CCI - Principes des langages de programmation
TD 1 : Introduction

Nicolas Bredeche[*], Marwan Burelle[†]

26 septembre 2005

1 Éditeur de texte

Avant de commencer à écrire vos programmes, il vous faut un éditeur de texte. Plusieurs sont disponibles, nous allons essayer les deux éditeurs les plus répandus sur les système de type UNIX : `emacs` et `vi`.

Il existe d'autres éditeurs de texte sous UNIX, l'éditeur `emacs` à l'avantage de fournir de nombreuse possibilités en terme d'édition de programme (coloration syntaxique, indentation intelligente, complétion, exécution de commande, contrôle des résultat de compilation ...) L'éditeur `vi` est l'un des éditeurs historiques d'UNIX, une version est toujours disponible sur toutes les variantes de système UNIX.

1.1 L'éditeur `emacs`

- Commencez par ouvrir un terminal
- Créer vous un répertoire `programmes` avec la commande `mkdir(1)`[1]
- Placez vous le répertoire ainsi créé (`cd programmes`) et lancez l'éditeur `emacs` en ouvrant un fichier `td1-emacs.c` à l'aide de la commande : `emacs td1-emacs.c &`
- Le fichier `td1-emacs.c` est vide, ajoutez-y un commentaire : `/* Commentaire */`
- Sauvez le fichier ainsi obtenu avec `C-x C-s`[2]
- Enfin pour quitter utiliser `C-x C-c`.

Vous pouvez activer la coloration syntaxique en tapant `M-x font-lock-fontify-buffer`. Pour effectuer une recherche dans le texte vous pouvez utiliser `C-s` (recherche vers l'avant) ou `C-r` (recherche vers l'arrière.) Pour effectuer des remplacement utiliser `M-%` (ou pour une version plus évoluer utilisant les *REGEXP* `M-C-%`.)

1.2 L'éditeur `vi`

L'éditeur de texte `vi` est basé sur un principe légèrement différent de ce que vous pouvez avoir l'habitude de rencontrer. Il existe deux modes : le mode commande qui vous permet d'effectuer les opérations ordinaires de types sauver, rechercher, quitter ou ouvrir un fichier ; le mode insertion qui vous permet de rajouter du texte dans le fichier ouvert. Passons à un petit exemple pour commencer :

- Placez vous dans le même répertoire que pour l'exercice précédant.
- Lancez l'éditeur `vi` avec la commande `vi td1-vi.c`
- Passer en mode insertion en appuyant sur la touche `i` et insérer un commentaire comme précédemment.
- Repasser en mode commande (`Esc`) et sauver grâce à la séquence de touche `:w`

[*] bredeche@lri.fr

[†] burelle@lri.fr

[1] lorsqu'un nom de commande est suivie d'un numéro entre parenthèse, il s'agit d'une référence à la section des pages du manuels présentant cette commande, ici la section 1, ainsi peut-on lire la page de manuel de `mkdir` grâce à la commande : `man 1 mkdir`. Notez que le numéro de section est facultatif s'il n'y a pas d'ambiguité sur la commande.

[2] `C` représente la touche *control*, `C-x` signifie pressez les touches `C` et `x` en même temps, tandis que `Esc x` signifie pressez la touche `Esc` puis la touche `x`. `M` représente la touche *Meta*, que vous pouvez obtenir de deux façon, soit en pressant la touche *escape* puis la touche suivante à utiliser soit en pressant la touche *alt* en même temps que la touche suivante à presser, par exemple `M-x` peut donc se faire en tapant `Esc x` ou `Alt-x`.

– Quitter `vi` en tapant `:q`

Il existe plusieurs versions de `vi`, plus ou moins évoluées, dans certaines versions (prévues à l'origine pour des claviers sans flèche de direction, entre autre) les déplacements (en mode commande) devaient s'effectuer à l'aide des touches `h` (une lettre à gauche), `j` (descendre d'une ligne), `k` (monter d'une ligne) et `l` (une lettre à droite), ces déplacement sont toujours possibles avec les versions récentes de vi, familiarisez vous avec.

De même, en mode commande, on peut supprimer un caractère à l'aide de la touche `x`, une ligne grâce à la combinaison `dd`, on peut copier une ligne grâce à `yy` et coller une ligne précédemment copiée ou effacée avec `p`.

`vi` dispose de bon nombre de commandes d'édition, comme les recherches (`/` en mode commande) et les remplacements à l'aide de *REGEXP*. Si vous décidez d'utiliser `vi` lisez la page de manuel correspondante qui vous expliquera toutes les subtilités.

2 Premier pas en C

Maintenant que vous avez choisi un éditeur de texte, nous pouvons passer à la programmation en C.

