

## **La récursivité**

### **1. Introduction**

Ecrire les définitions récursives des fonctions suivantes, puis écrivez-les en langage C :

- $\text{add}(a, b) = a + b$  ;
- $\text{prod}(a, b) = a * b$ .

Soit le programme suivant :

```
void recursive(){  
    int i ;  
    recursive() ;  
}
```

Expliquez pourquoi l'exécution de ce programme est dangereuse. En déduire une règle pour les programmes récursifs.

### **2. Miroir**

Ecrire une fonction récursive affichant une phrase dans l'ordre inverse de celle reçue en argument.

### **3. Palindrome**

Un palindrome est une chaîne de caractères pouvant être lue dans les deux sens (par exemple "été" est un palindrome).

Ecrire une fonction récursive déterminant si une chaîne reçue en argument est un palindrome.



A partir d'un point donné du labyrinthe quelles sont les possibilités de déplacement. Généraliser à tous les points en écrivant un algorithme de parcours récursif. Ajouter la condition d'arrêt du parcours qui indique que le point A a été trouvé. Comment faire pour que le chemin parcouru du point de départ au point d'arrivée soit marqué de caractères + ?

## **Les structures**

### **1. Choix de la structure de données**

Quelle structure de données utiliser pour stocker les informations suivantes sur une personne ?

- nom ;
- prénom ;
- âge.

Ecrire une fonction permettant d'initialiser une telle structure à partir d'un nom, d'un prénom et d'un âge passée en paramètre. Ecrire des fonctions permettant de récupérer chaque champs d'une structure passé en paramètre. Ecrire un programme principal permettant de tester les fonctions précédentes.

### **2. Cas où le nombre de structures est connu**

Quelle structure de données choisir pour constituer un ensemble fini et non ordonné de personnes ? Ecrire les fonctions suivantes permettant de manipuler cet ensemble :

- initialiser l'ensemble de structures ;
- ajouter une nouvelle structure dans l'ensemble ;
- rechercher un structure par son nom ;
- supprimer une structure à partir de son nom.

Ecrire un programme principal permettant de tester les fonctions précédentes.

On souhaite à présent ordonner cet ensemble d'après le nom des personnes qui le constitue. Réécrire les fonctions précédentes pour ce nouvel ensemble en veillant à utiliser les mêmes en-têtes de fonction. Que gagne-t'on à ordonner l'ensemble ? Quel est l'intérêt de conserver les en-têtes des fonctions ?

### **3. Cas où le nombre de structures est quelconque**

Reprendre les questions du paragraphe 2 dans le cas où le nombre de structures est quelconque (on cherchera à conserver les mêmes en-têtes des fonctions).

## Les pointeurs de fonctions

Les langages à objets implémentent une technique appelée le polymorphisme permettant avec une seule instruction d'appeler des fonctions différentes. Le but de cet exercice est de mettre en œuvre cette technique en C à l'aide de pointeurs de fonctions. Le programme suivant est un exemple d'utilisation d'une instruction polymorphe (`ptrA->f()`) :

```
void Af(){
    printf( "Je suis la fonction A\n" );
}

void Bf(){
    printf( "Je suis la fonction B\n" );
}

void main(){
    A a = initA();
    B b = initB() ;
    A* ptrA = &a ;
    ptrA->f() ;    /* appel de Af */
    ptrA = &b ;
    ptrA->f() ;    /* appel de Bf */
}
```

Pour bien comprendre ce qui se passe, on va commencer par travailler sur une version simplifiée du programme ci-dessus :

```
void Af(){
    printf( "Je suis la fonction A\n" );
}

void main(){
    A a = initA();
    a.f() ;        /* appel de Af() */
}
```

où A est une structure.

Ce qui est surprenant avec le programme ci-dessus, c'est que à droite du point de « `a.f()` », on trouve une fonction et non un champs comme d'habitude avec les structures. Ceci est à la base de la programmation par objet où on regroupe dans une structure, des données, mais aussi des fonctions pour manipuler ces données. Comment ce programme peut-il fonctionner ?

En fait, il y a dans la structure A un pointeur de fonction. Le but de la fonction `initA()` du programme ci-dessus est alors d'initialiser ce pointeur en le faisant pointer vers la fonction `Af()`. Pour réaliser cela, écrivez la déclaration de la structure A, et la définition de la fonction `initA()`.

