Fiche n° 6 de TP

Instructions itératives (1)

Objectifs : manipulation des instructions itératives `for` et `while` ; itération d'une instruction sur une suite de valeurs.

Prérequis : syntaxe des instructions itératives `for` et `while`.

Travail minimum : exercices 1 à 3.

Exercice 1

Téléchargez le fichier `boucle.c` à partir de la page universitice :

TP6_src/boucle.c

```c
/**
 * Équipe pédagogique BPI
 * Exemple de boucle
 * ex 1 tp 6
 */

/* Appel des bibliothèques */

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

/* Déclarations des fonctions */

int main(void) {
  /* Invariant de boucle :
   * ??? <= k && k <= ??? && les entiers de ??? à ??? ont été affichés
   */
  for (int k = 0; k < 10; ++k ) {
    printf("%d\n", k);
  }
  return EXIT_SUCCESS;
}

/* Définitions des fonctions */
```

TP6_src/boucle.c

Que fait ce programme ? Complétez l'invariant de boucle

Exercice 2

La factorielle d'un entier n est notée $n!$. Elle est définie par

$$\forall n \geqslant 0, \quad n! = \begin{cases} 1 & \text{si } n = 0, \\ 1 \times 2 \times \cdots \times n & \text{sinon.} \end{cases}$$

La factorielle d'un nombre strictement négatif n'est pas définie

1) Récupérez le fichier `factorielle.c` sur universitice. C'est le fichier qu'il faudra compléter.

2) Complétez les assertions d'entrées et de sortie de la fonction `fact` dans les commentaires. Faites attention à bien prendre en compte le domaine de définition de la factorielle et aux limites du type `int`.

3) Complétez son prototype.

4) Écrivez le corps de la fonction en respectant l'invariant de boucle proposé. Quelle doit être la condition d'arrêt ?

5) Utilisez des `assert` pour provoquer une sortie du programme si l'assertion d'entrée n'est pas respectée.

6) Écrivez ensuite un programme qui prend en entrée un entier positif n, puis qui calcule et affiche $n!$.

Exercice 3

1) Téléchargez le répertoire `Chrono` à partir du site universitice. Ouvrez-le et observez les différents fichiers qui le composent.

2) Ouvrez un terminal dans ce répertoire et tapez la commande `make`. Qu'observez vous ? Exécutez le programme `chrono`.

3) Écrivez le corps des fonctions `sub_1s` et `time_is_zero` dans le fichier `ext_time.c` en respectant les spécifications du fichier `timeio.h`.

4) Modifiez le fichier `Makefile` afin qu'il compile le fichier `ext_time.c`. Modifiez le fichier `main.c` afin de tester les deux fonctions que vous avez écrites.

5) Modifiez le fichier `main.c` afin que le programme `chrono` demande à l'utilisateur de saisir une durée sous la forme « heures:minutes:secondes », puis qui affiche le décompte à chaque seconde. Pour cela, on fera appel à `sleep(1)` qui attend 1 seconde avant de poursuivre l'exécution du programme. La fonction `sleep` est déclarée dans le fichier d'en-tête `unistd.h`.
Remarque : Sous windows, la fonction `sleep` n'est pas reconnue. Vous devez remplacer `sleep(1)` par `Sleep(1000)` et `#include <unistd.h>` par `#include <windows.h>`

```
./chrono
Combien de temps au chrono (hh:mm:ss) ?
0:1:1
00:01:01
00:01:00
00:00:59
00:00:58
 ...
00:00:07
00:00:06
00:00:05
00:00:04
00:00:03
00:00:02
00:00:01
```

Indication : Vous pourrez respecter un invariant de la forme

```
/* Invariant de boucle:
   * Tous les horaires entre init_val(h) : init_val(m) : init_val(s)
   * et h : m : s + 1 seconde ont été affichés
   */
```

avec comme condition d'arrêt `(h, m, s) == (0, 0, 0)`.

Exercice 4

Téléchargez le fichier `puissance.c` à partir de la page UniversiTICE :

TP6_src/puissance.c

```c
/**
 * Équipe pédagogique BPI
 * Lit un entier n sur l'entrée standard puis affiche ???
 */

/* Appel des bibliothèques */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

/* Déclarations des fonctions et des macros */

#define DEBUG
#define STEP 5

/* Fonction principale */
int main(void) {
  int x;
  if (scanf("%d", &x) != 1) {
    printf("Erreur de saisie\n");
    return EXIT_FAILURE;
  }

  int v = 1;
  /* Invariant de boucle:
   * ?? <= k && k <= ?? && v == ??
   * (les entiers  ???? ont été affichés ^^ DEBUG n'est pas défini)
   */
  for (int k = 1; k <= x; ++k) {
    v = 2 * v;
    /* En mode DEBUG
     * affiche la progression du calcul
     */
    #ifdef DEBUG
      if (k % STEP == 0) {
        printf("%d", k);
        fflush(stdout);
        sleep(1);
        printf("\r");
      }
    #endif
  }
  printf("\nLe résultat du calcul est %d\n", v);

  return EXIT_SUCCESS;
}

/* Définitions des fonctions */
```

TP6_src/puissance.c

1) Exécutez le avec les valeurs 0, 1, 3, 10 et 30 en *commentant* puis en *décommentant* la ligne `#define DEBUG`.

2) Étudiez l'utilisation des macroconstantes ainsi que celle de `#ifdef`.

3) Que signifie `\r` dans le `printf` ? Que se passe-t-il si on enlève la ligne `fflush` ?

4) À quoi sert `sleep` ?

5) Commentez la ligne avec `sleep`, enlever le commentaire de la ligne `#define DEBUG` et donnez la valeur `1000000` à la macroconstante `STEP`. Exécutez le programme avec la valeur `2147483647` en entrée.

6) Mettez la ligne `#define DEBUG` en commentaire et recommencez la même opération. Que constatez vous ? Proposez une solution pour que le comportement soit le même dans les deux cas.

7) En fait, les problèmes arrivent déjà pour des valeurs plus petites. Que se passe-t-il pour les entrées 31, 32, 33, ... ? Expliquez.

8) Empêchez ce comportement en ajoutant une instruction `assert` appropriée.

9) Complétez l'invariant de boucle. En déduire une expression de la valeur de `v` affichée par l'instruction `printf`.