Chaîne de compilation

Lycée Thiers

Crédits

- Un cours de Christophe Rippert à l'ENSIMAG
- Un tuto sur KOOR.fr

Avertissement

Nous utilisons toujours le langage **C** tel que défini dans la norme « C99 ».

Historique

- C : langage de programmation impératif généraliste, de *bas niveau* (il gère les échanges RAM/processeur). Mais pas de *très bas niveau* sinon il gérerait aussi ce qui se passe dans le processeur. Toujours très utilisé.

Historique

- C : langage de programmation impératif généraliste, de *bas niveau* (il gère les échanges RAM/processeur). Mais pas de *très bas niveau* sinon il gérerait aussi ce qui se passe dans le processeur. Toujours très utilisé.
- Création début des années 70 pour réécrire UNIX

Historique

- C : langage de programmation impératif généraliste, de *bas niveau* (il gère les échanges RAM/processeur). Mais pas de *très bas niveau* sinon il gérerait aussi ce qui se passe dans le processeur. Toujours très utilisé.
- Création début des années 70 pour réécrire UNIX
- A inspiré de nombreux langages plus modernes comme C++, C#, Java et PHP ou Javascript.

Historique

- C : langage de programmation impératif généraliste, de *bas niveau* (il gère les échanges RAM/processeur). Mais pas de *très bas niveau* sinon il gérerait aussi ce qui se passe dans le processeur. Toujours très utilisé.
- Création début des années 70 pour réécrire UNIX
- A inspiré de nombreux langages plus modernes comme C++, C#, Java et PHP ou Javascript.
- Utilisé par exemple pour réaliser les bases (compilateurs, interpréteurs...) des langages plus modernes.

Historique

- C : langage de programmation impératif généraliste, de *bas niveau* (il gère les échanges RAM/processeur). Mais pas de *très bas niveau* sinon il gérerait aussi ce qui se passe dans le processeur. Toujours très utilisé.
- Création début des années 70 pour réécrire UNIX
- A inspiré de nombreux langages plus modernes comme C++, C#, Java et PHP ou Javascript.
- Utilisé par exemple pour réaliser les bases (compilateurs, interpréteurs...) des langages plus modernes.
- C offre au développeur une marge de contrôle importante sur la machine (notamment sur la gestion de la mémoire)

Du fichier source au langage machine

La traduction d'un (ou plusieurs) fichiers sources en C vers le langage machine se fait en 2 étapes indépendantes et souvent enchaînées automatiquement (donc difficile à distinguer pour le débutant).

Prétraitement Opération purement textuelle consistant par exemple à supprimer les *commentaires*,

Du fichier source au langage machine

La traduction d'un (ou plusieurs) fichiers sources en C vers le langage machine se fait en 2 étapes indépendantes et souvent enchaînées automatiquement (donc difficile à distinguer pour le débutant).

Prétraitement Opération purement textuelle consistant par exemple à supprimer les *commentaires*,

Compilation . En 3 sous-étapes : Compilation proprement dite, Assemblage, Édition de liens

Du fichier source au langage machine

Compilation proprement dite : transforme le fichier généré par le préprocesseur en *assembleur* (une suite d'instructions du microprocesseur lisibles -avec de l'entraînement- par un humain),

Du fichier source au langage machine

Compilation proprement dite : transforme le fichier généré par le préprocesseur en *assembleur* (une suite d'instructions du microprocesseur lisibles -avec de l'entraînement- par un humain),

Assemblage : transforme le code assembleur en instructions binaires (donc directement compréhensibles par le processeur) Cette instruction et la précédente sont généralement effectuées dans la foulée l'une de l'autre sauf précision contraire passée en argument du compilateur.. Fichier binaire produit appelé *fichier objet* (ou *module objet*).

Du fichier source au langage machine

Compilation proprement dite : transforme le fichier généré par le préprocesseur en *assembleur* (une suite d'instructions du microprocesseur lisibles -avec de l'entraînement- par un humain),

Assemblage : transforme le code assembleur en instructions binaires (donc directement compréhensibles par le processeur) Cette instruction et la précédente sont généralement effectuées dans la foulée l'une de l'autre sauf précision contraire passée en argument du compilateur.. Fichier binaire produit appelé *fichier objet* (ou *module objet*).

Edition de liens génération des liens entre les différents fichiers objets du projet. En effet, un programme complexe est en général écrit sur plusieurs fichiers différents. L'éditeur de lien produit un fichier *exécutable*.

Environnement

Choix pour la MP2I à Thiers

- Un éditeur de texte **emacs** tournant sous Linux couplé au compilateur C **gcc** lancé depuis un terminal

Environnement

Choix pour la MP2I à Thiers

- Un éditeur de texte **emacs** tournant sous Linux couplé au compilateur C **gcc** lancé depuis un terminal
- Avantage : s'installe sans (trop) de difficultés pour la plupart des configurations Linux.

