Devoir d'Informatique n°3

27 janvier 2024

*
* *

Durée de l'épreuve : 3 heures

L'usage de tout dispositif électronique est interdit.

Consignes

Pour répondre à une question, il vous est permis de réutiliser le résultat d'une question antérieure même si vous n'avez pas réussi à établir ce résultat.

Quand l'énoncé demande de coder une fonction, sauf demande explicite, il n'est pas nécessaire de justifier la correction ou la terminaison de cette fonction, ou de la commenter.

Vous attacherez la plus grande importance à la clarté, à la précision et à la concision de la rédaction.

Si vous repérez ce qu'il vous semble être une erreur d'énoncé, signalez-le sur votre copie et poursuivez la composition en expliquant les éventuelles initiatives que vous aurez pris.

Vous devrez traiter les questions de programmation dans le langage C. On suppose que les librairies `<stdio.h>`, `<stdlib.h>`, `<stdbool.h>`, `<assert.h>` *sont toujours incluses dans le code que vous écrirez.*

Présentation du sujet

L'épreuve est composée de quatre problèmes indépendants. Le premier problème porte sur l'étude d'un algorithme mystère. Le deuxième problème porte sur l'écriture d'un interpréteur d'expressions arithmétiques. Le troisième problème porte sur les listes chaînées et sur la présentation d'un algorithme de détection de cycles. Le quatrième problème porte sur la représentation binaire gauche des entiers.

Algorithme mystère

On considère l'algorithme suivant :

Algorithme 1 : Mystère

Entrée(s) : T tableau de taille n
Sortie(s) : ...

```
1  i ← 0;
   // Invariant extérieur: ...
2  tant que i < n faire
3      a ← i;
4      b ← i + 1;
       // Invariant intérieur: ...
5      tant que b < n faire
6          si T[b] < T[a] alors
7              a ← b;
8          b ← b + 1;
       // En sortie de la boucle intérieure: ...
9      x ← T[i];
10     T[i] ← T[a];
11     T[a] ← x;
12     i ← i + 1;
   // En sortie de la boucle extérieure: ...
```

Q1. Exécuter l'algorithme sur le tableau $T = [2, 3, 1, 4, 5, 4]$. Noter l'état de T et la valeur de i à chaque passage sur la condition de la boucle extérieure (ligne 2).

Q2. On considère que l'unique opération élémentaire est la comparaison d'éléments avec $<$. Donner la complexité de cet algorithme sous la forme d'un Θ, en justifiant.

Q3. Proposer une spécification pour cet algorithme.

Q4. Proposer (sans la démontrer) une propriété sur a, T, i et n qui est vérifiée en sortie de la boucle intérieure.

Q5. Proposer un invariant de la boucle intérieure, démontrer qu'il est vrai en entrée de cette boucle, qu'il se conserve, et qu'il permet de montrer la propriété de la question précédente.

Q6. Montrer la correction de l'algorithme mystère.

Q7. Quels algorithmes efficaces connaissez-vous qui ont la même fonction que l'algorithme mystère ? Quelles sont leurs complexités ?

Interpréteur d'expression

On s'intéresse à la lecture d'expressions arithmétiques en notation infixe. On considère des expressions pouvant contenir des nombres entiers, les opérateurs binaires $+, \times$, ainsi que des parenthèses et des crochets. Les parenthèses et les crochets jouent le même rôle, mais on ne peut pas fermer une parenthèse par un crochet et inversement. Dans le sujet, "**parenthèses**" désignera à la fois les parenthèses et les crochets.

On représentera les expressions par des tableaux de symboles. Par exemple, l'expression `(32+7)*[2+4]` sera représentée par le tableau de taille 12 suivant :

$$T_0 = [\texttt{(}, 3, 2, \texttt{+}, 7, \texttt{)}, \times, \texttt{[}, 2, \texttt{+}, 4, \texttt{]}]$$

On souhaite écrire un interpréteur capable de calculer la valeur associée à une expression arithmétique. Par exemple, sur le tableau précédent, l'interpréteur répondra que l'expression s'évalue en $39 \times 6 = 234$.

