Premiers pas avec GCC

Fondements du langage C

Le langage C est très utilisé depuis des décennies dans différents domaines du développement logiciel. Il s'agit d'un langage généraliste élaboré au début des années 1970 par Ken Thomson et Dennis Ritchie afin de répondre aux exigences d'écriture du premier système d'exploitation Unix. Le langage C compte parmi les langages plus utilisés aujourd'hui. Il comporte des instructions et des structures de données de haut niveau tout en permettant des opérations de bas niveau notamment sur la mémoire et les ressources matérielles. Le langage C présente l'avantage d'être portable et il très performant du point de vue de son exécution et de son empreinte mémoire. Sa syntaxe est standardisée et répond à une norme ANSI [1].

Écriture du code source

Au-delà de la phase de conception, la première étape du développement d'un programme consiste à écrire le code source destiné à répondre aux besoins du programme à générer. Ce code source doit rigoureusement respecter la syntaxe du langage C mais aussi, le plus souvent, des conventions de codage destinées à apporter une meilleure lisibilité au code. Il s'agit alors de se conformer à un format prédéfini et structuré pour l'organisation des lignes de code écrites. Le code source correspond à du texte écrit à l'aide d'un éditeur classique ou proposant une coloration syntaxique correspondant au langage C.

Des commentaires sont également à insérer dans le code source afin d'améliorer la compréhension lors de l'analyse ou de la relecture du code. Il est en effet important que le code source soit correctement commenté notamment lorsqu'un certain niveau de complexité des traitements codés est atteint. En langage C et selon la norme ANSI, les commentaires doivent être insérés entre les marques `/*` et `*/`. Ils n'ont strictement aucune influence sur l'exécution du programme.

Le code ci-dessous montre un exemple simplifié de programmation en langage C :

```c
#include <stdio.h>

/* Fonction principale */
int main(void)
{
  int nb = 4;

  printf("Valeur : %d\n", nb);
  nb++;  /* Incrémentation */
  printf("Valeur : %d\n", nb);

  return 0;
}
```

Le traitement effectué par le code présenté en exemple se limite à l'affichage de la valeur initiale d'une variable entière et de sa valeur après une opération d'incrémentation.

Selon la syntaxe du langage C, chaque instruction se termine par un point virgule.

Il est important d'utiliser l'indentation pour améliorer la lisibilité et la forme du code. Il s'agit du décalage des lignes de code vers la droite notamment au sein d'un bloc de code placé entre les accolades { et }. Dans l'exemple présenté ci-dessus, les accolades délimitent le code associé à la fonction principale `main()`.

1. American National Standards Institute

Les fichiers relatifs au code source d'un programme écrit en langage C portent toujours l'extension `.c` ou `.h`. Les traitements sont écrits dans un ou plusieurs fichiers `.c` alors que l'extension `.h` est réservée aux fichiers d'entête décrivant notamment le prototype de certaines fonctions utilisés dans le reste du code. Par exemple, le fichier d'entête `stdio.h` contient la description et le format de la fonction `printf()` appelée dans la fonction principale `main()`.

Génération d'un fichier binaire exécutable

Une fois que le code source est écrit et mis en forme, il doit être traduit en langage machine afin de pouvoir être exécuté par la machine pour lequel il a été écrit. Cette opération est appelée *compilation*. Le résultat de la compilation est un fichier au format binaire. Il est ainsi constitué d'une succession de `0` et de `1` que le processeur de la machine est en mesure de traiter.

Il est nécessaire de disposer d'un logiciel spécifique afin de réaliser la compilation et donc la transformation du code source en code binaire. Ce logiciel est désigné par le terme *compilateur*.

L'un des compilateurs le plus utilisés pour traiter du code écrit en langage C est GCC [2]. Il est gratuit et très largement mis en œuvre pour le développement des logiciels libres.

L'appel au compilateur GCC se fait au moyen de la commande `gcc`. Cette commande est lancée avec un certain nombre d'arguments et d'options en fonction des besoins de la compilation. Le nom du fichier source à compiler est l'argument à passer au minimum à la ligne de commande pour l'appel du compilateur. Il est aussi conseillé d'activer les options d'affichage des avertissements liés à la correction du code source écrit. Par ailleurs, et sans précision du nom à donner au binaire généré, celui-ci est par défaut nommé `a.out`. Il est cependant possible d'utiliser l'option `-o` afin d'indiquer le nom à donner au fichier binaire à générer.

La ligne de commande présentée ci-dessous permet la compilation d'un code source contenu dans un unique fichier `exemple.c` et la génération d'un fichier binaire `exemple` :

```
gcc -Wall exemple.c -o exemple
```

2. *GNU C Compiler* à l'origine puis *GNU Compiler Collection* suite au support progressif d'autres langages de programmation

L'option `-Wall` active l'affichage des messages d'avertissement. Elle est très utile lorsqu'il s'agit de produire un programme performant, de le tester et de s'assurer de sa bonne exécution. Elle permet notamment de mettre en évidence des variables déclarées mais qui ne sont jamais utilisées et qui occupent ainsi inutilement de la mémoire.