Environnement

Choix pour la MP2I à Thiers

- Un éditeur de texte **emacs** tournant sous Linux couplé au compilateur C **gcc** lancé depuis un terminal
- Avantage : s'installe sans (trop) de difficultés pour la plupart des configurations Linux.
- Par exemple, sous Ubuntu, puisque **gcc** et le terminal sont fournis par défaut, il ne reste qu'à installer **emacs** :

```
$ sudo apt-get install emacs
```

Un code

```c
/* Le fichier hello.c
   Mon premier code C */

#include <stdio.h> // pour communiquer avec des fichier
#include <stdlib.h> // pour gérer la mémoire

int main()
{
    int i = 10;
    printf("Hello world!\n");// affichage puis passage à la ligne
    printf("%i\n",i);// affichage de i
    return 0;// renvoyer 0 : c.a.d tout s'est bien passé
}
```

Compiler puis exécuter

```
$ gcc hello.c -o hello
$ ./hello
Hello world!
i=10
```

Présentation

GCC GNU Compiler Collection, abrégé en GCC, est un ensemble de compilateurs créés par le projet GNU. Nous l'utilisons pour C mais il peut aussi compiler Java, Ada, Fortran...

Présentation

GCC GNU Compiler Collection, abrégé en GCC, est un ensemble de compilateurs créés par le projet GNU. Nous l'utilisons pour C mais il peut aussi compiler Java, Ada, Fortran...

Trois composantes GCC désigne :

Présentation

GCC GNU Compiler Collection, abrégé en GCC, est un ensemble de compilateurs créés par le projet GNU. Nous l'utilisons pour C mais il peut aussi compiler Java, Ada, Fortran...

Trois composantes GCC désigne :

- la collection complète de compilateurs (le ≪ projet GCC ≫)

Présentation

GCC GNU Compiler Collection, abrégé en GCC, est un ensemble de compilateurs créés par le projet GNU. Nous l'utilisons pour C mais il peut aussi compiler Java, Ada, Fortran...

Trois composantes GCC désigne :

- la collection complète de compilateurs (le « projet GCC »)
- la partie commune à tous les compilateurs (« GCC ») ;

Présentation

GCC GNU Compiler Collection, abrégé en GCC, est un ensemble de compilateurs créés par le projet GNU. Nous l'utilisons pour C mais il peut aussi compiler Java, Ada, Fortran...

Trois composantes GCC désigne :

- la collection complète de compilateurs (le « projet GCC »)
- la partie commune à tous les compilateurs (« GCC ») ;
- le compilateur C lui-même (le frontend `gcc`), écrit en minuscule).

Présentation

GCC GNU Compiler Collection, abrégé en GCC, est un ensemble de compilateurs créés par le projet GNU. Nous l'utilisons pour C mais il peut aussi compiler Java, Ada, Fortran...

Trois composantes GCC désigne :

- la collection complète de compilateurs (le « projet GCC »)
- la partie commune à tous les compilateurs (« GCC ») ;
- le compilateur C lui-même (le frontend `gcc`), écrit en minuscule).

frontend `gcc` est un une application *frontale* (en anglais « frontend »), ce qui signifie que l'utilisateur interragit directement avec lui.

L'application `gcc` ♡

`gcc` permet d'appeler dans l'ordre le préprocesseur, le compilateur, l'assembleur et le *linker* (lieur) pour le langage C (et aussi C++).
Opérations réalisées par **gcc** :

- appel du préprocesseur C (nommé **cpp**) ;

L'application `gcc` ♡

`gcc` permet d'appeler dans l'ordre le préprocesseur, le compilateur, l'assembleur et le *linker* (lieur) pour le langage C (et aussi C++).
Opérations réalisées par **gcc** :

- appel du préprocesseur C (nommé **cpp**) ;
- appel du compilateur C (nommé **cc**). Le compilateur génère un fichier assembleur ;

L'application `gcc` ♡

`gcc` permet d'appeler dans l'ordre le préprocesseur, le compilateur, l'assembleur et le *linker* (lieur) pour le langage C (et aussi C++).
Opérations réalisées par **gcc** :

- appel du préprocesseur C (nommé **cpp**) ;
- appel du compilateur C (nommé **cc**). Le compilateur génère un fichier assembleur ;
- appel de l'assembleur (**as**) pour générer le fichier objet ;

L'application `gcc` ♡

`gcc` permet d'appeler dans l'ordre le préprocesseur, le compilateur, l'assembleur et le *linker* (lieur) pour le langage C (et aussi C++).
Opérations réalisées par **gcc** :

- appel du préprocesseur C (nommé **cpp**) ;
- appel du compilateur C (nommé **cc**). Le compilateur génère un fichier assembleur ;
- appel de l'assembleur (**as**) pour générer le fichier objet ;
- appel de l'éditeur de liens (**ld**) pour générer l'exécutable ou la bibliothèque. La librairie C standard est incluse par défaut dans la ligne de commande de l'édition de liens.

