

Algorithmique et Programmation – Examen (durée 3 heures)

Deuxième session du 22 juin 2021

NOTES : Aucun document autorisé. Sont interdits les calculatrices, les téléphones, ainsi que tout autre ustensile de calcul ou de communication.

REMARQUE 1 : Une indication sur le niveau atteint est précisée après chaque exercice. Le niveau de difficulté allant croissant, il est préférable de traiter les exercices dans l'ordre. La réussite de l'exercice 1 est requise pour valider l'examen ; les exercices peuvent ensuite être traités indépendamment de leur ordre.

REMARQUE 2 : Dans la suite, les indications concernant les nombres de lignes sont données en comptant toute ligne de code non-vide (`#include`, prototypes, accolades...).

REMARQUE 3 : Les petits oublis d'un point-virgule, d'une parenthèse, d'une virgule, d'une apostrophe double ou simple, ne seront pas pénalisants tant qu'ils restent **ponctuels**. En revanche, soignez le placement des accolades.

Exercice 1 : Intervalles d'acceptabilité

Notre logiciel de calcul permet de résoudre un problème (mécanique, électronique, gestion industrielle, ...) et utilise pour cela des intervalles d'acceptabilités. Lorsqu'une valeur est testée, elle est soit dans l'intervalle, soit hors de l'intervalle. Nous mettons en place dans cet exercice les fonctionnalités qui permettront de manipuler des intervalles d'acceptabilité.

a) Écrire une **FONCTION** en **LANGAGE C** qui prend en paramètre :

- la borne inférieure de l'intervalle ;
- la borne supérieure de l'intervalle ;
- et la valeur testée ;

et retourne la valeur nulle (zéro) si valeur est en dehors de l'intervalle, ou sinon une valeur positive correspondant à la longueur de l'intervalle lorsque la valeur est comprise dans l'intervalle. Nous considérons que les bornes sont incluses dans les intervalles : `[borne_inf ; borne_sup]`. La longueur d'un intervalle sera mesurée comme la différence entre sa borne supérieure et sa borne inférieure.

Le prototype de la fonction sera le suivant :

```
int dans_intervalle (int borne_inf, int borne_sup, int valeur) ;
```

b) Pour tester la fonction du (a), écrire un **PROGRAMME** en **LANGAGE C** dans lequel l'utilisateur commence par indiquer une borne inférieure et une borne supérieure, puis donne une valeur. Ensuite, le programme informe si la valeur est dans l'intervalle, si le double de la valeur est dans l'intervalle et si le carré de la valeur est dans l'intervalle.

