UGA - UFR IM^{2}AG **Master 2 CCI**

Programmation en C

Examen du 8 décembre 2023 (session 1)

Durée : deux heures – **Document autorisé :** une feuille A4 recto-verso

Exercice 1 : exécution d'un programme C (3 pts)

Donnez la séquence des trois affichages produits à l'écran lors de l'exécution du programme suivant :

```
#include <stdio.h>

/* variables globales */
int a ;
int *p ;

int f1(int *x) {
  int a ; // variable locale
  a = 42 ;
  *x = a ;
  printf("f1 : a=%d *p=%d\n", a, *p) ;
  return *x+1 ;
}

void f2(int x, int *y) {
  x = f1(y) ;
  a = x+1 ;
  printf("f2 : a=%d x=%d\n", a, x) ;
}

int main() {
   a=12 ;
   p = &a ;
   f2(a, p) ;
   printf("main : a=%d *p=%d\n", a, *p) ;
  return 0 ;
}
```

Exercice 2 : comparaison de chaînes de caractères (5 pts)

En C la fonction `strcmp` fournie par la bibliothèque standard `string.h` est définie de la manière suivante :

> La fonction `strcmp` compare deux chaînes de caractères s1 et s2. Elle renvoie un entier inférieur, égal, ou supérieur à 0 si s1 est respectivement plus petite, identique, ou plus grande que s2. La comparaison des deux chaînes se fait selon l'ordre lexicographique (celui utilisé pour classer les mots dans un dictionnaire).

Q0. Lorsque l'on teste la fonction `strcmp` fournie par la bibliothèque `string.h` on obtient le résultat suivant :
`strcmp("M2 CCI", "M2CCI")` renvoie la valeur -1.
Expliquez pourquoi ...

Un développeur souhaite écrire sa propre implémentation de la fonction `strcmp`, la bibliothèque `string.h` n'étant pas disponible sur sa machine.

Q1. Ecrivez un programme principal lui permettant de tester son implémentation de `strcmp` sur **deux** chaînes de son choix entrées au clavier. On se limitera à des chaînes de moins de 255 caractères (sans la marque de fin). On ne cherchera donc pas à tester la fonction sur des chaînes plus grandes.

Comment pourrait-on faire si l'on ne voulait pas limiter la longueur des chaînes dans ce programme de test ?

Q2. Quels exemples de couples de chaines (s1,s2) suggérez-vous à ce développeur d'utiliser pour tester son programme ? Proposez au moins 5 exemples qui vous semblent pertinents en expliquant pourquoi vous les avez choisis.

Q3. Le développeur propose l'implémentation de `strcmp` donnée ci-dessous.

```
int strcmp(char *s1, char *s2) {
  unsigned int i=0 ;
  char result=0 ;
  while (s1[i] != '\0') {
    if (s1[i] < s2[i]) {
        result = -1 ;
        break ;
    } ;
    if (s1[i] > s2[i]) {
        result = 1 ;
        break ;
    } ;
    i = i+1 ;
  } ;
return result ;
}
```

Cette implémentation est **erronée**. Est-ce que l'erreur est detectée par un de vos tests proposé à la question **Q2** ? Si oui lesquels, et, si non, quel test faut-il ajouter ?

Q4. Donnez une version corrigée de la solution proposée dans la question **Q3.**

Problème : gestion de ressources partagées (12 pts)

On souhaite développer une bibliothèque [1] de nom ResAlloc permettant d'attribuer des *ressources* à des *utilisateurs*. Informellement un utilisateur peut demander l'accès à une ressource, puis libérer cette ressource lorsqu'il a fini de l'utiliser. Notons que l'accès à une ressource n'est garantie que pour une certaine durée de δ secondes. Au delà de cette durée la ressource utilisée pourra être pré-emptée pour satisfaire une nouvelle demande.

Partie 1

On considère tout d'abord qu'il y a un nombre total de N ressources, toutes identiques, numérotées de 0 à $N-1$ et que chaque utilisateur ne peut pas utiliser plus d'une seule ressource à la fois.

Lorsqu'un utilisateur u souhaite **acquérir** une ressource à une date t alors la politique suivante doit être appliquée :

- s'il existe des ressources disponibles (c'est-à-dire non utilisées), alors une (quelconque) de ces ressources est attribuée à l'utilisateur u à partir de la date t ;
- si aucune ressource n'est disponible alors :
  - si toutes les ressources sont utilisées depuis moins de δ secondes alors la demande d'attribution de l'utilisateur u est rejetée ;
  - s'il existe des ressources utilisées depuis plus de δ secondes alors la ressource i **utilisée depuis le plus longtemps** est pré-emptée et attribuée à l'utilisateur u à partir de l'instant t.

