IN 104 – Initiation au Makefile

L'objectif de cette initiation non notée est de prendre en main l'outil Makefile. Si vous n'avez pas le temps de finir cet exercice lors de la séance, il est fortement recommandé de la travailler à la maison pour être à l'aise avec.
Il est nécessaire de comprendre le fonctionnement de l'outil Makefile pour avancer correctement lors du projet.

1 Fonctionnement

```
# Au début du fichier, on place les assignations de variables qu'on pourra
# rappeler à l'aide de la command $(...)
GCC = gcc # choix du compilateur

# Puis on place, les règles de compilation sous la forme
# cible :dependance
#     ligne de commande
# Attention, on doit indenter à l'aide de tabulations et non d'espaces !

compile :fichier1.o
    $(GCC) fichier1.o -o application

fichier1.o :fichier1.c
    $(GCC) fichier1.c -c -o fichier1.o
```

2 Exercices

1. Écrire un programme qui prend en paramètres deux entiers et renvoie la somme des deux entiers. Voici un rendu attendu : `./add 1 2` doit afficher 3.

   ```
   prompt>  ./add 1 2
   1 et 2 font 3.
   prompt>
   ```

2. Écrire un fichier `Makefile` avec deux cibles `compile` (pour la compilation du programme) et `run` (pour l'exécution)
   Exécuter

   ```
   prompt>  make compile
   prompt>  make run
   ```

3. Modifier votre programme de test avec un commentaire, puis relancer

   ```
   prompt>  make run
   ```

   La compilation a-t-elle lieu ?
   Pourquoi ? Rajouter une dépendance à la cible `run`.

À présent, vous devez avoir un fichier `Makefile` avec une bonne gestion des dépendances.

4. Écrire une vraie fonction d'addition et la déporter dans un second fichier.
   Compiler les deux fichiers séparément sans édition de lien, puis ensuite faire l'édition de lien séparément :

```
prompt>  gcc -c fichier1.c -o fichier1.o
prompt>  gcc -c fichier2.c -o fichier2.o
prompt>  gcc fichier1.o fichier2.o -o add
```

5. Créer autant de cibles que nécessaire pour faire réaliser la compilation de la question précédente par le `Makefile`.

6. Observez la règle suivante :

```
%.o :%.c
    $(GCC) -c %< -o %@
```

   D'après vous, à quoi sert-elle ?
   Modifier votre Makefile pour le simplifier au maximum grâce à cette règle.

3 Correction

On crée un fichier `test.c` :

```c
#include <stdlib.h> // nécessaire à atoi
#include <stdio.h>  // nécessaire à printf

// On pourrait inclure cette ligne dans un .h mais peu importe pour cet exercice.
int add(int a, int b);

int main(int argc, char* argv[]) {
  int a = atoi(argv[1]), b = atoi(argv[2]);
  printf("%d et %d font %d.\n", a, b, add(a,b));
  return 0;         // code de retour; 0 si tout va bien
}
```

et un fichier `add.c` :

```c
int add(int a, int b) {
  return a+b;
}
```

Le `Makefile` final pourra ressembler à :

```
GCC = gcc # choix du compilateur

run :compile
    ./add 1 2

compile :test.o add.o
    $(GCC) test.o add.o -o add

# %< correspond au champ dépendances
# %@ correspond au champ cible
%.o :%.c
    $(GCC) -c %< -o %@
```