Ci-après quatre exemples d'exécution de ce programme :

```

C:\ExemplesC> exercicelb.exe
Donnez la borne inferieure : 20
Donnez la borne superieure : 200
Donnez une valeur : 60
La valeur est dans l'intervalle.
Le double de la valeur est dans l'intervalle.
C:\ExemplesC>

C:\ExemplesC> exercicelb.exe
Donnez la borne inferieure : 5
Donnez la borne superieure : 50
Donnez une valeur : 6
La valeur est dans l'intervalle.
Le double de la valeur est dans l'intervalle.
Le carre de la valeur est dans l'intervalle.
C:\ExemplesC>

C:\ExemplesC> exercicelb.exe
Donnez la borne inferieure : 20
Donnez la borne superieure : 200
Donnez une valeur : 10
Le double de la valeur est dans l'intervalle.
Le carre de la valeur est dans l'intervalle.
C:\ExemplesC>

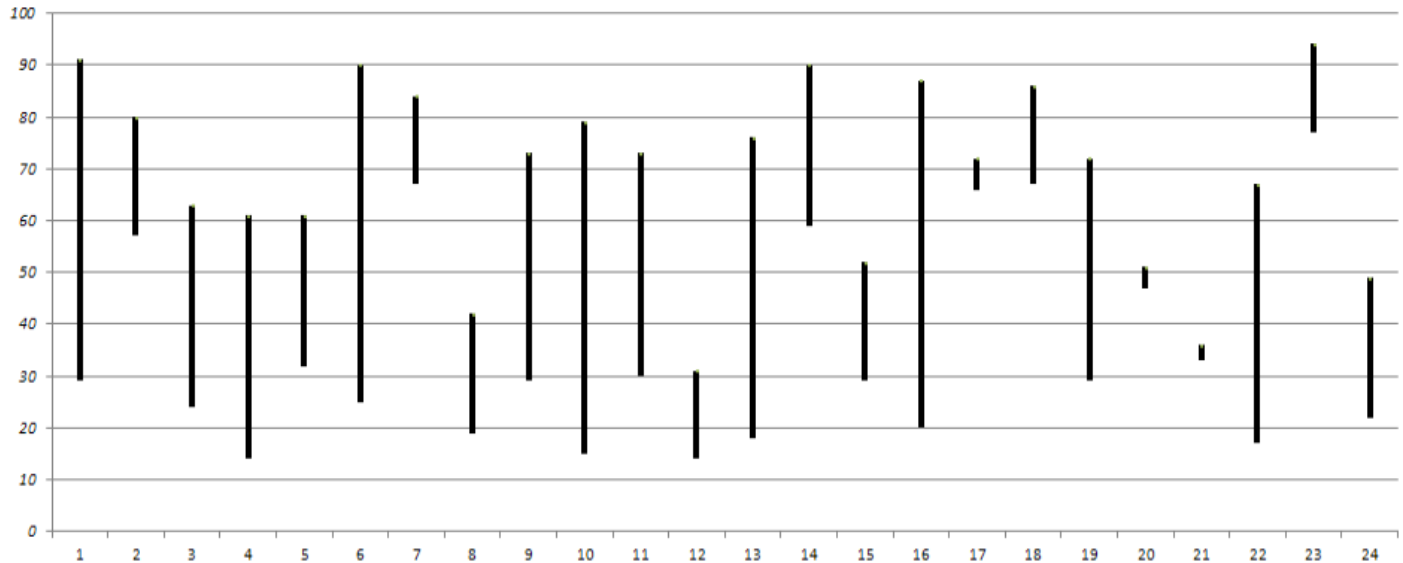
C:\ExemplesC> exercicelb.exe
Donnez la borne inferieure : 5
Donnez la borne superieure : 50
Donnez une valeur : 30
La valeur est dans l'intervalle.
C:\ExemplesC>

```

c) Pour amorcer le développement du logiciel de calcul, écrire un **PROGRAMME** en **LANGAGE C** dans lequel sont déclarés les deux tableaux suivants, définissant 24 bornes inférieures et les bornes supérieures :

```
int inf[24] = { 29, 57, 24, 14, 32, 25, 67, 19, 29, 15, 30, 14, 18, 59, 29, 20,
               66, 67, 29, 47, 33, 17, 77, 22 } ;
int sup[24] = { 91, 80, 63, 61, 61, 90, 84, 42, 73, 79, 73, 31, 76, 90, 52, 87,
               72, 86, 72, 51, 36, 67, 94, 49 } ;
```

Ces bornes définissent ainsi 24 intervalles. Par exemple, le premier intervalle est [29 ; 91]. Ces intervalles ont été représentés graphiquement sur la figure suivante :



Le programme commencera par demander à l'utilisateur de saisir une valeur, puis affichera le nombre d'intervalles dans lesquels la valeur est acceptée (càd comprise entre la borne inférieure et la borne supérieure), puis affichera, avant de s'arrêter, le plus grand intervalles acceptant la valeur.

Ci-après trois exemples d'exécution de ce programme :

<pre>Invite de commandes C:\ExemplesC> exercicelc.exe Donnez votre valeur : 10 0 intervalles sont acceptables. C:\ExemplesC></pre>	<pre>Invite de commandes C:\ExemplesC> exercicelc.exe Donnez votre valeur : 50 14 intervalles sont acceptables. Le plus grand intervalle acceptable est [20;87]. C:\ExemplesC></pre>
<pre>Invite de commandes C:\ExemplesC> exercicelc.exe Donnez votre valeur : 20 7 intervalles sont acceptables. Le plus grand intervalle acceptable est [20;87]. C:\ExemplesC></pre>	<pre>Invite de commandes C:\ExemplesC> exercicelc.exe Donnez votre valeur : 90 4 intervalles sont acceptables. Le plus grand intervalle acceptable est [25;90]. C:\ExemplesC></pre>

N.B. : Un seul parcours de tableau suffit.

