2 tout entier strictement positif,
- diviser par 2 un entier strictement positif pair.

Voici un exemple qui multiplie 14 par 13 en utilisant uniquement ces opérations :

| | | |
|---|---|---|
| $14 \times 13$ | $= 14 + \mathbf{14} \times \mathbf{(13 - 1)}$ | $= 14 + \mathbf{14} \times \mathbf{12}$ |
| | $= 14 + \mathbf{(14} \times \mathbf{2)} \times \mathbf{(12 / 2)}$ | $= 14 + \mathbf{28} \times \mathbf{6}$ |
| | $= 14 + \mathbf{(28} \times \mathbf{2)} \times \mathbf{(6 / 2)}$ | $= 14 + \mathbf{56} \times \mathbf{3}$ |
| | $= 14 + 56 + \mathbf{56} \times \mathbf{(3 - 1)}$ | $= 70 + \mathbf{56} \times \mathbf{2}$ |
| | $= 70 + \mathbf{(56} \times \mathbf{2)} \times \mathbf{(2 / 2)}$ | $= 70 + \mathbf{112} \times \mathbf{1}$ |
| | $= 70 + 112$ | $= 182$ |

### 1.2 Algorithme

Nous allons écrire l'algorithme récursif qui permet la multiplication de 2 naturels suivant cette méthode :

1. Déterminer le ou les cas terminaux. Déterminer le ou les cas généraux.
2. Donner le corps de la fonction suivante en utilisant un algorithme **récursif** :

   **fonction** multiplicationEgyptienne (a,b : **Naturel) : Naturel**

## 2 Drapeau tricolore (3 points)

Soit le type énuméré `Couleur` et le type `TableauDeCouleur` définis de la façon suivante :
- **Type** Couleur **=** {Bleu,Blanc,Rouge}
- **Type** TableauDeCouleur **= Tableau[**1..MAX**] de** Couleur

Dans le cadre de la coupe du monde de football, on se propose de développer une procédure (`construireDrapeauFrancais`) qui pour un tableau de couleurs (contenant $nb$ éléments significatifs), **trie** les éléments de façon à ce qu'il y ait successivement tous les bleus en début de tableau, puis tous les blancs enfin tous les rouges.

1. Proposer une analyse descendante de ce problème.
2. Donner le corps de la procédure `construireDrapeauFrancais` et des fonctions ou procédures de votre analyse.

# 3 Les pointeurs en C (3 points)

Soit $p$ un pointeur qui "pointe" sur un tableau $t$ :

```
int t[] = {12, 23, 43, 45, 56, 67, 79, 89, 90};
int *p;
p = t;
```

Quelles valeurs ou adresses fournissent les expressions suivantes :

1. `p+(*p-10)`
2. `*(p+2)`
3. `*p+2`
4. `t+3`
5. `*(&(t[4])-3)`
6. `*(p+*(p+8)-t[7])`

# 4 Encore un peu de C (5 points)

Dans un espace orthonormé à deux dimensions, on définit un carré par les coordonnées de son point bas gauche et par la longueur d'un coté.

1. Définir un type `Carre` qui permet de stocker les informations définissant un carré.
2. On considère la relation d'inclusion entre les carrés suivante : "un carré C2 est inclus dans un carré C1 si, graphiquement, le carré C2 est à l'intérieur du carré C1".
   (a) Écrire une fonction `int estInclusDans(Carre C2, Carre C1)` qui retourne 1 si $C2$ est inclus dans $C1$, 0 sinon.
   (b) Écrire une fonction qui, pour un tableau de carrés, permet de trouver l'indice dans le tableau, du carré qui inclut le plus de carrés.

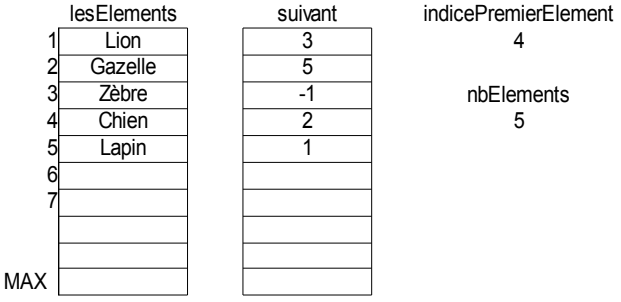
# 5 Et pour finir un peu d'algo (6 points)

