INF123 - Travaux pratiques, séance 3

Compilation séparée, Makefile - Variables d'environnement

[INF123] Utilisez la commande `./installeTP.sh 3` pour créer un répertoire ***TP3*** et y copier les fichiers nécessaires à cette séance.

Compilation séparée, Makefile

Compilation séparée.
Compilez séparément les fichiers ***entrees_sorties_codage.c***, ***code1.c***, ***code2.c*** et ***codage.c*** :

```
gcc -Wall -Werror -c entrees_sorties_codage.c
gcc -Wall -Werror -c code1.c
gcc -Wall -Werror -c code2.c
gcc -Wall -Werror -c codage.c
```

Normalement, les trois premières compilations se passent bien, mais la dernière produit un message d'erreur.
[a] Quelle est la fonction utilisée dans ***codage.c*** alors qu'elle n'est pas déclarée ? ■
Utilisez la commande `grep` pour savoir dans quel fichier cette fonction est déclarée :
[b] Donnez la commande complète commençant par `grep` que vous utilisez. Dans quel fichier la fonction est-elle déclarée ? ■

Dans ***codage.c***, ajoutez une ligne (commençant par `#include`) permettant d'inclure le fichier ***.h*** dans lequel la fonction est déclarée. Vérifiez que la compilation de ***codage.c*** se passe maintenant normalement.

Créez maintenant le fichier exécutable ***codage*** par la commande

```
gcc -o codage codage.o code1.o code2.o entrees_sorties_codage.o
```

Exécutez ***codage***.
Puis supprimez les fichiers ***code1.o***, ***code2.o***, ***codage.o***, ***entrees_sorties_codage.o*** et ***codage***.
[c] Expliquez pourquoi il n'est pas "dangereux" (risque de perte du travail effectué) de supprimer ces fichiers : les fichiers ***.o*** et l'exécutable ***codage***. En serait-il de même de l'exécutable ***installeTP.sh***, par exemple ? ■

Utilisation du Makefile
Préliminaire : la commande `touch`. La commande **touch** est utile pour effectuer des tests : sans changer le contenu d'un fichier, elle permet de mettre à jour sa date de dernière modification. Elle permet ainsi de simuler la modification d'un fichier texte sans passer par un éditeur. Elle permet aussi de créer un fichier vide.
Expérimentez les commandes suivantes :

```
ls -l
```

Notez la date de dernière modification de ***code1.c***

```
touch code1.c
ls -l
```

et remarquez que ***code1.c*** est maintenant le fichier le plus récent de votre répertoire.
Lisez attentivement le contenu du fichier ***Makefile***. **Notez bien** que les lignes de *commande* commencent par un caractère de tabulation, et **non pas par des espaces**.
[d] Dessinez, comme vu en TD, le graphe de dépendance de l'exécutable ***codage***, et demandez à votre enseignant de vérifier ce graphe avant de continuer. ■

Exécutez la commande

```
make
```

[e] Quels fichiers ont-ils été successivement crées ? ■
Recommencez :

```
make
```
[f] Que signifie le message obtenu ? ■
Faites semblant d'avoir modifé ***code1.c***, et recompilez :
```
touch code1.c
make
```
[g] Quels fichiers sont-ils successivement recréés ? Vérifiez que cela correspond au chemin du graphe de dépendance entre ***code1.c*** et ***codage***. ■

Refaites la même chose (`touch ...` puis `make`) avec chacun des fichiers ***.c*** et ***.h*** apparaissant dans le graphe, et vérifiez que les fichiers recréés sont conformes au graphe de dépendance.

Exercice complémentaire :

Écrivez votre propre codage :
- sur le modèle de ***code1.c*** et ***code2.c***, créez un fichier ***code3.c*** avec une nouvelle fonction de codage, de votre choix (conseil : faites simple).
- créez le fichier ***code3.h*** correspondant.
- modifiez ***codage.c*** pour pouvoir faire appel à votre codage.
- le cas échéant, modifiez aussi la fonction `lire_choix` du fichier ***entrees_sorties_codage.c*** afin de pouvoir lire le choix `3`.
- en prenant modèle sur ce qui existe dans ce fichier, complétez le ***Makefile*** afin de prendre en compte les fichiers ***code3.c*** et ***code3.h***.

