Exercices de langage C (TP1)

1. Premier programme

Afin de vous familiariser avec l'environnement de travail utilisé à ce cours, il vous est demandé de :

- Vous connecter au serveur courslinux (Debian) dans VS Code
  → cf. ssh_vscode_courslinux.pdf
- Ouvrir un terminal et créer un répertoire de travail pour ce cours ainsi qu'un sous-répertoire pour la première séance (ex : `~/langageC/TP1`)
- Écrire un programme C qui affiche le message de votre choix à l'écran et le sauvegarder dans le répertoire du cours sous le nom « `ex1.1.c` »
- Compiler votre programme à l'aide de la commande *cc* (cf. man gcc)
- Lancer le fichier exécutable généré et vérifier si le comportement est correct
- Recompiler votre programme en spécifiant le nom de l'exécutable généré et tester son exécution.

2. Connaissances de base : entrées/sorties

Pour ces exercices, utilisez les fonctions de la librairie standard C : `scanf`[1] pour lire une valeur numérique à l'entrée standard (i.e. clavier) et `printf`[2] pour afficher les résultats à la sortie standard (i.e. l'écran). Consultez le manuel `man` ou le chapitre 6 du eSyllabus pour avoir plus d'infos sur ces fonctions.

a) Écrire un programme qui saisit deux entiers au clavier et affiche leur produit.

b) Modifiez le programme pour que les nombres soient des réels.

c) Écrire un programme qui échange les valeurs de deux variables entières dont les valeurs sont lues au clavier. Affichez les valeurs des variables avant et après l'échange.

d) Écrire un programme qui affiche les codes ASCII (`man 7 ascii` pour en savoir plus) des lettres majuscules et des chiffres sous la forme suivante :

```
Caractère = 'A'    code déc. = 65    code hexa. = 41
Caractère = 'B'    code déc. = 66    code hexa. = 42
...
Caractère = '1'    code déc. = 49    code hexa. = 31
...
Caractère = '9'    code déc. = 57    code hexa. = 39
```

e) Écrire un programme qui affiche tous les diviseurs (plus grands ou égal à 1) d'un nombre entier strictement positif dont la valeur est lue au clavier.

Si l'entier lu n'est pas strictement positif, le comportement du programme est indéterminé. En d'autres termes, vous gérez ce cas comme bon vous semble.

---

1 La lecture d'une valeur entière au calvier se fait grâce à la fonction `scanf` avec le format `%d` comme dans l'exemple `scanf("%d",&variable)` où *`variable`* est l'identificateur d'une variable entière.

2 L'affichage du contenu d'une variable entière `i` se fait grâce à la fonction `printf` avec le format `%d` comme dans l'exemple `printf("Le contenu de la variable i est %d\n", i)`.

f) Écrire un programme qui simule l'opération de division entière (aussi appelée *division euclidienne*) entre deux entiers positifs a et b dont les valeurs sont saisies au clavier. On divise le plus grand par le plus petit, sans utiliser les opérateurs /, % ou *. Afficher le quotient et le reste.

   Si la plus petite valeur vaut 0, demandez à l'utilisateur d'entrer de nouvelles valeurs.

3. Connaissances de base : limites des types

En mathématiques, la fonction factorielle est définie comme suit :

```
0! = 1
n! = 1 * 2 * ... * (n-2) * (n-1) * n    (pour n>0)
```

Ecrivez un programme qui affiche[3] la factorielle d'un nombre. Il est demandé d'utiliser une boucle `for` pour effectuer le calcul et de ne pas définir, ni utiliser de fonctions. Voici l'algorithme :

```
fact = 1
pour i allant de 2 à n faire
      fact ← fact * i
```

Différentes variantes vous sont demandées :

a) la valeur du nombre n est lue au clavier avec `scanf` et il est demandé d'utiliser des nombres entiers (*`int`*) pour faire les calculs. Testez le programme avec les valeurs suivantes pour la constante :

   0 ⇒
   1 ⇒
   5 ⇒
   12 ⇒
   13 ⇒

   Quels résultats sont non plausibles ?

b) ajoutez dans votre programme l'affichage de la valeur de `INT_MAX` (la valeur maximale d'un `int`) et expliquez à partir de quelle valeur le calcul de la factorielle ne donne plus un résultat correct[4].

c) modifiez le programme en utilisant une variable de type *`double`* pour les calculs intermédiaires et le résultat final. Testez votre programme avec les mêmes valeurs qu'au point 1 et discutez les résultats[5].