2.1 Un premier programme

Pour commencer nous allons partir d'un programme de type *Hello World* et rajouter quelques affichages dans la suite de l'exercice. Taper le programme suivant dans un fichier `td1.c` :

```
0  /* on commence par inclure les fonctions d'affichages de la
1     bibliotheque standard */
2  #include <stdio.h>
3
4  /* Fonction principale */
5  int main() {
6    printf("Hello World !") ;
7    exit(0) ;
8  }
```

2.2 Compilation et exécution

Pour pouvoir exécuter votre nouveau programme, il faut le compiler. Nous allons utiliser pour ça le compilateur `gcc` :

```
> gcc -o td1 td1.c
>
```

Si vous avez bien taper le programme, vous devriez récupérer la main directement après. vous pouvez maintenant exécuter votre programme :

```
> ./td1
>
```

Et là, en règle générale, il ne se passe rien ...

Surpris ? En fait, c'est tout à fait normal. Nous avons affiché une série de caractères sans indiquer de fin de ligne, or l'affichage attend l'indication de fin de ligne pour afficher ce que nous lui avons transmis, il faut donc rajouter cette indication (`\n`), on remplace la ligne 6 par :

```
6    printf("Hello World !\n") ;
```

Maintenant, reprenons la compilation et l'exécution :

```
> gcc -o td1 td1.c
> ./td1
Hello World !
>
```

2.3 Affichages plus poussés

Nous allons maintenant affichés des entiers et des flottants pour explorer un peu plus loin `printf`.

Commençons par un entier, pour afficher 1 on peut se contenter de `printf("1");`, mais ce n'est pas satisfaisant si cet entier est le résultat d'un calcul ou provient d'une variable. La technique consiste à insérer des *caractère de contrôle* dans la chaîne de caractères que l'on affiche et ensuite de fournir les valeurs qu'attendent ces caractères de contrôle. Pour notre entier ça nous donnera : `printf("%i",1)`.

On peut mélanger ces caractères de contrôle avec une chaîne de caractères ordinaire : `printf("Un entier : %i\nUn autre : %i\n",1,2)`. Insérer cette ligne entre la ligne 6 et la ligne 7 de notre programme, compilez et exécutez le, nous obtenons l'affichage :

```
> gcc -o td1 td1.c
> ./td1
Hello World !
Un entier : 1
Un autre : 2
>
```

Pour afficher un nombre flottant, le caractère de contrôle correspondant est `%f` que l'on peut accompagner de sa précision (nombre de caractère après la virgule, sous la forme `%.Xf` ou `X` est la précision, par défaut 6.)

Rajouter une ligne à notre programme pour afficher 1.23456789 avec une précision de 4 chiffres après la virgule.

3 Les différentes phases de la compilation

En langage C le code source peut être découpé en plusieurs fichiers sources. L'extension des fichiers à compiler est .c. Les fichier d'entête ont pour extension .h, ils ne contiennent que les prototypes des fonctions définies dans les .c correspondants.

La compilation en C se déroule en quatre étapes :

1. préprocessing ou prétraitement : il s'agit de la modification du texte d'un fichier source basée essentiellement sur l'interprétation d'instructions particulières. Celles-ci sont reconnaissables dans le code par le fait qu'elles commencent par le symbole #
2. compilation en assembleur (extension : .s)
3. assemblage : production d'un module objet (extension : .o) (rien à voir avec la programmation objet)
4. édition de liens : un module objet n'est pas un exécutable tel quel car il peut lui manquer d'autre modules objet lorsque par exemple le source est découpé en plusieurs fichiers. Il lui manque aussi les instructions exécutable des fonctions standard appelées dans le fichier source (par exemple printf de la bibliothèque stdio).

`gcc` permet de décomposer chaque phase de la compilation, l'option `-E` arrête la compilation après le préprocessing (renvoie le résultat sur la sortie standard), `-S` arrête la compilation avant l'assemblage et renvoie un fichier avec l'extension .s, enfin `-c` arrête la compilation après l'assemblage (et produit un fichier avec l'extension .o).

À partir de votre programme :

– engendrez la version après préprocessing (sauvez la dans un fichier `td1-preproc.c`)
– engendrez le code assembleur correspondant
– engendrez le version assemblé

ANNEXES

Quelques raccourcis `emacs`

`C-x C-f`	ouvrir un fichier dans le buffer courant
`C-x k`	fermer un buffer
`C-x C-c`	quitter
`C-x i`	Insérer un fichier dans le buffer courant

`C-x b`	passer à un autre buffer
`C-x 1`	aucune division de la fenêtre
`C-x 2`	division horizontal
`C-x 3`	division vertical
`C-x o`	passer dans la prochaine sous-fenêtre
`C-h i`	affichage des pages infos (version améliorer des page de manuel)

Quelques commandes `vi`

`:w`	sauver
`:q`	quitter
`:q!`	quitter même si le fichier n'a pas été sauvé
`:wq`	sauver et quitter
`:x`	sauver et quitter
`ZZ`	sauver et quitter
`x`	supprimer un caractère
`dd`	effacer une ligne
`yy`	copier une ligne
`p`	coller une ligne
`/`	rechercher