# TP 1 (1,5h): Chaine de compilation C/UNIX

**Avant de commencer**, rappelez-vous les fondamentaux du *shell* UNIX et les principales commandes, et en particulier la commande **man**. A partir de ce point, lorsqu’une commande est mentionnée (en **gras** dans le texte), il est conseillé de consulter son manuel avec **man.**

**Rappel de la chaîne de compilation C vue en cours :**

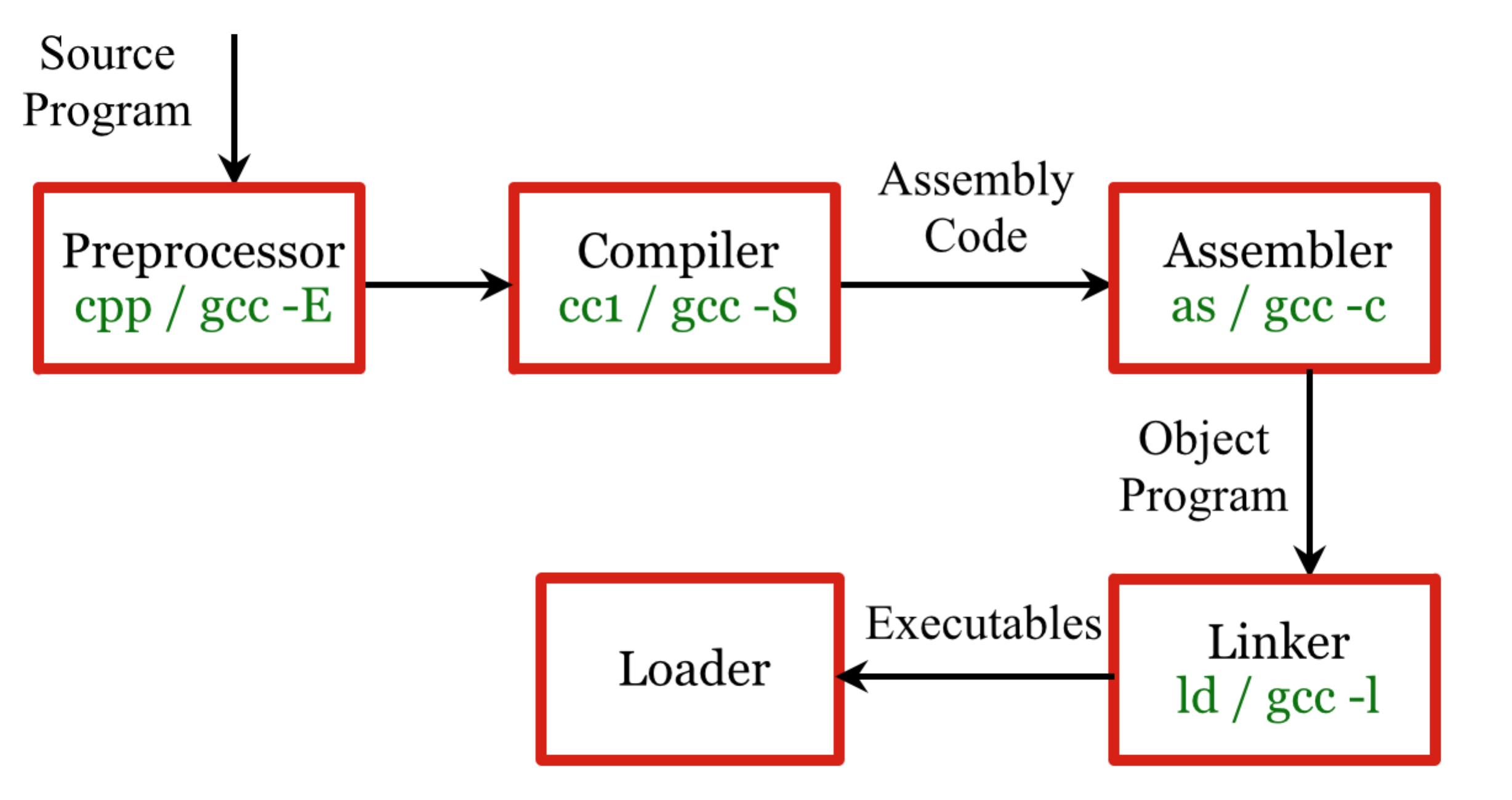


Figure 1. Etapes de la compilation d’un programme C

1. **Compilation d’un programme mono-fichier.**
2. Ecrivez un programme C (*p1.c*) qui affiche un caractère. Utilisez le shell Linux pour le compiler avec **gcc** et l’exécuter.
3. Ajoutez l’option **–v** à la compilation du fichier. Examinez les différents affichages et identifiez chacune des étapes de compilation de la Figure 1.
4. Utilisez les commandes en vert dans la Figure 1 pour générer les formats intermédiaires. Examinez les fichiers ainsi générés :

* Utilisez la commande **file** pour déterminer le type de chaque fichier
* Pour les fichiers texte, afficher le contenu (par ex., avec **cat**)
* Pour les fichiers binaires, afficher le contenu avec la commande **objdump –D**