Remarques :

- entre 3 et 6 lignes (environ) pour le (a)
- plus ou moins 16 lignes (environ) pour le (b)
- plus ou moins 20 lignes (environ) pour le (c)
- les programmes du (b) et du (c) doivent utiliser la fonction du (a)

E Être capable d'écrire des fonctions et des programmes simples

Exercice 2 : Paquetage de gestion d'intervalles

Nous souhaitons créer un paquetage permettant de manipuler des intervalles, décrits chacun par une borne inférieure et une borne supérieure. Une partie de ce paquetage est déjà écrite, mais il manque encore deux fonctions. L'objectif de l'exercice est d'écrire les deux fonctions manquantes.

Ci-dessous le fichier d'entête `TRange.h` déjà écrit :

```

1 /* File: TRange.h */
2 #ifndef TRANGE_H
3 #define TRANGE_H
4
5 struct range {
6     int inf ;
7     int sup ;
8 } ;
9
10 typedef struct range TRange ;
11
12 void range_print (TRange *range) ;
13
14 char range_read (TRange *range, FILE *desc) ;
15
16 char range_contains (TRange *range1, TRange *range2) ;
17
18 #endif /* TRANGE_H */

```

Ci-dessous le fichier d'implantation `TRange.c` partiellement écrit :

```

1 /* File: TRange.c */
2 #include <stdio.h>
3
4 #include "TRange.h"
5
6 void range_print (TRange *range) {
7     printf ("[_%d_ ; _%d_]\n", range->inf, range->sup) ;
8 }

```

a) La première **FONCTION EN LANGAGE C** manquante permet de lire un (et un seul) intervalle depuis un descripteur de fichier. Nous considérons que le descripteur de fichier est déjà correctement créé et que chaque ligne du fichier ouvert est structurée comme suit :

- Chaque ligne décrit un et un seul intervalle ;
- Son premier élément est la borne inférieure de l'intervalle ;
- Son deuxième élément est la borne supérieure de l'intervalle.

Ci-contre l'exemple du fichier `bornes.txt` décrivant vingt quatre intervalles (correspondant au vingt quatre intervalles de la figure de l'exercice 1).

La fonction retourne une valeur vraie en cas de réussite de la lecture, ou faux sinon.

Écrivez le code de cette fonction `range_read()` permettant d'initialiser un intervalle en lisant depuis un descripteur ouvert vers un tel fichier.

b) La deuxième **FONCTION EN LANGAGE C** manquante permet de savoir si un premier intervalle contient un deuxième intervalle. Ainsi, la fonction retourne vrai si `range2` est inclu dans `range1`, ou faux sinon.

Écrivez le code de cette fonction `range_contains()`.

