Examen

8 janvier 2019 — Durée 2h

Document autorisé : **Mémento C** vierge de toute annotation
Les deux parties sont indépendantes et peuvent être traitées dans un ordre quelconque.

Éléments de correction

Partie I (7 pt + 1 pt bonus)

Soit le programme C suivant, écrit dans un fichier nommé `premiers.c` :

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

/* est_premier(n) = vrai ssi n est un nombre premier */
/* Précondition : n >= 0 */
bool est_premier(int n) {
  if (n <= 1) {
    return false;
  } else if (n == 2) {
    return true;
  } else {
    // n >= 3
    int d = 2;
    while ((d*d < n) && (n%d != 0)) {
      d = d + 1;
    }
    return (d*d > n);
  }
}

int main(int argc, char ** argv) {
  int a = atoi(argv[1]);
  int b = atoi(argv[2]);

  printf("La liste des nombres premiers dans l'intervalle [%d, %d] est :\n", a, b);
  for (int i = a; i < b; i++) {
    if (est_premier(i)) {
      printf("%d\n", i);
    }
  }
}
```

Exercice 1. (1 pt) Décrire en une phrase ou deux ce que fait ce programme.

Ce programme prend en argument de la ligne de commande deux entiers a et b, et écrit sur la sortie standard la liste des nombres premiers compris dans l'intervalle $[a, b]$.

Exercice 2. (1 pt) Quelles sont les entrées de ce programme ? Quelles conditions suffisantes doivent satisfaire ces entrées pour que ce programme fonctionne correctement ?

Les entrées sont deux entiers a et b (fournis comme argument de la ligne de commande).
Ces deux entrées doivent être entières et positives.
(Condition suffisante acceptée : $a \leq b$)

Exercice 3. (1 pt) Donner la (ou les) ligne(s) de commande permettant de compiler ce programme, générant un exécutable nommé `premiers`.

```
clang premiers.c -o premiers
```

Exercice 4. (3 pt) Décrire un jeu de tests fonctionnels pour ce programme.

Le jeu de tests peut être construit à partir : 1) de la taille de l'intervalle, 2) des bornes de l'intervalle (cas particuliers : 0 et 1), 3) du nombre de nombres premiers attendus, et de leur position dans l'intervalle.

- *Cas limite : intervalle vide, exemple :* $[1, 0]$
- *Cas limite : intervalle de taille 1*
  - *Cas limite : intervalle contenant 0 :* $[0, 0]$
  - *Cas limite : intervalle contenant 1 :* $[1, 1]$
  - *Intervalle contenant un nombre premier :* $[2, 2]$
  - *Intervalle ne contenant pas de nombre premier :* $[4, 4]$
- *Intervalle de taille quelconque*
  - *Cas limite : borne 0*
    - *Intervalle ne contenant pas de nombre premier :* $[0, 1]$
    - *Intervalle finissant par un nombre premier :* $[0, 2]$
    - *Intervalle ne finissant pas par un nombre premier :* $[0, 4]$
  - *Cas limite : borne 1*
    - *Intervalle finissant par un nombre premier :* $[1, 3]$
    - *Intervalle ne finissant pas par un nombre premier :* $[1, 4]$
  - *Cas général :*
    - *Intervalle ne contenant aucun nombre premier :* $[8, 10]$
    - *Intervalle contenant un seul nombre premier :* $[7, 10]$, $[8, 11]$, $[6, 8]$
    - *Intervalle contenant plusieurs nombres premiers :* $[2, 4]$, $[5, 8]$, $[4, 7]$, $[5, 11]$

Exercice 5. (1 pt) Donner un exemple de test de robustesse pour ce programme.

Quelques exemples possibles :

- *appel du programme sans argument :* `./premiers`
- *entrée non entière :* `./premiers a 12`
- *entrée négative :* `./premiers -1 1`

Exercice 6. (bonus : 1 pt) Le programme fourni comprend une erreur : laquelle ? Le jeu de tests que vous avez fourni à l'exercice 4 aurait-il permis de détecter cette erreur ?

Partie II (13 pt)

Soient les paquetages `arbres`, `es_arbres` et `op_arbres`, dont les spécifications sont données dans les fichiers en-têtes en annexe. On souhaite dans cette partie tester la fonction `profondeur_noeud`.

NB : pour cette partie, vous n'avez pas besoin de connaître l'implémentation des paquetages, ni du format des arbres binaires dans un fichier. Si vous avez besoin de décrire un arbre binaire, vous pouvez simplement le dessiner.

Exercice 7. (3 pt) Écrire un programme de test `test_profondeur.c` qui :

1. lit un arbre dans un fichier dont le nom est donné en argument de la ligne de commande,
2. demande la saisie d'un entier n sur l'entrée standard,
3. affiche la profondeur de l'élément n dans l'arbre.

NB : vous pouvez répondre à l'exercice 12 sur le même programme.

(Avec la gestion d'erreurs)

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

#include "arbres.h"
#include "es_arbres.h"
#include "op_arbres.h"

int main(int argc, char ** argv) {

  FILE * f;
  erreur_lecture e;
  arbre a;
  int n, p;
  erreur_prof ep;

  f = fopen(argv[1], "r");

  e = lire_arbre(f, &a);

  switch(e) {
  case OK_LECTURE:
    printf("Saisir l'entier àchercher : ");
    scanf("%d",&n);
    ep = profondeur_noeud(a, n, &p);
    switch (ep) {
    case OK_PROF:
      printf("Profondeur de l'entier %d : %d\n", n, p);
      break;
    case ERREUR_VIDE:
      printf("Erreur : l'arbre est vide\n");
      break;
    case ERREUR_VALEUR:
      printf("Erreur : la valeur recherchée n'est pas dans l'arbre\n");
      break;
    }
    break;
  case ERREUR_FICHIER:
    printf("Erreur de lecture du fichier\n");
    break;
  case ERREUR_FORMAT:
    printf("Erreur dans le format de l'arbre\n");
    break;
  }
}
```

Exercice 8. (1 pt) Donner le domaine de validité de la fonction `profondeur_noeud`.