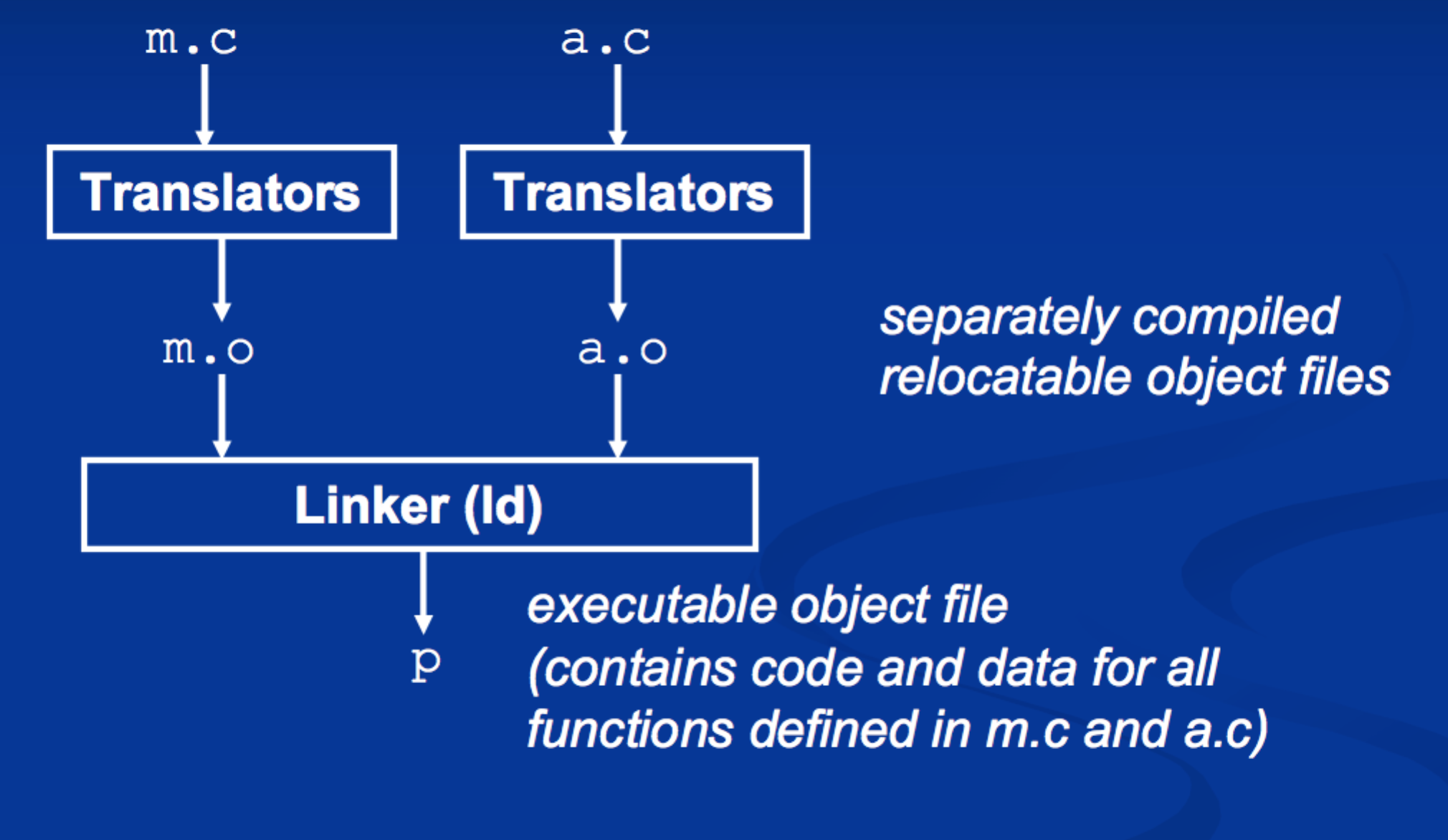


Figure 2. Schéma de compilation avec plusieurs fichiers

1. **Compilation d’un programme multi-fichier.**
2. Ecrivez un programme C (*p2.c*) qui appelle une fonction *f* située dans un autre fichier (*f.c)* et qui affiche un caractère (utilisez un fichier *f.h* pour la déclaration de *f()* ).
3. Avec l’ option -c de **gcc**, générez séparément les deux fichiers objet *p2.o* et *f.o* puis faire l’édition de liens pour obtenir l’exécutable, en vous servant du schéma de la Figure 2 (commande : **gcc –o p2 f.o p2.o**).
4. Utilisez la commande **objdump –D –r** pour visualiser le contenu des fichiers objets et examinez comment est codé le traitement de l’appel à la fonction *f()*. Visualisez ensuite le contenu de l’exécutable et essayez de tirer des conclusions sur ce qu’a fait l’éditeur de liens.
5. **Utilisation de la commande make.** Créer un fichier *Makefile* permettant de compiler séparément *p2.c* et *f.c* et de générer l’exécutable. La compilation d’une source ne doit se faire que si elle a été modifiée.  
     
   Ceci est un exemple de Makefile très basique :

# nommer le fichier « Makefile » afin que make le trouve automatiquement

# règle 1 : l'executable p.x est généré à partir du fichier objet p.o

**p.x : p.o**

**gcc -o p.x p.o**

# règle 2 : l'objet p.o est généré à partir de certains fichiers source

**p.o: p.c p.h**

**gcc -g -c p.c**

# exemple de règle sans dépendance (lancer avec la commande "make clean")

**clean:**

**rm p2.x p2.o f.o**

1. **Utilisation de gdb.**