Lecture des entiers

On s'intéresse tout d'abord à la lecture d'entiers dans un tableau de symboles correspondant à une expression arithmétique. Pour T un tableau de taille n et $i \leq j$ deux indices du tableau tels que $T[i], \dots T[j]$ sont tous des chiffres, on note $\overline{T[i..j]}$ l'entier obtenu en lisant les cases $T[i], \dots, T[j]$, avec le chiffre des unités à droite.

Par exemple, pour le tableau T_0 précédent, $\overline{T_0[1..2]}$ vaut 32, et $\overline{T_0[4..4]}$ vaut 7. $\overline{T_0[4..5]}$ n'est pas défini car $T[5]$ ne contient pas un chiffre.

Q8. Donner une expression de $\overline{T[i..j]}$ sous la forme d'une somme.

Q9. Écrire en pseudo-code un algorithme correspondant à la spécification suivante :

Algorithme 2 : LectureEntier

Entrée(s) : T un tableau de symboles de taille n, $i, j \in [\![0, n-1]\!]$ avec $i \leq j$ tels que $T[i], \dots, T[j]$ sont des chiffres

Sortie(s) : $\overline{T[i..j]}$

Q10. Si ce n'est pas déjà le cas, modifier l'algorithme pour qu'il s'exécute en $\mathcal{O}(j-i)$.

Q11. Montrez la correction de l'algorithme que vous avez écrit.

Bon parenthésage

Avant d'interpréter l'expression, on vérifie qu'elle est correctement parenthésée. Pour cela, on utilise une structure de données **First In First Out** : la pile. Pour vérifier si une expression est bien parenthésée, on la lit de gauche à droite, en gardant dans une pile les parenthèses ouvrantes n'ayant pas encore été fermées. On utilise cette pile pour traiter les parenthèses fermantes que l'on lit dans T.

L'objectif de cette partie est donc d'écrire en pseudo-code un algorithme implémentant le principe décrit ci dessus :

Algorithme 3 : BonParenthésage

Entrée(s) : T un tableau de symboles de taille n

Sortie(s) : Booléen : True si T représente une expression bien parenthésée, False sinon

Q12. Donner un jeu de test pertinent pour cet algorithme, en donnant pour chaque entrée à tester la valeur attendue.

En particulier, on veillera à couvrir les deux valeurs de retour, ainsi que les différents cas pouvant faire qu'un mot n'est pas bien parenthésé.

Q13. Écrire en pseudo-code l'algorithme **BonParenthésage** et donner sa complexité en fonction de n.

Interprétation des expressions

Le schéma général que nous allons suivre pour interpréteur l'expression stockée dans un tableau T est le suivant :

1. Déterminer s'il y a un opérateur à l'extérieur des parenthèses, noter i sa position dans T.
2. S'il n'y a aucun opérateur à l'extérieur des parenthèses :
   a) si T contient uniquement des chiffres, lire T avec **LectureEntier**,
   b) sinon, alors $T[0]$ et $T[n-1]$ forment un couple de parenthèses, auquel cas on interprète $T[1..n-2]$.
3. Sinon, interpréter les expressions stockées dans $T[0..i-1]$ et $T[i+1..n-1]$, et appliquer l'opérateur sur les deux valeurs obtenues.

Par exemple, pour l'expression $(32+7)\times[2+4]$, représentée par le tableau T suivant :

i	0	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	(	3	2	+	7	)	×	[	2	+	4	]

L'opérateur le plus extérieur est le $\times$ à l'indice $i=6$. On commence donc par interpréter l'expression à sa gauche :

i	0	1	2	3	4	5
$T[i]$	(	3	2	+	7	)

Il n'y a aucun opérateur à l'extérieur : on interprète donc l'expression comprise entre la deuxième et l'avant dernière case :

i	1	2	3	4
$T[i]$	3	2	+	7

L'opérateur extérieur est $+$, à la case $i=3$. On interprète donc les deux sous-expressions de gauche et de droite : aucune n'a d'opérateur extérieur, et les deux sont uniquement constituées d'entiers. $T[1..2]$ s'évalue en 32 et $T[4..4]$ en 7. L'expression $T[1..4]$ s'évalue donc en $32+7=39$.

