Introduction au module : exercices corrigés en C
Corrigé

Objectifs

- Installer l'environnement de développement ;
- Se familiariser avec l'environnement de développement ;
- Choisir les bons types ;
- Écrire un premier programme ;
- Manipuler les entrées/sorties ;

Exercice 1 : Installation et utilisation de `Code::Blocks`

`Code::Blocks` est un IDE [1] *open source* pour les langages C et C++ qui fonctionne sur les plateformes Linux, Mac et Windows.

1.1 *Installer `Code::Blocks`.* `Code::Blocks` peut être téléchargé depuis le site `http://www.codeblocks.org/`. Il suffit alors de sélectionner l'onglet « *download* » puis « *Download the binary release* », de télécharger la version qui vous convient et, enfin, de l'installer. Pour les utilisateurs de Windows, il faut prendre la version dont l'extension est « `mingw-setup.exe` » car elle inclut le compilateur GCC [2] que nous allons utiliser.

1.2 *Créer un projet de type console.* Lorsque `Code::Blocks` est lancé, il affiche dans la fenêtre principale une plusieurs actions qui peuvent être faites, en particulier créer un nouveau projet. En cliquant sur « *Create a new project* », plusieurs types de projet sont proposés. Nous utiliserons « *Console application* ». Dans les boîtes de dialogue qui suivent, il faut sélectionner C (plutôt que C++), faire *Next* puis donner un nom au projet, par exemple `S0s2e1` (pour Semaine 0, série 2, exercice 1), sélectionner un répertoire (*Folder*) où seront conservés les fichiers du projet, faire *Next*. Sur l'écran suivant, il suffit de faire *Finish*.

Un nouveau projet arrive dans le *workspace*. Il s'appelle naturellement `S0s2e1`. Dans le sous-répertoire *Source* se trouve le fichier `main.c` qui correspond au squelette du programme principal. On peut cliquer deux fois dessus pour voir son contenu dans la fenêtre principale.

On peut alors le compiler, puis l'exécuter ou faire les deux en même temps soit en utilisant le menu *Build* ou les icônes correspondantes.

1. Environnement de développement intégré.
2. GNU C Compiler, `http://gcc.gnu.org/`.

Quand on a fini de travailler avec un projet, il est conseillé de le fermer : clique droit puis « *Close project* ».

1.3 *Quitter* `Code::Blocks`. Lorque l'on quitte l'IDE, il est conseillé de sauver les projets et les workspaces. C'est ce qui est proposé dans les boîtes de dialogue.

Exercice 2 : Récupérer et compléter les sources fournies

Commençons par créer un nouveau projet (S0s2e2, pour Semaine 0, série 2, exercice 2). Il suffit de sélectionner *File / New / Project*. On choisira « *Empty project* » puis on donnera le titre. Notons qu'il n'est pas utile de préciser le répertoire. Celui sélectionné précédemment est utilisé.

Ensuite, on clique à droite sur le projet, on sélectionne « *Add files...* ». Dans la boîte de dialogue, on sélectionne le ou les fichiers à inclure. Ici, on sélectionne le fichier nommé `pgcd-euclide.c`. On coche les deux options *Debug* et *Release* puis on fait *Valider*.

On peut alors ouvrir le fichier, le compiler et l'exécuter.

On peut aussi le compléter, le compiler et l'exécuter à nouveau.

Exercice 3 : Utilisation du débogueur de `Code::Blocks`

Le débogueur est un outil qui permet d'exécuter pas à pas (instruction par instruction) votre programme et de consulter la valeur des variables de votre programme. Ceci peut éviter d'avoir à mettre des traces dans le code source.

Il est possible de mettre (ou d'enlever) des points d'arrêt en cliquant à droite sur une instruction et en sélectionnant *Toggle breakpoint* ou en cliquant à droite dans la marge et en faisant *Add breakpoint* (ou *Remove breakpoint*). Un disque rouge apparaît dans la marge quand un point d'arrêt est positionné sur une instruction.