Prétraitement

Le pré-processeur est un module de pré-traitement qui effectue des transformations sur le code écrit avant qu'il ne soit traité par le compilateur.

- Consiste en la modification du texte d'un fichier source basée sur l'interprétation des *directives à destination du préprocesseur*. ♡

Prétraitement

Le pré-processeur est un module de pré-traitement qui effectue des transformations sur le code écrit avant qu'il ne soit traité par le compilateur.

- Consiste en la modification du texte d'un fichier source basée sur l'interprétation des *directives à destination du préprocesseur*. ♡
- On les reconnaît parce qu'elles commencent par un hastag **#**. Elle ne se terminent pas par un point-virgule `;` .

Prétraitement

Le pré-processeur est un module de pré-traitement qui effectue des transformations sur le code écrit avant qu'il ne soit traité par le compilateur.

- Consiste en la modification du texte d'un fichier source basée sur l'interprétation des *directives à destination du préprocesseur*. ♡
- On les reconnaît parce qu'elles commencent par un hastag **#**. Elle ne se terminent pas par un point-virgule `;` .
- Les deux directives les plus importantes sont

Prétraitement

Le pré-processeur est un module de pré-traitement qui effectue des transformations sur le code écrit avant qu'il ne soit traité par le compilateur.

- Consiste en la modification du texte d'un fichier source basée sur l'interprétation des *directives à destination du préprocesseur*. ♡
- On les reconnaît parce qu'elles commencent par un hastag **#**. Elle ne se terminent pas par un point-virgule `;` .
- Les deux directives les plus importantes sont

  #include : inclusion d'autres fichiers source. Récursif : si **A** inclut **B** qui inclut **C**, après pré-processing, **A** contient **C**. ♡

Prétraitement

Le pré-processeur est un module de pré-traitement qui effectue des transformations sur le code écrit avant qu'il ne soit traité par le compilateur.

- Consiste en la modification du texte d'un fichier source basée sur l'interprétation des *directives à destination du préprocesseur*. ♡
- On les reconnaît parce qu'elles commencent par un hastag **#**. Elle ne se terminent pas par un point-virgule `;` .
- Les deux directives les plus importantes sont

#include : inclusion d'autres fichiers source. Récursif : si **A** inclut **B** qui inclut **C**, après pré-processing, **A** contient **C**. ♡

#define : définition de macros ou symboles (c'est à dire des bout de code qui seront recopiées tels-quels dans le code C). ♡

Prétraitement

Le pré-processeur est un module de pré-traitement qui effectue des transformations sur le code écrit avant qu'il ne soit traité par le compilateur.

- Consiste en la modification du texte d'un fichier source basée sur l'interprétation des *directives à destination du préprocesseur*. ♡
- On les reconnaît parce qu'elles commencent par un hastag **#**. Elle ne se terminent pas par un point-virgule `;` .
- Les deux directives les plus importantes sont

  #include : inclusion d'autres fichiers source. Récursif : si **A** inclut **B** qui inclut **C**, après pré-processing, **A** contient **C**. ♡

  #define : définition de macros ou symboles (c'est à dire des bout de code qui seront recopiées tels-quels dans le code C). ♡

- D'autres directives **# if, # ifndef** et **# endif**, utilisées lors de l'écriture de fichiers d'en-tête, seront présentées ultérieurement (pour la compilation de projets sur plusieurs fichiers).

Pré-processeur

- Effectue des remplacements textuels (et non sémantiques) : le pré-processeur n'est pas le compilateur, il n'interprète pas ce qu'il manipule. Analogie : fonction **Remplacer** d'un traitement de texte. ♡

Pré-processeur

- Effectue des remplacements textuels (et non sémantiques) : le pré-processeur n'est pas le compilateur, il n'interprète pas ce qu'il manipule. Analogie : fonction **Remplacer** d'un traitement de texte. ♡
- Avec `#define NOM val`, le pré-processeur remplace chaque occurence de la chaine **NOM** par la valeur **val** dans tout le texte du fichier. ♡

Pré-processeur

- Effectue des remplacements textuels (et non sémantiques) : le pré-processeur n'est pas le compilateur, il n'interprète pas ce qu'il manipule. Analogie : fonction **Remplacer** d'un traitement de texte. ♡
- Avec `#define NOM val`, le pré-processeur remplace chaque occurence de la chaine **NOM** par la valeur **val** dans tout le texte du fichier. ♡
- Entrer `cpp hello.c hello.i` puis vérifier l'explosion de la taille de **hello.i**.

Compilateur

Traduire le code C en code assembleur (programme `cc`).

- Compilation séparée : chaque fichier source est compilé sans regarder les autres fichiers, même s'ils font partie du même programme

Compilateur

Traduire le code C en code assembleur (programme `cc`).