Donc $T[0..5]$ s'évalue en 39. En appliquant le même procédé, on trouve que $T[6..11]$ s'évalue en 6. Finalement, $T=T[0..11]$ s'évalue en $39\times 6=234$.

Q14. On considère l'expression $(e_1)+(e_2)+(e_3)$ où e_1, e_2, e_3 sont des expressions, de telle sorte qu'il y a deux symboles $+$ à l'extérieur des parenthèses. Lequel des deux choisir à l'étape 1 de l'algorithme ?

Et pour l'expression $(e_1)+(e_2)\times(e_3)$?

Q15. De manière générale, si plusieurs opérateurs sont à l'extérieur des parenthèses, lequel doit-on choisir pour séparer l'expression en 2 ?

Q16. En déduire un algorithme en pseudo-code ayant la spécification suivante :

Algorithme 4 : PositionOpérateur
Entrée(s) : T un tableau de symboles de taille n, $i, j \in [\![0, n-1]\!]$ avec $i \leq j$, tels que $T[i..j]$ est bien parenthésé
Sortie(s) : Position $p \in [\![i, j]\!]$ de l'opérateur à choisir parmi ceux à l'extérieur. Renvoie -1 si aucun opérateur ne se trouve à l'extérieur

(*Indication : On lira T de gauche à droite, en stockant dans une pile les parenthèses non fermées, afin de pouvoir déterminer lorsqu'on se trouve à l'extérieur des parenthèses.*)

Q17. En déduire un algorithme récursif en pseudo-code ayant la spécification suivante :

Algorithme 5 : Interpréter
Entrée(s) : T un tableau de symboles de taille n, $i, j \in [\![0, n-1]\!]$ avec $i \leq j$, tels que $T[i..j]$ est bien parenthésé
Sortie(s) : Valeur obtenue en évaluant l'expression $T[i..j]$

Q18. Expliquer comment modifier l'algorithme de choix d'opérateur pour pouvoir traiter également l'opérateur binaire $-$, selon les règles habituelles de calcul. Par exemple, l'expression `3-2-1+4` doit d'évaluer en 4.

Variable

On rajoute à nos expressions la possibilité de contenir des variables, dont on suppose pour simplifier qu'elles sont constituées uniquement de lettres minuscules.

Définition 1. Un **environnement** est la donnée d'un nombre fini de variables $x_1, \ldots, x_k$ et d'associations $x_1 \mapsto v_1, \ldots, x_k \mapsto v_k$, où les v_i sont des entiers.

Si ρ est un environnement, alors **l'évaluation d'une expression** e dans ρ est la valeur obtenue en évaluant e, où chaque variable x_i est évaluée à v_i.

Q19. Quelle structure de données abstraite peut-on utiliser pour représenter un environnement ? Quelle implémentation concrète efficace connaissez-vous pour cette structure abstraite ?

Q20. Expliquer comment modifier **Interpreter** en un algorithme **InterpréterEnv** prenant également en entrée un environnement, et évaluant l'expression dans cet environnement.

Listes chaînées

On rappelle qu'une liste chaînée est une structure composée de cellules mises bout à bout, où chaque cellule possède un pointeur vers la cellule suivante. On propose le type struct C suivant pour les listes chaînées :

```c
struct cell {
   struct cell* next;
   int data;
};
typedef struct cell cell_t;
typedef cell_t* list_t;
```

Une liste est donc la donnée d'un pointeur vers un maillon, appelé la **tête** de liste.
Par exemple, le code suivant crée une liste contenant, dans l'ordre, les éléments 1, 2 et 3 :

```c
int main(){
   list_t l = malloc(sizeof(cell_t)); // 1er maillon
   cell_t* c = l;
   c->data = 1;

   c->next = malloc(sizeof(cell_t)); // 2eme maillon
   c = c->next;
   c->data = 2;

   c->next = malloc(sizeof(cell_t)); // 3eme maillon
   c = c->next;
   c->data = 3;

   c->next = NULL;
}
```

Pour créer une liste vide, on renvoie un pointeur nul :

```c
list_t list_create(){
   return NULL;
}
```

Q21. Écrire une fonction C `void list_print(list_t l)` qui affiche une à une les données des cellules de la liste, avec un saut de ligne entre chaque valeur affichée.

