Licence Informatique 2014-2015 **J1INAW02 - Programmation 1**
Université de Bordeaux **Devoir surveilé du 13-11-2014**

Éléments de correction

Exercice 1. Questions de cours

1. Expliquer ce que représentent l'interface et l'implémentation d'un module.
   Points à évoquer :

   Interface : contenu (les prototypes des fonctions du module, éventuellement des déclarations de types, des inclusions de fichiers, ..), suffixe du nom (.h), rôle joué dans un programme *client*.

   Implémentation : contenu (des définitions de fonctions du module, dont le prototype se trouve dans l'interface, des définitions de fonctions auxiliaires (locales au fichier implémentation), des déclarations de types et des variables (locales au fichier implémentation) , des inclusions de fichiers d'en-tête, suffixe du nom (.c).

2. Quel est l'intérêt de la compilation séparée ?
   Points à évoquer :

   découpage du programme en plusieurs fichiers correspondants aux différentes composantes logiques : structuration en modules réutilisables.

   lorsqu'on modifie seulement une partie du programme, seuls les fichiers concernés par les modifications doivent être recompilés.

Exercice 2. Fonctions *"echanger"*

— **Première version :** fonction qui prétend échanger, mais elle n'échange que le contenu de ses paramètres, qui sont de variable locales (et ont donc la vie courte....) :

```
void echange-version-1(int x, int y)
{
  int tmp;

  tmp = x;
  x   = y;
  y   = tmp;
}
```

Si on appelle : `echange-version-1(a,b);` après avoir défini, par exemple : `{int a = 4, b = 8;}`, les contenus de `a` et de `b` seront copiés respectivement dans les deux variables `x` et `y`, locales à la fonctions `echange-version-1` et donc stockées sur la pile d'éxécution. Les contenus de `x` et `y` sont bien échangés, mais cela n'a aucun impact sur les contenus de `a` et de `b`. À la fin de l'exécution de la fonction `echange-version-1` ses variables sont hors portée pour le programme, donc toute trace de l'echange est perdue. L'instruction : `{printf("a = %d, b = %d\n",a,b);}` affiche donc, dans l'exemple, `a = 4, b = 8` .

— **Deuxième version :** la seule qui a des effets de bord, car elle échange effectivement les contenus de `a` et de `b` !

```
void echange-version-2(int *x, int *y)
{
  int tmp;

  tmp = *x;
  *x  = *y;
  *y  = tmp;
}
```

Si on appelle : `echange-version-2(&a,&b);` après avoir défini, par exemple : `{int a = 4, b = 8;}`, les variables `x` et `y` contiennent respectivement les adresses de `a` et de `b`. On dit que `x` et `y` "pointent" sur `a` et sur `b` respectivement. Donc :

`tmp = *x;` copie dans la variable `tmp` la valeur de la variable pointée par `x`, c'est à dire la valeur de `a` (`tmp` prend la valeur 4)

`*x = *y;` copie dans la variable pointée par `x`, c'est à dire `a`, le contenu de la variable pointée par `y`, c'est à dire `b`. La variable `a` prend donc la valeur 8.

`*y = tmp;` copie dans la variable pointée par `y`, c'est à dire `b`, le contenu de `tmp` : `b` prend donc la valeur 4.

L'instruction : `{printf("a = %d, b = %d\n",a,b);}` affiche donc, dans l'exemple, `a = 8, b = 4` .

— **Troisième version :** elle ne modifie que des pointeurs, qui sont des variables locales, et non pas les variables pointées !

```
void echange-version-3(int *x, int *y)
{
  int *tmp;

  tmp = x;
  x   = y;
  y   = tmp;
}
```

Suite à :
`{int a = 4, b = 8; echange-version-3(&a,&b);}`
les variables `x` et `y` pointent respectivement sur `a` et sur `b`. Donc :

`int *tmp;` déclare une variable de type pointeur sur un entier.

`tmp = x;` copie dans `tmp` le contenu de `x`, c'est à dire l'adresse de `a` ; `tmp` et `x` sont deux pointeurs qui pointent sur `a`.

`x = y;` copie dans `x` le contenu de `y`, c'est à dire l'adresse de `b` ; aussi bien `x` que `y` pointent sur `b`.

`y = tmp;` copie dans `y` le contenu de `tmp`, c'est à dire l'adresse de `a` ; aussi bien `y` que `tmp` pointent sur `a`.

L'instruction : `{printf("a = %d, b = %d\n",a,b);}` affiche donc, dans l'exemple, `a = 4, b = 8` car seulement les contenus de `x` et de `y` ont été modifiés par la fonction, et non pas les contenus des variables pointées par `x` et de `y`.

