Série 1 :
Programmation *C* - Bases

Buts

Le but de cette série d'exercices est de vous permettre de pratiquer les bases de la programmation en C sur des exemples simples de premiers programmes : variables, expressions, opérateurs.

Informations importantes :

Thèmes et niveaux de difficulté

Pour vous aider dans votre pratique (voire votre [auto-]évaluation), un thème et un niveau de difficulté (de 1 = facile à 3 = avancé) est associé à chaque exercice. Pratiquez donc en priorité les thèmes avec lesquels vous vous sentez le moins à l'aise. De même, si vous êtes doué pour la matière, attaquez directement les exercices niveau 2 et 3. Si par contre vous êtes en difficulté, focalisez vous sur les niveau 1 et 2 puis revenez plus tard sur le niveau 3.

Notez que les niveaux sont déterminés en fonction du moment où la série est prévue... il est clair qu'un même exercice donné plus tard au cours de l'année (par exemple au moment de l'examen) serait considéré comme beaucoup plus facile !

L'objectif final est d'atteindre au moins le niveau 2. (Il faut donc pour cela s'être déjà confronté plusieurs fois au niveau 3 !)

Ne vous contentez pas de faire les premiers exercices, mais assurez-vous d'avoir fait suffisement d'exercices niveau 2.

Durée

La totalité de la série n'est pas prévue pour être finie en 2 heures ! La série est à considérer comme un regroupement de plusieurs exercices, un peu comme un chapitre de livre d'exercices, où chacun peut choisir (à l'aide des indications ci-dessus) ce qui lui semble adapté à sa progression. Le niveau **moyen** pour 2 heures d'exercices correspond à un exercice niveau 1 et deux exercices niveau 2.

Pour finir, je vous conseille également, en plus des 2 heures d'exercices hebdomadaires à l'emploi du temps, de travailler par vous-même chez vous de façon régulière (pratique personnelle). Ce travail supplémentaire devrait en moyenne correspondre à 2 à 3 heures hebdomadaires.

En résumé

- **Niveau 1:** Ces exercices élémentaires devraient pouvoir être faits par tous dans un temps raisonnable (20 à 30 minutes maximum). Ils permettent de travailler les bases.
- **Niveau 2:** Ces exercices plus avancés **devraient être abordés par tous**, sans forcément être finis. La reprise de l'exercice avec la correction devrait constituer un bon moyen de progresser.
- **Niveau 3:** Ces exercices d'un niveau avancé sont pour les plus motivés/habiles d'entre vous. Ils peuvent **dans un premier temps** être ignorés. Ils doivent par contre absolument être repris, avec la correction si nécessaire, au plus tard lors des révisions.

Préliminaires :

Pour les exercices de ce cours, je vous conseille de créer un répertoire de travail spécifique, par exemple `$HOME/ProgOS/`.

Pour créer un répertoire : utilisez la commande `mkdir`.

Exercice 0 : environnement de travail

Vous allez donc devoir écrire du « code source » en C. Cela peut bien sûr se faire dans n'importe quel éditeur de texte, même s'il est préférable d'avoir un environnement qui apporte « un peu d'aide » (coloration syntaxique, indentation, compilation, gestion des messages d'erreur, debugging, etc.), bref, un « environnement de développement intégré » (IDE).

Au niveau de ce cours, nous ne vous imposons aucun outil. Libre à vous de choisir ce que vous voulez pour écrire vos codes C.

Sur les VM des salles CO, vous pourrez utilisez (par ordre conseillé pour ce cours) : *Geany*, *Emacs*, *Gedit*, *vim*, ...

Au niveau plus général (votre propre machine), on peut citer : ***CodeLite***, ***Geany***, ***Xcode*** (pour Mac), ***kDevelop*** (sous KDE), ***NetBeans***, ***CDT pour Eclipse***, ***Code::Blocks***, ***Anjuta***, ***Sublime Text***, ... A vous de choisir !

Pour compiler vos programmes :

- comme dit en cours, à la ligne de commande cela se fait simplement avec :

  ```
  gcc -o nom_de_l_executable code_source.c
  ```

- Dans *Geany* : appuyer sur F9 (ou "Build" dans le menu "Build")
- Pour une version plus rigoureuse, suivant un standard :

  ```
  gcc -ansi -Wall -pedantic -o executable code_source.c
  gcc -std=c11 -Wall -pedantic -o executable code_source.c
  ```

  Dans *Geany* : modifiez les préférences de la commande "Build" ("Set build commands" dans le menu "Build").