Q22. Écrire une fonction C `void list_free(list_t l)` qui libère toute la mémoire allouée à une liste.

Q23. Implémenter une fonction d'ajout en tête de liste, ayant la spécification suivante :

```c
/* Ajoute la valeur x à la tête de la liste l.
   Renvoie un pointeur vers la nouvelle cellule de tête. */
list_t list_prepend(list_t l, int x);
```

Votre fonction devra avoir une complexité de $\mathcal{O}(1)$.

Q24. Implémenter une fonction d'ajout en queue de liste, ayant la spécification suivante :

```
/* Ajoute la valeur x à la queue de la liste l.
   Renvoie un pointeur vers la nouvelle cellule de tête. */
list_t list_append(list_t l, int x);
```

Votre fonction devra avoir une complexité de $\mathcal{O}(n)$ et n'allouer que $\mathcal{O}(1)$ espace mémoire supplémentaire.

On exécute le code suivant :

```
int main(){
   list_t l = list_create();
   l = list_append(l, 1);
   l = list_append(l, 2);
   l = list_append(l, 3);
   list_t q = list_prepend(l, 4);
   l = list_prepend(l, 5);
   // A
   list_free(l);
   list_free(q);
   // B
}
```

Q25. Représenter schématiquement la mémoire à l'instant A. On fera apparaître clairement la distinction entre le tas et la pile d'appel, et on pourra représenter les pointeurs par des flèches : $p \to X$ signifiera que p a pour valeur l'adresse de la zone X.

Q26. Déterminer si les instructions entre l'instant A et l'instant B permettent ou non de libérer correctement la mémoire réservée, et proposer une correction si ce n'est pas le cas.

On considère le code suivant :

```
int main(){
   list_t l = list_create();
   l = list_append(l, 0);
   list_t l0 = l;
   for (int i = 1; i <= 4; i++){
      l = list_prepend(l, i);
   }

   cell_t* c = l0;
   for (int i = 5; i <= 8; i++){
      l = list_append(l, i);
      c = c->next;
   }
   c->next = l0;
   // A
   list_print(l);
}
```

Q27. A l'instant A, représenter schématiquement l'état de la liste `l`. On ne fera pas figurer la pile et le tas, on se contentera de représenter les différents pointeurs.

Q28. Qu'affiche le programme lors de l'appel à `list_print` ?

On dit qu'une liste est mal formée si une cellule c pointe vers une cellule c' qui apparaît avant elle dans la liste, formant ainsi un cycle. On propose l'algorithme suivant pour détecter les cycles dans une liste chaînée :

Algorithme 6 : Détection de cycle ?

Entrée(s) : L une liste chaînée
Sortie(s) : "Oui" si L contient un cycle, "Non" sinon

```
1 C_0 ← cellule de tête de L;
2 C ← C_0.next;
3 tant que C ≠ NULL et C ≠ C_0 faire
4     C ← C.next;
5 retourner "Oui" si C = C_0, "Non" sinon
```

Q29. Dessiner une liste chaînée mal-formée pour laquelle cet algorithme détecte le cycle, et une autre pour laquelle il ne le détecte pas, en justifiant sommairement.

L'algorithme du **lièvre et de la tortue** est un algorithme classique de détection des cycles. Il fonctionne à l'aide de deux pointeurs qui parcourent la liste : le lièvre et la tortue.

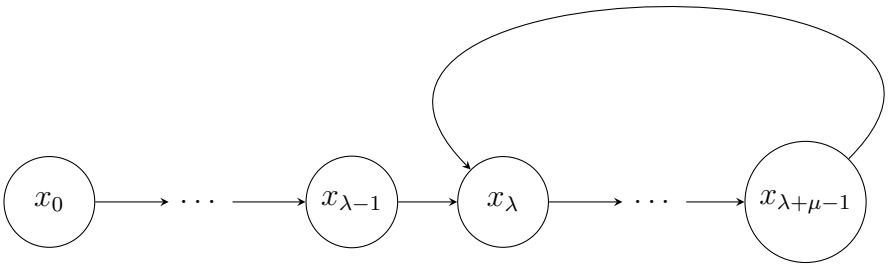
La tortue parcourt la liste cellule par cellule, et le lièvre la parcourt par sauts de deux cellules :