- Compilation séparée : chaque fichier source est compilé sans regarder les autres fichiers, même s'ils font partie du même programme
- Le compilateur a juste besoin de connaitre les *prototypes* (on dit aussi *signatures*) de toutes les fonctions utilisées dans le fichier compilé, mais il n'a pas besoin d'avoir à sa disposition le code des fonctions externes.

Compilateur

Traduire le code C en code assembleur (programme `cc`).

- Compilation séparée : chaque fichier source est compilé sans regarder les autres fichiers, même s'ils font partie du même programme
- Le compilateur a juste besoin de connaitre les *prototypes* (on dit aussi *signatures*) de toutes les fonctions utilisées dans le fichier compilé, mais il n'a pas besoin d'avoir à sa disposition le code des fonctions externes.
- Les prototypes des fonctions sont définies dans des fichiers `.h`, qui sont inclus par le pré-processeur (via la directive `#include`) dans le fichier C qui les utilise.

Compilateur

Traduire le code C en code assembleur (programme `cc`).

- Compilation séparée : chaque fichier source est compilé sans regarder les autres fichiers, même s'ils font partie du même programme
- Le compilateur a juste besoin de connaitre les *prototypes* (on dit aussi *signatures*) de toutes les fonctions utilisées dans le fichier compilé, mais il n'a pas besoin d'avoir à sa disposition le code des fonctions externes.
- Les prototypes des fonctions sont définies dans des fichiers `.h`, qui sont inclus par le pré-processeur (via la directive `#include`) dans le fichier C qui les utilise.
- Entrer `cc -S hello.i`. On obtient **hello.s**.
  L'ouvrir et vérifier par exemple la mention du registre 32 bits **%eax**, du pointeur de sommet de pile `%rsp` et du pointeur de cadre de pile `%rbp`

L'assembleur

- L'assembleur `as` prend un fichier écrit en langage assembleur et le traduit en code binaire (langage machine).

L'assembleur

- L'assembleur `as` prend un fichier écrit en langage assembleur et le traduit en code binaire (langage machine).
- Avec `as -o hello.o hello.s`, l'assembleur produit un fichier objet `hello.o` illisible pour un humain et non encore exploitable tel quel (il manque les *liens*).

Édition de liens

- L'éditeur de lien `ld` a pour rôle de fusionner les différents fichiers objets pour produire l'exécutable final. Il fusionne :

Édition de liens

- L'éditeur de lien `ld` a pour rôle de fusionner les différents fichiers objets pour produire l'exécutable final. Il fusionne :
  - les codes contenus dans les différentes fonctions de tous les fichiers objets du programme, pour obtenir une seule grosse zone de code ;

Édition de liens

- L'éditeur de lien `ld` a pour rôle de fusionner les différents fichiers objets pour produire l'exécutable final. Il fusionne :
  - les codes contenus dans les différentes fonctions de tous les fichiers objets du programme, pour obtenir une seule grosse zone de code ;
  - les données statiques allouées dans tous les fichiers objets du programme, pour obtenir une grosse zone de données statiques commune à tout le programme.

Édition de liens

- L'éditeur de lien `ld` a pour rôle de fusionner les différents fichiers objets pour produire l'exécutable final. Il fusionne :
  - les codes contenus dans les différentes fonctions de tous les fichiers objets du programme, pour obtenir une seule grosse zone de code ;
  - les données statiques allouées dans tous les fichiers objets du programme, pour obtenir une grosse zone de données statiques commune à tout le programme.
- Un programme C utilise toujours d'autres fichiers objets contenus dans la bibliothèque standard **libc.a**. Par exemple le fichier objet contenant le code de `printf` mais aussi d'autres fichiers qui permettent de rendre le code exécutable.

Édition de liens `ld`

- Dans la compilation de **hello.c**, l'assembleur a laissé un « trou » à l'endroit de l'appel à `printf`. C'est une indication pour insérer à cet endroit un véritable appel à la fonction `printf` (on trouve sur ma machine le fichier objet `stdio.o` dans l'archive **/usr/lib/x86_64-linux-gnu/libc.a**)

Édition de liens `ld`

- Dans la compilation de **hello.c**, l'assembleur a laissé un « trou » à l'endroit de l'appel à `printf`. C'est une indication pour insérer à cet endroit un véritable appel à la fonction `printf` (on trouve sur ma machine le fichier objet `stdio.o` dans l'archive **/usr/lib/x86_64-linux-gnu/libc.a**)
- Si on appelle dans un fichier **.c** une fonction mal orthographiée ou qui n'existe pas, c'est l'éditeur de lien qui repère l'erreur.

Édition de liens `ld`

- Dans la compilation de **hello.c**, l'assembleur a laissé un « trou » à l'endroit de l'appel à `printf`. C'est une indication pour insérer à cet endroit un véritable appel à la fonction `printf` (on trouve sur ma machine le fichier objet `stdio.o` dans l'archive **/usr/lib/x86_64-linux-gnu/libc.a**)
- Si on appelle dans un fichier **.c** une fonction mal orthographiée ou qui n'existe pas, c'est l'éditeur de lien qui repère l'erreur.
- Oublions d'écrire un `main` dans le projet. Un des fichiers inclus automatiquement par l'éditeur de lien inclu un appel à `main` lequel est cherché dans les fichiers sources qu'on lui donne. `ld` déclenche une erreur.

