Perfectionnement à la programmation en C
Fiche de TP 3

L2 Informatique 2020-2021

Chronomètre

Ce TP se déroule en **une seule séance** et est à faire par binômes. Le travail réalisé doit être envoyé au plus tard exactement **une semaine après** le début de la séance de TP. Il sera disposé sur la plate-forme prévue à cet effet et constitué des programmes répondant aux questions et des éventuels fichiers annexes qui peuvent être demandés.

L'objectif de ce TP est d'implanter un chronomètre muni de quelques fonctions plus avancées (minuteur, comptage de tours, *etc.*). Nous introduisons et/ou approfondissons ici des notions relatives à

— l'interface NCURSES en gestion en temps réel ;

— la manipulation et la mesure du temps ;

— la mise en place de pré-assertions ;

— la consolidation d'un programme et la prévention d'erreurs provoquées par l'utilisateur.

1 Spécification

L'application à programmer est un chronomètre qui permet de mesurer des durées allant de 0 secondes à 100 heures moins 1 centième de seconde, au centième de seconde près. Il sera muni des fonctionnalités suivantes :

— décompte du temps en temps réel ;

— mise en pause du chronomètre ;

— mise à zéro du chronomètre ;

— marquage de tours ;

— fonction de minuterie paramétrable avec avertissement.

Ces fonctionnalités seront décrites en détails un peu plus loin et sont organisées en une fenêtre NCURSES devant avoir l'allure

```
                          == Chronometre ==

                Tour 1   :  0 :  0 :  2 : 10
                Tour 2   :  0 :  0 :  3 : 40
                           0 :  0 :  6 : 25
           Avertissement :  0 :  2 :  5 :  0
------------------------------------------------------------------------
Espace : lancer / mettre en pause
r      : reinitialiser
t      : marquer tour
F1/F2  : incrementer / decrementer heure avertissement
F3/F4  : incrementer / decrementer minute avertissement
F5/F6  : incrementer / decrementer seconde avertissement
q      : quitter
```

L'application doit fonctionner de la façon suivante. Lorsqu'elle vient d'être lancée, le chronomètre est à l'arrêt, aucune marque de tour n'est affichée et l'avertissement pour la minuterie est paramétré sur 25 secondes. L'interface présentée est donc

```
                          == Chronometre ==




                           0 :  0 :  0 :  0
           Avertissement :  0 :  0 : 25 :  0
------------------------------------------------------------------------
Espace : lancer / mettre en pause
r      : reinitialiser
t      : marquer tour
F1/F2  : incrementer / decrementer heure avertissement
F3/F4  : incrementer / decrementer minute avertissement
F5/F6  : incrementer / decrementer seconde avertissement
q      : quitter
```

Une pression sur la touche espace lance le chronomètre qui s'affiche en temps réel. Une nouvelle pression met la mesure du temps en pause et ainsi de suite. À chaque instant, une pression sur la touche `'r'` remet le chronomètre à zéro. Ce faisant, il se trouve mis en pause. De plus, à chaque instant (même lorsque le chronomètre est en pause), une pression sur la touche `'t'` fait apparaître sur une ligne au dessus du chronomètre l'état courant de la mesure de temps, accompagnée du message `"Tour N :"` où `N` est le nombre de fois que la touche `'t'` a été pressée depuis le début du programme. Finalement, la fonctionnalité d'avertissement fonctionne de la manière suivante. Lorsque le chronomètre vient de dépasser le temps dédié pour l'avertissement, l'écran émet un avertissement visuel. Il est possible de paramétrer ce temps à l'aide des touches `F1`, `F3` et `F5` qui respectivement incrémentent les heures, minutes et secondes, et `F2`, `F4`, `F6` qui respectivement les décrémentent.

Par exemple, en lançant l'application, appuyant sur la touche espace, attendant environ 7 secondes, appuyant ensuite sur `'t'` trois fois à environ 2 secondes d'intervalle, l'interface ressemblera à

```
                          == Chronometre ==
                Tour 5   :  0 :  0 :  7 : 15
                Tour 6   :  0 :  0 :  9 : 40
                Tour 7   :  0 :  0 : 11 : 52
                           0 :  0 : 12 : 34
           Avertissement :  0 :  0 : 25 :  0
--------------------------------------------------------------------------
Espace : lancer / mettre en pause
r      : reinitialiser
t      : marquer tour
F1/F2  : incrementer / decrementer heure avertissement
F3/F4  : incrementer / decrementer minute avertissement
F5/F6  : incrementer / decrementer seconde avertissement
q      : quitter
```

Voici, pour finir cette explication générale, quelques conventions de plus à respecter :

— lorsque le chronomètre arrive à 99 heures, 59 minutes, 59 secondes, et 99 centièmes de seconde, il retourne à zéro ensuite (sans se mettre en pause) ;

— le temps dédié pour l'avertissement est d'au moins d'une seconde et pas plus de 99 heures, 59 minutes et 59 secondes ;

