TP4 – 'make' et pointeurs

Langage C (LC4) – semaine du 27 février 2012

1 Premiers pas avec `make`

1.1 L'outil `make` sans Makefile

Exercice 1 Considérez un ou plusieurs programmes des TPs précedents. Verifiez le sens de l'option `-c` de `gcc`, puis
- observez le temps d'execution de la commande `gcc -c <votre programme>.c` (commande `time`),
- puis observez le temps d'execution de la commande `make <votre programme>.o`.
- Effacez le fichier `<votre programme>.o` de votre répertoire courant et recommencez l'opération ci-dessus.

Qu'en concluez vous à propos de l'utilisation de `make` ?

1.2 Compilation separée et premier Makefile

Considérez les trois programmes suivants, et copiez les dans trois fichiers distincts que vous nommerez respectivement : `hello.c`, `hello.h`, `main.c`.

```
#include <stdio.h>

void hello()
{
  printf("Hello World\n");
}
```

```
#ifndef H_GL_HELLO
#define H_GL_HELLO

void hello();

#endif
```

```
#include <stdio.h>
#include "hello.h"

int main()
{
  hello();
  return 0;
}
```

Exercice 2 Une fois compilé que produit l'execution de ce programme ?

Exercice 3 À quoi semble servir la suite de directives `#ifndef`, `#define`, et `#endif` ?

Exercice 4 Établissez un ordre de dépendance entre les fichiers ci-dessus et les fichiers intermédiaires que vous pourriez construire au cours de la compilation (les fichiers objets, l'executable).

Exercice 5 En suivant le tutoriel `http://gl.developpez.com/tutoriel/outil/makefile/`, créez un `Makefile` simple pour compiler le programme ci-dessus, tout en générant tous les fichiers objets intermédiaires.

Améliorez votre `Makefile` afin qu'il vous permette aussi de nettoyer votre répertoire, c'est à dire d'en retirer tous les fichiers objets. Il s'agit d'écrire une règle supplémentaire dont la cible sera conventionnellement appelée `clean`.

2 Retour vers des exercices plus classiques

Dans les prochains exercices et TPs, il est désormais recommandé de créer un dossier spécifique et un `Makefile` spécifique à ce dossier. Pour ne pas aller trop vite nous vous fournissons un schéma de Makefile que vous pouvez modifier pour compiler vos exercices.

Normalement, il suffit d'insérer les noms de vos fichiers `*.c` dans la première ligne, apres vous pouvez compiler votre programme avec la commande `make` dans le terminal.

```
SOURCES=fichier1.c fichier2.c ...
OBJECTS=$(SOURCES:.c=.o)
CFLAGS= -std=c89 -Wall -Wextra
EXECUTABLE=exo

$(EXECUTABLE): $(OBJECTS)
        gcc -o $(EXECUTABLE) $(OBJECTS)

%.o : %.c
        gcc -c $(CFLAGS) $< -o $@

clean:
        rm -f $(EXECUTABLE) $(OBJECTS)
```

Exercice 6 Avant de passer à la suite essayez de comprendre le sens du `Makefile` proposé.

3 Des pointeurs

Exercice 7 Recopiez, compilez et exécutez le programme `ptr-return.c` suivant :

```
#include <stdio.h>

int *toto() {
    int x = 42;
    return &x;
}

void tata() {
    int a = 23;
}

int main(int argc, char *argv[]) {
    int *y = toto();
    tata();
    printf("%d\n", *y);
    return(0);
}
```

Essayez de comprendre ce qui s'est passé.

Exercice 8 Écrire une fonction

```
    char *recherche(char *s, char c)
```

qui renvoie un pointeur vers la première occurrence dans la chaîne `s` du caractère `c` passé en argument. Si ce caractère n'apparaît pas dans la chaîne, la fonction devra renvoyer `NULL`.

Exercice 9 À l'aide de la fonction précédente, écrire une fonction

```
    int compte(char *s, char c)
```

qui renvoie le nombre d'occurrences de `c` dans `s`.