TRAVAUX PRATIQUES

TP 2 - Make et Makefile

2022/2023

Un logiciel est en général écrit par morceaux testés, compilés et archivés en bibliothèque. Il ne reste plus qu'à écrire le programme principal de l'application qui utilise ces morceaux. En effet, il est inutil qu'à chaque exécution, ou si on modifie juste une partie, de recompiler tout l'ensemble pour reconstituer le logiciel.

Compilation séparée.

1. Créer, compiler (`cc prog.c`) et exécuter le fichier suivant

`prog.c`

```
#include <stdio.h>
float fahr (int);
main() {
      int f; float c;
      scanf ("%d", &f);
      c = fahr(f);
      printf ("En Celcius = %f\n", c);
}
float fahr(int f) {
      /*  Convertit Temperature fahrenheit
      en Celcius */
      float c;
      c = 5./9 * (f - 32);
      return c;
}
```

• Créer maintenant deux fichiers: `ppal.c` et `fahr.h`

`ppal.c`

```
#include <stdio.h>
#include "fahr.h"   /* noter ce nouveau include */
main() {
      int f; float c;
      scanf ("%d", &f);
      c = fahr(f);
      printf ("En Celcius = %f\n", c);
}
```

fahr.h

```
float fahr(int f) {
      /*  Convertit Temperature fahrenheit
      en Celcius */
      float c;
      c = 5./9 * (f - 32);
      return c;
}
```

• Compiler (`cc ppalc`) et exécuter (`a.out`).

NB: Les fichiers include <*file*.h> sont à rechercher dans `/usr/include`.
Avec "*file*.h", le répertoire courant est *aussi* examiné.

2. Séparer cette fois ci en deux fichiers: `ppal.c` et `fahr.c`

ppal.c

```
#include <stdio.h>
float fahr (int);          /* Ici le Prototype*/
main() {
      int f; float c;
      scanf ("%d", &f);
      c = fahr(f);
      printf ("En Celcius = %f\n", c);
}
```

fahr.c

```
float fahr(int f) {
/*  Convertit Température fahrenheit
      en Celcius */
float c;
c = 5./9 * (f - 32);
return c;
}
```

• Compiler cette fois-ci avec `cc ppal.c fahr.c` , et exécuter.

• Créer un fichier objet (compilé en `.o`) correspondant à `fahr.c` par:

```
cc -c fahr.c
```

(Faire `ls` et vérifier la présence de `fahr.o`)

• Compiler et exécuter avec : `cc fahr.o ppal.c`

(Seul `ppal.c` est compilé en `ppal.o` et ensuite édition de lien)

3. Créer un répertoire *dir* (`mkdir dir`).

Archiver maintenant la fonction *fahr*() par:

```
ar rv dir/libfh.a fahr.o
```

qui ajoute le code objet `fahr.o` au fichier bibilothèque `libfh.a` du répertoire `dir`.
Le nom `lib` et l'extension `.a` sont voulus par le système UNIX (voir plus loin).

Faire ensuite (sur linux ce n'est pas nécessaire)

```
Ranlib dir/*
```

Cette commande optimise le chargement des fichiers *lib*.

• Compiler et exécuter par

```
cc ppal.c -Ldir -lfh
```

L'option **–l** introduit le nom de la bibliothèque (`fh`) dans laquelle il faut chercher les modules complémentaires à éditer. Ce nom sert à déterminer le fichier **`libfh.a`**

L'option **`-L`**`dir` , quand à elle, indique qu'il faut le chercher dans le répertoire `dir`, car sinon, on cherche dans le répertoire système `/usr/lib`. (Verifier "`ls /usr/lib`" quels autres bibliothèques existent. En particulier `libm.a`, utilisée avec `cc p.c -lm`)

• Maintenant, dans ppal.c on retire la déclaration `float fahr (int);`
pour la mettre dans un fichier fahr.h et on met à la place `include "fahr.h"`.