Quand on exécute le programme en choisissant *Debug / Start*, le programme s'exécute jusqu'à ce qu'un point d'arrêt soit rencontré. On peut alors continuer l'exécution du programme ligne par ligne ou instruction par instruction. Il peut être utile de faire apparaître l'état des variables locales en faisant *Debug / Debugging windows / Watches*. On peut alors voir l'état de toutes les variables locales (quand le programme est en cours d'exécution !). Il suffit d'ouvrir « *Local variables* ».

Remarque : Il est souvent plus simple de sélectionner une ligne dans le fichier source et de faire *Debug / Run to cursor*. Le programme s'exécute alors jusqu'à atteindre la ligne indiquée par le curseur.

Attention, le débogueur est très consommateur de temps et il est souvent préférable de relire le source de son programme à tête reposée.

Exercice 4 : Affichage d'une ligne d'une facture

Écrire un programme pour saisir le code d'un article (un seul caractère), un prix unitaire hors taxe (exprimé en euros), une quantité entière. Il affiche ensuite un récapitulatif en donnant en plus le prix total hors taxes et TTC.

Modifier le programme pour que les données soient tabulées.

Par exemple, l'article de code T et de prix unitaire 2,5 a été commandé en 100 exemplaires. La ligne de facture affichée est donc la suivante :

```
|  T |           2.50 |   100 |          250.00 |          299.00 |
```

Solution :

```
R0 : Afficher une ligne d'une facture

tests : celui fournit dans le sujet.

R1 : Raffinage de « Afficher une ligne d'une facture »
  | Saisir les caractéristiques de l'article commandé
  | Calculer les prix
  | Afficher la ligne de la facture

R2 : Raffinage De « Saisir les caractéristiques de l'article commandé »
  | Saisir le code
  | Saisir le prix unitaire
  | Saisir la quantité

R2 : Raffinage De « Calculer le prix »
  | prix_ht <- quantité * prix_unitaire
  | prix_ttc <- prix_ht * (1 + TVA)

R2 : Raffinage De « Afficher la ligne de la facture »
  | Écrire(code, prix_unitaire, quantité, prix_ht, prix_ttc)
```

On en déduit alors l'algorithme suivant :

```
Algorithme ligne_facture

        -- Afficher une ligne d'une facture

Constante
    TVA = 0.196         -- Taux de TVA

Variable
    code: Caractère     -- le code de l'article
    prix_unitaire: Réel -- le prix unitaire
    quantité: Entier    -- quantité de l'article
    prix_ht: Réel       -- prix hors taxes
    prix_ttc: Réel      -- prix TTC

Début
    -- Saisir les caractéristiques de l'article commandé
    Saisir le code
    Saisir le prix unitaire
    Saisir la quantité

    -- Calculer les prix
    prix_ht <- quantité * prix_unitaire
    prix_ttc <- prix_ht * (1 + TVA)

    -- Afficher la ligne de la facture
    Écrire(code, prix_unitaire, quantité, prix_ht, prix_ttc)
Fin
```

```
/*****************************************************************
 *  Auteur   : Xavier Crégut <cregut@enseeiht.fr>
```

```c
 *  Version  : 1.2
 *  Objectif : Afficher une ligne d'une facture (sans contrôle)
 ******************************************************************/

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

#define TVA .196        /* Taux de TVA */

int main()
{
    char code;          /* le code de l'article */
    double prix_unitaire;       /* le prix unitaire */
    int quantite;       /* quantité de l'article */
    double prix_ht;     /* prix hors taxes */
    double prix_ttc;    /* prix TTC */

    /* Saisir les caractéristiques de la ligne de facture */
    printf("Code article : ");
    scanf("%c", &code);
    printf("Prix unitaire : ");
    scanf("%lf", &prix_unitaire);
    printf("Quantité : ");
    scanf("%d", &quantite);

    /* Calculer les prix */
    prix_ht = quantite * prix_unitaire;
    prix_ttc = prix_ht * (1 + TVA);

    /* Afficher la ligne de la facture */
    printf("| %2c | %15.2f | %5d | %15.2f | %15.2f |\n",
            code, prix_unitaire, quantite, prix_ht, prix_ttc);

    return EXIT_SUCCESS;
}
```