Lorsqu'un utilisateur u souhaite **libérer** une ressource r :

- si celle-ci est toujours attribuée à u alors elle est rendue disponible ;
- sinon rien n'est modifié (la ressource r a eté pré-emptée)

Pour implémenter cette bibliothèque on propose d'utiliser le lexique suivant :

N : la constante entière 255
Delta : la constante entière 360
Ressource : le type 0..N-1
Utilisateur : le type entier ≥ 0
Attribution : un tableau sur Ressource de couples <Utilisateur, Date>

Le tableau Attribution indique si une ressource est attribuée ou non :

- si Attribution$_i$ vaut (u,0), alors la ressource i est disponible ;
- si Attribution$_i$ vaut (u,t) avec $t \neq 0$, alors la ressource i est attribuée à l'utilisateur u depuis la date t

1. un ensemble de fonctions C

Les trois primitives suivantes doivent être fournies par la bibliothèque ResAlloc :

initialiser : une action
{état initial : indifférent}
{état final : toutes les ressources sont disponibles}

acquérir : une action (la donnée u : un Utilisateur, le résultat r : une Ressource ; le résultat ok : un booléen)
{état initial : indifférent}
{état final : la ressource r est attribuée à l'utilisateur u selon la politique spécifiée ci-dessus, ok vaut faux ssi aucune ressource n'a pu être attribuée}

libérer : une action (la donnée u : un Utilisateur, la donnée r : une Ressource)
{état initial : indifférent}
{état final : si la ressource r est attribuée à u alors elle est rendue disponible}

On dispose par ailleurs d'une bibliothèque `date` dont l'interface est fournie par le fichier `date.h` dont le contenu est le suivant :

```
// une date est exprimée en seconde depuis l'origine des dates fixée au 01/01/1970
typedef unsigned int Date ;

Date date() ;
// renvoie la date courante (en secondes depuis l'origine des dates)

void attendre (Date x) ;
// interrompt l'exécution du programme pendant x secondes
```

On suppose que les fonctions `date` et `attendre` sont implémentées dans un fichier `date.c` qu'il n'est pas demandé d'écrire.

Q1. Ecrivez le fichier `ResAlloc.h` contenant une implémentation en C du lexique et les **en-têtes** des trois primitives fournies par la bibliothèque ResAlloc. Expliquer votre choix de représentation du type Ressource.

Q2. Ecrivez le contenu du fichier `ResAlloc.c` contenant une implémentation en C du **corps** des trois primitives fournies par la bibliothèque ResAlloc. Le cas échéant vous pouvez également utiliser (en les écrivant !) d'autres fonctions intermédiaires pour rendre votre code plus lisible ....

Q3. Ecrivez le contenu complet du fichier `test.c` qui fournit un programme principal permettant :

1. d'appeler l'action Initialiser
2. de répéter **en permanence** :
   acquérir une ressource par l'utilisateur 1
   attendre 200 seconde
   acquérir une ressource par l'utilisateur 2
   attendre 300 secondes
   si l'utilisateur 1 a pu acquérir une ressource alors il la libère
   attendre 100 secondes
   si l'utilisateur 2 a pu acquérir une ressource alors il la libère

Q4. Ecrivez un Makefile permettant la **compilation séparée** de l'ensemble des fichiers nécessaires à la génération du programme exécutable `test`.

Partie 2

On considère maintenant que les N ressources ne sont pas identiques – elles sont donc choisies par les utilisateurs – et que le **nombre total d'utilisateurs par ressource** (à un instant donné) est limité à P. On suppose toujours que chaque utilisateur ne peut pas utiliser plus d'une seule ressource à la fois.

Lorsqu'un utilisateur u souhaite **acquérir** une ressource r à une date t alors la politique suivante doit être appliquée :

- s'il y a moins de P utilisateurs utilisant la ressource r alors celle-ci est attribuée au nouvel utilisateur u à partir de l'instant t ;
- sinon :
  - si tous les P utilisateurs de la ressource r l'utilisent depuis moins de δ secondes alors la demande d'attribution de u est rejetée ;
  - sinon la ressource r est attribuée à l'utilisateur u **aux dépens** de l'utilisateur qui utilisait cette ressource r **depuis le plus longtemps**.

Lorsqu'un utilisateur u souhaite **libérer** une ressource r :

- si celle-ci est toujours attribuée à u alors elle est rendue disponible ;
- sinon rien n'est modifié (la ressource r a eté pré-emptée)

On complète et modifie le précédent lexique de la façon suivante :

- on ajoute la constante entière P de valeur 10
- Attribution$_i$ fournit la liste des utilisateurs qui utilisent la ressource i et depuis quelle date ; c'est donc une **liste de couples** (u,t) où u est un utilsateur et t la date depuis laquelle u utilise la ressource i

Les trois primitives suivantes doivent être fournies par la bibliothèque ResAlloc :

initialiser : une action
{état initial : indifférent}
{état final : toutes les ressources sont disponibles}

acquérir : une action (la donnée u : un Utilisateur, le résultat r : une Ressource ; le résultat ok : un booléen)
{état initial : indifférent}
{état final : la ressource r est attribuée à u selon la politique spécifiée ci-dessus, ok vaut faux ssi aucune ressource n'a pu être attribuée}

libérer : une action (la donnée u : un utilisateur, la donnée r : une Ressource)
{état initial : indifférent}
{état final : si la ressource r est attribuée à u alors elle est rendue disponible}

Q5. Ecrivez le nouveau fichier `ResAlloc.h` contenant une implémentation en C du lexique et les **en-têtes** des trois primitives fournies par la bibliothèque ResAlloc. Pour représenter le contenu du tableau Attribution vous utiliserez des **listes chaînées**.

Q6. Ecrivez le nouveau fichier `ResAlloc.c` contenant une implémentation en C du **corps** des trois primitives fournies par la bibliothèque ResAlloc. Le cas échéant vous pouvez également utiliser (en les écrivant !) d'autres fonctions intermédiaires pour rendre votre code plus lisible ....

Q7. Le type `Date` est implémenté dans `date.h` par un type entier (`unsigned int`), chaque date étant représentée par le nombre de secondes écoulées depuis le 01/01/1970. Ce choix peut-il poser un problème en pratique ? Justifiez votre réponse en quelques lignes ....