Université Polytechnique Hauts-de-France
Institut National des Sciences Appliquées Hauts-de-France **2023-2024**
1er cycle INSA Algorithmique et Programmation
Semestre 3

Fiche TP 1

---

1 Environnement de travail

Si vous utilisez une machine de la salle de TP vous utiliserez le logiciel `Code::Blocks` pour traiter les différents sujets de TP. Ce logiciel gratuit est ce qu'on appelle un IDE (Environnement de Développement Intégré) : il permet de saisir des programmes écrits en langage `C`, et de construire le fichier exécutable correspondant [1]. `Code::Blocks` est *multi-plateformes* et *gratuit*. Vous pouvez aussi utiliser un autre IDE, mais cette section se limite à une présentation rapide de l'interface de `Code::Blocks`.

Note :
On vous recommande de télécharger `Code::Blocks` pour votre propre machine, ce qui vous permettra de disposer d'un environnement de développement en dehors des séances de TP

http://www.codeblocks.org/downloads/26

`Code::Blocks` est disponible pour Windows, Linux et OS-X.

Attention :
pour ceux qui utilisent un PC sous `Windows`, il faut télécharger la version du logiciel qui inclut le compilateur GCC : il s'agit actuellement de `codeblocks-20.03mingw-32bit-setup.exe`, mais le numéro de version peut changer. Consultez les notes sur la page de téléchargement et considérer la version qui fait référence à `MinGW-W64 project`.

Avant tout chose, vous devez créer un répertoire dans votre espace personnel, `TP_INFO` par exemple, pour y stocker les programmes que vous allez écrire. Vous pouvez également créer un sous-répertoire `TP1` pour ce premier TP. Ne mettez **jamais** de blanc dans les noms des dossiers ou des fichiers.

`Code::Blocks` fonctionne via la notion de `projet`. Il est ainsi nécessaire de créer un projet pour pouvoir commencer à coder. Pour cela vous passez par le chemin suivant : `File\New\Project...`
Dans la fenêtre qui s'ouvre vous sélectionnez alors l'image avec l'intitulé `Console application` (puis `Go`).
Une nouvelle fenêtre s'ouvre. **Attention** à bien sélectionner `C` et pas `C++`, puis `Next`.
Vous pouvez alors saisir le nom du projet (`tp1` par exemple pour ce TP), puis indiquer où il doit être sauvegardé, puis `Next`.
Dernière étape, vous vous assurez que le logiciel affiche bien `GNU GCC Compiler` dans la fenêtre du haut, et vous cliquez sur `Finish`. Le logiciel va alors créer le projet et y inclure automatiquement un fichier `main.c` qui contient une fonction nommée `main`.

Pour pouvoir tester un programme après en avoir saisi le code source, il faut d'abord générer son exécutable. Pour cela sous `Code::Blocks` vous pouvez cliquer sur `Build\Build`. Cette commande lance le processus de compilation et d'édition de liens, que vous pouvez suivre dans le fenêtre en bas. Si *aucune erreur et aucun avertissement* ne sont indiqués, vous pouvez alors tester le programme en passant par `Build\Run`.

1. La construction d'un exécutable est réalisée en deux phases : compilation des différents fichiers sources composant ce programme, puis édition de liens.

Important :

- Il **n'est pas possible** de tester un programme tant qu'il y a au moins une erreur indiquée par le compilateur.
- Il est **fortement déconseillé** de tester un programme si au moins un avertissement (*warning*) est indiqué par le compilateur (la plupart des avertissements signalent une erreur de conception ou de codage qui peut être grave).
- **Après chaque modification** dans le code il faut à nouveau reconstruire l'exécutable pour pouvoir tester et voir l'impact des modifications.

2 Rappels / compléments du cours

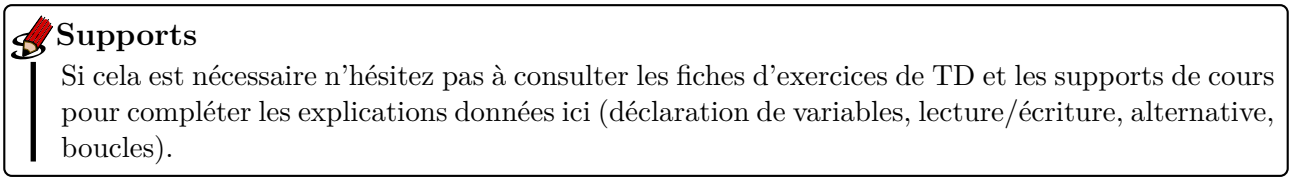
Un programme en langage `C` est structuré grâce à l'écriture de **fonctions**. Une fonction est caractérisée par un type de résultat, un identifiant et une liste de paramètres passés entre parenthèses, chacun étant lui-même caractérisé par un type et un identifiant. Ces points seront précisés lors du CM numéro 4.