— le nombre d'informations de tour dépend de la hauteur de la fenêtre mais ne doit jamais dépasser six lignes. Il faut de plus de la place pour afficher au moins deux lignes de telles informations et l'information renseignant sur le dernier tour apparaît le plus en bas. Si plus d'informations de tours doivent être affichées que de nombre de lignes disponibles, ces informations sont décalées (par exemple, s'il est possible de renseigner au plus trois lignes et que trois informations de tours sont déjà affichées, une nouvelle demande d'affichage fait que ce sont, de haut en bas, les information du 2^e, 3^e et 4^e tour qui sont affichées) ;

— la fenêtre dans laquelle l'application est lancée doit avoir un nombre de lignes minimal de 14 et un nombre de colonnes minimal de 58. Quand ces conditions ne sont pas vérifiées, l'exécution s'interrompt proprement et affiche un message d'erreur ;

— l'application doit être sensible au redimensionnement de la fenêtre : lorsque la la géométrie de la fenêtre est modifiée, ses éléments doivent s'y replacer harmonieusement.

2 Écriture du programme

Exercice 1. (Gestion du temps)

Concernant la gestion du temps, nous utilisons ici uniquement la fonction

```c
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

accessible depuis le fichier d'en-tête `sys/time.h`. Cette fonction permet d'obtenir des informations temporelles avec une précision théorique proche de la microseconde. Elle s'utilise de la façon suivante

```c
/* Variables de recuperation du temps. */
struct timeval temps_debut, temps_fin;

/* Variables pour recueillir les mesures. */
int sec, usec;
...
...

/* Mesure de temps initiale. */
gettimeofday(&temps_debut, NULL);
...
/* Mesure de temps finale. */
gettimeofday(&temps_fin, NULL);

sec = temps_fin.tv_sec - temps_debut.tv_sec;
usec = temps_fin.tv_usec - temps_debut.tv_usec;
```

Comme nous pouvons l'observer, nous appellerons toujours ici la fonction `gettimeofday` avec `NULL` comme 2e argument. Son premier paramètre est un pointeur vers une variable d'un type structuré qui est modifiée de sorte à prendre comme valeur le temps passé depuis une certaine date **fixée** dans le passé. Ainsi, comme réalisé en lignes 15 et 16, calculer les différences entre les champs secondes `tv_sec` et microsecondes `tv_usec` permet de connaître le temps passé entre l'exécution de la ligne 10 et la de la ligne 13.

1. Écrire une fonction

   ```c
   int intervalle_ms(struct timeval debut, struct timeval fin);
   ```

   qui renvoie en millisecondes la durée écoulée entre `debut` et `fin`. On décide d'adopter pour toute la suite la **milliseconde** comme **unité de temps de base**.

2. Écrire les quatre fonctions

   ```c
   int nb_ms_vers_centiemes(int nb_ms);
   int nb_ms_vers_secondes(int nb_ms);
   int nb_ms_vers_minutes(int nb_ms);
   int nb_ms_vers_heures(int nb_ms);
   ```

   qui renvoient respectivement le nombre de centièmes de seconde, secondes, minutes ou heures selon le nombre de millisecondes `nb_ms` donné. Attention, le nombre de centièmes de seconde renvoyé doit être compris entre 0 et 99, et les nombres de secondes et de minutes renvoyés doivent être compris entre 0 et 59. Ainsi par exemple, l'appel `nb_ms_vers_centiemes(1213)` renvoie

   $$\frac{1213}{10} \pmod{100},$$

   ce qui donne 21. Encore en guise d'exemple, l'appel `nb_ms_vers_minutes(6404440)` renvoie

   $$\frac{6404440}{1000 \times 60} \pmod{60},$$

   ce qui donne 46.

3. Pour se familiariser avec ces notions, écrire un programme `ChronoSimple.c` qui, lorsqu'il est lancé, affiche sur la **sortie standard** le temps écoulé depuis le lancement du programme avec une fréquence de mise à jour d'une fois toutes les 500 millisecondes. L'utilisateur peut interrompre le programme (par `Ctrl + C`). Le format d'affichage est

   `H : M : S : C` (1)

où `H` est le nombre d'heures, `M` de minutes, `S` de secondes et `C` de centièmes de seconde.

Utiliser la fonction