Fichiers provisoires ♡

Au cours de la compilation, des fichiers provisoires sont produits. Ils sont effacés par le compilateur quand ce dernier termine sa tâche :

- **.c** : fichier source contenant le programme (et ses commentaires) écrit par le programmeur,

Fichiers provisoires ♡

Au cours de la compilation, des fichiers provisoires sont produits. Ils sont effacés par le compilateur quand ce dernier termine sa tâche :

- **.c** : fichier source contenant le programme (et ses commentaires) écrit par le programmeur,
- **.i** : fichiers générés par le préprocesseur (ce sont des fichiers textes),

Fichiers provisoires ♡

Au cours de la compilation, des fichiers provisoires sont produits. Ils sont effacés par le compilateur quand ce dernier termine sa tâche :

- **.c** : fichier source contenant le programme (et ses commentaires) écrit par le programmeur,
- **.i** : fichiers générés par le préprocesseur (ce sont des fichiers textes),
- **.s** : fichiers assembleurs (donc lisibles par un humain),

Fichiers provisoires ♡

Au cours de la compilation, des fichiers provisoires sont produits. Ils sont effacés par le compilateur quand ce dernier termine sa tâche :

- **.c** : fichier source contenant le programme (et ses commentaires) écrit par le programmeur,
- **.i** : fichiers générés par le préprocesseur (ce sont des fichiers textes),
- **.s** : fichiers assembleurs (donc lisibles par un humain),
- **.o** : fichiers objets (binaires) produits après l'assemblage.

Bibliothèques

- « Bibliothèque » est la traduction de l'anglais « library ». On utilise improprement en français le mot « librairie ».

Bibliothèques

- « Bibliothèque » est la traduction de l'anglais « library ». On utilise improprement en français le mot « librairie ».
- Les fichiers **.a** : Archives constituées de fichiers objets correspondant aux librairies précompilées (par exemple la librairie standard d'entrées-sorties).

Bibliothèques

- « Bibliothèque » est la traduction de l'anglais « library ». On utilise improprement en français le mot « librairie ».
- Les fichiers **.a** : Archives constituées de fichiers objets correspondant aux librairies précompilées (par exemple la librairie standard d'entrées-sorties).
- Exemple **libc.a** : archive contenant notamment **stdio.o**.

gcc et bibliothèques

Option `-l` de `gcc`

- Les éventuelles librairies précompilées utilisées sont déclarées par la chaîne **-l*bibliothèqueDésirée*** et se situent souvent dans un sous-répertoire de /**usr**/**lib**/.

gcc et bibliothèques

Option `-l` de `gcc`

- Les éventuelles librairies précompilées utilisées sont déclarées par la chaîne **-l*bibliothèqueDésirée*** et se situent souvent dans un sous-répertoire de /**usr/lib**/.
- Exemple : la librairie mathématique **libm.a** est située sur mon ordinateur dans /**usr/lib/x86_64-linux-gnu**/
  Pour la trouver, entrer :

```
$ find /usr/lib -name libm.a # affiche localisation de libm.a
```

gcc et bibliothèques

Option `-l` de `gcc`

- Les éventuelles librairies précompilées utilisées sont déclarées par la chaîne **-l*bibliothèqueDésirée*** et se situent souvent dans un sous-répertoire de /**usr/lib**/.
- Exemple : la librairie mathématique **libm.a** est située sur mon ordinateur dans /**usr/lib/x86_64-linux-gnu**/
  Pour la trouver, entrer :

```
$ find /usr/lib -name libm.a # affiche localisation de libm.a
```

- Un nom de librairie commence par **lib**. A la suite de `-l` on n'écrit que ce qui est après « **lib** » et avant le point.

gcc et bibliothèques

Option `-l` de `gcc`

- Les éventuelles librairies précompilées utilisées sont déclarées par la chaîne **-l*bibliothèqueDésirée*** et se situent souvent dans un sous-répertoire de /**usr/lib**/.
- Exemple : la librairie mathématique **libm.a** est située sur mon ordinateur dans /**usr/lib/x86_64-linux-gnu**/
  Pour la trouver, entrer :

```
$ find /usr/lib -name libm.a # affiche localisation de libm.a
```

- Un nom de librairie commence par **lib**. A la suite de `-l` on n'écrit que ce qui est après « **lib** » et avant le point.
- Pour utiliser **libm.a** et ses fonctions dans le programme **myprog** : ♡

```
$ gcc myprog.c -lm -o myprog
```

Encore plus sur **libm.a**

- Un certain nombre de fonctions de la librairie **libm.a** sont des fonctions intégrées du compilateur (*built-in functions*).