Lors de la création d'un projet dans `Code::Blocks` le logiciel génère automatiquement un programme contenant une seule fonction `main` ressemblant au code suivant [2].

```
1 #include <stdio.h>

3 int main()
4 {
5   printf("Bonjour le monde\n");
6   return 0;
7 }
```

Sans rentrer dans les détails, l'instruction `#include <stdio.h>` ajoute dans notre programme les déclarations des fonctions d'entrées-sorties fournies par le langage. La ligne 3 est la première ligne de la fonction principale `main` qui retourne **obligatoirement** un `int` en `C`. Cette fonction consiste uniquement ici à afficher un message et à retourner la valeur 0.



3 Entrées / sorties

Le terme d'entrées / sorties fait référence aux mécanismes mis en œuvre par un programme pour échanger des données depuis / vers un flux. Les flux sont des abstractions qui peuvent par exemple représenter un périphérique (écran, clavier en particulier) ou un fichier. On considère ici uniquement ce qu'on appelle l'entrée standard qui correspond au clavier et la sortie standard qui correspond à l'écran [3].

Par défaut, l'accès au flux est ***tamponné*** : une zone de mémoire (*buffer* en anglais) sert de tampon durant les transferts. Ainsi, lors de la saisie de valeurs via le flux d'entrée standard, les données issues du clavier sont stockées dans le tampon en attendant que le programme les réclame. Un intérêt de cette utilisation du *buffer* est que, tant que le programme ne demande pas l'accès au tampon, il est possible de modifier les données qu'il contient. Les tampons permettent aussi d'optimiser les vitesses d'échange avec les périphériques prévus pour gérer les données *par paquets*.

2. Avec un message différent
3. En langage `C` le flux d'entrée standard est appelée `stdin` et le flux de sortie standard `stdout`

3.1 Entrées / sorties non formatées

Il existe plusieurs fonctions prédéfinies[4] pour saisir ou afficher un ou plusieurs caractères en `C`. On considère ici en particulier la fonction `getchar` qui permet de récupérer dans un `int` le code `ASCII` d'un caractère (un seul) lu dans le flux d'entrée standard. Cette fonction prélève le premier caractère du tampon et le supprime de celui-ci. Si le tampon est vide, le programme attend jusqu'à ce qu'un caractère y soit placé (c'est-à-dire jusqu'à un caractère soit saisi). Le prototype de la fonction est le suivant :

```
int getchar(void) ;  /* getchar est une fonction qui ne recoit rien et retourne un int */
```

On peut donc l'utiliser comme suit :

```
int c ;
c = getchar();
```

Le fonction `putchar` permet de son côté afficher à l'écran le caractère dont le code `ASCII` est passé en paramètre. Son prototype est :

```
int putchar(int c) ;
```

La fonction retourne un `int` qui permet de savoir si l'exécution de la fonction s'est bien passée ou non (pas la peine d'en dire plus pour le moment). On peut utiliser cette fonction simplement :

```
int c ;
c = getchar();
putchar(c);
```

Pour utiliser correctement la fonction `getchar` il faut avoir à l'esprit qu'elle ne peut accéder aux caractères présents dans le tampon qu'une fois que la touche `Entrée` a été pressée. En fait les caractères saisis au clavier sont ajoutés les uns à la suite des autres dans le buffer jusqu'à ce que l'utilisateur presse cette touche. La fonction `getchar` peut alors être utilisée pour traiter les caractères un à un. Remarquez que la pression de la touche `Entrée` (qui correspond à un retour à la ligne) provoque aussi l'introduction du caractère noté `\n` dans le buffer.
Les exercices suivants doivent être traités dans le même fichier source (`main.c` du projet `Code::Blocks`). Une fois un exercice validé vous pourrez le "supprimer logiquement" avant de passer à l'exercice suivant, qui doit être placé à la suite. On obtient ainsi un programme ressemblant à l'illustration suivante.

```
#include <stdio.h>

int main()
{
 /* exercice 1 */
 #if 0
    code de l'exercice 1
 #endif

 /* exercice 2 */
 #if 0
    code de l'exercice 2
 #endif

 /* exercice 3 */
    code en cours de realisation

  return 0;
}
```

Attention

N'effacez surtout pas ce que vous avez fait ! L'utilisation de la compilation conditionnelle permet de demander au compilateur d'ignorer les lignes placées entre `#if 0` et `#endif` .

4. Ces fonctions, (et les autres fonctions d'entrées/sorties) sont déclarées dans `stdio.h`.

Exercice 1

Compléter la fonction `main` dans `Code::Blocks` afin d'obtenir le comportement suivant, en utilisant les fonctions `getchar` et `putchar`. L'utilisateur saisit un premier caractère, valide en appuyant sur a touche "Entrée", puis saisit un second caractère, et valide à nouveau en appuyant sur la touche "Entrée". Les instructions ajoutées doivent permettre de récupérer les deux caractères saisis et de les afficher, une fois les deux saisies faites, sur deux lignes consécutives.

