2I006

Algorithmique appliquée et structures de données

2014-2015

Fascicule de TD-TME
Séance 1 à 6

Gr2 Mercredi 16h00-19h45 :	Nawal Benabbou et Pierre Fouilhoux
Gr3 Jeudi 10h45-12h30 / 14h00-15h45 :	Grégoire Cotté et Thibaut Lust
Gr2 Vendredi 14h00-17h45 :	Bruno Escoffier

2I006 : Algorihmique appliquée et Structure de Données

Calendrier 2014-2015

Semaine	Cours	Gpe	TD/TME	
Semaine 1 (19/01)	Cours 1			
Semaine 2 (26/01)	Cours 2	Gpes 1-3	TD1 + TME 1)	
Semaine 3 (02/02)	Cours 3	Gpes 1-3	TD2 + Mini-projet (1)	
Semaine 4 (09/02)	Cours 4	Gpes 1-3	TD3 + Mini-projet (2)	
Semaine 5 (16/02)	Cours 5	Gpes 1-3	TD4 + Projet (1)	Soutenance Mini-projet
Semaine 6 (23/02)	Cours 6	Gpes 1-3	TD5 + Projet (2)	
Semaine 7 (02/03)	Cours 7	Gpes 1-3	TD6 + Projet (3)	
Semaine 8 (09/03)	Cours 8	Gpes 1-3	TD7 + Projet (4)	
Semaine 9 (16/03)	Cours 9	Gpes 1-3	TD8 + Projet (5)	Rendu 1ere partie
Semaine 10 (23/03)	Cours 10	Gpe 1-3	TD9 + Projet (6)	
Semaine 11 (30/03)	Semaine libre			
Semaine 12 (06/04)	Cours 11	Gpes 1-3	TD10 + Projet (7)	
Semaine 13 (13/04)	-	Gpes 1-3	TD11 + Projet (bonus)	Soutenance Projet (total)

Vacances Pâques (2 semaines)

Semaine 14 (04/04)	Révision L2
Semaine 15 (11/05)	Examen 1ere session
Semaine 16 (18/05)	
Semaine 17 (25/05)	
Semaine 18 (01/06)	Examen 2eme session

Cours :	Mercredi	10h45-12h30	Amphi 15	Pierre Fouilhoux Pierre-Henri Wuillemin
Gpe 2	Mercredi	16h00-17h45 24-25-103	18h00-19h45 24-25-304	Pierre Fouilhoux/Nawal Benabbou Adel Aithamlat
Gpe 3	Jeudi	10h45-12h30 14-24-108	14h00-15h45 14-15-403	Thibaut Lust/Grégoire Cotté Sylvain Lobry
Gpe 1	Vendredi	14h00-15h45 Bât Salle 08	16h00-17h45 14-15-401	Bruno Escoffier Raphaël Tackx

TD1: Retour sur le C: compilation, pointeurs et tableaux

Exercice 1 – Rappel : code, indentation

Le style de codage n'est qu'une convention. Toutefois, lorsqu'elle est connue et suivie, cette convention permet d'améliorer sensiblement la lisibilité d'un code pour tous (et particulièrement pour les correcteurs). Nous vous demanderons dans ce module de suivre au mieux la convention K&R (cf. *Le Langage C ANSI*, Brian Kernighan et Dennis Ritchie, 1988) dont voici un exemple parlant :

```
1  /* Voici un commentaire de description d'une fonction */
2  int une_function(void)
3  {
4      int x, y;
5      if (x == y) {
6          /* voici un commentaire local, indent'e comme le code */
7          quelquechose1();
8          quelquechose2();
9      } else {
10         autrechose1();
11         autrechose2();
12     }
13     chosefinale();
14     return une_valeur_retournee;
15 }
```

Remarquez :

- Accolade ouvrante sur la même ligne que l'instruction de contrôle (L5 et L9),
- à l'exception des accolades de début de fonction, placées à la ligne (L3).
- Indentation incrémentée à chaque accolade ouvrante et décrémentée à chaque accolade fermante.
- Commentaires indentés comme le code (L5).
- Utilisation de caractères basiques. En particulier, pas d'accent ni dans le code ni dans les commentaires (L5).

Q 1.1 Faites quelque chose avec ça :

```
1  #include <stdio.h>
2  int xy(int n){ if (n) return n*/*warum nicht ? */xy(n-1); else return 1;} void
3  main(void) { printf("%d", xy(5));}
```

PS : ce programme compile...

Exercice 2 – Rappel : compilation séparée

Séparer un programme en plusieurs fichiers est une bonne idée quand le code prend de l'ampleur. La création de bibliothèques de fonctions accroit fortement la maintenabilité d'un code C et permet de penser à la ré-utilisabilité de code écrit.
Si un projet est composé de 3 fichiers `A.c`, `B.c` et `prog.c` (`A` et `B` sont des librairies de fonctions utilisées par `prog.c` par exemple), on peut demander à compiler l'ensemble dans un seul programme (de nom `prog`) par :

```
gcc A.c B.c prog.c -o prog
```

Toutefois, il est nécessaire de pouvoir inclure dans `prog.c` au moins une spécification (une signature) des fonctions que `A.c` apportera au projet. Cela se fait dans un fichier *header* : `A.h` qui sera inclus dans chaque fichier `*.c` qui voudra utiliser les fonctions définies dans `A.c`.
Par ailleurs, devoir compiler `A.c` et `B.c` chaque fois qu'on fait une modification mineure dans `prog.c` n'est pas forcément utile. En effet, il s'avère que la compilation est composée de 2 phases bien différentes :

1. **La compilation** proprement dite : du `.c` au `.o`.
   Un fichier source est compilé en langage machine (fichier objet). Mais le code est 'relogeable' : on ne connait pas l'adresse des fonctions ni des variables. Plus exactement, les adresses sont encore 'translatables', données dans un mémoire virtuelle.
2. **L'édition de lien** : les mémoires virtuelles des différents fichiers objets sont unifiées et toutes les adresses sont résolues de manière univoque.

C'est uniquement cette dernière étape qui doit être faite d'un seul tenant, avec l'ensemble des fichiers-objets, résultats de compilations qui, elles, peuvent être séparées.
Enfin un fichier `Makefile` suit un format permettant de déclarer les différentes dépendances entre fichiers d'un même projet, afin de trouver le nombre minimum d'opérations (compilations ou édition de lien) nécessaire afin de produire un exécutable final. La commande `make` utilise le fichier `Makefile` pour effectuer ces opérations nécessaires.

Q 2.1 Proposer une organisation, le squelette de chaque fichier et le Makefile associé d'un projet vérifiant que :
- la fonction `int f(int)` est définie dans un fichier source `A`,
- la fonction `void g(int)` est définie dans un fichier source `B` et utilise la fonction `int f(int)`,
- le fichier `prog.c` contient une fonction `main(void)` appelant `int f(int)` et `void g(int)`.

Exercice 3 – Rappels : pointeurs

Un pointeur est une variable contenant l'adresse d'une zone mémoire. Cette zone mémoire est interprétée comme une valeur d'un certain type, dépendant du type du pointeur. La syntaxe des pointeurs est la suivante :
- **accèder à la zone mémoire pointée par un pointeur `p` et typée d'après le type de `p` : `*p`**
- **récupérer l'adresse de la zone mémoire représentant une variable `i` : `&i`**

L'utilisation la plus fréquente des pointeurs : **création dynamique de valeurs en mémoire** .
L'erreur la plus fréquente des pointeurs : **oubli de suppression des valeurs en mémoire créées dynamiquement**.

```c
#include <stdio.h>
#include <malloc.h>

void main(void)
{
    int *p;                     /* p est un pointeur sur un entier */
    int i = 1;

    /* r'ecup'eration de l'adresse d'un int pr'e-existant */
    p = &i;
    printf("%d\n", *p);
```

```
12
13      /* cr'eation dynamique d'un entier */
14      p = (int *) malloc(sizeof(int));
15      printf("%d\n", *p);          /* ? */
16
17      /* nettoyage */
18      free(p);
19  }
```

Q 3.1 Pointeurs et appels de fonction (1)
Comment faire une fonction qui modifie la valeur de son argument ?

Q 3.2 Pointeurs et appels de fonction (2)
Quelle autre raison pour une fonction C de passer un pointeur plutôt qu'une valeur ?

Q 3.3 Pointeurs et tableau (1)
Un tableau est un pointeur sur une zone de mémoire statiquement allouée. L'arithmétique des pointeurs permet de retrouver facilement un élément à partir du pointeur initial plus un déplacement :

```
1       int t[5];
2       printf("%d",t[3]);
```

Comment définir un tel tableau dynamiquement ? Comment accéder à ses éléments ?