Encore plus sur **libm.a**

- Un certain nombre de fonctions de la librairie **libm.a** sont des fonctions intégrées du compilateur (*built-in functions*).
- Pour des fonctions comme `cos`, la directive `#include <math.h>` est toujours nécessaire mais pas (ou pas toujours) l'option `-lm` de **gcc**.

Encore plus sur **libm.a**

- Un certain nombre de fonctions de la librairie **libm.a** sont des fonctions intégrées du compilateur (*built-in functions*).
- Pour des fonctions comme `cos`, la directive `#include <math.h>` est toujours nécessaire mais pas (ou pas toujours) l'option `-lm` de **gcc**.
- En revanche, d'autres fonctions comme `nextafterf` ne sont toujours pas des fonctions intégrées du compilateur et nécessite encore l'option `-lm` de **gcc**. Du moins sur ma machine...

Accès aux bibliothèques

- Les fonctions dont les prototypes sont dans **stdlib.h** et **stdio.h** sont écrites dans la librairie **libc** laquelle est importée par défaut par `gcc`.
  `gcc truc.c` se comprend donc comme `gcc truc.c -lc`.
  L'usage de `-lc` est donc implicite.

Accès aux bibliothèques

- Les fonctions dont les prototypes sont dans **stdlib.h** et **stdio.h** sont écrites dans la librairie **libc** laquelle est importée par défaut par `gcc`.

  `gcc truc.c` se comprend donc comme `gcc truc.c -lc`.

  L'usage de `-lc` est donc implicite.
- Les librairies précompilées sont cherchées par défaut dans le répertoire **/usr/lib** et ses descendants.

Accès aux bibliothèques

- Les fonctions dont les prototypes sont dans **stdlib.h** et **stdio.h** sont écrites dans la librairie **libc** laquelle est importée par défaut par `gcc`.

  `gcc truc.c` se comprend donc comme `gcc truc.c -lc`.

  L'usage de `-lc` est donc implicite.
- Les librairies précompilées sont cherchées par défaut dans le répertoire **/usr/lib** et ses descendants.
- Mais il peut se produire qu'elles ne se trouvent pas dans ce répertoire usuel, on spécifie alors leur chemin d'accès par l'option `-L`.

Options de production de fichiers

On peut demander à gcc de produire les fichiers intermédiaires manipulés par les différents programmes qu'il appelle si on veut les observer :

- `gcc -E -o hello.i hello.c` produit le fichier en sortie du pré-processeur,

Options de production de fichiers

On peut demander à gcc de produire les fichiers intermédiaires manipulés par les différents programmes qu'il appelle si on veut les observer :

- `gcc -E -o hello.i hello.c` produit le fichier en sortie du pré-processeur,
- `gcc -S hello.c` produit le fichier en sortie du compilateur (après avoir appelé aussi le pré-processeur)

Options de production de fichiers

On peut demander à gcc de produire les fichiers intermédiaires manipulés par les différents programmes qu'il appelle si on veut les observer :

- `gcc -E -o hello.i hello.c` produit le fichier en sortie du pré-processeur,
- `gcc -S hello.c` produit le fichier en sortie du compilateur (après avoir appelé aussi le pré-processeur)
- `gcc -c hello.c` produit le fichier en sortie de l'assembleur (après avoir appelé aussi le pré-processeur et le compilateur)

Options de production de fichiers

On peut demander à gcc de produire les fichiers intermédiaires manipulés par les différents programmes qu'il appelle si on veut les observer :

- `gcc -E -o hello.i hello.c` produit le fichier en sortie du pré-processeur,
- `gcc -S hello.c` produit le fichier en sortie du compilateur (après avoir appelé aussi le pré-processeur)
- `gcc -c hello.c` produit le fichier en sortie de l'assembleur (après avoir appelé aussi le pré-processeur et le compilateur)
- `gcc -o hello hello.c` appelle tous les programmes nécessaires pour produire directement le binaire hello.

Application de gcc à un format de fichier

Le programme `gcc` se base simplement sur l'extension du fichier qu'on lui donne pour déterminer quelles opérations il doit effectuer. Par exemple, si le fichier se termine par :

- `.c`, il appelle dans l'ordre : le pré-processeur, le compilateur, l'assembleur et l'éditeur de lien.

Application de gcc à un format de fichier

Le programme `gcc` se base simplement sur l'extension du fichier qu'on lui donne pour déterminer quelles opérations il doit effectuer. Par exemple, si le fichier se termine par :

- `.c`, il appelle dans l'ordre : le pré-processeur, le compilateur, l'assembleur et l'éditeur de lien.
- `.i` il appelle dans l'ordre : le compilateur, l'assembleur et l'éditeur de lien.

Application de gcc à un format de fichier

Le programme `gcc` se base simplement sur l'extension du fichier qu'on lui donne pour déterminer quelles opérations il doit effectuer. Par exemple, si le fichier se termine par :

- `.c`, il appelle dans l'ordre : le pré-processeur, le compilateur, l'assembleur et l'éditeur de lien.
- `.i` il appelle dans l'ordre : le compilateur, l'assembleur et l'éditeur de lien.
- `.s` : il appelle dans l'ordre : l'assembleur et l'éditeur de lien.

