TP - gcc et ddd

1 Étude du fonctionnement de gcc [1]

Tapez avec votre editeur de texte favori [2] le programme C suivant (que vous nommerez exemple1.c) :

```
1 #define VAL 10
2 /* Le programme principal */
3 int main() {
4   int i=VAL;
5   i=i+1;
6   printf("Bonjour, la valeur de i est %d\n",i);
7   return i;
8 }
```

1.1 Compilation, exécution et récupération du résultat

1. À quoi servent les options `-Wall` et `-pedantic` de gcc ?
2. Qu'obtenez vous en tapant la commande `gcc -Wall -pedantic exemple1.c` (pour l'instant ne vous souciez pas du *warning* affiché) ?
3. Quelle option devez vous donner à `gcc` pour changer le nom de l'exécutable produit ?
4. Lancez le programme, et affichez juste après la valeur de la variable d'environnement `$?` à l'aide de la commande `echo`. Que constatez vous ?

1.2 Étude des différentes phases de la compilation

Comme l'indique le site Web donné en référence, la transformation d'un programme en langage C en un exécutable passe en fait par 4 phases que nous allons étudier.

1.2.1 Pré compilation

La première phase est la précompilation qui a pour but :
— de supprimer les commentaires,
— d'interpréter les macros,
— de vérifier la syntaxe.

1. Très fortement inspiré de http ://www.cmi.univ-mrs.fr/ contensi/coursC/index.php ?section=env&page=comp

2. Nous vous conseillons d'utiliser emacs, éditeur de texte "couteau suisse", qui bien qu'un peu difficile à appréhender est l'un des éditeurs de texte, avec vi, le plus efficace. Pour plus d'information : http ://www.linux-france.org/article/appli/emacs/debutant/howto/Emacs-Beginner-HOWTO-fr.html

On peut demander à `gcc` de s'arréter juste après cette étape à l'aide de l'option `-E` (attention dans ce cas, `gcc` affiche le code produit sur la sortie standard). Utilisez cette option sur l'exemple précédent, que constatez vous ?

1.2.2 Production du code assembleur

Cette seconde phase traduit le programme C de le phase précédente en code assembleur [3], c'est-à-dire dans un langage très proche de la machine mais encore compréhensible par un humain. On peut obtenir ce code en utilisant `gcc` avec l'option `-S`. Identifiez les lignes assembleur correspondant aux lignes 4 à 6 du programme C initial.

1.2.3 Production du code objet

Cette troisième phase produit le code objet (une suite d'octets interprétable par le microprocesseur de la machine que vous utilisez) à partir du code assembleur précédent et d'une table de correspondance "instruction assembleur - représentation machine(octet)" [4] : ce code objet est humainement illisible.

Avec `gcc` on obtient ce code objet grâce à l'option `-c`.

Que signifie les *warning* que nous avons continuellement depuis le début du TP ?

1.2.4 Édition des liens

Le code précédent n'est pas encore un exécutable car il utilise peut-être des bibliothèques annexes. L'édition des liens a pour objectif de rassembler l'ensemble des codes objet utilisés en un exécutable en associant pour chaque appel de fonction l'adresse mémoire de cette dernière.

Avec `gcc` on demande l'édition des liens en désignant simplement les codes objet utilisés. Faites l'édition de liens du programme *exemple1*. Pourquoi, malgré les *warning* précédent, votre programme fonctionne normalement ?

2 Utilisation de ddd

`ddd` est un débugger (surcouche graphique au débugger texte `gdb`).

2.1 Option de compilation pour le débuggage

1. Quelle option de `gcc` doit on utiliser pour pouvoir debugger un programme ?
2. Compilez le programme avec cette option. Analysez de nouveau le code assembleur. Que remarquez vous ?

2.2 Débuggage

On ajoute une petite fonction au programme précédent :

3. Cf. `http://en.wikipedia.org/wiki/X86_instruction_listings`
4. Par exemple, vous trouverez à l'adresse suivante la table des instructions du microprocesseur Z80 : http ://fms.komkon.org/MSX/Docs/Z80-1.txt

```c
#define VAL 10

int inc(int a) {
  return a+1;
}
/* Le programme principal */
int main() {
  int i=VAL;
  i=inc(i);
  printf("Bonjour, la valeur de i est %d\n",i);
  return i;
}
```

1. Compilez ce programme avec l'option de débuggage.
2. Exécutez ce programme pas à pas en visualisant les variables (mettez un point d'arrêt sur la première instruction). Quelle est la différence entre les commandes "next" et "step" ?

3 Compilation d'un programme de jeu

L'objectif maintenant est de compiler un programme de jeu (code issu du projet nsnake, `http://nsnake.sourceforge.net/`, modifié pour ce TP), composé d'un programme principal `main.c` et de plusieurs modules :

— `arguments.c`
— `engine.c`
— `fruit.c`
— `hscores.c`
— `nsnake.c`
— `player.c`

De plus ce jeu utilise une bibliothèque nommée `ncurses`.

Téléchargez l'archive du jeu sur moodle. Compilez les `.c` de façon à ce que les `.o` générés (de même nom) soit créés dans le répertoire `src`. Faites l'édition des liens pour que l'exécutable (nommé `nsnake`) soit créé dans le répertoire `bin`. À vous de jouer !

Informations complémentaires

Si vous voulez plus d'information concernant les différentes étapes de compilation d'un programme sous linux, je vous invite à lire l'article d'Alexandre Courbot, paru dans le « Linux Magazine » de Juin 2010, « Conception et vie d'un programme : les sous-traitants de GCC » (disponible à la biliothèque de l'INSA).