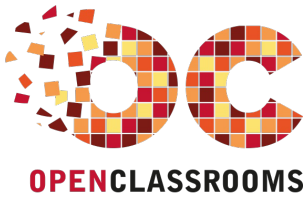
Créer des bibliothèques avec Code::Blocks

Par seb13



www.openclassrooms.com

Dernière mise à jour le 18/06/2011

Sommaire

Créer des bibliothèques avec Code::Blocks

Par seb13
Mise à jour : 18/06/2011

Salut à tous.
Dans ce mini tuto je vais vous parler de la création de bibliothèques en C. Comme j'utilise l'IDE Code::Blocks, les manipulations données le seront pour ce programme. Mais cela ne vous empêche pas de créer des Bibliothèques avec un autre IDE comme Dev-C++ ou Visual C++. Je précise aussi que je travaille sous Windows XP (et alors ? ...).
Après la lecture de ce tuto vous serez normalement capable de:

- comprendre les bibliothèques
- créer des bibliothèques statiques,
- créer des bibliothèques dynamiques (deux types).

Comme base pour lire et comprendre ce tuto, vous devez savoir utiliser les fonctions et les headers. Cela correspond à la fin de la partie 1 et au premier chapitre de la partie 2 du cours C/C++ de M@teo21.

Autant le dire tout de suite, on emploie couramment le mot librairie de manière abusive. Cela vient du fait que le mot anglais library est un faux ami et signifie bibliothèque.
Sommaire du tutoriel :

Que sont les bibliothèques

Les Bibliothèques contiennent généralement des fonctions qui peuvent êtres utilisées par plusieurs programmes. Elles peuvent aussi contenir des icônes comme le fichier shell32.dll de Windows XP par exemple. Il existe deux types de bibliothèques : les bibliothèques statiques et les bibliothèques dynamiques.

Les bibliothèques statiques

Dans le cas des bibliothèques statiques, les fonctions contenues dans la bibliothèque sont chargées lors de l'édition des liens. C'est pour cela que le fichier .h est nécessaire, il permet d'indiquer quelles fonctions sont utiles. Les fonctions sont inclues dans le fichier exécutable de votre programme, il n'y à rien d'autre à fournir donc le programme peut être exécuté sur un ordinateur ne possédant pas la bibliothèque.
Les bibliothèques statiques sont des fichiers .lib ou des .a

Les bibliothèques dynamiques

Pour les bibliothèques dynamiques, les fonctions ne sont pas inclues dans l'exécutable. Les fonctions sont appelées pendant l'exécution du programme (d'où le "dynamique"). L'avantage est donc que l'exécutable est plus léger mais il faut fournir la bibliothèque avec le programme. Donc au final l'ensemble exécutable + dll est plus lourd que l'exécutable contenant la bibliothèque (comme dans le cas précédent).
Il faut savoir aussi que si plusieurs programmes ont besoin de la même dll, ils peuvent tous y accéder alors que la bibliothèque n'est chargée qu'une fois. Dans ce cas on parle de bibliothèque partagée.
Sous Windows les bibliothèques dynamiques sont des fichiers .dll et sous GNU/Linux se sont des fichiers .so.

Il existe deux types de bibliothèque dynamique: les bibliothèques à chargement implicite et les bibliothèques à chargement explicite (ne fonctionne que sous Windows).
La différence est que pour une bibliothèque à chargement implicite il faut linker avec un fichier .a. Si vous voulez utiliser seulement la dll il faut faire une bibliothèque à chargement explicite. En fait la construction de la bibliothèque est la même, c'est la

conception de l'exécutable qui diffère.
La SDL par exemple s'utilise par chargement implicite. En effet cela est portable et plus simple à coder.

Les bibliothèques statiques

Je vais d'abord vous parler des bibliothèques statiques. En effet c'est le plus simple à réaliser.

Le projet de la bibliothèque

Dans Code::Blocks vous devez créer un nouveau projet en choisissant "Static Library". Cochez la case "Do not create any files" et appuyez sur le bouton create.
Ajoutez à votre projet un fichier .c que vous pouvez par exemple nommer lib.c.

Comme je vous avais dit que ces bibliothèques étaient simples à réaliser, on passe tout de suite à la pratique. Il vous faut une fonction donc on va prendre comme exemple une fonction qui retourne le carré d'un nombre passé en paramètre:

Code : C

```c
#include "dll.h"

int carre(int entier) {
    int carre  = 0;
    carre = entier*entier;
    return carre;
}
```

Maintenant il faut faire un fichier dll.h qui va contenir les prototypes de vos fonctions:

Code : C

```c
int carre(int entier);
```

Il faut inclure ce fichier dans votre lib.c cela peut être utiles lorsque votre bibliothèque contient plusieurs fonctions qui se servent les unes des autres.