Q 3.4 Pointeurs et tableau (2)
Un tableau statique (`float t[5]`) correspond donc à allouer une zone mémoire de 5 `float` et à définir un pointeur `t` qui contient l'adresse de cette zone. Mais où se trouve l'information sur la taille de cette zone ? Est-ce un problème ? Quelle solution proposée ?

Exercice 4 – Tester vos connaissances sur les pointeurs

Q 4.1 Répondre aux questions posées dans le programme suivant. Indiquer quelle opération est effectuée par ce programme.

```
1   #include <stdio.h>
2   #include <malloc.h>
3
4
5   int exchange_pointeur(int *p, int *q)
6   {
7       [ Donner les valeurs de &p , &q, p et q ]
8       [ A quoi correspondent les valeurs de p et q ]
9
10      p = q;
11
12      [ Donner les valeurs &p et p ]
13  }
14
15  int main(void)
16  {
17      int *p;  /* p est un pointeur sur un entier */
18      int *q;
19      printf("adresse de p dans main = %p\n", &p); //renvoie : 0xbfaf368c
20      printf("adresse de q dans main = %p\n", &q); // renvoie : 0xbfaf3688
21
22      int i = 1;
```

```
23      int j = 2;
24      printf("adresse␣i␣=␣%p\n", &i); //renvoie : 0xbfaf3684
25      printf("adresse␣j␣=␣%p\n", &j); //renvoie : 0xbfaf3680
26
27      p = &i;
28      q = &j;
29
30      [ Donner les valeurs de p et *p ]
31      [ A quoi correspond la valeur *p ]
32
33
34      exchange_pointeur(p,q);
35
36      [ Donner les valeurs de &p, p, *p et q ]
37
38
39      return 0;
40  }
```

Q 4.2 Mêmes questions concernant le programme suivant.

```
1   #include <stdio.h>
2   #include <malloc.h>
3
4
5   int exchange_pointeur(int **ad_p, int *q)
6   {
7      [ Donner les valeurs de *ad_p, &q, **ad_p  et q ]
8      [ A quoi correspond ad_p, *ad_p et **ad_p ]
9
10     *ad_p = q;
11
12     [ Donner les valeurs de *ad_p et **ad_p ]
13  }
14
15  int main(void)
16  {
17      int *p;  /* p est un pointeur sur un entier */
18      int *q;
19      printf("adresse␣de␣p␣dans␣main␣=␣%p\n", &p); //renvoie : 0xbfbfef5c
20      printf("adresse␣de␣q␣dans␣main␣=␣%p\n", &q); // renvoie : 0xbfbfef58
21      int i = 1;
22      int j = 2;
23      printf("adresse␣i␣=␣%p\n", &i); //renvoie : 0xbfbfef54
24      printf("adresse␣j␣=␣%p\n", &j); //renvoie : 0xbfbfef50
25
26      p = &i;
27      q = &j;
28
29      [ Donner les valeurs de p et *p ]
30
31      exchange_pointeur(&p,q);
32
33      [Donner les valeurs de &p, p, *p et q ]
34
35      return 0;
36  }
```

Exercice 5 – Allocation de tableau dynamique à 2 dimensions

Une matrice se déclare statiquement ainsi : `int m[5][6];`. La syntaxe indique clairement qu'une matrice est en fait un tableau de tableaux d'entiers. Il est nécessaire de bien comprendre que

- `m` est de type "pointeur de pointeur d'entier",
- `m[1]` est de type pointeur d'entier,
- `m[1][2]` est un entier.

Q 5.1 On considère la déclaration `int **m;` définissant une variable pouvant pointer sur une matrice à deux dimensions. Indiquer comment allouer cette matrice dynamiquement ?

Q 5.2 Proposer une fonction qui permet d'allouer la variable `m` de la question précendent, comme étant une matrice de taille (n, m), n et m étant passé en argument. Cette fonction rendra `1` si elle s'est bien passé (assez de mémoire) et `0` sinon.

Exercice 6 – Matrice triangulaire

Q 6.1 Des types de matrices sont créables dynamiquement et impossible à créer statiquement. Lesquels ?

Q 6.2 Utiliser un struct pour représenter complètement une structure de matrice triangulaire.

Q 6.3 Écrire une fonction d'allocation d'une telle matrice.

Q 6.4 Écrire une fonction d'affichage d'une telle matrice.

Q 6.5 Écrire une fonction qui transpose une telle matrice.
Voilà un résultat d'exécution attendu :

```
Matrice(5,3)
    1    2    3
    2    4    6
    3    6    9
    4    8   12
    5   10   15


Matrice(3,5)
    1    2    3    4    5
    2    4    6    8   10
    3    6    9   12   15
```

TD2 - Grouper des données en mémoire: les listes

Exercice 1 – Liste doublement chaînée

Pour permettre un accès facile à l'élément fin de la liste ou pour accéder facilement à l'élément précédent un élément, on peut imaginer une liste doublement chaînée (voir figure 1). Le pointeur `premier` permet de parcourir la liste en partant du premier élément et le pointeur `dernier` en partant du dernier élément. On allouera de la mémoire au moment utile et elle sera libérée lors de la destruction de l'élément.

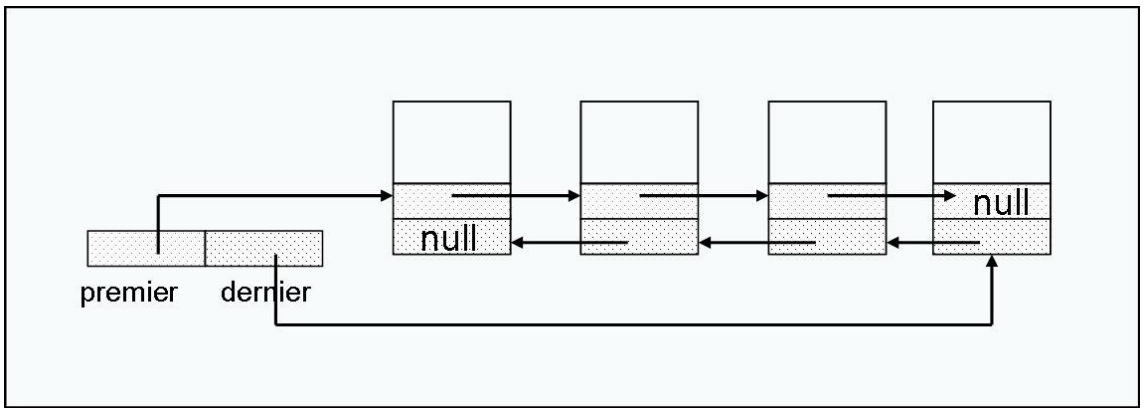

FIGURE 1 – Liste doublement chaînée

Pour programmer les différentes fonctions de base utilisées avec cette structure les valeurs stockées dans la liste seront des valeurs entières.

Remarque : les fonctions programmées ici sont indépendantes du type de la valeur rangée dans la structure de liste. Il suffirait de remplacer le `int` de la valeur par un autre type et d'avoir une fonction de comparaison entre valeurs pour disposer d'un même ensemble de fonctions sur une liste doublement chaînée.

Q 1.1 Définir la structure d'un élément de la liste

Q 1.2 Ecrire la fonction `créerElement` qui crée un élément dont la valeur sera passée en paramètre.

Q 1.3 Définir la structure d'une liste doublement chaînée

Q 1.4 Ecrire la fonction `listeVide` qui retourne un booléen vrai si la liste est vide, faux dans le cas contraire.

Q 1.5 Ecrire la fonction `insererEnTete`qui insère un nouvel élément en tête de liste

Q 1.6 Ecrire la fonction `insererEnFin`qui insère un nouvel élément en fin de liste.

Q 1.7 Ecrire la fonction `afficher` qui affiche une liste.

Q 1.8 Ecrire la fonction `rechercher` qui recherche si une valeur est présente dans la liste et retourne l'élement correspondant s'il existe, NULL s'il n'est pas trouvé.

Q 1.9 Ecrire la fonction `enleverElement` qui enlève l'élément passé en paramètre de la liste donnée.

Q 1.10 Ecrire la fonction `DetruireListe` qui libère toute la mémoire utilisée pour la liste.

Q 1.11 Ecrire le fichier listeDC.h, fichier d'en tête correspondant à cette structure

Q 1.12 Ecrire une fonction main qui créera un liste vide et affichera un menu permettant à l'utilisateur de faire toutes les opérations définies dans les questions 6 à 10.