Préalablement à l'opération de compilation proprement dite, l'exécution de la commande `gcc` effectue automatiquement un appel à un préprocesseur qui assure des étapes préparatoires à la compilation. Il s'agit notamment de traiter les inclusions de fichiers d'entête réalisées par la directive `#include` en recopiant le contenu du fichier passé en paramètre dans le fichier courant. Il peut également s'agir de directives spécifiques destinées à gérer une compilation conditionnelle mais aussi à définir des macros ou des constantes au moyen de la directive `#define`. Par exemple, la définition de la constante π peut se faire au moyen de la ligne suivante :

```
#define PI 3.14159
```

Concernant les macros, le code ci-dessous montre un exemple de définition d'une macro :

```
#define BONJOUR() printf("Bonjour\n")
```

Par convention, les constantes et les macros sont exprimées en lettres majuscules. Selon les exemples présentés ci-dessus, le préprocesseur remplacera systématiquement toutes les occurences de `PI` et de `BONJOUR()` rencontrées dans le code respectivement par `3.14159` et `printf("Bonjour\n")` avant l'exécution de la compilation.

Le code source ci-dessous présente un exemple de définition et d'utilisation d'une macro et d'une constante :

```
#include <stdio.h>

/* Définition de macro
   et de constante      */
#define MSG() printf("Début...\n")
#define NB 9

/* Fonction principale */
int main(void)
{
  MSG();
  printf("Valeur : %d\n", NB);

  return 0;
}
```

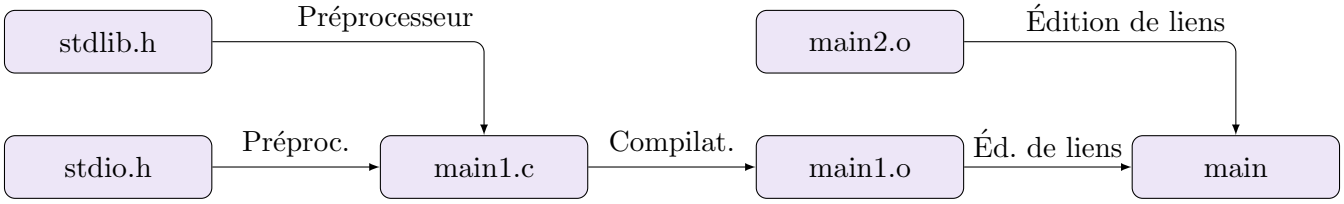

Fig. 1 – *Exemple de déroulement d'une séquence complète de génération d'un fichier binaire*

La figure 1 montre le déroulement complet de la séquence de génération d'un fichier binaire correspondant à un programme. La compilation, à proprement parler, n'est ainsi qu'une opération spécifique parmi les autres. Cependant, il est dans l'usage courant de considérer la compilation comme le traitement global des sources pour générer le fichier binaire.

La terminologie la plus adaptée à cette succession d'opérations et à l'ensemble des outils logiciels associés est de désigner la séquence comme une étant une *chaîne de compilation* incluant le préprocesseur, le compilateur et l'éditeur de liens. D'autres outils logiciels destinés à des opérations spécifiques complétent généralement une chaîne de compilation.

Le compilateur ne produit en fait que des fichiers binaires de type *objet* portant l'extension `.o`. Ces fichiers ne sont pas directement exécutables et sont généralement destinés à être fusionnés avec d'autres lorsque le code source est constitué de plusieurs fichiers source. L'opération de fusion des différents fichiers objet est appelée *édition de liens* et c'est au terme de cette opération que le fichier binaire exécutable est effectivement généré.

Il est tout à fait possible de faire appel au préprocesseur puis au compilateur GCC pour générer un fichier objet sans activer l'éditeur de liens. Il faut alors passer l'option `-c` à la ligne de commande. La commande est alors la suivante :

```
gcc -Wall exemple.c -c
```

Pour effectuer ensuite l'édition de liens et transformer ainsi le fichier objet en binaire exécutable, il suffit de lancer la commande ci-dessous :

```
gcc exemple.o -o exemple
```

La compilation sans effectuer l'édition de liens est utile dans le contexte où le programme à produire est réparti sur plusieurs fichiers source. Il est alors nécessaire de compiler individuellement chaque fichier source avant de les fusionner au sein d'un même binaire exécutable. Ce mode opératoire est appelé *compilation séparée* ou *compilation modulaire*. Un exemple d'enchaînement des commandes à exécuter pour la génération d'un fichier binaire exécutable à partir de deux fichiers source est présenté ci-dessous :

```
gcc -Wall exemple1.c -c
gcc -Wall exemple2.c -c
gcc exemple1.o exemple2.o -o exemple
```

Par ailleurs, un fichier objet peut être utilisable dans différents projets de programmation. En effet, il peut être utile d'isoler certains traitements spécifiques dans un fichier objet s'ils doivent être pris en compte dans plusieurs programmes plutôt que de les réécrire et de les compiler systématiquement.

Exécution du programme

Dès lors que la compilation du code s'est correctement déroulée et que le fichier binaire a été généré, l'exécution du programme se limite à une simple commande :

```
./exemple
```

L'exécution du programme se lance donc en ligne de commande à partir d'un émulateur de terminal.