Ce que vous venez d'écrire est déjà capable de faire fonctionner le programme ci-dessous

```
void Af(){
    printf( "Je suis la fonction A\n" );
}

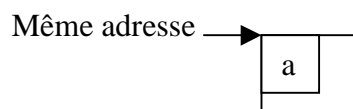
void main(){
    A a = initA();
    A* ptrA = &a ;
    a->f() ;          /* appel de Af() */
}
```

où on passe à présent par un pointeur pour appeler la fonction `f()`.

Revenons maintenant au premier programme principal où ce même pointeur contient successivement l'adresse d'une structure de type A et l'adresse d'une structure de type B ! Pour pouvoir réaliser ce tour de passe-passe, il faut que l'adresse d'une structure de type B puisse être convertie en un adresse de type A. C'est possible à condition d'écrire

```
struct B{
    A a ;
}
```

car cela se représente en mémoire par :



Structure de type B

Pour finir, écrire la fonction `initB()` qui réalise l'initialisation d'une structure de type B en faisant pointer le pointeur de fonction, de la structure a contenue dans la structure B, vers la fonction `Bf()`.

## **Les listes chaînées**

### **1. Les piles et les files**

Les listes chaînées sont des structures de données bien adaptées aux piles car il est facile d'ajouter un élément en tête de liste, et elle permettent de stocker un nombre quelconque d'éléments. Ecrivez les primitives de base de gestion d'une pile permettant :

- de créer une pile ;
- d'empiler un élément ;
- de dépiler un élément.

Ecrivez à l'aide d'une liste chaînée des primitives de gestion d'une file permettant :

- de créer une file ;
- d'ajouter un élément dans la file ;
- d'extraire un élément de la file.

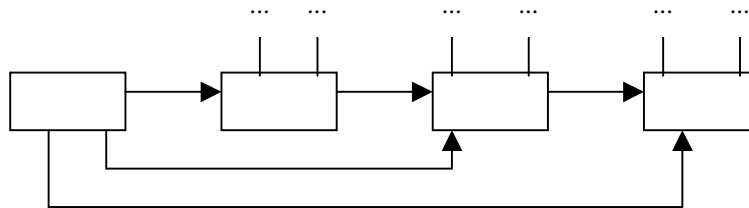
### **2. Le mystère du temple solaire**

Cet exercice se propose, ni plus ni moins, d'aider la police en lui révélant que le suicide des membres du Temple Solaire a laissé un survivant ! ce problème, bien connu des milieux inform...atisés, porte le nom de « problème du Josephus » :

N personnes décident de se suicider ensemble. Pour cela elles forment un cercle. La Mième personne du cercle se suicide et le cercle se referme alors sur les survivants. Quelle est la dernière personne à se suicider ? Quel est l'ordre des suicides ?

### 3. Arbre généalogique

Pour représenter un arbre généalogique on utilise une liste chaînée où en plus du champs qui permet de pointer sur le suivant de la liste, deux champs supplémentaires sont utilisés pour réaliser des liens de parenté.



Rédiger une fonction qui affiche le contenu de la liste sous la forme :

« nom »

Père : « nom du père »

Mère : « nom de la mère »

Ecrire une fonction qui vérifie si deux individus sont frères ou sœurs. Rédiger une fonction qui ajoute un individu en tête de liste à partir de son nom et de celui de ses parents (on suppose que les parents ont déjà été enregistrés dans la liste). Ecrire une fonction qui retourne le nombre d'enfants d'une mère et d'un père.

## **Les fichiers**

Le but de ce TD est de pouvoir sauvegarder sur un support persistant, en l'occurrence un fichier sur un disque dur, les données des programmes du TD sur les structures.

### **1. Lecture et écriture d'une structure dans un fichier binaire**

Ecrire deux fonctions permettant d'insérer et d'extraire une structure de type personne dans un fichier binaire à partir de la position courante.

### **2. Cas où le nombre de structures est connu**

Rédiger une fonction réalisant la sauvegarde d'un ensemble fini de structures dans un fichier binaire. Comment connaître le nombre de structures à partir du seul fichier ? Ecrire une fonction permettant d'initialiser l'ensemble fini de structures à partir du fichier. Ecrire un programme de test.

### **3. Cas où le nombre de structures est quelconque**

Reprendre les questions du paragraphe 2 dans le cas où le nombre de structures est quelconque (on cherchera à conserver les mêmes en-têtes des fonctions).