Exercice 2 – Tableau de files

La poste a de nombreux guichets, à l'entrée du bureau de poste vous donnez votre numéro et l'opération que vous voulez réaliser. Le préposé vous enregistre au guichet correspondant. Chaque guichet correspond ainsi à une file d'attente. Ainsi, on aura pour chaque guichet une liste doublement chainée dans laquelle un nouvel arrivant est ajouté en tête et la personne qui sera appelée au guichet est en fin de liste.

Q 2.1 Quelle structure de données proposez-vous pour représenter les guichets sachant que ceux-ci sont numérotés de `0` à `n`.

Q 2.2 Ecrire la fonction `creerBureauPoste` qui crée un bureau de poste à n guichets.

Q 2.3 Ecrire la fonction afficherPoste qui affiche pour chauqe guichet la liste d'attente.

Q 2.4 Ecrire la fonction `ajouterAuGuichet` qui a pour paramètres outre le tableau de guichets, le numéro du guichet concerné et le numéro d'identification de la personne et ajoute la personne en tête de la liste concernée.

Q 2.5 Ecrire la fonction `appelAuGuichet` qui rend le numéro de la première personne appelée à ce guichet et enleve cette personne de la liste correspondante. On rajoutera pour cela à l'ensemble des fonctions sur les listes doublement chaînées une fonction `valDernier` pour récupérer la valeur du dernier élément d'une liste.

Q 2.6 Ecrire une fonction main qui crée un bureau de postes et propose à l'utilisateur un menu lui permettant d'afficher l'état du bureau de poste à un instant donné, d'inscrire des personnes aux différents guichets, et de les appeler aux guichets.

Exercice 3 – Évaluation d'expressions complètement parenthésées

On propose d'écrire un algorithme permettant d'évaluer une expression mathématique complètement parenthésée ne contenant que des opérateurs binaires. Pour ce faire, on utilisera une pile, appelée pile d'évaluation, qui contiendra des valeurs ou des opérateurs. L'algorithme consiste à parcourir de gauche à droite l'expression parenthésée et à empiler ou dépiler des éléments suivant le type de symbole rencontré. Pour chaque symbole de l'expression on exécute les instructions suivantes :

- Si le symbole est une valeur, on empile cette valeur.
- Si le symbole est une variable, on empile la valeur de cette variable.
- Si le symbole est un opérateur, on empile cet opérateur.

— Si le symbole est une parenthèse fermante, on dépile les 3 premiers éléments de la pile (un opérateur et deux valeurs) et on empile le résultat de l'opération.

La figure 2 décrit l'évolution de la pile au cours de l'évaluation de l'expression $((x+2)*3)$ avec $x=5$.

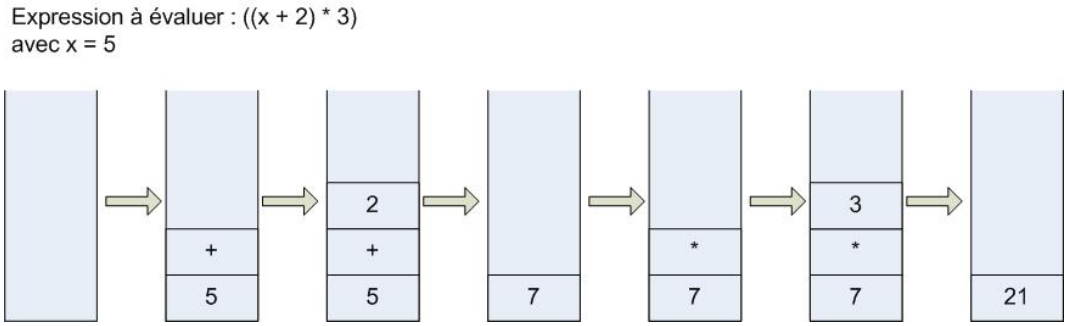

FIGURE 2 – Évolution de la pile d'évaluation

Q 3.1 Écrivez l'algorithme qui permet d'évaluer une expression complètement parenthésée.

Exercice 4 – Notation postfixée et Détection des erreurs de syntaxe

La notation postfixée (ou polonaise inversée) permet de décrire une expression mathématique sans utiliser de parenthèse. Cette notation place les opérandes avant les opérateurs.

La notation postfixe d'un expression E complètement parenthésée (ou infixe) se construit comme suit :

1. Si E est une constante ou une variable, E en notation postfixe donne E elle-même.
2. Si E est une opération binaire $E1$ *op* $E2$, alors E en notation postfixe donne $E'1$ $E'2$ *op* où $E'1$ (respectivement $E'2$) correspond à la notation postfixe de $E1$ (respectivement $E2$).
3. Si E est une expression entre parenthèses ($E1$), alors E en notation postfixe est $E'1$ $E'1$ (respectivement $E'2$) correspond à la notation postfixe de $E1$.

On se propose de transformer, une expression complètement parenthésée en notation postfixe, en ne parcourant qu'une seule fois l'expression parenthésée. Pour ce faire, on parcourt l'expression complètement parenthésée et on construit l'expression postfixe au fur et à mesure. Une pile est utilisée afin de stocker les opérateurs en attente. Pour chaque symbole de l'expression on exécute les instructions suivantes :

— Si le symbole est une valeur ou une variable, on le place en fin de l'expression postfixe courante.
— Si le symbole est un opérateur, on empile cet opérateur.
— Si le symbole est une parenthèse fermante, on dépile un opérateur que l'on place en fin de l'expression postfixe courante.

La figure 3 décrit l'évolution de la pile et de l'expression postfixe au cours du parcours de l'expression $(3*(x+2))$. Cette expression s'écrit en notation postfixe : 3 x 2 + *.

Q 4.1 Écrivez l'algorithme qui permet de transformer une expression complètement parenthésée, supposée syntaxiquement correcte, en notation postfixe.

Q 4.2 Adaptez l'algorithme avec de détecter les erreurs de syntaxe dans l'expression complètement parenthésée.