Pour exécuter :

- dans le terminal :

  ```
  ./nom_de_l_executable
  ```

- dans *Geany* : touche F5 (ou "Execute" dans le menu "Build")

Enfin, pour vous aider avec les principales commandes Unix, il pourra être utile de regarder **ce complément que j'ai écrit** pour un autre cours.

Exercice 1 : Variables (niveau 1)

Écrivez un programme `age.c` qui :

1. demande son âge à l'utilisateur ;
2. lit la réponse de l'utilisateur et l'enregistre dans une variable `age` de type entier ;
3. calcule l'année de naissance (à un an près) de l'utilisateur et l'enregistre dans la variable `annee` de type entier ;
4. affiche l'année de naissance ainsi calculée.

Compilez et exécutez votre programme.

Indications :

- Pour demander à l'utilisateur son âge, affichez un message à l'aide de l'instruction suivante :

  ```
  printf("Quel age avez-vous ?");
  ```

- Pour lire la réponse de l'utilisateur et l'enregistrer dans la variable `age` de type entier, utilisez l'instruction suivante :

  ```
  scanf("%d", &age);
  ```

- Pour afficher du texte et des variables entières, indiquez chaque variable par un code spécifique (`%d` pour les entiers, `%lf` pour les réels. Attention : devant le `f` c'est un L minuscule et pas in 1 (un) !), et passez ensuite ces variables comme paramètres supplémentaire à `printf`.

  Exemple :

  ```
  printf("message 1 %d, message2 %d\n", var1, var2);
  ```

  Pour plus de détails :

  ```
  man 3 printf
  ```

- "`\n`" indique un saut de ligne.
- Pour pouvoir utiliser `printf` et `scanf`, il faut inclure `stdio.h` :

```
#include <stdio.h>
```

Exercice 2 : Variables (niveaux 2 (début) et 3 (explication))

Écrivez un programme `calcul.c` qui :

1. déclare les variables *x* et *y* de type entier ;
2. déclare les variables *a*, *b*, *c* et *d* de type réel ;
3. affecte la valeur 2 à *x* et 4 à *y* ;
4. calcule la somme, la différence, le produit, et le quotient de *x* par *y*, et affecte les résultats à *a*, *b*, *c* et *d* ;
5. affiche les valeurs de *a*, *b*, *c* et *d*.

Compilez et exécutez le programme.

Vous devez constater que le quotient de *x* par *y* (c'est-à-dire `x / y`) donne un résultat nul.

Modifiez ensuite votre programme en déclarant *x* et *y* avec le type *réel*.
Recompilez et exécutez.

Que constatez-vous ?

Même question avec *x* de type entier et *y* de type réel.
(Expliquez [à votre voisin] les différents résultats obtenus.)

Exercice 3 : Expressions conditionnelles (niveau 1)

Soit **I** l'intervalle [-1,1[dans l'ensemble des réels.

Écrivez le programme `intervalle.c` qui :

1. demande à l'utilisateur d'entrer un réel ;
2. enregistre la réponse de l'utilisateur dans une variable *x* de type réel ;
3. teste l'appartenance de *x* à l'ensemble **I** et affiche le message:
   ```
   x appartient à I
   ```
   si c'est le cas, ou:
   ```
   x n'appartient pas à I
   ```
   dans le cas contraire.

Testez votre programme en l'exécutant plusieurs fois de suite en donnant successivement à *x* les valeurs -2.5, -1, 0.5, 1, et 1.5.

Indications :

- pour les points (1) et (2), vous pouvez utiliser le code suivant :

```
#include <stdio.h>

int main(void) {
  double x ;                    /* déclaration */
  printf("Entrez un réel : ");  /* demande à l'utilisateur d'entrer un réel */
  scanf("%lf", &x);             /* enregistre la réponse dans x */
  return 0;
}
```

Exercice 4 : Expressions conditionnelles (niveau 2)

Même problème que l'exercice précédent avec : **I** = [2,3[U]0,1] U [-10,-2]

Dans cet exercice, nous autorisons **uniquement** l'utilisation des opérateurs relationnels `<` et `==`. Tous les opérateurs logiques (`&&`, `||`, etc) sont, par contre, autorisés.

Testez avec les valeurs -20, -10, -2, -1, 0, 1, 1.5, 2, 3 et 4.

Dernière mise à jour le 15 février 2016
Last modified: Mon Feb 15, 2016