On se propose dans cet exercice d'utiliser une structure de données qui ordonne nativement des éléments. Cette structure de données est composée de 4 éléments :

1. un tableau permettant de stocker les éléments (`lesElements`),

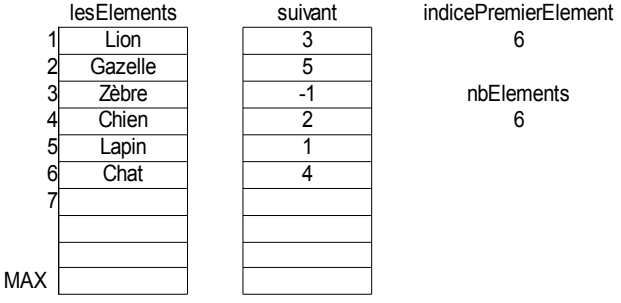
2. un tableau de naturels, associé au premier, qui permet de connaître l'indice de l'élément qui suit un autre avec la valeur -1 s'il n'y en a pas (`suivant`),
3. un naturel qui indique le nombre d'éléments (`nbElements`),
4. un naturel qui indique l'indice du premier élément ( `indicePremierElement`).

Pour illustrer cette structure nous allons considérer que nous voulons stocker des noms d'animaux. Les ajouts successifs de "Lion", "Gazelle", "Zèbre", "Chien" et "Lapin" donnerait la structure suivante :

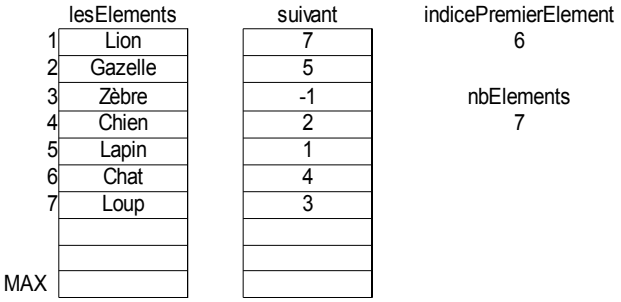



On voit bien que le premier élément est celui qui est à l'indice 4 (valeur d'`indicePremierElement`) et c'est le "Chien" (valeur de `lesElements[4]`). Il est suivi de l'élément se trouvant à l'indice 2 (valeur de `suivant[4]`), c'est la "Gazelle" (valeur de `lesElements[2]`), etc. Le dernier élément est bien "Zèbre" (valeur de `lesElements[3]`) car il ne possède pas d'élément suivant (valeur de `suivant[3]` à -1).

L'ajout de l'élément "Chat" dans cet exemple modifiera la structure de la façon suivante :




Enfin, l'ajout de l'élément "Loup" modifiera la structure de la façon suivante :



On représente cette structure de données à l'aide du type suivant :
**Type** Liste **= Structure**
lesElements : **Tableau[**1..MAX**] de Chaîne de caractères**
suivant : **Tableau[**1..MAX**] de Naturel**
indicePremierElement : **Naturel**
nbElements : **Naturel**
**finstructure**

## 5.1 Initialiser

Écrire le corps de la procédure suivante qui initialise (aucun élément) une variable de type Liste :
– **procédure** initialiser **(** **Entrée/Sortie** l : Liste **)**

## 5.2 Parcourir

Écrire le corps de la procédure suivante qui affiche dans l'ordre les éléments contenus dans une variable de type Liste :
– **procédure** afficher **(** **Entrée** l : Liste **)**

## 5.3 Ajouter

### 5.3.1 Un autre exemple

En reprenant l'exemple donné dans l'énoncé, après l'ajout de "Loup", proposez le résultat de l'ajout de "Pie".

### 5.3.2 L'algorithme

Écrire le corps de la procédure suivante qui ajoute un nom d'animal dans une liste :
– **procédure** ajouter **(** **Entrée/Sortie** l : Liste , **Entrée** nom : **Chaîne de caractères )**
Dans ce cas on considère :
– qu'il y a assez de place pour stocker le nouvel élément
– que l'on possède la fonction `estInferieureOuEgale` qui permet de comparer deux chaînes de caractères :
  – **fonction** estInferieureOuEgale (chaine1, chaine2 : **Chaîne de caractères) : Booléen**