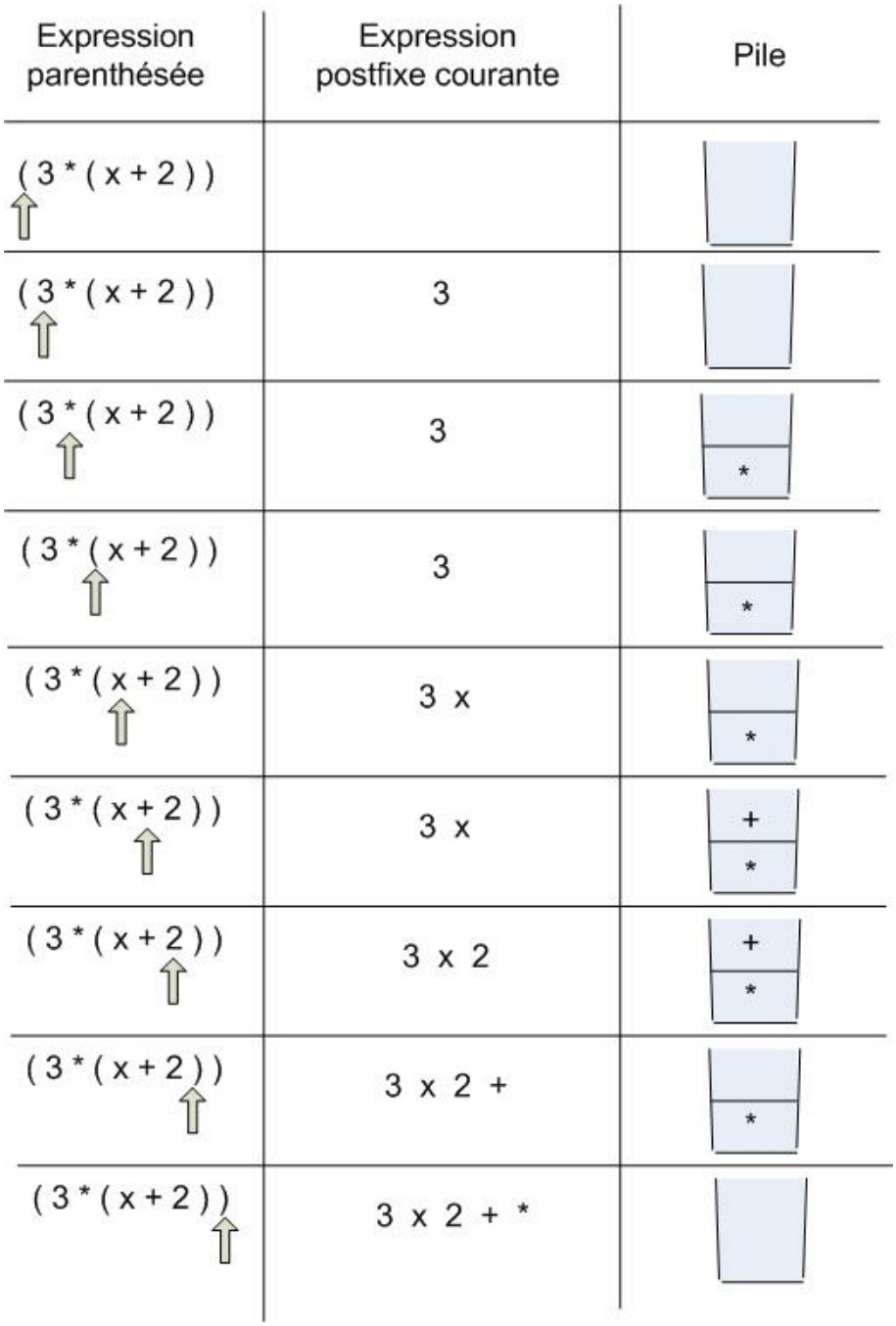

FIGURE 3 – Étapes de construction de la notation postfixe

TD 3 : Tables de hachage

Exercice 1 – Empreintes Digitales

On estime à 1/64 milliards la probabilité que deux individus possèdent les mêmes empreintes digitales. A l'échelle de la population humaine (plusieurs milliards d'individus), cette probabilité est suffisante pour assurer une identification fiable.
Une empreinte digitale est stockée sous forme d'un fichier image (JPEG) puis traitée afin de récupérer les informations nécessaires à l'identification. Ces informations sont représentées sous la forme d'une suite de 6 entiers compris entre 0 et 99 (i.e., [02, 17, 24, 04, 77, 20]).

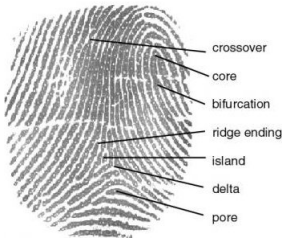

FIGURE 1 – Empreintes digitales

Les services de police disposent d'une banque de données associant une fiche personnelle (nom, prénom, âge, adresse, etc...) à une empreinte digitale.

Q 1.1 Quelle structure de données suggérez-vous pour le stockage des données en question ?

Q 1.2 La mémoire des ordinateurs dont dispose les services de polices étant limitée, la taille maximale des tableaux ne peut dépasser 100 000. Que peut-on en déduire sur la fonction de hachage à utiliser ?

Q 1.3 Soit $[x_1, x_2, x_3, x_4, x_5, x_6]$ le vecteur associé aux empreintes digitales d'un individu, que dire de la fonction de hachage suivante ?

```
1 int f1(int x1,  int x2, int x3, int x4, int x5, int x6) {
2   return (x1*x2*x3*x4*x5*x6) % 100000;
3 }
```

Comment diminuer le taux de collisions de cette fonction ?

Q 1.4 La banque de données des services de polices contient les entrées suivantes :

$$[02, 04, 10, 01, 00, 17]$$
$$[02, 10, 00, 02, 09, 05]$$
$$[00, 01, 17, 04, 02, 10]$$

Calculer la valeur renvoyée par la nouvelle fonction de hachage pour chacune de ces clés. Qu'observe-t-on ? Que peut-on proposer afin de résoudre ce problème ?

Exercice 2 – Comparaison de différents types de tables

On considère l'ensemble de 13 mots du tableau suivant. On suppose qu'il existe une fonction de hachage f qui associe à chaque mot une clé. Pour chaque mot μ, les clés $f(\mu)$ sont également données par la table suivante.

	mot μ	clés $f(\mu)$ associées (en hexadécimal)
01	"le"	FF2E
02	"cours"	178DD38
03	"de"	75EA33
04	"types"	35CE5
05	"et"	9AA8BF1
06	"structures"	2738
07	"est"	A4C74
08	"absolument"	1CA4C74
09	"génial"	14D26
10	"j'adore"	5A38
11	"faire"	1BAE5
12	"ses"	65B4EE5
13	"TD/TME"	8C74

Q 2.1 Nous allons utiliser la fonction de hachage $g(k) = k \bmod 16$ pour chaque clé k associée à un mot. Donner en base décimale, les valeurs entières $g(f(\mu))$ associée à chaque mot μ du tableau.

Q 2.2 Pour chaque type de table de hachage ci-dessous, indiquez ce que l'on obtiendrait si l'on partait d'une table vide de taille 16 et que l'on insérait dans l'ordre tous les mots ci-dessus avec la fonction de hachage $g(k) = k \bmod 16$.

1. Table de hachage avec résolution des collision par chaînage,
2. Table de hachage avec adressage ouvert et probing linéaire $h(k, i) = g(k) + i$,
3. Table de hachage avec adressage ouvert et probing quadratique $h(k, i) = g(k) + \frac{i}{2} + \frac{i^2}{2}$.

Pour les adressages ouverts, calculez le nombre de probes à effectuer en moyenne pour obtenir un élément donné dans la table. Déduisez-en le meilleur des deux probings.

Exercice 3 – fonction de hachage

On considère l'ensemble de couples (clé,valeur) suivant :

	mot	clé (en décimal)
01	“le”	123
02	“cours”	22
03	“de”	88
04	“types”	33
05	“et”	4
06	“structures”	28
07	“est”	73
08	“absolument”	7
09	“génial”	15

Q 3.1 Calculez les valeurs hachées des clés obtenues pour chaque couple par les fonctions ci-dessous :

```
1 int f1(int x) {
2   return x;
3 }
```

```
1 int f2(int x) {
2   return 10*x;
3 }
```

```
1 int f3(int x) {
2   return 2*x;
3 }
```

```
1 int f4(int x) {
2   return (x==0)?0:(8 * f4(x/10) + x % 10);
3 }
```

Q 3.2 Supposons que l'on veuille insérer les mots ci-dessus dans une table de hachage de longueur 10. On veut donc hacher les clés des mots en utilisant la fonction $f(x) = g(x) \bmod 10$, où $g \in \{\texttt{f1},\texttt{f2},\texttt{f3},\texttt{f4}\}$. Quelle est le meilleur choix pour g ? Expliquez pourquoi.

Exercice 4 – Implémentation des tables de hachage par chaînage

On considère ici les tables de hachage par chaînage. On désire étudier comment bien les implémenter.

Q 4.1 Quelle est la SDA qui va permettre de construire une telle implémentation ? Donner le code C correspondant à l'entête et l'initialisation de cette structure dans le cas où l'élément est un entier.

Q 4.2 On considère l'élement `int val` et la fonction de hachage (donnée) `int h(int val);`. Donner une fonction d'insertion de l'élément `val` dans la table de hachage.

Q 4.3 Une opération importante est de pouvoir récupérer facilement le nombre d'éléments introduits dans la table. Comment proposez-vous de réaliser cette opération ?

Q 4.4 L'opération de suppression d'un élément dans une table de hachage est une opération également importante. Quelle modification de la structure permettrait d'améliorer l'implémentation d'une telle suppression ?

TD4: File de priorité - Les tas

Exercice 1 – Rappels

Q 1.1 Donner la définition de la structure de données abstraite appelée "tas".

Q 1.2 En utilisant uniquement l'opération swap(père, fils), construire le tas obtenu par insertion successives des entiers $[10, 2, 5, 4, 7, 15, 1, 3]$, la valeur de ces entiers servant directement de clé.

Q 1.3 Combien d'éléments peut-on avoir dans un tas de hauteur h ?

Q 1.4 Étant donné n éléments à stocker dans un tas, écrire h, la hauteur du tas, en fonction de n.

Q 1.5 Quel est le coût, dans le pire des cas, d'insertion d'un élément dans un tas contenant n éléments.

Exercice 2 – Tas ternaire

Afin de réduire la hauteur de l'arbre sous-jacent, on propose d'utiliser un tas ternaire. Dans ce type de tas, chaque noeud possède au plus trois fils. La valeur de chaque noeud est inférieure à celle de ses fils.

Q 2.1 Représentez le tas ternaire construit à partir des éléments $[3, 6, 1, 13, 17, 18, 2]$.

Q 2.2 Étant donné n éléments à stocker dans un tas ternaire, écrire h, la hauteur du tas, en fonction de n. En déduire la complexité-temps des différentes opérations. Donner également la complexité-mémoire de la structure.

Q 2.3 Est-ce que l'utilisation d'un tas ternaire améliore le coût d'insertion dans le pire des cas par rapport à un tas binaire ? Dans le meilleure des cas ?