```
int usleep(int usec);
```

de `unistd.h`, déjà rencontrée dans les TP précédents et qui permet de suspendre l'exécution pendant `usec` microsecondes.

4. Enrichir le programme de la question précédente pour en faire un nouveau programme `ChronoMoyen.c` qui peut être mis en pause et utilise la librairie NCURSES. Plus précisément, le programme affiche sous le format (1) initialement un chronomètre à zéro et attend pour démarrer que l'utilisateur appuie sur espace. À chaque pression de la touche espace, le chronomètre change d'état entre décompter le temps et être en pause.

   Un algorithme possible, pour mettre en place ce programme, est le suivant.

   (1) Affecter 0 à une variable `duree_totale` ;

   (2) positionner l'état du chronomètre à « en pause » ;

   (3) tant que le programme est lancé :

   (a) si la touche espace est pressée :

   (i) si le chronomètre est en pause :

   — lire le temps courant dans `temps_debut` ;

   — positionner l'état du chronomètre à « en activité » ;

   (ii) sinon :

   — positionner l'état du chronomètre à « en pause » ;

   (b) attendre un certain délai de temps ;

   (c) si le chronomètre est en activité :

   (i) lire le temps courant dans `temps_fin` ;

   (ii) incrémenter `duree_totale` de l'intervalle entre `temps_debut` et `temps_fin` ;

   (iii) affecter à `temps_debut` la valeur de `temps_fin` ;

   (d) rafraichir l'affichage du chronomètre en utilisant le contenu de `duree_totale`.

   Vérifier l'exactitude du chronomètre en utilisant un autre chronomètre sur des périodes de temps assez longues. Tester divers délais de rafraichissement de l'affichage. Il faut également être vigilant sur les erreurs d'arrondi provenant des calculs de division de conversion du temps en microseconde vers les millisecondes.

Exercice 2. (État de l'application et graphisme)

L'état d'un chronomètre est l'amalgame des données suivantes :

— l'information de l'état du chronomètre (`1` si en activité et `0` sinon) ;

— la durée totale écoulée, exprimée en millisecondes ;

— la durée en millisecondes provoquant un avertissement ;
— l'indice du dernier tour enregistré (initialement `0`) ;
— le nombre de tours enregistrés (initialement `0`) ;
— un tableau statique de taille raisonnable contenant les temps en millisecondes de chacun des tours enregistrés.

1. Déclarer un type structuré `Chronometre` qui permet de représenter un chronomètre.
2. Définir une fonction
```c
Chronometre initialiser_chronometre();
```
qui renvoie un chronomètre dans l'état tel qu'il doit l'être lors du lancement de l'application.
3. Écrire une fonction
```c
void afficher_duree(int y, int x, int nb_ms);
```
qui affiche à partir du point `(y, x)` de la fenêtre la durée `nb_ms` sous le format (1).
4. Écrire une fonction
```c
void afficher_interface(Chronometre chrono);
```
qui affiche sur la fenêtre l'état du chronomètre suivant ce qui est spécifié au début du projet. Il n'est pas obligatoire d'avoir une interface identique à celle donnée : toute amélioration ou variante pertinente est la bienvenue.
5. Écrire une fonction
```c
void afficher_flash();
```
qui affiche sur la fenêtre rapidement et de manière alternée des caractères `'*'` de couleurs différentes. Cette fonction sera utilisée pour avertir que la durée spécifiée par le minuteur est arrivée à son terme.
6. Écrire une fonction
```c
void ajouter_tour(Chronometre *chrono);
```
qui modifie le chronomètre pointé par `chrono` pour faire en sorte d'enregistrer le tour par la durée totale courante. Pour cela, il faut incrémenter l'indice du dernier tour enregistré, incrémenter le nombre de tours enregistrés (si ce nombre ne dépasse pas une certaine limite fixée), décaler le tableau statique contenant les informations des tours d'une case vers la droite et affecter à la première case de ce tableau la durée totale courante. De cette façon, le tableau contient les informations des tours du plus récent au plus ancien.

Exercice 3. (Chronomètre, étape par étape)

Nous allons maintenant mettre au point véritablement l'application décrite au début du sujet. Pour cela, nous allons procéder étape par étape en augmentant les fonctionnalités mises au point.

Le programme à écrire est nommé `Chrono.c` et, comme point de départ, est constitué du programme réalisé dans la question 4 de l'exercice 1, augmenté des types et fonctions implantés dans l'exercice 2.

1. Incorporer un mécanisme permettant de quitter l'application sur pression de la touche `'q'`.
2. Incorporer un mécanisme permettant de réinitialiser le chronomètre sur pression de la touche `'r'`. La réinitialisation ne porte que sur la durée écoulée que témoigne le chronomètre et les informations des tours enregistrés. Elle ne doit pas changer la durée du minuteur provoquant l'avertissement. Elle a aussi pour effet de mettre le chronomètre en pause.
3. Incorporer un mécanisme permettant de marquer un tour sur pression de la touche `'t'`. Ceci enregistre dans la case dédié du champ adéquat du chronomètre la durée courante et l'affiche comme expliqué préalablement.
4. Incorporer un mécanisme permettant de produire un avertissement visuel (réalisé par la fonction `afficher_flash` de l'exercice 2) lorsque la durée mesurée par le chronomètre vient de dépasser la durée spécifiée du minuteur. La mesure du temps réalisée par le chronomètre ne doit pas être altérée par ce processus.
5. Incorporer un mécanisme permettant de modifier la durée du minuteur à l'aide des touches `F1`, …, `F6` comme expliqué précédemment.
6. Consolider l'application au maximum pour éviter que des cas non prévus provoquent des erreurs d'exécution. Prévoir ce qu'il se passe en cas de redimensionnement de la fenêtre, en cas de nombre de tours à marquer trop élevé, en cas de durée totale enregistrée trop grande, *etc.* Ne surtout pas oublier de **mettre en place les pré-assertions** nécessaires à toutes les fonctions du projet.