Et bien c'est déjà fini, vous n'avez plus qu'à compiler et vous obtenez un fichier libstaticlib.a. Si le nom ne vous plait pas allez dans "Project", "Properties" puis dans l'onglet "Target". Là vous modifiez "Output Filename".

Le projet de votre programme

Pour ce projet vous pouvez faire ce que vous voulez (console, SDL, OpenGL, ...) mais ici je vais prendre l'exemple d'un projet console pour faire simple. J'ose espérer que vous savez créer un projet console avec Code::Blocks. Mettez dans votre projet un fichier main.c qui contiendra votre fonction main().
Comme le but est de se servir de la fonction contenue dans la bibliothèque, il faut évidemment faire un code qui se sert de cette fonction:

Code : C

```c
#include <stdio.h>
#include <stdlib.h>
#include "dll.h"

int main(int argc, char *argv[]) {
   int resultat = 0;
   resultat = carre(3);
   printf("Le resultat est %d\n", resultat);
   printf("Appuyez sur une touche pour continuer...");
   getchar();
   return 0;
```

```
}
```

Comme vous le voyez j'ai inclu de fichier dll.h.
Ensuite n'oubliez pas de linker avec votre belle bibliothèque. Pour cela vous allez dans "Project" puis "Build options" et enfin vous allez dans l'onglet "Linker". Là vous ajoutez votre libstaticlib.a ou autre si vous avez changé le nom.

Et maintenant le grand moment tant attendu, la compilation et l'exécution. Si vous n'avez pas de problème et je l'espère (sinon c'est que ce tuto est nul) la console affiche fièrement:

Code : Console

```
Le resultat est 9

Appuyez sur une touche pour continuer...
```

Et voila vous savez créer et utiliser une bibliothèque statique. Maintenant nous allons voir ensemble le cas des bibliothèques dynamiques, les biens connues dlls.

Les bibliothèques dynamiques

Je vais maintenant vous parler des bibliothèques dynamiques. Alors là cela va être un peu plus compliqué.

Le projet de la dll

Tout d'abord dans Code::Blocks vous devez créer un nouveau projet en choisissant "Dynamic Link Library". Cochez la case "Do not create any files" et appuyez sur le bouton create.
Ajouter un fichier .c à votre projet. Moi je le nomme dll.c (pas très original c'est vrai).

Alors que mettre dans ce fichier ?
En fait ce n'est pas très compliqué, il suffit d'avoir une fonction. Je vais prendre l'exemple d'une fonction qui renvoie la somme de deux entiers passés en paramètre. Bon ça devrait aller je pense :

Code : C

```
int addition(int entier1, int entier2) {
   return entier1 + entier2;
}
```

Donc, on met cette fonction dans le fichier dll.c mais cela ne suffit pas. En effet, il faut indiquer que cette fonction devra être exporté pour être utilisé par votre programme.
Pour cela, on utilise __declspec (dllexport) placé entre le type de la fonction et son nom. On obtient donc la fonction suivante:

Code : C

```
int __declspec (dllexport) addition(int entier1, int entier2) {
   return entier1 + entier2;
}
```

Si vous avez plusieurs fonctions il est intéressant de faire la macro suivante pour faciliter le travail:

Code : C

```
#define DllExport __declspec(dllexport)
```

```c
int DllExport addition(int entier1, int entier2) {
   return entier1 + entier2;
}
```

Et voilà c'est presque finit. Il faut juste créer un fichier header dll.h.
Comme vous le savez sûrement déjà (sinon relisez les cours de M@teo) ce fichier contiendra les prototypes des fonctions de la dll:

Code : C

```c
int addition(int entier1, int entier2);
```

Mais ici la fonction sera importée de la dll donc il faut rajouter __declspec (dllimport). La encore on place ce code entre le type de la fonction et son nom. On obtient:

Code : C

```c
#define DllImport __declspec(dllimport)

int DllImport addition(int entier1, int entier2);
```

Voilà c'est fini. Vous n'avez plus qu'à compiler votre projet. Après la compilation vous obtenez deux fichier importants: sample.dll et libsample.a (ne jetez pas non plus le fichier dll.h).
Ces deux fichiers vont vous êtres utiles par la suite.