---

3 L'affichage du contenu d'une variable entière se fait grâce à la fonction `printf` avec le format `%d` comme dans l'exemple `printf("Le contenu de la variable i est %d\n", i)` (cf. eSyllabus chap. 6)

4 La même problématique informatique est à l'origine du « bug de l'an 2038 » qui concerne les logiciels utilisant la représentation POSIX du temps.

5 L'affichage du contenu de la partie entière d'une variable réelle se fait grâce à la fonction `printf` avec le format `%.0f` comme dans l'exemple `printf("Le contenu de la partie entière de la variable double x est %.0f\n", x)` (cf. eSyllabus chap. 6)

d) modifiez votre programme du point b (calculs en `int`) afin que la boucle `for` s'interrompe en affichant un message d'erreur lorsque le calcul de factorielle dépasse les capacités du type `int`.

e) modifiez le programme en utilisant une boucle `while` (les trois parties constituant la boucle `for` - initialisation, condition, adaptation - doivent y être présentes).

f) modifiez le programme afin que le test de dépassement de capacité soit intégré à la condition de la boucle `while`.

4. Bonus: Lecture d'un nombre d'un seul chiffre avec la macro getchar

Reprenez le programme précédent qui calcule la factorielle de nombres entrés au clavier.

Modifiez votre programme afin de calculer la factorielle de plusieurs nombres successivement. Ces nombres sont formés d'un seul chiffre et sont introduits au clavier. Ils sont lus grâce à la macro `getchar` (attention, `getchar` lit des caractères, donc le code ASCII du chiffre[6]). Le programme s'arrête quand il rencontre la marque de fin de fichier `<Ctrl-D>`. Notez que la vérification de dépassement de capacité lié au type `int` ne doit pas être vérifiée dans le cas d'un seul chiffre.

Pour faciliter la compréhension du fonctionnement de `getchar`, affichez le message de debuggage suivant après chaque lecture :

```
printf("caractere lu '%c' - code ascii %d\n", c, c);
```

Ajoutez-y tous les tests nécessaires afin que seule l'introduction de nombres d'un seul chiffre soit acceptée. Pensez à :

- vérifier que la ligne lue n'est pas vide
- vérifier que le caractère lu est compris entre 0 et 9
- vérifier que la ligne lue ne comprend que 2 caractères : le chiffre décimal directement suivi du `return`
- vider le buffer de lecture au cas où l'utilisateur aurait introduit plus de deux caractères

Testez votre programme avec le fichier `entiers1.dta`. Pour ce faire, si votre exécutable est `factoriel`, tapez la commande suivante pour rediriger l'entrée standard de votre programme vers le fichier `entiers1.dta` :

```
./factoriel < entiers1.dta
```

Pour vérifier que votre programme traite correctement les données, comparez vos résultats avec le fichier d'output `entiers1.dta.out`.

6 Consultez le chapitre 6 de l'eSyllabus ou tapez la commande `man getchar` dans un terminal pour accéder à la page de manuel de la fonction.

5. Bonus : Lecture, avec la macro getchar, de nombres de 1 ou 2 chiffres

Ecrivez une autre version du programme précédant afin que celui-ci calcule la factorielle de nombres de un ou deux chiffres lus grâce à la macro `getchar`.

Ajoutez tous les tests nécessaires afin que seule l'introduction de nombres de maximum deux chiffres soit acceptée.

Assurez-vous que le résultat renvoyé par votre programme soit toujours correct. Pour ce faire, testez votre programme avec le fichier `entiers2.dta` et vérifiez les résultats avec le fichier `entiers2.dta.out`.

6. Bonus : Puissances entières

Ecrivez un programme qui calcule la puissance d'un entier par un autre entier (sans utiliser la fonction `pow` de la librairie *math.h*). Les nombres entiers sont lus au clavier grâce à la macro `getchar`. Ils sont composés de 1 ou 2 chiffres et peuvent être négatifs.

Rappelons que :

$a^{(-b)} = 1/a^b$.

Assurez-vous que votre programme traite correctement les cas limites :

- $0^0=1$
- $a^0=1$
- $0^{(-b)}=\infty$ si $b>0$