

Fiche annexe sur la création de bibliothèque.

Une **bibliothèque C** (ou librairie) est un ensemble formé de deux types de fichiers:

- Un ou plusieurs fichiers d'en tête (.h)
- Un fichier (potentiellement plusieurs) binaire contenant le code compilé implémentant les fonctions décrites dans les en-têtes.

Le fichier binaire peut être vu comme une archive regroupant un ou plusieurs fichiers objets (.o).

Il existe deux types de librairies:

1. **Bibliothèque statique** (extension .a), qui vont être incluse dans l'exécutable finale.
2. **Bibliothèque dynamique** (extension .so), qui ne sont pas incluse dans l'exécutable finale, mais liée dynamiquement à chaque lancement du programme. Ce sont les librairies les plus répandues.

Sous Windows, les librairies possèdent l'extension .dll

Les librairies permettent **d'échanger** aisément des **fonctionnalités**, sans avoir à fournir le code du corps des fonctions. Cela peut avoir deux buts:

- **Cacher** au client le code.
- **Simplifier** la démarche d'installation en évitant que le client ne doive compiler du code.

Un exemple typique d'utilisation concerne les mises-à-jour de programme. Lorsqu'une nouvelle fonctionnalité, ou une correction est mis en place. Son installation revient simplement à remplacer la librairie précédente par la nouvelle (changement de .so / ou de .dll sous Windows). L'exécutable du programme (souvent beaucoup plus volumineux) restant inchangé.

Organisation des bibliothèques sous Linux:

Sous Linux, les librairies sont généralement stockés dans des répertoires standards dans lesquels les programmes viennent les chercher par défauts.

➔ **Observez** les répertoires:

- **/usr/include/** : il contient les fichiers d'en tête des librairies installées sur votre système.
- **/usr/lib/** : il contient les librairies binaires installées sur votre système.

Toutes les librairies visibles possèdent des fonctionnalités (en C ou C++) que vous pouvez utiliser dans vos programmes.

Note: Ces répertoires de bibliothèques ne sont pas à confondre avec les programmes exécutables disponible dans **/usr/bin/**

Note: Sous *Windows*, il n'existe pas de répertoire standard pour stocker les librairies. Celles-ci sont généralement dupliquées localement dans chaque dossier d'un programme spécifique. Chaque programme viens alors directement charger sa version locale du fichier *dll*.

Creation d'une bibliothèque:

Considérez l'exemple_1 des fichiers fournis (répertoire 04_bibliotheque/).

Le dossier `code_librairie/` contient le code source de la librairie.

Le dossier `executables/` contient 2 programmes faisant appel à cette même librairie.

Nous allons générer les deux exécutables en les liant dynamiquement à la librairie.

Premièrement, nous générons la **bibliothèque dynamique**. Cette étape a lieu **une unique fois** à partir du code source de la librairie.

→ Placez vous dans le répertoire `code_librairie/`

→ **Compilez** le code source vers un fichier objet avec l'option “**fPIC**” (rend la création de librairie dynamique possible).

```
$ gcc -c vecteur.c -fPIC -Wall -Wextra
```

La librairie dynamique se réalise en appelant `gcc` avec l'option “**shared**” sur l'ensemble des fichiers objets que l'on souhaite inclure dans cette librairie.

→ Dans notre cas, on écrit alors:

```
$ gcc -shared vecteur.o -o libvecteur.so
```

Nous avons créé la librairie dynamique.

Celle-ci peut se lier à d'autres programmes sans avoir à utiliser le code source (.c).

Par contre, le fichier d'en tête .h reste **nécessaire** pour utiliser les fonctions définies dans cette librairie.

→ **Copiez** désormais le fichier d'en tête, et la librairie dans le répertoire `executables/`

→ **Compilez** les deux programmes exécutables:

```
$ gcc -c mon_programme_1.c -Wall -Wextra
```

```
$ gcc -c mon_programme_2.c -Wall -Wextra
```

Lors de l'**édition de liens**, la librairie doit être **associée** au programme.

Pour cela, on utilise l'option `-l<NOM_LIBRAIRIE>` pour une librairie contenue dans le fichier `libNOM_LIBRAIRIE.so`

→ Ecrivez:

```
$ gcc mon_programme_1.o -lvecteur -o mon_programme_1
```

Cette commande échoue. Elle vous indique que la librairie `vecteur` n'est pas trouvée. En effet, `gcc` viens chercher les librairies dans le chemin standard. Par exemple `/usr/lib/`

Le répertoire courant n'est pas contre pas dans les chemins de recherche par défaut. Pour ajouter un chemin spécifique non standard, on utilise l'option `-L<CHEMIN>`.

Dans notre cas, le chemin spécifique est le répertoire courant.

→ Ajouter le nom complet, ou simplement un point (.) après la lettre L.

```
$ gcc mon_programme_1.o -lvecteur -L. -o mon_programme_1
```

Cette fois l'exécutable est généré, la librairie à été trouvée.

→ Tentez désormais d'exécuter votre programme. Que ce passe t-il?

La ligne d'erreur indique que la librairie n'est pas trouvée. Encore une fois, lors de l'exécution d'un programme exécutable, les librairies dynamiques sont cherchées dans un chemin par défaut : par exemple `/usr/lib/`.

→ Lancez la commande:

```
$ ldd ./mon_programme_1
```

qui permet de *visualiser* quelles librairies dynamiques sont utilisées par un exécutable donnée.

Ici, notez que la librairie "vecteur" n'est pas trouvée.

Pour **ajouter** un chemin spécifique de recherche, on peut utiliser une **variable globale du Shell** (ligne de commande) qui contient la liste des répertoire à visiter.

Cette variable porte le nom de `LD_LIBRARY_PATH`. Pour ajouter un répertoire <CHEMIN>, utilisez la syntaxe suivante en ligne de commande:

```
$ export LD_LIBRARY_PATH=<CHEMIN>:$LD_LIBRARY_PATH
```

→ **Appliquez** cette ligne dans le cas où <CHEMIN>= . (répertoire courant).

```
$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

→ **Relancez** votre exécutable.

Cette fois la librairie est trouvée. Elle est cherchée et chargée dynamiquement à chaque lancement.

→ **Observez** à nouveau le résultat de:

```
$ ldd ./mon_programme_1
```

→ Ouvrez un nouveau terminal dans ce même répertoire. Tentez de lancer votre exécutable à partir de ce nouveau terminal. Que ce passe-il?

Cette fois, le chemin d'accès ne contient plus le répertoire courant, et la démarche d'affectation de la variable du Shell est à nouveau nécessaire.

Note: Sous *Windows*, le chemin par défaut est au contraire le répertoire courant. C'est pour cette raison qu'il est courant de devoir déplacer les `.dll` à coté d'un exécutable.

Pour aller plus loin:

Afin d'éviter cette manipulation à chaque lancement, il est possible de définir la ligne export `LD_LIBRARY_PATH` avec les chemins spécifique dans un script qui viens ensuite lui même lancer l'exécutable.

- **Réalisez** les autres exemples dont l'énoncé est contenu dans les fichiers `consignes.txt`.
- **Creez** une bibliothèque dynamique du comportement de déplacement de vos pièces (`piece_deplacement_standard`) sous le nom de `libdeplacement_standard.so`.
- **Creez un second** Makefile à votre projet tel que l'appel à `$ make -f Makefile_librairie` génère un exécutable du projet liée à `libdeplacement_standard.so`