Votre programme pour une bibliothèque à chargement implicite

Maintenant il faut utiliser cette belle bibliothèque. Pour ce projet nous allons encore faire un projet console.
Une fois votre projet créé, vous devez lui rajouter un fichier main.c contenant votre fonction main(). Puis vous mettez le code pour pouvoir tester l'importation de fonctions de la dll.
Par exemple:

Code : C

```c
#include <stdio.h>
#include <stdlib.h>
#include "dll.h"

int main(int argc, char *argv[]) {
   int resultat = 0;
   resultat = addition(3, 7);
   printf("Le resultat est %d\n", resultat);
   printf("Appuyez sur une touche pour continuer...");
   getchar();
   return 0;
}
```

Pensez bien à inclure le fichier dll.h pour avoir le prototype des fonctions de la bibliothèque. Maintenant vous compilez votre projet et ...
Et cela ne marche pas. Vous obtenez un horrible "undefined reference to `_imp__addition`".
C'est à ce moment là qu'il faut se rappeler du fichier libsample.a qui a été généré par Code::Blocks à la compilation du projet de dll. Vous devez donc linker votre projet avec ce fichier.
Vous recompilez et quelle joie cela fonctionne correctement.

Maintenant vous placez votre fichier dll.dll dans le même répertoire que votre exécutable. Vous exécutez et heureusement pour moi la console affiche:

Code : Console

```
Le resultat est 10

Appuyez sur une touche pour continuer...
```

Votre programme pour une bibliothèque à chargement explicite

Dans ce cas, c'est un peu plus compliqué. Il n'est plus nécessaire d'inclure le fichier dll.h dans votre code ni de linker avec libsample.a (au moins vous ne l'oublierez pas). Par contre, il faut inclure windows.h (donc ce code ne sera pas portable). Comme exemple je vais prendre le même que précédemment.

Code : C

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

   HINSTANCE DLLHandle;
   typedef int(*Type_Pointeur_De_Fonction)(int, int);
   Type_Pointeur_De_Fonction Pointeur_Fonction;
   DLLHandle = LoadLibrary("dll.dll");
   Pointeur_Fonction =
(Type_Pointeur_De_Fonction)GetProcAddress(DLLHandle,"addition");

   int resultat = 0;
   resultat = Pointeur_Fonction(3,7);
   printf("Le resultat est: %d\n", resultat);
   printf("Appuyez sur une touche pour continuer...");
   getchar();

   FreeLibrary(DLLHandle);
   return 0;
}
```

Alors là je crois que des explications s'imposent non ? On va faire ça ligne par ligne (enfin pas toutes quand même).

Code : C

```
HINSTANCE DLLHandle;
```

Déclare une variable de type HINSTANCE. Cela permet de garder le handle pour notre programme. Comme Windows est multitâche cela lui permet de savoir ce que fait notre programme.

Code : C

```
typedef int(*Type_Pointeur_De_Fonction)(int, int);
```

Là on définit un type de pointeur de fonction. Je vais juste vous expliquer un petit peu ce qu'est un pointeur de fonctions.
Ce n'est pas très compliqué si vous savez ce qu'est un pointeur. Un pointeur est une variable contenant l'adresse d'une autre variable. Pour une fonction, le pointeur contient donc l'adresse de la fonction. Où plus précisément l'adresse du point d'entrée dans la fonction.
Pour plus d'informations sur les pointeurs de fonctions, vous pouvez aller voir le tuto Les pointeurs de fonction de mleg.

Code : C

```
Type_Pointeur_De_Fonction Pointeur_Fonction;
```

Ici on déclare un pointeur de fonction du type ci-dessus. Ce pointeur pointera (c'est son métier) vers la fonction que l'on va importer de la dll.

Code : C

```
DLLHandle = LoadLibrary("dll.dll");
```

La fonction LoadLibrary permet de charger en mémoire notre bibliothèque. Celle ci est pointée par DLLHandle.

Code : C

```
Pointeur_Fonction =
(Type_Pointeur_De_Fonction)GetProcAddress(DLLHandle,"addition");
```

GetProcAddress renvoie l'adresse de la fonction. Le premier paramètre sert à identifier la dll que l'on utilise et le second est le nom de la fonction que l'on veut importer.

Code : C

```
resultat = Pointeur_Fonction(3,7);
```

Maintenant on utilise le pointeur de la même manière que si on utilisait la fonction.

Code : C

```
FreeLibrary(DLLHandle);
```

Et enfin il ne faut pas oublier de décharger la bibliothèque de la mémoire.

Et là votre programme marche correctement (surtout si vous pensez à mettre la dll dans le même répertoire que l'exécutable).
Voilà c'est déjà fini.
J'espère que vous avez appris quelque chose et que vous avez apprécié la lecture de ce tuto.
Mais il n'est pas nécessaire de créer une bibliothèque pour des fonctions aussi simples que mes exemples. Vous devez bien sûr utiliser des fonctions plus complexes et qui peuvent être utiles dans plusieurs de vos programmes. Sinon créer une bibliothèque n'est pas nécessaire.

Si vous avez des questions ou des suggestions (voire des corrections) vous pouvez me contacter par MP ou laisser un commentaire pour ceux qui ne sont pas membres du sdz.

Seb13