Exercice 2

Compléter maintenant avec des instructions permettant de lire une suite de caractères sur la même ligne via la fonction `getchar`. Chaque caractère lu par le programme (et récupéré dans le buffer) sera simplement affiché via `putchar`. La saisie s'arrête lors de la pression de la touche "Entrée" par l'utilisateur.

3.2 Sorties formatées

La fonction `putchar` ne permet d'afficher que des caractères, un à la fois. La fonction `printf` permet d'effectuer une sortie formatée, et éventuellement d'afficher plusieurs valeurs de différents types simples. Son prototype est le suivant :

```
printf(format, exp1, ..., expn);
```

La fonction affiche les valeurs des expressions `exp1, ..., expn`, alors que le paramètre `format` est une chaîne de caractères de formatage précisant comment les expressions doivent être affichées. Cette chaîne peut contenir du texte qui sera affiché tel quel et/ou des caractères qui vont spécifier les formats d'affichage de valeurs de variables. On pourra ainsi par exemple écrire

```
printf("Bonjour le monde\n");
```

comme plus haut dans l'énoncé pour afficher simplement du texte à l'écran. Dans ce cas le `format` est composé uniquement de texte, et il n'y a pas de valeurs d'expressions fournies. Si on souhaite afficher la valeur d'une variable on procédera comme suit, par exemple pour un entier `int` :

```
int i = 4 ;
printf("%d\n", i);
printf("La valeur de la variable i est %d, et son carre est %d\n", i, i*i);
```

Ici le `format` du premier `printf` est composé du spécificateur de conversion `%d` qui indique que l'on va afficher la valeur d'une expression de type `int`. Le caractère `\n` indique que l'on veut terminer par un retour à la ligne, et l'expression `i` indique quelle est la valeur à afficher. Le second `printf` fonctionne sur le même principe mais avec deux expressions, et en combinant texte et format d'affichage. Notez qu'il y a donc autant d'expressions dans un `printf` qu'il y a de spécifications de conversions (repérables avec le `%`).

Exercice 3

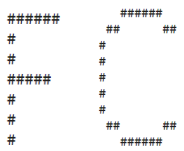
Compléter votre programme avec des instructions permettant de compter puis d'afficher le nombre de fois qu'un caractère saisi au clavier (via `getchar`) est présent dans une suite de caractères lus au clavier (via `getchar` également et sur une même ligne comme dans l'exercice 2). Le programme affichera ce nombre à l'écran avec la fonction `printf`.

Exercice 4

Ajouter les instructions pour recopier à l'écran les caractères saisis un à un au clavier, toujours selon le même fonctionnement. Le programme ne doit pas recopier les espaces ni les tabulations ('`\t`' en `C`).

Exercice 5

Ajouter les instructions pour afficher à l'écran un 'F' ou un 'C', selon les modèles suivants, en fonction du choix de l'utilisateur. La réalisation du "dessin" se fera en utilisant le plus possible de boucles.



3.3 Entrées formatées

La fonction `getchar` ne permet de lire que des données de type caractère, avec la gestion du buffer à assurer. On peut aussi utiliser la fonction `scanf` pour saisir une ou plusieurs valeurs formatées. Cette fonction repose sur le même principe que la fonction `printf` : on précise une chaîne de formatage, puis une suite de paramètres :

```
scanf(format, adr1, ..., adrn);
```

La différence **fondamentale** est que dans le cas de `scanf` les paramètres sont des adresses (voir cours). Si l'on souhaite récupérer la valeur d'un entier et la stocker dans une variable `i` préalablement déclarée on écrira ainsi :

```
int i;
scanf("%d",&i);
```

Exercice 6

Compléter votre programme pour afficher sur `n` lignes consécutives la valeur du prochain entier naturel (on commencera à 1), son carré et son cube. La valeur de `n` sera lue au clavier. Par exemple pour `n = 3` le programme doit afficher :

```
1 1 1
2 4 8
3 9 27
```

Exercice 7

Ajouter les instructions permettant de lire trois valeurs entières. Le programme affichera ensuite 1 si au moins une des trois valeurs est dans l'intervalle $[10; 20]$, et 0 sinon (bonus : faire le traitement avec un seul `if`).

Exercice 8

Ajouter les instructions pour lire deux valeurs entières (positives), disons `a` et `b`, et afficher `a` lignes contenant `b` entiers naturels consécutifs (on commencera à nouveau à 1). Par exemple pour `a = 4` et `b = 2` le programme doit afficher :

```
1 2
3 4
5 6
7 8
```

Exercice 9

Compléter le programme pour qu'il affiche un triangle sur la base de celui ci-dessous, dont le nombre de lignes est lu au clavier (5 dans l'exemple)

```
1
01
101
0101
10101
```