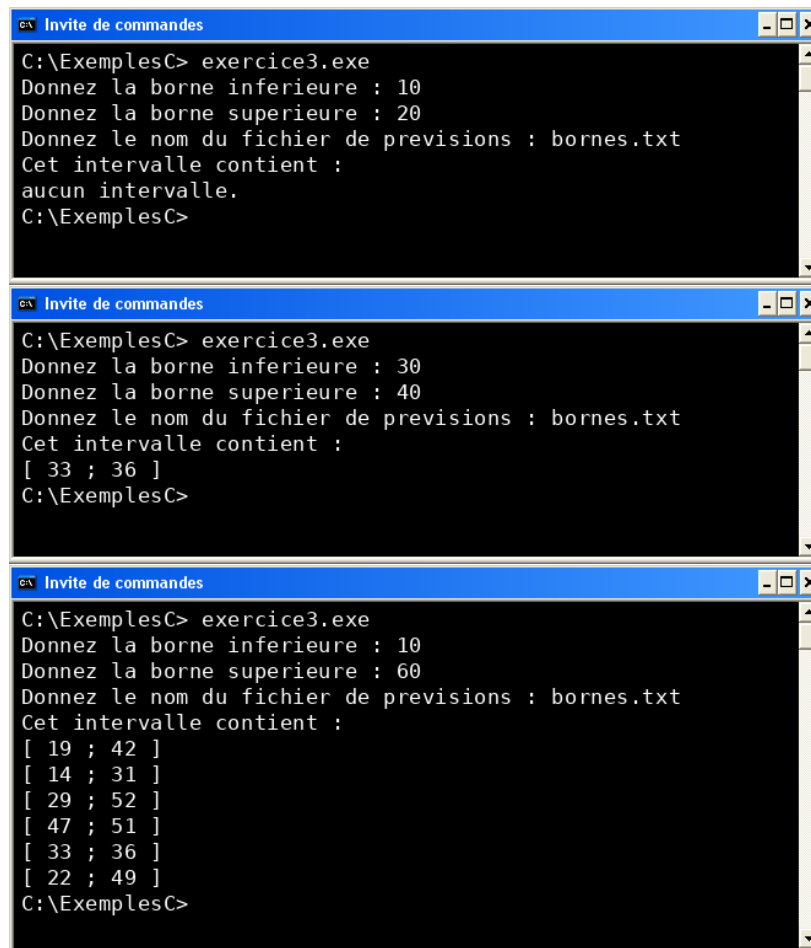
Remarque : Les fonctions utiles pour la manipulation de fichiers sont décrites en annexe 1 (page 6).

1	29	91
2	57	80
3	24	63
4	14	61
5	32	61
6	25	90
7	67	84
8	19	42
9	29	73
10	15	79
11	30	73
12	14	31
13	18	76
14	59	90
15	29	52
16	20	87
17	66	72
18	67	86
19	29	72
20	47	51
21	33	36
22	17	67
23	77	94
24	22	49

Exercice 3 : Programme de simulation d'une installation énergétique

Écrire un **PROGRAMME** en **LANGAGE C** qui demande de spécifier les bornes inférieures et supérieures d'un intervalle de référence, puis demande le nom du fichier des intervalles à lire. Le programme lit ensuite les intervalles contenus dans le fichier, et indique à chaque fois si l'intervalle lu est inclus dans l'intervalle de référence. Ce programme utilisera habilement les structures de données et les fonctions du paquetage de l'exercice 2.

Ci-dessous trois exemples d'exécution de ce programme :



```
Invite de commandes
C:\ExemplesC> exercice3.exe
Donnez la borne inferieure : 10
Donnez la borne superieure : 20
Donnez le nom du fichier de previsions : bornes.txt
Cet intervalle contient :
aucun intervalle.
C:\ExemplesC>

Invite de commandes
C:\ExemplesC> exercice3.exe
Donnez la borne inferieure : 30
Donnez la borne superieure : 40
Donnez le nom du fichier de previsions : bornes.txt
Cet intervalle contient :
[ 33 ; 36 ]
C:\ExemplesC>

Invite de commandes
C:\ExemplesC> exercice3.exe
Donnez la borne inferieure : 10
Donnez la borne superieure : 60
Donnez le nom du fichier de previsions : bornes.txt
Cet intervalle contient :
[ 19 ; 42 ]
[ 14 ; 31 ]
[ 29 ; 52 ]
[ 47 ; 51 ]
[ 33 ; 36 ]
[ 22 ; 49 ]
C:\ExemplesC>
```

Remarque : Les fonctions utiles pour la manipulation de fichiers sont décrites en annexe 1 (page 6).

C Être capable d'utiliser des pointeurs et des structures de données

Exercice 4 : Algorithmique et Structures de Données

B/A Être capable de résoudre des problèmes algorithmiques

Annexe 1 : Extraits de la section 3 des pages de manuel Linux

1. NOM

fopen - Fonctions d'ouverture de flux

2. SYNOPSIS

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

```
FILE *fopen(char *path, char *mode);
```

3. DESCRIPTION

La fonction **fopen()** ouvre le fichier dont le nom est contenu dans la chaîne pointée par **path** et lui associe un flux.