Algorithme 7 : Lièvre et tortue

Entrée(s) : L une liste chaînée
Sortie(s) : "Oui" si L contient un cycle, "Non" sinon

```
1 C_0 ← cellule de tête de L;
2 H ← C_0.next// ``Hare", le lapin
3 T ← C_0.next// ``Tortoise", la tortue
4 tant que H ≠ NULL et H.next ≠ NULL et H ≠ T faire
5     T ← T.next;
6     H ← H.next.next;
7 retourner "Oui" si H = T est NULL, "Non" sinon
```

On considère une liste L mal formée, constituée de $\lambda + \mu$ cellule, dont les μ dernières forment un cycle.



Q30. Sur quelle cellule x_i se trouve le lièvre lorsque la tortue arrive sur la cellule x_λ pour la première fois ?

Q31. On se place à l'instant où la tortue arrive sur la cellule x_λ pour la première fois. Déterminer combien d'étapes supplémentaires sont nécessaires avant que le lièvre et la tortue se trouvent sur la même cellule. En conclure la complexité de l'algorithme proposé, en fonction de λ et μ.

Représentation binaire gauche

Cette partie est tirée de l'épreuve 2023 de MPI du concours des Mines

Soient m un entier naturel et N un entier naturel non nul. Il est classique d'appeler **représentation binaire standard de l'entier m sur N chiffres** ou plus simplement **représentation standard**, toute suite finie $b = (b_n)_{0\leq n<N}$ de longueur N telle que, pour tout indice n compris entre 0 et $N-1$, le chiffre b_n appartient à $\{0,1\}$ et l'égalité suivante est vérifiée :

$$m = \sum_{n=0}^{N-1} b_n 2^n$$

Définition 2. On appelle **représentation binaire gauche** de l'entier m sur N chiffres toute suite finie $g = (g_n)_{0\leq n<N}$ de longueur N telle que :

1. pour tout indice n compris entre 0 et $N-1$, le chiffre g_n appartient à $\{0,1,2\}$,
2. l'égalité suivante est vérifiée :

$$m = \sum_{n=0}^{N-1} g_n(2^{n+1}-1),$$

3. le chiffre "2" n'apparaît qu'au plus une fois parmis les chiffres $(g_n)_{0\leq n<N}$,
4. s'il existe une position p tel que le chiffre g_p est 2, alors, pour tout indice n compris entre 0 et $p-1$, le chiffre g_n est nul.

De manière plus courte, on parlera simplement de **représentation gauche**.

La figure 1 ci-dessous donne une représentation standard et une représentation gauche sur quatre chiffres des seize premiers entiers. Conformément à l'usage habituel, on écrit toute représentation, qu'elle soit standard ou gauche, sous la forme d'un mot $b_{N-1}\dots b_0$ ou $g_{N-1}\dots g_0$ dans lequel les chiffres de poids faibles sont écrits à droite.

Entier	Repr. standard	Repr. gauche
0	0000	0000
1	0001	0001
2	0010	0002
3	0011	0010
4	0100	0011
5	0101	0012
6	0110	0020
7	0111	0100

Entier	Repr. standard	Repr. gauche
8	1000	0101
9	1001	0102
10	1010	0110
11	1011	0111
12	1100	0112
13	1101	0120
14	1110	0100
15	1111	1000

FIGURE 1 – Représentations standard et gauche des seize premiers entiers

Q32. Soit c un entier. Donner la représentation standard de l'entier dont une représentation gauche est $10\dots0$ (avec c chiffres nuls) en justifiant sommairement. Faire de même avec l'entier dont une représentation gauche est $20\dots0$ (avec c chiffres nuls).

Q33. Déterminer, en justifiant, le plus grand entier naturel M_N qui admet une représentation gauche sur N chiffres. Préciser la représentation gauche de cet entier.

Définition 3. Soit n_0 un indice compris entre 0 et $N-1$. On dit que l'indice n_0 est la **position du chiffre de plus fort poids** d'une représentation $g_{N-1}\dots g_0$ si l'indice n_0 est le plus petit entier tel que, pour tout indice $n > n_0$, le chiffre g_n est nul. On appelle g_{n_0} le **chiffre de plus fort poids**.