### **4. Modification du fichier**

Rédiger une fonction permettant de localiser dans le fichier un enregistrement correspondant à une personne à partir de son nom. Ecrire une fonction qui remplace l'enregistrement localisé par un autre.

### **5. Fichier texte**

Afin de faciliter la lecture et la modification manuelle du fichier, écrire une fonction de conversion de fichier binaire à fichier texte et vice versa.

## Les piles

Pour évaluer une expression infixée telle que  $(5 * ((9 + 8) * (4 * 6)) + 7)$ , une méthode consiste à utiliser une pile. Deux étapes sont alors nécessaires :

- transformer l'expression en notation polonaise inversée (notation postfixée) où l'opérateur figure après les opérandes sur lesquels il agit :  $5\ 9\ 8\ +\ 4\ 6\ *\ * 7\ +\ *$  ;
- évaluer l'expression en notation postfixée.

L'intérêt de la notation postfixée est l'absence de parenthèse. Cet exercice montre aussi comment les piles peuvent être utilisées pour mémoriser des résultats intermédiaires ; cette technique étant utilisée par certains processeurs.

Écrire un algorithme qui transforme une expression infixée saisie au clavier en une expression postfixée. Écrire un algorithme qui évalue une expression postfixée.

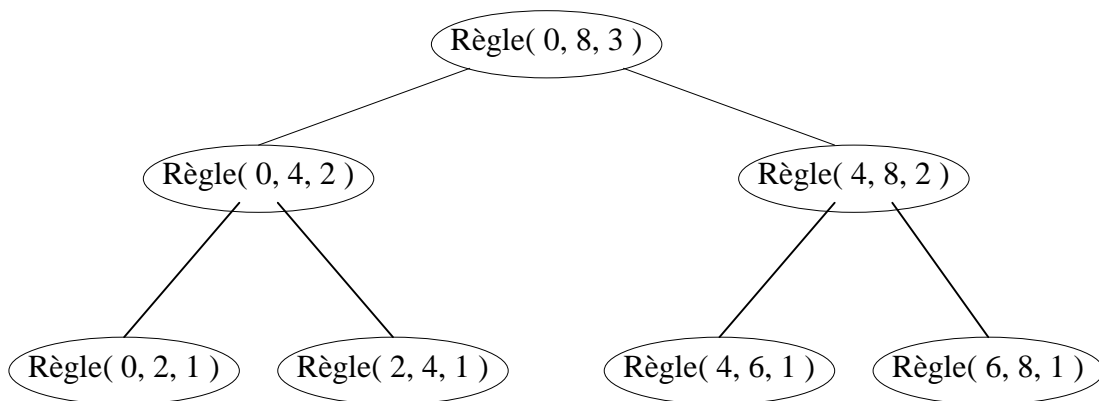
## Les arbres

### 1. Tracer une règle

La plupart des algorithmes récursifs effectuent deux appels récursifs, chacun portant sur environ la moitié des données. C'est le principe diviser pour résoudre. Une application directe de ce principe consiste à graduer une règle, sur laquelle une marque de plus en plus petite est tracée à mesure que la règle est divisée en deux :



Pour représenter le processus de graduation de la règle, un arbre fictif peut être construit, où chaque nœud correspond un appel d'une procédure chargée de tracer les graduations. Cette procédure reçoit comme deux premiers arguments, les coordonnées gauche et droite de la sous-règle où dessiner un trait ; le troisième argument donnant la hauteur de ce trait :



#### Construire la règle avec un parcours préfixé de l'arbre

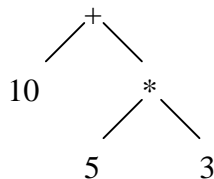
A quelle condition la profondeur maximale de l'arbre est-elle atteinte ? En déduire la condition d'arrêt d'un processus récursif de parcours de l'arbre. Ecrire l'algorithme de la procédure Règle réalisant un parcours préfixé de l'arbre, où la visite d'un nœud consiste à dessiner un trait à la position  $x$ , et de hauteur  $h$ , grâce à l'appel de la procédure Tracer( $x, h$ ).

### Construire la règle avec un parcours infixé de l'arbre

Réécrire l'algorithme précédent en réalisant cette fois un parcours infixé de l'arbre.