2. Afin d’exercer les fonctions de débogage, ajoutez quelques lignes à la fonction *f* qui affectent et affichent plusieurs variables locales (minimum 3).
3. Le débogueur ne permet d’utiliser les symboles du programme (noms de variables, de fonctions, etc.) que si celles-ci sont renseignées dans le fichier exécutable, ce qui par défaut n’est pas le cas (pour des raisons d’optimisation).   
   Compiler *p2* et ses dépendances et rajoutant les informations de débogage avec l’option **–g** :  
   **gcc –g p2.c f.c**
4. Utiliser l’environnement commande **gdb** pour :
   1. Apprendre à interroger l’aide (commande **help**)
   2. Exécuter le programme (commande **run**)
   3. Mettre un point d’arrêt au début de la fonction *f* (commande **break**)
   4. Exécuter pas à pas *f* (commande **next** après un **run/break)**
   5. Afficher les valeurs de variables en cours d’exécution (commande **print**), l’état de la pile d’appels (commande **bt**)
   6. Prendre quelques minutes pour explorer l’aide sur les autres commandes disponibles
5. L’éditeur de texte **emacs** intègre les fonctionnalités de **gdb** à travers une IHM graphique. Utiliser cette interface pour réaliser les actions du point précédent. Vous pourrez ensuite utiliser **emacs** comme environnement de développement pour la suite des TD/TP.