Application de gcc à un format de fichier

Le programme `gcc` se base simplement sur l'extension du fichier qu'on lui donne pour déterminer quelles opérations il doit effectuer. Par exemple, si le fichier se termine par :

- `.c`, il appelle dans l'ordre : le pré-processeur, le compilateur, l'assembleur et l'éditeur de lien.
- `.i` il appelle dans l'ordre : le compilateur, l'assembleur et l'éditeur de lien.
- `.s` : il appelle dans l'ordre : l'assembleur et l'éditeur de lien.
- `.o` : il appelle l'éditeur de lien.

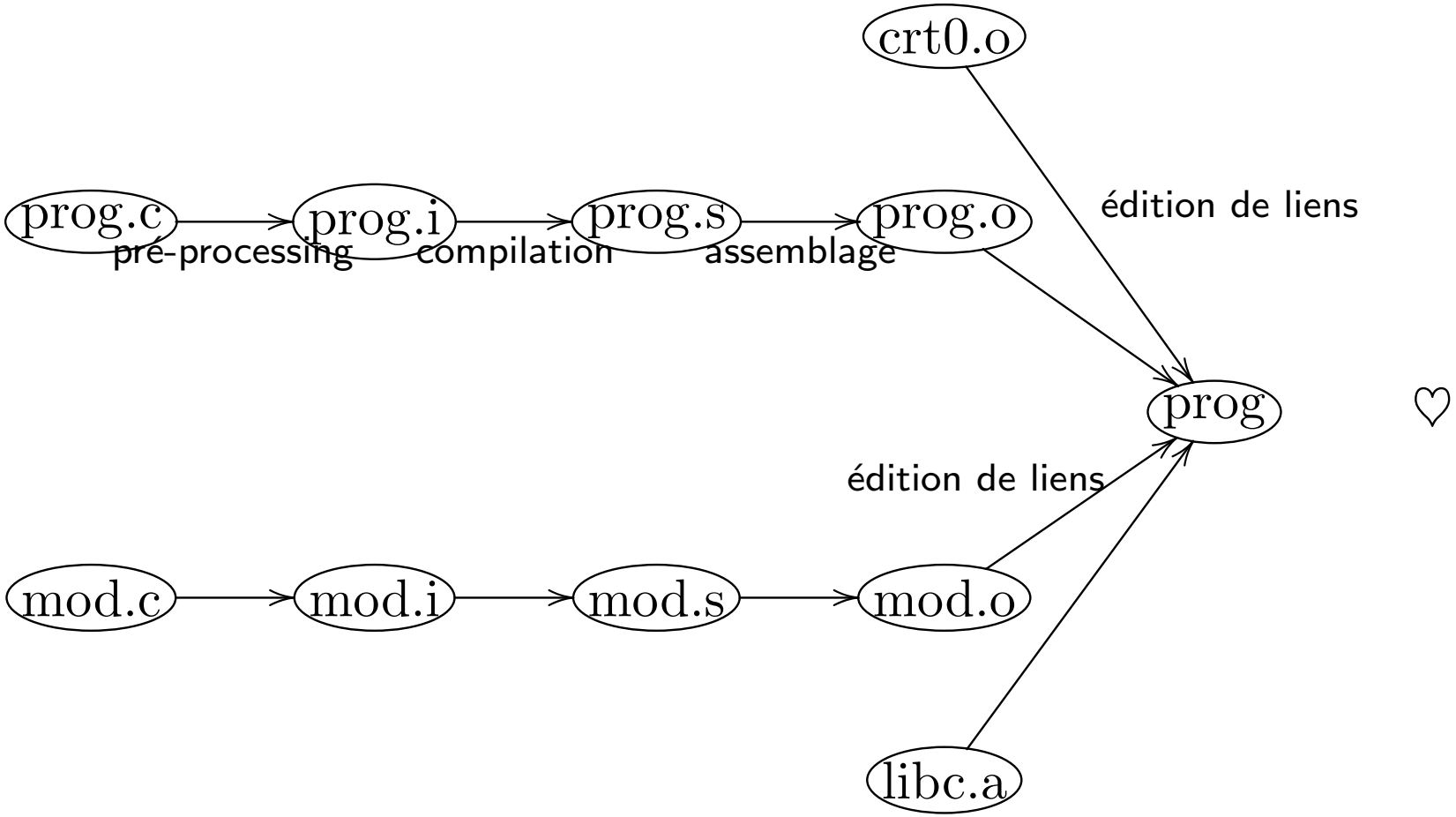
Application de gcc à un format de fichier

Le programme `gcc` se base simplement sur l'extension du fichier qu'on lui donne pour déterminer quelles opérations il doit effectuer. Par exemple, si le fichier se termine par :

- `.c`, il appelle dans l'ordre : le pré-processeur, le compilateur, l'assembleur et l'éditeur de lien.
- `.i` il appelle dans l'ordre : le compilateur, l'assembleur et l'éditeur de lien.
- `.s` : il appelle dans l'ordre : l'assembleur et l'éditeur de lien.
- `.o` : il appelle l'éditeur de lien.
- `gcc -o hello hello.c` appelle tous les programmes nécessaires pour produire directement le binaire **hello**.
  Détruit en outre les fichiers intermédiaires.