Q 2.4 En numérotant à partir de 1 les noeuds de l'arbre du haut vers le bas, niveau après niveau, et de la gauche vers la droite, quelles relations peut-on établir entre le numéro d'un noeud et ceux de ses trois fils ? Sachant que l'arbre est complet, décrire comment stocker un tas ternaire dans un tableau et écrire les fonctions nécessaires à sa gestion (s'inspirer des fonctions définies en cours).

Q 2.5 En utilisant les opérations définies précédemment, écrire une fonction qui supprime le plus petit élément d'un tas ternaire à p éléments et réorganise le reste des éléments en un tas contenant $p - 1$ éléments.
Note Afin de maintenir une structure d'arbre ternaire tassé à gauche, il faut remplacer la racine par la feuille qui se trouve à l'extrémité droite. Dérouler un petit exemple à la main.

Q 2.6 Quel est le coût, dans le pire des cas, de suppression d'un élément dans un tas ternaire contenant n éléments.

Exercice 3 – Tas contenant des éléments à deux clefs

On considère des couples (f, i) où f est un nombre réel et i est un entier compris entre 0 et K, K étant

un nombre entier connu à l'avance. Ces couples doivent être stockés dans une structure de données telle que, pour une valeur $i \in \{1, ..., K\}$, la structure contienne au plus un élément (f, i). De plus, on veut que cette structure de données possède les primitives suivantes :
- un accès en $O(1)$ au couple (f, i) ayant la plus petite valeur f.
- insertion en $O(log(n))$ d'un couple (f, i).
- suppression en $O(log(n))$ de l'élément de plus petite valeur f.
- test en $O(1)$ de la présence ou non d'un élément (f, i) pour une valeur i donnée.
- suppression, s'il existe, en $O(log(n))$, de l'élément (f, i) pour une valeur i donnée.

Q 3.1 Proposer une structure de données correspondant à cette description, ainsi que le code de ses primitives.

Q 3.2 Indiquer comment adapter ce code pour une structure de tas contenant des élements (f, c) où c est une valeur prise dans un ensemble quelconque fini $\mathcal{K}$. Indiquer la complexité des primitives de cette structure de données et discuter de la complexité moyenne.

TD5 - Structures de données non-linéaires

Exercice 1 – Une formation est un ensemble de formations

Un organisme de formation propose des cours et des formations complètes (ensemble comprenant des cours et/ou des formations complètes). Son catalogue est donc un ensemble contenant des cours simples et des formations complètes. Notre choix de représentation pour le catalogue est un tableau.

Q 1.1 Définir une structure permettant de représenter un cours ou une formation. Un cours sera modélisé par un nom (`char*`), un nombre d'heures (`int`), un prix (`double`).

Q 1.2 La formation Z, contient un cours $C1$ et une formation Y. Cette formation Y contient elle-même une formation X et un cours $C2$. Mais à la création du catalogue, il y a eu une erreur de frappe et la formation Y a été déclarée comme contenant la formation Z. Que se passe-t-il lors de l'affichage des formations ?

Q 1.3 Lors de la création du catalogue des formations, on veut pouvoir éviter les erreurs de frappe de la questions précédentes. Implémenter un catalogue et la fonction associée `ajouter un cours ou une formation`.

Q 1.4 Une autre erreur de frappe possible est qu'une formation contienne deux fois le même cours. Corrigez la fonction précédente pour que cette erreur ne se produise pas.

Q 1.5 On veut calculer le nombre d'heures de cours proposées dans le catalogue. Ecrire un programme qui calcule le nombre d'heures.

Exercice 2 – Maison des associations

On veut représenter les différentes associations présentes dans un quartier. La composition d'une association évoluant souvent (ajout ou retrait de membres), on choisit de représenter une association par un nom et une liste de membres. Chaque personne est représentée par son nom ou par son numéro d'adhérent.

De nouvelles associations voient le jour, d'autres disparaissent, ce qui nous conduit à choisir de représenter la maison des associations par une liste d'associations.
On aura une structure ressemblant à celle donnée par la figure 2.

Q 2.1 Représentation d'une association

a) Ecrire la structure correspondant à un Element de la liste des personnes dans une association (`Element` de la figure 2).
b) Ecrire la structure correspondant à une Association (`Association` de la figure 2).
c) Définir la fonction `creerAssociation` qui crée une association en créant son nom et une liste de membres vide.

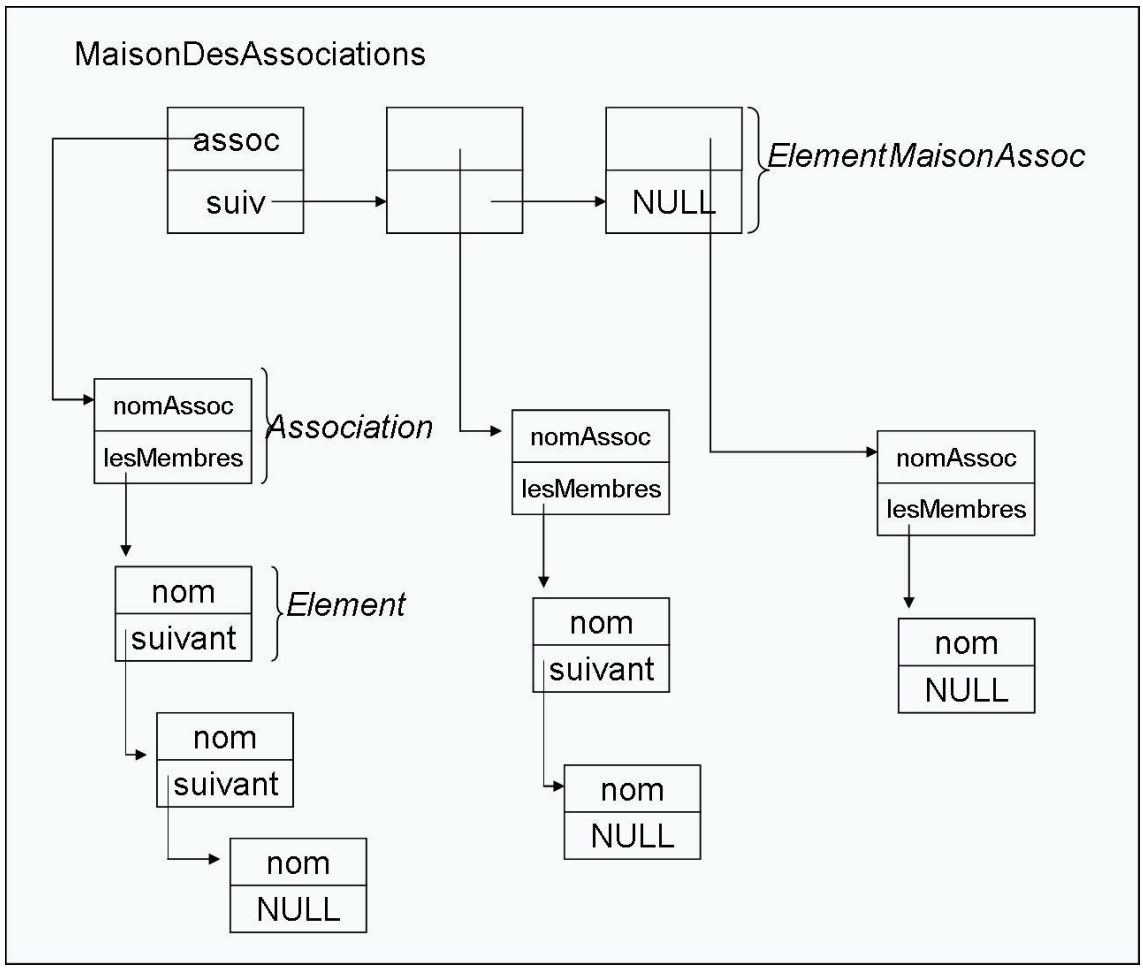

FIGURE 1 – Représentation de la maison des associations

d) Définir la fonction `ajoutPersonne` qui permet d'ajouter en-tête de la liste une personne à la liste des membres.
e) Définir la fonction `trouverPersonne` qui trouve le nom dans la liste et retourne un pointeur sur l'élément précédent cette personne dans la liste (NULL si la personne n'appartient pas à la liste).
f) Définir la fonction `enlever` qui enlève une personne de la liste.