Compilez et testez !
[h] Joignez les fichiers (imprimés ou recopiés) ***code3.c*** et ***code3.h*** à votre compte rendu. ■

Ajoutez au ***Makefile*** deux variables, `CC` et `CFLAGS`, contenant respectivement le nom de la commande de compilation à utiliser et les options à lui passer pour la traduction des `.c` en `.o` : autrement dit, ajoutez les deux lignes :
```
CC=gcc
CFLAGS=-Wall -Werror
```
puis, remplacez toutes les occurrences de `gcc` par `$(CC)` et toutes les occurrences de `-Wall -Werror` par `$(CFLAGS)`. Le ***Makefile*** obtenu est équivalent au précédent.
[i] Comment vérifiez-vous que votre ***Makefile*** "marche" toujours ? ■

Les variables d'environnement

[TP3] Lisez le fichier ***copiedir.sh*** de votre répertoire ***INF123***. La première commande utilise une variable nommée HOME. Ce n'est pas une variable locale à ce script, c'est une *variable d'environnement*. Pour visualiser sa valeur, exécutez la commande `echo $HOME` (vous pouvez aussi utiliser la commande `printenv HOME`)
[j] Notez soigneusement la valeur de cette variable. Pouvait-on se passer de cette variable (HOME) dans le script ***copiedir.sh*** ? Si oui, pourquoi l'avoir utilisée ? ■

D'une façon générale, les variables d'environnement mémorisent des informations propres à chaque utilisateur : par exemple l'interpréteur de commande par défaut (SHELL), le répertoire courant (PWD), le nom de l'utilisateur (USER), etc. Exécutez les commandes :
```
echo $PWD
echo PWD
echo $USER
```
[k] Expliquez chacune des réponses obtenues. ■
Exécutez la commande **printenv** pour avoir la liste de vos variables d'environnement, et repérez celles dont vous comprenez la signification.
[l] Quelle est la valeur de la variable SHELL ? ■

La variable HOME

Certaines variables d'environnement peuvent être définies par l'utilisateur à l'aide de la commande `export`.
Créez un répertoire ***Trucs*** dans votre répertoire personnel :
```
mkdir ~/Trucs
```

Vérifiez que le répertoire ***Trucs*** a bien été créé :

```
cd
ls
```

Modifiez votre variable HOME :

```
export HOME="/Public/123_Public/Maison_TP3"
```

Vérifiez avec la commande `printenv HOME` que la modification a été prise en compte.
Créez un répertoire ***Bidules*** dans votre répertoire personnel :

```
mkdir ~/Bidules
```

[m] Que se passe-t-il ? ■
Exécutez `cd`
[n] Dans quel répertoire êtes-vous ? Visitez-le ! ■
À l'aide de la commande `export`, redonnez à la variable `HOME` sa valeur normale ; vérifiez en exécutant `cd`.

Quelles commandes exécutez-vous ? La variable PATH

[TP3] Lisez le contenu du fichier ***test.c***, et compilez-le pour créer un fichier exécutable de nom ***test***. Exécutez la commande `test` (et non pas `./test`) . Pour comprendre ce qui se passe, exécutez la commande `which test`. Ceci affiche le nom complet du fichier exécutable correspondant à la commande `test`. Ici, ce n'est pas le programme ***test*** situé dans votre répertoire courant mais un autre programme de nom ***test*** qui a été exécuté. Pour exécuter votre propre programme ***test***, il faut le nommer explicitement (par exemple `./test`).
[INF123] Exécutez les commandes suivantes :

```
which less
which ls
which gcc
which gedit
which ploumploum
which rm
```

[o] Quelle commande exécutez-vous pour connaître la valeur de votre variable d'environnement PATH ? ■

La valeur de la variable d'environnement PATH est une liste de répertoires. C'est en examinant cette liste (**dans l'ordre**) que l'interpréteur de commandes recherche les programmes exécutables : par exemple **gedit**, les commandes Unix (**less**, **ls** ...), le compilateur (**gcc**), etc.
Ré-exécutez la suite de commandes `which ...` ci-dessus, et repérez dans votre variable PATH les répertoires où ont été trouvées ces commandes.
Remarque : le répertoire courant (`.`) n'appartient pas à la liste de répertoires définie par la variable PATH. C'est pour cette raison qu'il est nécessaire de nommer explicitement les programmes ou fichiers de commande que l'on a écrit pour pouvoir les exécuter (comme `./test` ou `./installeTP.sh`).
[TP3] Vous allez modifier la variable PATH mais *attention !* afin de pouvoir restaurer facilement sa valeur initiale, sauvegardez-la dans une nouvelle variable VRAIPATH avec la commande suivante :

```
export VRAIPATH=$PATH
```

Vérifiez que vous vous ne vous êtes pas trompés en exécutant les commandes :

```
[ $VRAIPATH == $PATH ]
echo $?
```

[p] Que doit afficher cette dernière commande `echo` lorsque les variables PATH et VRAIPATH ont la même valeur ? Pourquoi ? ■

Créez un fichier de nom ***ls*** avec le contenu suivant :

```
echo nous sommes le
date
```

et donnez-vous le droit d'exécuter ce fichier.
Modifiez votre variable PATH afin de mettre le répertoire courant (.) en tête de la liste[1] :

```
export PATH=.:$PATH
```

[q] Quel est le résultat de l'exécution de `ls` ? de `which ls` ? ■

[INF123] (changement de répertoire)
[r] Quel est le résultat de l'exécution de `ls` ? de `which ls` ? ■
Redonnez à la variable PATH sa valeur initiale :

1. L'opérateur noté " :" est la concaténation

```
export PATH=$VRAIPATH
```

[s] ***[TP3]*** Une question (un peu difficile) : exécutez la commande

```
*
```

et expliquez ce qui se passe. ■

Exercice complémentaire :

Les alias

[TP3] Les *alias* sont des définitions ou re-définitions de commandes dans votre environnement de travail. Leur durée de vie est celle de la session en cours. Certains alias sont initialisés au moyen de fichiers de configuration invoqués chaque fois qu'un nouveau *shell bash* est lancé : ce sont les fichiers ***/etc/bash.bashrc*** (commun à tous les utilisateurs) et ***∼/.bashrc*** (propre à chaque utilisateur).
Pour connaître la liste de vos *alias*, exécutez simplement la commande `alias`.