Schéma du processus de compilation

Chaîne de compilation pour un programme composé d'un programme principal **prog.c** qui utilise un module **mod.c** (Christophe Rippert).



♡

crt0.o : chargé par défaut par `ld` ; permet de rendre le fichier objet exécutable

Options

Quelques options du compilateur gcc :

`-c` : supprime l'édition de liens ; produit un fichier objet.

Options

Quelques options du compilateur gcc :

- `-c` : supprime l'édition de liens ; produit un fichier objet.
- `-E` : n'active que le préprocesseur (le résultat est envoyé sur la sortie standard).

Options

Quelques options du compilateur gcc :

`-c` : supprime l'édition de liens ; produit un fichier objet.

`-E` : n'active que le préprocesseur (le résultat est envoyé sur la sortie standard).

`-I`*`nom-de-répertoire`* : spécifie le répertoire dans lequel doivent être recherchés les fichiers en-têtes à inclure (en plus du répertoire courant).

Options

Quelques options du compilateur gcc :

`-c` : supprime l'édition de liens ; produit un fichier objet.

`-E` : n'active que le préprocesseur (le résultat est envoyé sur la sortie standard).

`-Inom-de-répertoire` : spécifie le répertoire dans lequel doivent être recherchés les fichiers en-têtes à inclure (en plus du répertoire courant).

`-Lnom-de-répertoire` : spécifie le répertoire dans lequel doivent être recherchées les librairies précompilées (en plus du répertoire usuel).

Options

Quelques options du compilateur gcc :

`-c` : supprime l'édition de liens ; produit un fichier objet.

`-E` : n'active que le préprocesseur (le résultat est envoyé sur la sortie standard).

`-I`*`nom-de-répertoire`* : spécifie le répertoire dans lequel doivent être recherchés les fichiers en-têtes à inclure (en plus du répertoire courant).

`-L`*`nom-de-répertoire`* : spécifie le répertoire dans lequel doivent être recherchées les librairies précompilées (en plus du répertoire usuel).

-o *nom-de-fichier* spécifie le nom du fichier produit. Par défaut, l'exécutable fichier s'appelle **a.out**.

Options

Quelques options du compilateur gcc :

`-c` : supprime l'édition de liens ; produit un fichier objet.

`-E` : n'active que le préprocesseur (le résultat est envoyé sur la sortie standard).

`-Inom-de-répertoire` : spécifie le répertoire dans lequel doivent être recherchés les fichiers en-têtes à inclure (en plus du répertoire courant).

`-Lnom-de-répertoire` : spécifie le répertoire dans lequel doivent être recherchées les librairies précompilées (en plus du répertoire usuel).

-o *nom-de-fichier* spécifie le nom du fichier produit. Par défaut, l'exécutable fichier s'appelle **a.out**.

-S : n'active que le préprocesseur et le compilateur ; produit un fichier assembleur.

Options

Quelques options du compilateur **gcc** :

- **-v** : pour *verbose*. Affiche la liste des commandes exécutées par les différentes étapes de la compilation.

Options

Quelques options du compilateur **gcc** :

- **-v** : pour *verbose*. Affiche la liste des commandes exécutées par les différentes étapes de la compilation.
- **-W** : imprime des messages d'avertissement (warning) supplémentaires.

Options

Quelques options du compilateur **gcc** :

-v : pour *verbose*. Affiche la liste des commandes exécutées par les différentes étapes de la compilation.

-W : imprime des messages d'avertissement (warning) supplémentaires.

-Wall imprime tous les messages d'avertissement. Attention, cela peut-être très verbeux !

Options

Quelques options du compilateur **gcc** :

-v : pour *verbose*. Affiche la liste des commandes exécutées par les différentes étapes de la compilation.

-W : imprime des messages d'avertissement (warning) supplémentaires.

-Wall imprime tous les messages d'avertissement. Attention, cela peut-être très verbeux !

...

Options

Quelques options du compilateur **gcc** :

-v : pour *verbose*. Affiche la liste des commandes exécutées par les différentes étapes de la compilation.

-W : imprime des messages d'avertissement (warning) supplémentaires.

-Wall imprime tous les messages d'avertissement. Attention, cela peut-être très verbeux !

...

Le manuel Pour plus d'informations sur **gcc**, entrer :

```
$ man gcc
```

Pour quitter la *manpage*, taper **q** puis « Entrée ».