Q34. Soient N entier naturel non nul, $g = g_{N-1}\dots g_0$ et $h = h_{N-1}\dots h_0$ deux représentations gauches d'un même entier m. Démontrer que les chiffres de plus fort poids de g et de h sont de même valeur et à la même position.

Q35. Démontrer que tout entier appartenant à $[\![0, M_N]\!]$, où l'entier M_N a été introduit à la question 33, ne possède au plus qu'une seule représentation gauche sur N chiffres.

Indication C : l'entier N est déclaré comme constante globale. Nous utilisons la structure C déclarée comme suit pour écrire la représentation gauche sur N chiffres d'un entier $g_{N-1}\dots g_0$:

```c
#define N 8

struct RepGauche {
   int position;
   bool chiffres[N];
};
typedef struct RepGauche rg;
```

Le champ `position` repère la position éventuelle du chiffre 2 ; il vaut -1 au cas où le chiffre 2 n'apparaît pas. Pour tout indice n compris entre 0 et $N-1$, la n^{e} case du champ `chiffres` vaut `true` si le chiffre g_n est non-nul, et vaut `false` sinon.

Par exemple, les entiers 15 et 21 ont pour représentation gauche les variables `entier_15` et `entier_21` suivantes :

```c
rg entier_15 = {
   .position = -1,
   .chiffres = {0, 0, 0, 1, 0, 0, 0, 0}
};
rg entier_21 = {
   .position = 1,
   .chiffres = {0, 1, 0, 1, 0, 0, 0, 0}
};
```

Q36. Donner une ou plusieurs propriétés formelles exprimant le fait qu'une valeur C de type `rg` est la représentation gauche d'un entier.

Q37. Écrire une fonction C `int rg_to_int(rg g)` qui renvoie l'entier dont g est la représentation gauche. On supposera que la propriété de la question 36 est satisfaite.

On propose l'algorithme suivant :

Algorithme 8 : Mystère

Entrée(s) : Représentation gauche $g = g_{N-1}\dots g_0$ d'un certain entier m
Sortie(s) : g a été modifié

si ***aucun des chiffres*** $(g_n)_{0\leq n<N}$ ***ne vaut 2*** **alors**
  Changer le chiffre g_0 en $g_0 + 1$
sinon
  En notant p la position du chiffre 2, changer le chiffre g_p en 0 et g_{p+1} en $g_{p+1} + 1$
On note m' l'entier dont la représentation gauche est g après exécution de l'algorithme.

Q38. Vérifier que la propriété de la question 36 est un invariant de l'algorithme mystère, i.e. qu'elle n'est pas rompue par cet algorithme. Avec les notations m et m' de l'algorithme, caractériser m' en fonction de m. Justifier.

Q39. Écrire une fonction C `bool rg_incr(rg* s)` dont la spécification suit :
Précondition : La variable `s` est un pointeur vers la représentation gauche d'un entier m.
Effet : La valeur pointée par `s` est modifiée afin de représenter l'entier $m+1$.
Valeur de retour : Booléen `true` si l'incrémentation de m peut avoir lieu et `false` si un débordement se produit car $m+1$ n'est pas représentable sur le même nombre de chiffres.

Q40. Calculer la complexité en temps dans le pire cas de la fonction `rg_incr` en fonction de N. Comparer avec la complexité de la même opération sur la représentation standard.

Q41. Écrire une fonction C `bool rg_decr(rg* s)` dont la spécification suit : *Précondition* : La variable `s` est un pointeur vers la représentation gauche d'un entier m.
Effet : La valeur pointée par `s` est modifiée afin de représenter l'entier $m-1$.
Valeur de retour : Booléen `true` si l'incrémentation de m peut avoir lieu et `false` si un débordement se produit car m est nul.
Il est recommandé d'expliquer son intention avant de donner son code.

Q42. Démontrer que tout entier appartenant à $[\![0, M_N]\!]$, où l'entier M_N a été introduit à la question 33, possède une représentation gauche sur N chiffres.