TP 03 - C sous Linux -

1 Les outils nécessaire

Sous GNU/Linux les outils nécessaire à l'édition et la compilation d'un programme C sont déjà présents, il s'agit du compilateur *gcc* et d'un éditeur de texte (Exemple: *gedit*, *mousepad*, *kate*, *nano*, *emacs*, *vi*, ...).

2 GCC (GNU Compiler Collection)

GCC est un ensemble de compilateurs créés par le projet GNU. C'est un logiciel libre capable de compiler divers langages de programmation, dont *C*, *C++*, *Objective-C*, *Java*, *Ada* et *Fortran*. Pour faire référence précisément aux compilateurs de chaque langage, on parle de:

- gcc pour C;
- G++ pour C++;
- GobjC pour Objective-C;
- GobjC++ pour Objective-C++;
- GCJ pour Java;
- GNAT pour Ada;
- Gfortran pour Fortran;
- gccgo pour Go.

3 Premier programme C sous Linux

- Créer le dossier *progc* dans votre répertoire *home*;
- Saisir le programme suivant:

```
#include<stdio.h>

void main() {
  printf("Bonjour\n");
}
```

- Enregistrer le programme dans le dossier *progc* sous le nom *bonjour.c*;
- Ouvrir un terminal et entrer dans le dossier *progc*;
- Afficher le contenu de ce dossier;

- Compiler le programme avec la commande suivante:

  ```
  $ gcc bonjour.c
  ```

- Afficher le contenu de ce répertoire une deuxième fois;
- Exécuter le fichier *a.out*:

  ```
  $ ./a.out
  ```

- Exécuter la commande suivante:

  ```
  $ gcc -o bonjour bonjour.c
  ```

- Afficher le contenu du dossier.

4 Quelques options de *gcc*

- **-E** Appelle le préprocesseur. N'effectue pas la compilation;
- **-S** Appelle le préprocesseur et effectue la compilation. N'effectue pas l'assemblage ni l'édition de lien. Seuls les fichiers assembleur "*.s*" sont générés;
- **-c** Appelle le préprocesseur, effectue la compilation et l'assemblage, mais ne fait pas l'édition de lien. Seuls les fichiers objets "*.o*" sont générés;
- **-o nom** Fixe le nom du fichier objet généré lors de la compilation d'un fichier source.

5 Paramètres de la ligne de commande

En C, on considère les paramètres de la fonction *main* comme les paramètres de la ligne de commande. Le premier paramètre *argc* indique le nombre de paramètres et le paramètre suivant *argv* est un tableau de chaines de caractères, chaque élément de ce tableau contient un paramètre.

Syntaxe de la fonction *main*

```
int main(int argc, char *argv[]) { ... }
```

Exemple Soit la commande *cp* suivante qui copie le *fichier1* vers *fichier2*

```
cp fichier1 fichier2
```

Dans ce cas, on aura *argc = 3* et

- *argv[0] = "cp"*
- *argv[1] = "fichier1"*
- *argv[2] = "fichier2"*

6 Les entrées et sorties de caractères

La bibliothèque standard fournit plusieurs fonctions pour lire ou écrire un caractère à la fois, parmis elle les fonctions *getchar* et *putchar*

6.1 Fonction *getchar*

Elle est utilisée pour lire un caractère.

Syntaxe

```
c = getchar();
```

La variable *c* va contenir le caractère suivant reçu en entrée. Ce caractère vient normalement du clavier.

6.2 Fonction *putchar*

Elle est utilisée pour écrire un caractère.

Syntaxe

```
putchar(c);
```

Elle affiche le contenu de la variable *c* sous la forme d'un caractère, en général sur l'écran.

Remarque Avec ces deux fonctions on peut écrire un nombre surprenant de programmes utiles sans rien savoir de plus sur les entrées-sorties.

6.3 EOF (*End Of File*)

Le problème avec ces deux fonctions est de détecter la fin des données en entrées. La fonction *getchar* retourne une valeur particulière lorsqu'il n'y a plus rien en entrée, cette valeur s'appelle *EOF*.

7 Exercices

Exercice 1 Écrire un programme qui permet d'afficher son nombre de paramètres suivi de la valeur de chacun des paramètres.

Exercice 2 Écrire un programme C qui copie son entrée sur sa sortie caractère par caractère.

Exercice 3 Écrire un programme C qu'on va appeller *eko* qui sera similaire à la commande *echo*, c'est-à-dire il va réécrire ses paramètres sur la sortie standard. Sa syntaxe sera comme suit:

```
eko [-Mm] [String ...]
```

Il y a deux options: *-M* indique que l'écriture se fait en majuscule, et *-m* en minuscule.