Q 2.2 Représentation d'une maison des associations

a) Ecrire la structure correspondant à un élément de la maison des associations (`ElementMaisonAssoc` de la figure 2)
b) Définir la fonction `creerMaisonAssoc` qui crée une maison vide.
c) Définir la fonction `ajouterAssociation` qui ajoute une association en tête de la liste.
d) Définir la fonction `rechercherAssociation` qui rend l'association précédant l'association recherchée dans la liste, NULL si elle n'existe pas.
e) Définir la fonction `enleverAssociation`.

Q 2.3 Ecrire une fonction main qui crée une maison des associations, et propose un menu pour créer des associations et les ajouter à la maison, rechercher une association, ajouter une personne à une association, enlever une personne d'une association et enlever une association.

Exercice 3 – Maison des associations : gestion acyclique

Nous allons reprendre l'exercice précédent correspondant à la maison des associations en ajoutant la notion suivante : chaque association est composée de membres qui peuvent être soit des personnes physiques, soit des personnes morales. Une association peut donc elle-même être membre d'une autre association.

Q 3.1 Définir la structure *membreAssoc* qui permet de représenter soit une personne physique caractérisée par son nom (char*) et son adresse (char*), soit une personne morale qui aura, bien sûr, un nom, une adresse et une liste de membres. Prévoir de différencier une personne morale d'une personne physique. Donner le schéma UML correspondant.

Q 3.2 Implémenter une `association`, ensemble de `membreAssoc` en utilisant une liste. Créer les fonctions `ajouter` et `rechercher`.

Q 3.3 On veut faire un courrier à tous les membres physiques et moraux d'une association A en considérant que tous les membres d'une association adhérente à A sont membres de A. Si l'on envoie un courrier à toutes les personnes qui sont pointées dans la liste, que peut-il se passer ? Donner des exemples d'instances dans les différents cas possibles.

Q 3.4 Comment peut-on remédier aux problèmes rencontrés dans la question précédente ? Ecrire la fonction `envoiCourrier` correspondante

Q 3.5 Implémenter la fonction `supprimer un membre`.

TD6 - Tris

Soit $L = (e_1, e_2, e_3, \ldots, e_n)$ une liste d'éléments. A chaque élément on associe une clé telle que les clés soient comparables entre elles. Les éléments seront triés selon leurs clés. Attention, souvent, dans les algorithmes décrits ici, clés et éléments sont parfois confondus.

Exercice 1 – Tri par insertion dichotomique

Dans cette méthode de tri, on ajoute de nouveaux éléments à un tableau déjà trié T. La recherche de la place de l'élément à ajouter est faite par une méthode dichotomique. Si x est l'élément à insérer, on le compare à $T[M]$ où M est l'élément milieu du tableau trié, si $x < T[M]$ on recommence la recherche dans la partie gauche du tableau, si $x > T[M]$ on recommence dans la partie droite. La recherche s'arrête lorsque la partie du tableau dans laquelle on recherche est vide (indice min=indice max).

Q 1.1 Programmer de façon récursive, la fonction rang qui à partir du tableau T trouve la place de l'élément x.

Q 1.2 Utiliser cette fonction pour créer une tableau trié en insérant les divers éléments au fur et à mesure.

Exercice 2 – Tri rapide

Rappel du principe du tri rapide :
Le tri rapide consiste à faire une partition dans le tableau à trier, par rapport à une valeur du tableau nommée `clé` ou `valeur pivot`. On séparera le tableau en deux, une partie du tableau contenant les éléments qui sont inférieurs à la clé, et l'autre ceux qui sont supérieurs à la clé. Après partitionnement le `pivot` est positionné à sa place définitive, on refera le même travail sur chacun des sous-tableaux nouvellement créés jusqu'à ce que tous les tableaux soient triés.

Q 2.1 Appliquer la méthode de partitionnement sur le tableau tab suivant en prenant comme pivot le premier élément : { 3, 1, 5, 2, 4}

Q 2.2 Ecrire la fonction partition qui à partir d'un tableau d'entier tab, de la valeur de l'indice inférieur de ce tableau imin et de l'indice supérieur de ce tableau imax, rend le tableau tab modifié et l'indice de la valeur pivot.

Q 2.3 En utilisant cette fonction, écrire la fonction `triRapide` qui trie un tableau d'entiers `tab` de taille `nb`.

Q 2.4 Définir les fonctions qu'il faut implémenter sur un `objet` pour que les fonctions `partition` et `triRapide` puissent être exécutées sur un tableau dont les éléments sont de type `objet`.

Q 2.5 Dans le cas le plus défavorable quel sera en fonction de la taille du tableau le nombre d'échanges réalisés ?

Q 2.6 En admettant qu'il est possible de trouver la médiane d'un tableau de n entiers en $O(n)$, pouvez-vous améliorer la complexité au pire des cas du tri rapide (on pourra supposer pour simplifier

que tous les éléments sont différents) ?

Exercice 3 – Tri par tas

On considère un tableau T de n éléments à trier. On va utiliser une structure de tas tassé à gauche pour construire un algorithme de tri. Pour cela, on définit un tas 'Max' h de taille maximale n.

Q 3.1 Considérons le tableau $T = [2, 5, 23, 13, 10]$. On insère dans h les éléments de T un à un. Donner le tas correspondant.

Q 3.2 Utiliser h pour trier le tableau T par ordre décroissant.

Q 3.3 Quelle complexité obtient-on pour l'opération de tri ainsi obtenu ?

Q 3.4 Peut-on utiliser ce tri pour trier une liste ?

Exercice 4 – Tri par dénombrement (ou tri postal)

Ce tri ne peut s'appliquer que lorsque les différentes valeurs à trier sont en nombre fini et connues à l'avance. Dans ce tri, on ne compare pas les éléments à trier entre eux. On définit un nouveau tableau d'entiers qui va nous servir comme ensemble de cases de tri (comme les bacs du tri postal). Lorsqu'on rencontre un nouvel élément à trier on ajoutera 1 dans la case correspondante. Puis on relira le tableau des cases de tri et on créera un nouveau tableau en vidant les cases.

Exemple : Tableau à trier {5,4,6,9,1,5,8,9,1,9}, les valeurs du tableau sont dans l'intervalle [1 .. 9]. On crée un tableau de 9 cases et on obtient :

2			1	2	1		1	3
1	2	3	4	5	6	7	8	9

ce qui permet de trier le tableau initial en relisant le tableau intermédiaire : {1,1,4,5,5,6,8,9,9,9}.

Q 4.1 Ecrire la fonction `tri` à trois paramètres, le tableau à trier `tab`, et les deux bornes de l'intervalle dans lequel sont définies les valeurs du tableau.

Q 4.2 Donner la complexité-temps et complexité-mémoire d'une telle fonction. Est-ce que cette complexité peut-être atteinte pour un tri de valeurs quelconques ?

Q 4.3 On considère à présent un tableau de n réels dont chaque valeurs est prise parmi seulement K valeurs différentes. Comment adapter le tri par dénombrement pour ces valeurs ? Conserve-t-on la même complexité ?

Exercice 5 – Le tri comme primitive

Q 5.1 On considère un tableau de n entiers, ainsi qu'un entier B. Le but est de savoir s'il existe deux entiers dans le tableau dont la somme égale B.

1. Proposez une méthode simple - sans utiliser de tri - pour résoudre ce problème. Quelle est la complexité de cette méthode ?
2. Proposez une méthode en $O(n \log n)$ pour résoudre le problème.

Q 5.2 On considère n appartements $(a_1, a_2, \ldots, a_n)$ évalués sur deux critères : le loyer mensuel ℓ_i et le temps de trajet pour venir à l'UPMC t_i. On supposera pour simplifier que les loyers sont tous différents, ainsi que les temps de trajet. On dit qu'un appartement a_i domine un appartement a_j si $\ell_i < \ell_j$ et $t_i < t_j$ (a_i est meilleur sur les deux critères). Comment (et avec quelle complexité) répondriez-vous aux questions suivantes :

1. Déterminer si l'appartement a_1 est dominé (par un autre appartement) ou pas.
2. Trouver un appartement qui n'est dominé par aucun autre.
3. Déterminer s'il existe un appartement qui est dominé par un autre.

Q 5.3 Vous avez n tâches $t_1, \cdots, t_n$ à exécuter séquentiellement sur un processeur ; chaque tâche t_i a une durée $p_i \geq 0$. Vous voulez minimiser la date de fin moyenne des tâches. Le but est de déterminer dans quel ordre les tâches doivent être exécutées par le processeur.

1. On considère 3 tâches de durée respective $p_1 = 8$, $p_2 = 2$ et $p_3 = 5$. Quelle est la date de fin moyenne si on exécute ces tâches dans cet ordre ? Pouvez-vous proposer une solution meilleure ?
2. De manière générale, comment doit-on procéder ? Quelle est sa complexité de votre algorithme ? (question subsidiaire : pouvez-vous prouver que votre algorithme donne effectivement la meilleure solution ?)