L'argument **mode** pointe vers une chaîne commençant par l'une des séquences suivantes (éventuellement suivie par des caractères supplémentaires, conformément à la description ci-dessous).

r : Ouvre le fichier en lecture. Le pointeur de flux est placé au début du fichier.

w : Tronque le fichier à son début ou ouvre le fichier en écriture. Le pointeur de flux est placé au début du fichier.

a : Ouvre le fichier en ajout (écriture à la fin du fichier). Le fichier est créé s'il n'existait pas. Le pointeur de flux est placé à la fin du fichier.

4. VALEUR RENVOYÉE

Si elle réussit intégralement, **fopen()** renvoie un pointeur de type **FILE**. Sinon, elle renvoie **NULL** et **errno** contient le code d'erreur.

1. NOM

perror - Afficher un message d'erreur système

2. SYNOPSIS

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

```
void perror(char *s);
```

```
#include <errno.h>
```

```
int errno;
```

3. DESCRIPTION

La fonction **perror()** affiche un message sur la sortie d'erreur standard, décrivant la dernière erreur rencontrée durant un appel système ou une fonction de bibliothèque décrite par **errno**.

Notez que **errno** est indéfinie après un appel de fonction de bibliothèque réussi. Cette fonction peut modifier **errno** même si elle réussit, ne serait-ce que par des appels système internes qui peuvent échouer. Ainsi, si un appel qui échoue n'est pas immédiatement suivi par **perror()**, la valeur de **errno** doit être sauvegardée.

1. NOM

feof - Vérifier l'état d'un flux

2. SYNOPSIS

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

```
int feof(FILE *stream);
```

3. DESCRIPTION

La fonction **feof()** teste l'indicateur de fin de fichier du flux pointé par **stream** et renvoie une valeur non nulle si cet indicateur est actif.

1. NOM

fscanf - Entrées formatées

2. SYNOPSIS

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

```
int fscanf(FILE *stream, char *format, ...);
```

3. DESCRIPTION

La fonction **fscanf()** lit ses entrées depuis le flux pointé par **stream**. La fonction **fscanf()** analyse les entrées au même format que la fonction **scanf()**.

4. VALEUR RENVOYÉE

La fonction **fscanf()** renvoie le nombre d'éléments d'entrées correctement mis en correspondance et assignés. Ce nombre peut être plus petit que le nombre d'éléments attendus, et même être nul, s'il y a une erreur de mise en correspondance. Le filtre pour lire une ligne entière sous forme d'une chaîne de caractères est `%[^\r\n]s`.

1. NOM

fprintf - Formatage des sorties

2. SYNOPSIS

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

```
int fprintf(FILE *stream, char *format, ...);
```

3. DESCRIPTION

La fonction **fprintf()** écrit sa sortie sur le flux **stream** indiqué. Comme la fonction **printf()**, cette fonction crée sa sortie sous le contrôle d'une chaîne de format qui indique les conversions à apporter aux arguments qui suivent.

3.1. VALEUR RENVOYÉE

En cas de succès, la fonction **fprintf()** renvoie le nombre de caractères affichés (sans compter l'octet nul final utilisé pour terminer les sorties dans les chaînes).

1. NOM

fclose - Fermer un flux

2. SYNOPSIS

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

```
int fclose(FILE *fp);
```

3. DESCRIPTION

La fonction **fclose()** vide le flux pointé par **fp** (en écrivant toute donnée de sortie en tampon avec **fflush(3)**) et ferme le descripteur de fichier sous-jacent.

4. VALEUR RENVOYÉE

Si la fonction réussit intégralement, elle renvoie 0, sinon elle renvoie EOF et **errno** contient le code d'erreur. Dans tous les cas, tout autre accès ultérieur au flux (y compris un autre appel de **fclose()**) conduit à un comportement indéfini.