"Aliasage" provisoire de la commande `rm`
Créez un répertoire ***Provisoire***, et placez-vous dans ce répertoire.
[Provisoire] Créez le fichier ***temporaire***, puis supprimez-le avec la commande ***rm*** : le fichier est effacé sans autre forme de procès.
Recréez maintenant ***temporaire***, et définissez un alias pour la commande `rm` :

```
alias rm='rm -i'
```

puis tentez à nouveau d'effacer ***temporaire*** : vous devriez obtenir cette fois un message de demande de confirmation, répondez *non* (ou recréez encore le fichier si vous avez répondu *oui* trop vite ...)

Les scripts exécutent les "vraies" commandes, pas les alias
Lorsqu'on exécute un script, un nouveau *shell* est lancé, mais sans utiliser les fichiers de configuration. Faites-en l'expérience :

- Créez de nouveau un fichier ***temporaire***, et un fichier de commandes ***rm.sh*** qui supprime le fichier dont le nom est donné en argument.
  [t] Quel est le contenu du fichier ***rm.sh***? ■
  Exécutez la commande

  ```
  ./rm.sh temporaire
  ```

  Avez-vous obtenu le message de demande de confirmation ? le fichier ***temporaire*** existe-t-il encore ?
- Créez un fichier de commandes ***affiche_alias.sh*** avec le contenu

  ```
  #!/bin/bash
  echo debut des alias
  alias
  echo fin des alias
  ```

[u] Quels sont les *alias* définis pendant l'exécution de ce script ? ■

"Désaliasage" de la commande `rm` Supprimez maintenant la définition de l'alias `rm` :

```
unalias rm
```

puis exécutez de nouveau la commande `rm temporaire` : que constatez-vous ?

La commande de la semaine : `sed`.

La commande ***sed*** permet de transformer un fichier en appliquant différents traitements (suppression, substitution) sur un sous-ensemble de ses chaînes de caractères.

Essayez par exemple la commande suivante, en observant le contenu du fichier ***names.txt*** avant **et** après son exécution :

```
sed /NOM_A/d names.txt
```

[v] Expliquez l'effet de cette commande. ■

Essayez maintenant :

```
sed s/NOM_B/NOM_A/ names.txt > new_names.txt
```

Puis encore :

```
sed s/NOM/PRENOM/ names.txt > new_new_names.txt
```

[w] Expliquez précisément l'effet de cette autre utilisation de la commande `sed`. ■

[x] Comment utilisez-vous `sed` pour modifier le fichier ***Candide_ chapitre1.txt*** du TP1 en remplaçant toutes les occurrences de "Candide" par "Romeo" et toutes les occurrences de "Cunegonde" par "Juliette". ■

Un script pour extraire les dépendances d'un fichier C

Lors de la création du graphe de dépendance d'un programme, il est nécessaire de connaître, pour chaque fichier .c, les fichiers .h inclus grâce à la directive

```
#include "header.h"
```

Nous allons réaliser ensemble un script qui permet d'extraire d'un fichier .c tous les fichiers .h qu'il inclut (hormis les librairies standard telles que `stdio.h`).

[y] Complétez le script suivant (attention à l'orientation des guillemets ' et ‘) :

```
#!/bin/bash
#Extraction des lignes contenant '#include "' dans le fichier passé en argument
LIGNES=‘grep ... $1‘
#Suppression de '#include "' puis du '"' restant en fin de ligne
echo "$LIGNES" | sed ... | sed 's/"//'
```

Utilisez ce script sur le fichier ***codage.c***. Quel est le résultat obtenu ? ■

Par défaut, la commande ***echo*** revient à la ligne. L'option ***-n*** passée à la commande ***echo*** permet de rester sur la même ligne :

```
>echo -n "AB" ; echo -n "CD"
 ABCD
```

[z] En utilisant une boucle ***for*** et l'option ***-n*** de ***echo***, modifiez le script précédent pour afficher les dépendances d'un fichier C en une seule ligne. ■

En fin de séance ...

N'oubliez pas de recopier l'ensemble de votre travail sur le compte de votre binôme (nous ne le répéterons plus dans les prochains énoncés !) :

```
scp -r TP3 login_binome@turing.e.ujf-grenoble.fr:INF123
```