## 2. Les arbres syntaxiques

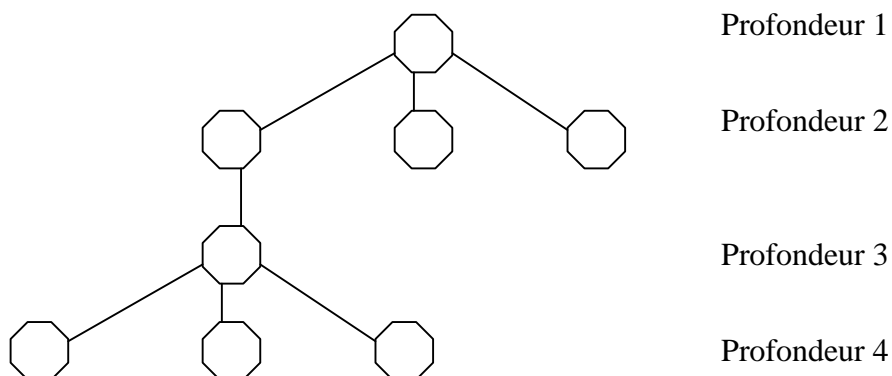
Pour calculer une expression, on peut utiliser un arbre syntaxique :



Proposer un algorithme qui parcourt un tel arbre en calculant l'expression qu'il recèle (on se limitera aux expressions ayant deux opérandes).

## 3. Explorer un arbre n-aire

Pour écrire un algorithme récursif de parcours d'arbre n-aire, on peut remarquer qu'à tous les niveaux, on peut appliquer le même traitement : rien ne différencie un nœud d'un autre si ce n'est la profondeur à laquelle il est dans l'arbre.



En supposant disposer d'une fonction qui permet d'énumérer les nœuds fils d'un nœud quelconque, et d'une fonction permettant d'extraire un nœud de l'ensemble des nœuds fils énumérés, écrivez une fonction de parcours d'un arbre n-aire.

### Parcours en profondeur d'abord

Quelle doit être la structure de données de l'ensemble de nœuds fils pour que l'arbre soit parcouru en profondeur d'abord ?

**Parcours en largeur d'abord**

Quelle doit être la structure de données de l'ensemble de nœuds fils pour que l'arbre soit parcouru en largeur d'abord ?

**Enumération**

Ecrire les procédures énumérerFils et extraireNoeud pour les structures de données des questions précédentes.

## **Tri par fusion**

Le tri par fusion consiste à diviser l'ensemble des enregistrements en sous-ensembles plus petits, et donc plus faciles à trier (le tri se faisant lors de la fusion des sous-ensembles). Ce tri peut être utilisé sur des tableaux d'enregistrements avec des bonnes performances, bien qu'il nécessite une place mémoire supplémentaire souvent proportionnelle à la taille des enregistrements. Cependant, c'est un des tris les plus efficaces sur les listes chaînées.

Ecrire l'algorithme du tri par fusion pour des listes chaînées.

## Recherche dichotomique

### 1. La recherche dichotomique

Transformer l'algorithme de la recherche dichotomique itératif en une version récursive.

### 2. Recherche d'un minimum et d'un maximum

La méthode « diviser pour résoudre » peut être utilisée pour rechercher le minimum et le maximum dans une suite de nombres. L'idée de base qui a permis de construire cet algorithme est qu'il est très facile de rechercher le maximum et le minimum par comparaison de deux nombres seulement. Partant de là, la suite de nombres peut être divisée jusqu'à ce que la comparaison puisse s'effectuer sur deux nombres adjacents dans la liste :

$$4 \quad -5 \quad 20 \quad 6 \quad 0 \quad -36$$

$$\quad \quad \quad \vee \quad \quad \vee$$

Les résultats de ces comparaisons sur les éléments adjacents peuvent être comparés entre eux jusqu'à ce que le maximum et le minimum de la liste soient trouvés :

$$4 \quad -5 \quad 20 \quad 6 \quad 0 \quad -36$$

$$\quad \quad \quad \vee \quad \quad \vee$$

$$\quad \quad \quad \quad \quad \vee$$

$$\quad \quad \quad \quad \quad \dots$$

#### Diviser ...

Quelle structure de données peut être utilisée pour constituer une suite facilement divisible en deux ? En déduire l'en-tête de la procédure de recherche à écrire, et décrire grossièrement son corps en se limitant à la division de la suite.

#### ... pour résoudre

Quel test permet de savoir si la division est suffisante ? Compléter alors l'algorithme pour parvenir au résultat souhaité.