## Pour aller plus loin (exercices optionnels)

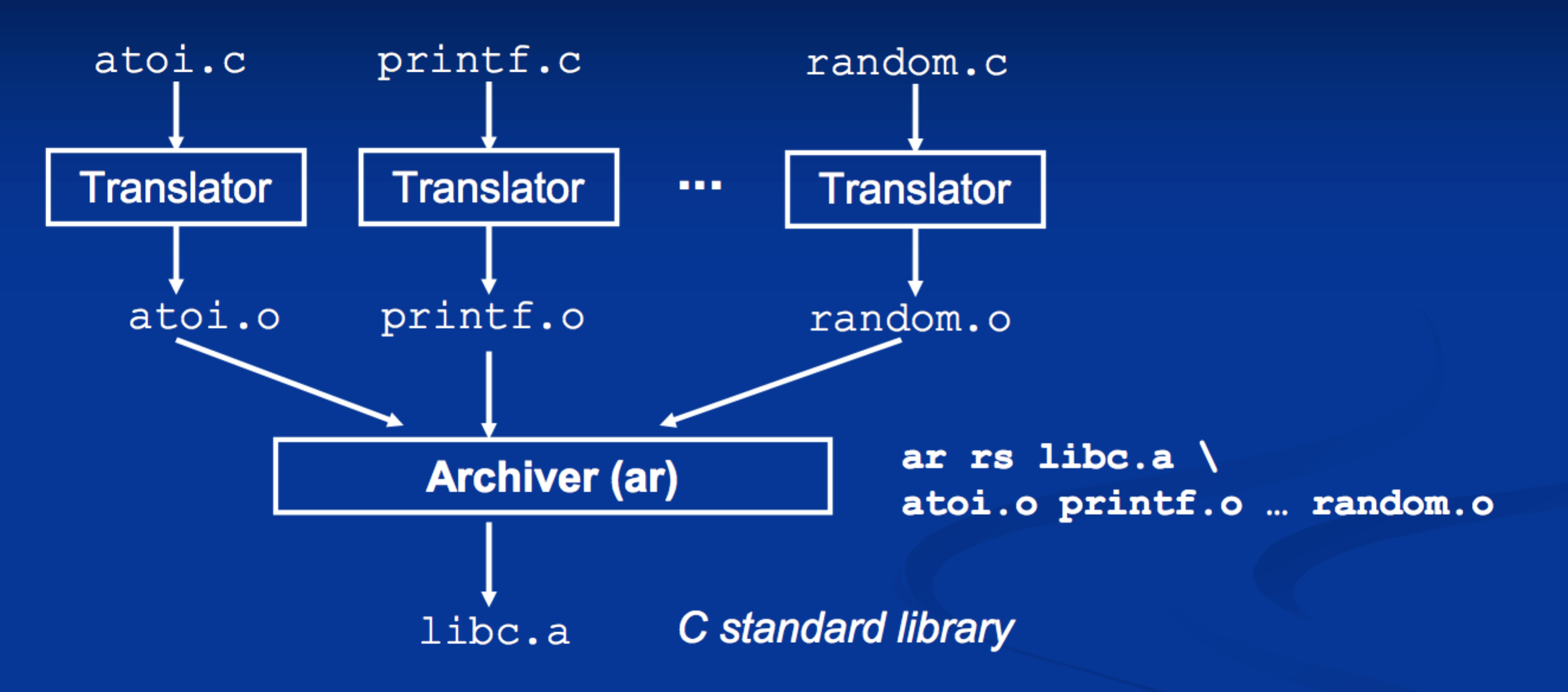


Figure 3. Génération d'une librairie statique

1. **Création et utilisation d’une librairie statique.**
2. Copiez le programme de l’Ex. 2 dans un fichier *p5.c* etajoutez l’appel à une autre fonction *g* située dans son propre fichier (*g.c*), qui affiche un entier.
3. En vous servant du schéma de la Figure 3, créez une librairie statique contenant *f* et *g*, puis l’utiliser pour la compilation de *p5.c*.
4. Ecrivez un fichier *Makefile* pour la création de la librairie à partir des sources *f, g*.
5. **Création et utilisation d’une librairie dynamique.**
6. Générer une librairie dynamique avec la commande suivante :  
   **gcc –shared –o fg.so f.c g.c**
7. Créer un exécutable avec la commande suivante :  
   **gcc –o p5dyn p5.c ./fg.so**
8. En utilisant la commande **ldd**, visualisez les dépendances de *p5dyn*. Déplacez *fg.so* dans un autre répertoire et lancez *p5dyn*. Que se passe-t-il ?
9. Lancez la commande   
   **gcc –static p5.c f.c g.c –o p5static**Comparez ensuite le type (commande **file**), la taille (**ls -l**) et les dépendances (**ldd**) de **p5static** et de **p5dyn**. Expliquez les résultats.