TME1 : Outil de débogue, tableaux et complexité

Les deux exercices de ce premier TME permettent de découvrir deux techniques qui seront utiles pour l'ensemble des TME. Le premier exercice traite d'une technique de débogue. Le deuxième exercice est l'occasion d'étudier une façon d'évaluer la vitesse d'un programme et de comparer les vitesses de plusieurs programmes. Les outils fournis pour ces deux exercices ne sont pas obligatoires et peuvent être remplacés par des outils équivalents.
Les fichiers nécessaires à ce TME peuvent être récupérés sur le site web de l'UE :
`https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2014/ue/2I006-2015fev/`

Exercice 1 – Utilisation d'un outil de débogue

La bonne compilation d'un programme écrit en C n'assure malheureusement pas son bon fonctionnement. Il existe souvent des erreurs qui ne sont révélées qu'au moment de l'exécution et qu'il faut trouver. Parmi ces erreurs, la plus courante est l'« erreur de segmentation » qui survient souvent lors de la manipulation de tableaux, ou de pointeurs. L'exercice suivant vous présente l'utilisation d'un outil de débogue qui vous permettra la plupart du temps de localiser vos erreurs.

Partie 1 : *indice de boucle*
Récupérez le fichier C nommé `ddd_exo1.c` :

```c
#include<stdio.h>
#include<stdlib.h>

const static int len = 10;

int main(void) {
    int *tab;
    unsigned int i;

    tab = (int*)malloc(len*sizeof(int));

    for (i=len-1; i>=0; i--) {
        tab[i] = i;
    }

    free(tab);
    return 0;
}
```

Q 1.1
À la lecture du code, qu'est censé faire le programme d'après vous ? Compilez le avec la commande suivante `gcc -o ddd_exo1 ddd_exo1.c` et lancer le. Que se passe-t-il ?

Q 1.2
Pour trouver l'erreur dans le programme, nous allons utiliser DDD, un outil de débogue graphique. Recompilez votre programme avec la commande suivante : `gcc -ggdb -o ddd_exo1 ddd_exo1.c`. L'option `-ggdb` permet d'inclure dans l'exécutable des informations supplémentaires qui facilite le débogue.

Lancez l'outil `ddd` sur l'exécutable de votre programme dans un terminal : `ddd ddd_exo1`. Pour trouver l'erreur du programme, vous allez l'exécuter pas-à-pas et vous allez "donner la trace" (ou "tracer") de la variable de boucle `i`, c'est-à-dire examiner l'évolution de sa valeur dans le temps.
Pour cela, commencer par définir un point d'arrêt au niveau de l'instruction qui se trouve dans la boucle d'exécution (`tab[i] = i;`). Avec la souris, placer le curseur à la fin de l'instruction en question, dans le code C affiché à l'écran, et cliquez droit : choisissez l'option *Set Breakpoint*. Ensuite lancez l'exécution en cliquant sur le bouton *Run* dans le menu *Program*. Vous devriez voir une flèche verte qui s'affiche à l'endroit où le point d'arrêt a été défini. Cela signifie que le programme a été exécuté jusqu'au point d'arrêt et qu'il est maintenant arrêté. Pour tracer la variable `i`, cliquez dessus dans le code C affiché à l'écran pour sélectionner la variable, et cliquez droit : choisissez l'option *Display i*. Pour suivre l'évolution de la variable, il vous faut activer l'écran d'affichage des données : activez le bouton *Data Window* dans le menu *View*.
Vous pouvez maintenant exécuter le programme, c'est-à-dire instruction par instruction. Pour cela, cliquez sur le bouton *Step* dans le menu *Program* (ou sur la touche `F5` du clavier). À chaque appui, une instruction est exécutée. Faites alors itérer la boucle jusqu'au bout et suivez l'évolution de la valeur de `i` dans l'écran de données.
Après l'itération dans laquelle `i` valait 0, que vaut la valeur de `i` ? Que devrait valoir `i` pour sortir de la boucle ? À quelle case du tableau essaie-t-on d'accéder ? Déduisez-en le sens de l'« erreur de segmentation ».

Q 1.3
Sachant que le mot clé `unsigned` force un type C standard (`char, short, int`) à ne jamais être négatif (c'est-à-dire « non signé »), que proposez-vous pour résoudre cette erreur ?

Partie 2 : *Déférencement de pointeur*
Récupérez le fichier C nommé `ddd_exo2.c` :

```c
#include<stdio.h>
#include<stdlib.h>

struct mystruct {
    int value;
    struct mystruct *next;
};

struct mystruct* insert_list (struct mystruct *list, int value) {
    struct mystruct *new;
    new = malloc(sizeof(struct mystruct));

    new->value = value;
    new->next = list;

    return new;
}

int main(void) {
    struct mystruct *head;

    head = insert_list(NULL, 5);
    head = insert_list(head, 3);
    head = insert_list(head, 6);
    head = insert_list(head, 10);
    head = insert_list(head, 17);
```

```
27
28      struct mystruct *scan = head;
29      do {
30          printf("value=%d\n", scan->value);
31          scan = scan->next;
32      } while(1);
33
34      return 0;
35  }
```

Q 1.4
À la lecture du code, qu'est censé faire le programme d'après vous ? Compilez le programme et lancez le. Que se passe-t-il ?

Q 1.5
Recompilez le programme en activant l'option de débogue et lancez l'outil `ddd` sur l'exécutable résultant. Tracez la variable `scan` et exécuter la boucle d'affichage pas-à-pas. Que vaut la variable `scan` au moment où survient l'erreur de segmentation ? Expliquez la cause de l'erreur.

Q 1.6
Proposez une autre condition de boucle (à la place du `while(1)`) pour la boucle d'affichage afin que cette dernière s'arrête correctement.

Exercice 2 – Algorithmique et tableaux

Partie 1 : Tableaux à une dimension

Q 2.1
Créez les fonctions suivantes :

1. une fonction `alloue_tableau` permettant d'allouer la mémoire utilisée par un tableau T d'entiers de taille n. Noter qu'il existe deux versions possibles pour cette fonction :

   `int * alloue_tableau(int n);` et `void alloue_tableau(int **T, int n);`
2. une fonction `remplir_tableau` permettant de remplir le tableau avec des valeurs entières aléatoirement générées entre 0 et V (non-compris).
3. une fonction `afficher_tableau` permettant d'afficher les valeurs du tableau.

Q 2.2
Ecrivez un algorithme permettant de calculer la somme des carrés des différences entre les éléments du tableau pris deux à deux :

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\Big(T(i)-T(j)\Big)^2$$

1. Ecrire un premier algorithme de complexité $O(n^2)$
2. Ecrire un second algorithme de meilleure complexité. Aidez-vous du fait que
   $\sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j = \Big(\sum_{i=1}^{n} x_i\Big)^2$

Q 2.3 Comparez les temps de calcul des deux algorithmes. Pour mesurer le temps mis par le CPU pour effectuer une fonction nommée `fct`, on peut utiliser le code suivant où `temps_cpu` contient le temps CPU utilisé en secondes.

```
#include <time.h>
...
clock_t temps_initial; /* Temps initial en micro-secondes */
clock_t temps_final;   /* Temps final en micro-secondes */
double temps_cpu;     /* Temps total en secondes */
...
temps_initial = clock();
  fct();
temps_final = clock ();
temps_cpu =  ((double)(temps_final - temps_initial))/CLOCKS_PER_SEC;
printf("%d %f ",n,temps_cpu);
```

Faites varier la taille n du tableau et analysez les temps de calcul obtenus.

Q 2.4
On veut visualiser par des courbes les séries de nombres obtenues. On peut utiliser pour cela un logiciel extérieur lisant un fichier texte créé par le programme. Par exemple, le logiciel `gnuplot` qui est utilisable en ligne sous linux. Le fichier d'entrée de `gnuplot` est simplement la donnée en lignes de `n` suivi des mesures de temps.
On peut alors lancer `gnuplot` et taper interactivement les commandes. On peut également utiliser une redirection `gnuplopt < commande.txt` avec un fichier du type :

```
plot "01_sortie_vitesse.txt" using 1:2 with lines
replot "02_sortie_vitesse.txt" using 1:3 with lines
set term postscript portrait
set output "01_courbes_vitesse.ps"
set size 0.7, 0.7
replot
```

Ces lignes de commande créent sur le disque le fichier postscript contenant deux courbes sur un même graphique (les lignes précédées par # dans gnuplot sont en commentaires).