`ppal.c`

```
#include "fahr.h"
main() {
   int f; float c;
   scanf ("%d", &f);
   c = fahr(f);
   printf ("En Celcius = %f\n", c);
}
```

`fahr.h`

```
float fahr(int );
```

• Refaire pour vérifier:

```
cc ppal.c -Ldir -lfh
```

C'est comme cela que la plupart des fonctions bibliothèques C fonctionnent : les fichiers `include.h` contiennent les prototypes, e.g. `double sqrt (double);` ... dans `math.h`, et les codes sont compilés et mis en bibliothèque (`libm.a`).

```
$ cc prog.c –lm
```

Usage de make et Makefile.

(Voir aussi l'annexe *Makefile* du poly Unix)

1. Ecrire deux fonctions *somme*() et *produit*() telle que:

```
int somme (int x, int y){ return x+y;}
```
fichier `som.c`

```
int produit (int x, int y){ return x*y;}
```
fichier `prod.c`

et faire un programme *main*(), fichier `ppal.c` qui les utilise.

Il s'agit, de relancer automatiquement la (re)compilation et la (re)formation de l'exécutable, quant un des composants `som.c prod.c` change. A une échelle plus grande, une application plus importante, cela devient nécessaire.

2. Créer un fichier qui s'appelle **`Makefile`** (M majuscule) avec les lignes suivantes:

Important: mettre un espace de type tabulation (touche →|, à côté de la lettre 'A' ou 'Q'), en début de la ligne `cc`.

```
som.o : som.c
→|    cc -c som.c
```

```
prod.o: prod.c
→|    cc -c prod.c
```

La ligne `som.o : som.c` dit que le fichier `som.o` dépend de `som.c`, càd, si la date de dernière màj de `som.c` est postérieure (plus récente) à `som.o`, alors il faut exécuter la ligne qui suit, soit: `cc -c som.c`. Cela se comprend, car `som.o` est probablement périmé car le source du programme a changé. De même pour `prod.c` et `prod.o` .

Le fichier `Makefile` est programmé par l'utilisateur, car c'est lui qui spécifie les dépendances de son application et les actions à entreprendre.

Supposons maintenant que l'on ait modifié `som.c` et qu'on veille refabriquer `som.o`. Il suffit de faire:

```
% make som.o
```

Le programme `make` ira regarder s'il existe une ligne dépendance avec `som.o` en partie gauche, et si oui, il regarde la partie droite (le fichier) dont elle dépend pour vérifier si la date de dernière màj de ce fichier est plus récente que celle de l'argument. Auquel cas, `make` lance l'exécution de la ligne commande qui suit la ligne dépendance examinée.

• Créer les fichiers demandés.
• Créer les .o correspondants.
• Simuler une modification de `som.c` ou `prod.c`. (il suffit d'éditer le fichier , de rajouter (et supprimer) un blanc ou un caractère et de sauvegarder le fichier).
• Faire
`make som.o`
(constater le résultat par `ll`)
• Refaire `make`. Résultat?
• Faire d'autres essais par exemple avec:

```
som.o : som.c
→|    echo 'Je recompile som.c, patienter ...'
→|    cc -c som.c
```

(<*tab*> = touche tabulation)

3. Modifier le fichier Makefile pour y rajouter:

```
prog  : ppal.c som.o prod.o
→|    cc -o prog ppal.c som.o prod.o
```

• « *Faire* » le ficher `prog` par `make prog`.
• Faire des modifications dans les `som.c` et/ou `prod.c` et (re) faire `som.o` et/ou `prod.o`.
• Faire ensuite `prog`.
• Reprendre des modifications et faire `prog` directement cette fois-ci. Constater que `som.o` et/ou `prod.o` sont automatiquement refaits. `make` travail en cascade.

4. (TP à rendre) Envisager maintenant d'archiver en bibliothèque (libsp.a) les fichier som.o et prod.o. Programmer un Makefile adéquat.

PS: A propos de *make*, voir aussi la commande `touch`, qui permet de changer la date de modification d'un fichier.

NB. `make -n` ... permet de montrer les commandes à exécuter (*just print*) sans les exécuter, et `make -s` ... permet d'exécuter ces commandes, sans les afficher en même temps (*silent*)