— **Quatrième version :** avertissement à la compilation [1] et "erreur de segmentation" [2]

```
void echange-version-4(int *x, int *y)
{
  int *tmp;

  *tmp = *x;
  *x   = *y;
  *y   = *tmp;
}
```

1. si on compile avec l'option `-Wall` on obtient le message `'tmp' is used uninitialized in this function`, d'où l'utilité de cette option...

2. interruption anormale d'un programme suite à un tentatif d'accès à un emplacement mémoire qui ne lui avait pas été alloué ; ici on essaye d'affecter une valeur à la variable pointée par `tmp`, mais `tmp` n'a pas été initialisée et contient en général une adresse indeterminée correspondant à une zone mémoire que le programme ne peut pas modifier

Suite à :
`{int a = 4, b = 8; echange-version-4(&a,&b);}`
les variables `x` et `y` pointent respectivement sur `a` et sur `b`. Donc :

`int *tmp;` déclare une variable de type pointeur sur un entier.

`*tmp = *x;` on essaye de copier dans la variable pointée par `tmp` le contenu de la variable pointée par `x` : mais quelle est la variable pointée par `tmp` ? ? ? Erreur de segmentation ! ! ! Moralité : attention à initialiser les pointeurs et à les initialiser avec des adresses valides !

Exercice 3. Module *temps*

1. Implémentation du module

```
#ifndef _TEMPS_H
#define _TEMPS_H


typedef long int temps;

extern long int convertirEnMinutes(temps t);
extern long int convertirEnHeures(temps t);

extern void afficherTemps(temps t);

#endif



// temps.c

#include <stdio.h>

#include "temps.h"

long int
convertirEnMinutes(temps t){
  return t/60;
}

long int
convertirEnHeures(temps t){
  return t/3600;
}

void
afficherTemps(temps t){
  printf("Le temps est de %ld secondes\n", t);
}
```

2. Programme de test

```
// programme de test: test-temps.c

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <assert.h>

#include "temps.h"

typedef struct horaire{
  long int heures;
  long int minutes;
  long int secondes;
}horaire;

void echangerTemps(temps *tabTemps, int tailleTab, int ix1, int ix2);
bool estCroissant(temps *tabTemps, int tailleTab);
horaire convertirHoraire(temps t);
void afficherHoraire(horaire h);



static void usage(char * commande){
  fprintf(stderr,"usage: %s temps1 temps2 ...\n", commande);
  exit(EXIT_FAILURE);
}

int
main(int argc, char *argv[]){
  if (argc < 2)
    usage(argv[0]);

  int taille = argc-1;
  temps *tab= malloc(taille*sizeof(temps));
  assert(tab!=NULL);

  for (int i=0; i<argc-1; i++){
    tab[i] = atoi(argv[i+1]);
    afficherTemps(tab[i]);
    printf("temps numero %d en minutes: %ld \n", i, convertirEnMinutes(tab[i]));
    printf("temps numero %d en heures: %ld \n", i, convertirEnHeures(tab[i]));
```

```
  }

  echangerTemps(tab,taille,0,taille-1); // on echange le premier et le dernier
  for (int i=0; i<taille; i++)
    afficherTemps(tab[i]);

  printf("%s croissant\n", estCroissant(tab,taille)?"est":"n'est pas");

  for (int i=0; i<taille; i++)
    afficherHoraire(convertirHoraire(tab[i]));

  free(tab);

  return EXIT_SUCCESS;
}


void
echangerTemps(temps *tabTemps, int tailleTab, int ix1, int ix2){
  assert(ix1>=0 && ix2>=0 && ix1<tailleTab && ix2<tailleTab);
  temps aux = tabTemps[ix1];
  tabTemps[ix1]=tabTemps[ix2];
  tabTemps[ix2]= aux;
}

bool
estCroissant(temps *tabTemps, int tailleTab){
  for (int i=1; i<tailleTab; i++)
    if (tabTemps[i-1] > tabTemps[i])
      return false;
  return true;
}

horaire
convertirHoraire(temps t){
  horaire h;
  h.heures= convertirEnHeures(t);
  long int aux = t - 3600 * h.heures;
  h.minutes= convertirEnMinutes(aux);
  h.secondes= aux - (h.minutes) * 60;
  return h;
}

void
afficherHoraire(horaire h){
  printf("%ld h %ld min %ld sec \n", h.heures, h.minutes, h.secondes);
}
```

Pour obtenir un exécutable du programme de test il faut générer le fichier objet de `test-temps.c` et ensuite faire l'édition des liens avec le fichier objet du module.

Pour obtenir le fichier objet du module :
`gcc -c -std=c99 -Wall temps.c`

Pour obtenir le fichier objet du programme de test :
`gcc -c -std=c99 -Wall test-temps.c`

L'édition de liens :
`gcc test-temps.o temps.o -o test-temps`