La fonction `profondeur_noeud` *prend en entrée un arbre binaire a et un entier n. L'arbre a doit être non vide, et l'élément n doit être présent dans a.*

Exercice 9. (3 pt) Décrire un jeu de tests fonctionnels pour la fonction `profondeur_noeud`.

- *Cas limite : un arbre avec un seul nœud, l'élément à chercher est la valeur de ce nœud*
- *Cas général : arbres de hauteur $h \geq 2$*
  - *Cas limite : valeur à chercher présente à la racine*
  - *Cas limite : valeur à chercher présente sur une feuille de l'arbre*
    - *feuille présente à gauche/à droite de la racine*
    - *chemin alternant fils gauche/fils droit de la racine à la feuille contenant la valeur recherchée*
    - *arbres de différentes structures : arbres complets, «peignes», ...*
  - *Cas général : valeur sur un nœud interne*
    - *nœud présent à gauche/à droite de la racine*
    - *chemin alternant fils gauche/fils droit de la racine au nœud contenant la valeur recherchée*
    - *arbres de différentes structures : arbres complets, «peignes», ...*

Exercice 10. (2 pt) Décrire un jeu de tests de robustesse pour la fonction `profondeur_noeud`.

- *Arbre vide*
- *Arbre ne contenant pas la valeur cherchée*
  - *Cas limite : arbre unaire*
  - *Cas général : arbres de différentes structures...*

Exercice 11. (2 pt) Écrire un Makefile permettant de compiler le programme de l'exercice 7. L'exécution de la commande `make` sans argument doit générer un exécutable nommé `test_profondeur`.

```
1  CC = clang
2
3  all: test_profondeur
4
5  arbres.o: arbres.h
6
7  es_arbres.o: es_arbres.h arbres.h
8
9  op_arbres.o: op_arbres.h arbres.h
10
11 test_profondeur.o: arbres.h es_arbres.h op_arbres.h
12
13 %.o: %.c
14        $(CC) -c $<
15
16 test_profondeur: test_profondeur.o es_arbres.o op_arbres.o arbres.o
17        $(CC) $^ -o $@
18
19 clean:
20        rm -f *.o test_profondeur
```

Exercice 12. (2 pt) Compléter le programme de l'exercice 7 pour afficher un message d'erreur en cas d'utilisation erronée de la fonction `profondeur_noeud`.

Annexes

Paquetage arbres : contenu du fichier arbres.h

```c
#ifndef ARBRES_H
#define ARBRES_H

#include <stdbool.h>

/* Un arbre binaire est défini comme une référence vers le noeud racine de l'arbre.
 * Un arbre binaire vide est représenté par une référence NULL. */
typedef struct noeud* arbre;

/* Retourne un arbre vide */
arbre arbre_vide();

/* Crée un noeud de valeur v, de fils gauche g, de fils droit d */
arbre nouveau_noeud(arbre g, int v, arbre d);

/* est_vide(a) = vrai ssi a est vide */
bool est_vide(arbre a);

/* Renvoie la valeur entiè re àla racine de l'arbre a
 * Précondition : a est non vide */
int valeur_racine(arbre a);

/* Renvoie le fils gauche de l'arbre a
 * Précondition : a est non vide */
arbre gauche(arbre a);

/* Renvoie le fils droit de l'arbre a
 * Précondition : a est non vide */
arbre droit(arbre a);

#endif
```

Paquetage es_arbres : contenu du fichier es_arbres.h

```c
#ifndef ES_ARBRES_H
#define ES_ARBRES_H

#include <stdio.h>

#include "arbres.h"

typedef enum {
  OK_LECTURE,
  ERREUR_FICHIER,
  ERREUR_FORMAT
} erreur_lecture;

/* Construit un arbre lu depuis le fichier f.
 * L'arbre construit est placé àl'adresse a
 * Précondition : f est ouvert en lecture. */
erreur_lecture lire_arbre(FILE * f, arbre * a);

/* Ecrit l'arbre dans le fichier f
 * Préconditions :
 * - a est un arbre correctement construit
 * - f est ouvert en écriture */
void ecrire_arbre(FILE * f, arbre a);

#endif
```

Paquetage op_arbres : contenu du fichier op_arbres.h

```
1  #ifndef OP_ARBRES_H
2  #define OP_ARBRES_H
3
4  #include "arbres.h"
5
6  typedef enum {
7    OK_PROF, /* Précondition remplie */
8    ERREUR_VIDE, /* Arbre vide */
9    ERREUR_VALEUR /* Arbre ne contenant pas la valeur */
10 } erreur_prof;
11
12 /* Calcule la profondeur dans l'arbre a du noeud de valeur n
13  * Le noeud àla racine est de profondeur 0
14  * Précondition : a contient un noeud de valeur n
15  * Résultat : *p contient la profondeur calculée
16  * Renvoie :
17  * - OK_PROF si la précondition est remplie
18  * - ERREUR_VIDE si l'arbre est vide
19  * - ERREUR_VALEUR si l'arbre ne contient pas la valeur recherchée
20  * si la fonction renvoie ERREUR_VIDE ou ERREUR_VALEUR, la valeur de
21  * *p n'est pas définie
22  */
23 erreur_prof profondeur_noeud(arbre a, int n, int * p);
24
25 #endif
```