Q 2.5 Justifiez les courbes obtenues en fonction de la complexité pire-cas attendue.

Partie 2 : Tableaux à deux dimensions (matrices)

Q 2.6
Créez les fonctions suivantes :

1. une fonction `alloue_matrice` permettant d'allouer la mémoire utilisée par une matrice entière carrée de taille m x m.
2. une fonction `remplir_matrice` permettant de remplir la matrice avec des valeurs entières aléatoirement générées entre 0 et V (non-compris).
3. une fonction `afficher_matrice` permettant d'afficher les valeurs de la matrice.

Q 2.7 Ecrivez un algorithme permettant de vérifier que tous les éléments de la matrice ont des valeurs différentes.

1. Ecrire un premier algorithme de complexité $O(n^2)$ avec n égal au nombre d'éléments de la matrice.
2. Ecrire un second algorithme de meilleure complexité dans le cas où la borne maximale V sur les valeurs contenues dans la matrice est telle qu'un tableau de taille V puisse être alloué en mémoire.

Q 2.8 Comparez les temps de calcul des deux algorithmes et représentez les courbes obtenues en fonction du nombre n d'éléments de la matrice.

Q 2.9 Justifiez les courbes obtenues en fonction de la complexité pire-cas attendue.

Mini-Projet: Gestion d'une bibliothèque

Ce mini-projet s'étale sur deux semaines (séances 2 et 3 de TME). Il est noté et doit être rendu par mail à vos chargés de TD/TME lors du TME numéro 4. A cette occasion, vous lui ferez une démonstration rapide de votre code.

L'objectif de ce mini-projet est d'apprendre à comparer des structures de données. Nous utiliserons dans ce projet les listes chaînées de struct et les tables de hachage.

L'ensemble des fichiers nécessaires à ce mini-projet peuvent être récupérés sur le site web de l'UE : `https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2014/ue/2I006-2015fev/`

Exercice 1 – Gestion d'une bibliothèque avec une liste chaînée de struct

Dans ce premier exercice, nous allons coder une bibliothèque comme une liste chaînée de struct de type livre.
Un livre est repéré par son titre, le nom de son auteur et un numéro d'enregistrement. Plus précisément, nous manipulerons le type `struct` suivant :

```
1 typedef struct livre{
2   char *titre;
3   char *auteur;
4   int num;
5 }s_livre;
```

Dans ce projet, nous appelons bibliothèque un ensemble de livres.

Remarque : il est fortement recommandé de construire votre exercice en codant dès le départ un "menu" permettant à l'utilisateur d'utiliser les différentes fonctions du programme.

Q 1.1 Récupérez le fichier GdeBiblio.txt. Créez une fonction permettant de lire n entrées de ce fichier et de les afficher à l'écran.
Pour lire facilement les champs du fichier, vous pouvez utiliser les fichiers `entree_sortie.h` et `entree_sortie.c`. Le module `entree_sortie` permet de manipuler facilement des textes en langage C. Vous pouvez lire les commentaires décrivant les fonctions dans le fichier `entree_sortie.h`.

Q 1.2 Transformez la fonction de la question 1 pour construire une fonction permettant de lire n entrées de ce fichier et de les stocker dans la structure de données (l'insertion dans la liste se fera en insérant un livre en tête de liste).

Q 1.3 Créez les fonctions suivantes permettant
- la recherche d'un ouvrage par son numéro
- la recherche d'un ouvrage par son titre
- la recherche de tous les livres d'un même auteur
- l'insertion d'un nouvel ouvrage
- la suppression d'un ouvrage

— la recherche des ouvrages au moins en double. Deux ouvrages sont identiques s'ils ont le même auteur et le même titre. Cette fonction devra renvoyer une liste comprenant les ouvrages qui sont au moins en double.

Exercice 2 – Gestion d'une bibliothèque avec une table de hachage

Bien que les listes soient très souples (allocation et libération dynamiques de la mémoire pour l'ajout et la suppression de données), si on cherche à récupérer un élément précis de la liste, dans le pire des cas, il faudra parcourir la liste en entier jusqu'à ce qu'on le trouve.
Dans cette deuxième partie, pour accélérer l'accès à un élément de la bibliothèque, nous allons utiliser une table de hachage. La résolution des collisions se fera par chaînage.

Pour implémenter votre table de hachage, vous pourrez utiliser les struct suivantes (par exemple) :

```
1  typedef struct cell{
2    int clef;
3    s_livre *data;
4    struct cell *suivant;
5  }cell_t;
6
7  typedef struct{
8    int nE; /*nombre d'elements contenus dans la table de hachage */
9    int m; /*taille de la table de hachage */
10   cell_t **T; /*table de hachage avec resolution des collisions par chainage */
11 } tableHachage_t;
```

Q 2.1 Créez une fonction `tableHachage_t* initTableHachage(int m);` permettant d'allouer l'espace mémoire nécessaire à votre table de hachage de taille m.

Q 2.2 Fonction clef : La fonction "clef" de la table de hachage à implémenter doit permettre d'associer une valeur numérique au contenu présent dans la table de hachage. Le contenu à stocker est le struct de type `s_livre`, c'est-à-dire un titre, un nom d'auteur et un numéro de livre. Comme fonction clef, vous pouvez utiliser les caractères contenus dans le nom de l'auteur et utiliser la somme des valeurs ASCII de chaque lettre du nom.
Créez la fonction `int fonctionClef(char *nom);` réalisant cette opération.

Q 2.3 Fonction de hachage : Il est ensuite nécessaire de transformer la clef obtenue en une valeur entière utilisable par la table de hachage (c'est-à-dire entre 0 et m non compris) et permettant d'éviter au maximum les collisions.
Vous pourrez par exemple utiliser la fonction de hachage $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$ pour toute clef k, où $A = \frac{\sqrt{5}-1}{2}$ (nombre d'or diminué de 1, proposé par Donald Knuth [1]). Vous testerez expérimentalement plusieurs valeurs de m afin de déterminer la valeur la plus appropriée.
Il est également possible d'utiliser des fonctions de hachage bien différentes : vous pouvez proposer d'autres fonctions.
Créez la fonction `int fonctionHachage(int clef, int m);` réalisant cette opération.

1. Né le 10 janvier 1938 dans l'état du Wisconsin aux États-Unis, Donald Knuth est un informaticien et mathématicien américain de renom et professeur émérite en informatique à l'université Stanford (États-Unis). Il est un des pionniers de l'algorithmique et a fait de nombreuses contributions dans plusieurs branches de l'informatique théorique. Il est également l'auteur de l'éditeur de texte LaTeX.

Q 2.4 Adaptez les questions 1.2 et 1.3 à l'utilisation d'une table de hachage pour gérer la bibliothèque. Pour insérer un ouvrage dans une bibliothèque, vous devrez procéder en trois étapes :

1. Identifier la clé correspondant à l'ouvrage (grâce à son auteur).
2. Trouver dans quelle case de la table de hachage insérer l'ouvrage (grâce à la fonction de hachage).
3. Insérer l'ouvrage dans la liste chaînée correspondant à cette case.

Exercice 3 – Comparaison des deux structures

On veut comparer les deux structures (liste et table de hachage) par rapport au temps nécessaire pour effectuer les fonctions de recherche.

Q 3.1 Comparer les temps de calcul entre les deux structures pour réaliser la recherche d'un ouvrage par son numéro, son titre et son auteur.
Pour avoir des temps de calcul significatifs, vous pourrez procéder à plusieurs recherches consécutives. Quelle structure (liste ou table de hachage) est la plus appropriée pour chacune de ces recherches ?

Q 3.2 Modifiez la taille de votre table de hachage. Comment évoluent vos temps de calcul en fonction de cette taille ?

Q 3.3
On veut déterminer les temps de recherche des ouvrages au moins en double en fonction de la taille de la bibliothèque et de la structure de données utilisée. Pour cela, vous adapterez votre fonction pour qu'elle puisse relancer la fonction de lecture en lisant partiellement les n premières lignes du fichier, n prenant les valeurs de 1000 à 50 000 [2] avec un pas croissant.
Pour chaque valeur de n vous sauvegarderez les temps de calcul obtenus avec chacune des deux structures. Vous visualiserez ensuite par des courbes les séries de nombres obtenues.

Q 3.4 Justifiez les courbes obtenues en fonction de la complexité pire-cas attendue.

2. Si sur votre machine le temps de calcul devient trop long pour 50 000 entrées (c'est-à-dire supérieur à 5 minutes), vous pouvez diminuer le nombre maximal d'entrées.