LAB3040 – TP C

Première partie – Création d'une DLL

Aurélien TEXIER
texier@esiea.fr

21/09/2020

Création d'une DLL

But du projet

Le but de ce projet est de vous faire pratiquer le langage C en vous expliquant une nouvelle notion : ***les librairies***. Cette notion est importante pour comprendre comment intégrer du code issu d'une librairie et comment l'exploiter au mieux.

Le projet se décomposera donc en deux parties :

Dans un premier temps, vous devrez créer une librairie répondant à un cahier des charges décrit dans ce document. Une fois celle-ci réalisé, vous aurez à livrer votre production à un autre groupe qui devra s'en servir dans la seconde partie du projet.

Vous devrez donc fournir une librairie complète ainsi qu'une documentation technique (les instructions pour générer cette documentation vous seront données dans ce document). De plus, le "SAV" fourni au groupe auquel vous aurez livré votre librairie fera partie de la notation.

Dans un second temps, vous aurez à réaliser un projet utilisant la dll produite et fourni par un autre groupe. Le sujet de cette seconde partie vous sera donné après la livraison de la librairie.

NB : Le projet sera réalisé sous ***Visual Studio 2019*** (disponible sur les machines de l'école). Si vous utilisez un autre environnement de développement, ***il vous faudra pour le rendu générer une solution Visual Studio 2019 valide***.

Table des matières

Graphes et plus court chemins

L'objectif de votre librairie sera de permettre la gestion de graphes et la possibilité d'y appliquer l'algorithme de Dijkstra avec quelques variantes.

Graph & Dijkstra

Vous allez développer une bibliothèque nommée graphManager contenant les fichiers graph.c/graph.h. L'objectif de cette librairie est de modéliser le problème suivant :

Vous devrez représenter sous forme de graphe un terrain tel que chaque case a pour voisin les cases situées juste jute à côté. Chaque case représentera un type de terrain parmi 6 (grass, forest, mountain, city, road ou water). La librairie permettra de calculer Dijkstra sur un tel graphe sachant que :

- Il sera possible d'interdire le déplacement sur certains types de terrain ;
- La distance entre deux cases dépendra du terrain de l'arrivée.

Structures

Plusieurs concepts seront modélisés via des structures. Voici les structures qui devront se trouver dans votre librairie (***les noms sont importants***) :

- Une structure ***node*** contenant :
  - L'indice du nœud — ***m_id***
  - La position en abscisse du nœud — ***m_posX***
  - La position en ordonnée du nœud — ***m_posY***
  - Un masque représentant le type de terrain — ***m_layer***
  - L'ID du type de terrain — ***m_layerID***
  - Le tableau de ses voisins (tableau statique de 4 pointeurs sur nœud) — ***m_neighbors***
  - Un pointeur générique pour y stocker plus d'informations par la suite — ***m_data***

- Une structure ***graph*** contenant :
  - La taille en abscisse du terrain — ***m_sizeX***
  - La taille en ordonnée du terrain — ***m_sizeY***
  - Le tableau de ***node*** (tableau de pointeur à simple entrée) — ***m_data***

- Une structure ***dijkstraNode*** contenant :
  - Un pointeur vers le nœud — ***m_node***
  - La distance au nœud de départ — ***m_distance***
  - Un flag pour vérifier si le nœud est traité — ***m_flag***
  - Un pointeur vers le nœud Dijkstra précédent — ***m_prev***

Fonctions

Plusieurs fonctions seront à développer pour rendre votre librairie utilisable. Le nom des fonctions ainsi que les valeurs en entrée et en sortie sont définies dans le tableau ci-dessous :

Fonction	Que fait la fonction ?	Donnée retournée	Param 1	Param 2	Param 3
LoadGraphFromFile	Alloue et charge un graphe à partir d'un fichier	graph*	char*	-	-
FreeGraph	Libère la mémoire allouée pour un graphe	void	graph*	-	-
GetLayerIDFromChar	Permet de récupérer le masque associé à un terrain en fonction de son caractère	int	char	-	-
Dijkstra	Fonction calculant le chemin le plus court d'un nœud vers tous les autres en fonction d'un masque des types de terrains accessibles	dijkstraNode**	graph*	node*	uchar
GetNodeFromPosition	Récupère un nœud en fonction de ses positions en X et Y	node*	graph*	uchar	uchar
GetManhattanDistance	Calcul la distance de Manhattan entre deux nœuds	int	node*	node*	-
IsNeighbour	Renvoie 1 si les deux nœuds en paramètre sont voisins, 0 sinon	char	node*	node*	-
SetNodeData	Permet d'affecter un pointeur sur un espace de donnée au nœud	void	node*	void*	-
GetNodeData	Récupère le pointeur sur l'espace de donnée stocké dans le nœud	void*	node*	-	-

Programme de test

Il est conseillé de faire des programmes de test notamment une fonction pour afficher le graphe et une autre pour afficher le résultat de Dijkstra. Ces fonctions ne seront pas à exportées dans la librairie.

Librairie

Une fois votre développement réalisé, vous allez le transformer en ***librairie***. Elle devra être exploitable par vos camarades.

Chaque binôme devra produire une librairie qu'il fournira au binôme suivant, sauf le dernier binôme qui fournira sa librairie au premier binôme. Il faudra donc que la librairie soit fiable, testée et facile à comprendre (bien documentée donc).

Chaque binôme est responsable de sa DLL et doit apporter le soutien nécessaire au bon fonctionnement de celle-ci.

Dans chaque groupe, un responsable sera désigné et sera le contact privilégié par mail via son compte ESIEA. Les librairies seront ainsi fournies au « client » par mail (en mode Release) et les questions qui pourraient survenir suite à la livraison/utilisation de la DLL seront posées par le « client » par mail (vous devrez aussi mettre votre enseignant en copie de ces échanges).

Qu'est-ce qu'une librairie ?

Nous aborderons les librairies sous Windows, le concept est le même sous Linux avec des extensions différentes :

- .lib ⇔.a
- .dll ⇔.so

Une librairie est un ensemble de fonctions et de définitions de type accessibles par le biais du chargement d'un binaire (.LIB sous Windows) et d'un fichier d'entête (header : .h).

Il y a 2 types de librairies : les librairies statiques et les librairies dynamiques.

Librairie statique

Une fois le fichier LIB généré (après compilation du code source contenu dans un ou plusieurs fichiers c) il est possible d'utiliser les fonctions ou les types de cette librairie. Le code de la librairie sera copié dans le programme généré, il n'y aura donc pas besoin de fichier supplémentaire (DLL). La maintenance est donc facilitée (pas de DLL à mettre à jour, juste le .EXE).

Une librairie statique nécessite 2 fichiers :

- .h : en-têtes
- .lib : utilisé pour l'édition des liens (le code compilé est intégré au code de l'exécutable)

Librairie dynamique

Une librairie dynamique ressemble beaucoup à une librairie statique. Le fichier DLL contenant le code sera chargé dynamiquement au démarrage du programme utilisant la librairie et les fonctions utilisées seront ainsi « récupérées » au moment de l'exécution du programme. Le principal intérêt d'une telle librairie est qu'elle permet de mutualiser la librairie (DLL) pour partager son code entre plusieurs programmes. Cela implique par contre d'être vigilant lors de la mise à jour de la DLL : il faut recompiler tous les programmes exploitant l'ancienne version de la DLL pour les mettre à jour.

Une librairie dynamique nécessite 3 fichiers :

- .h : en-têtes
- .lib : utilisé pour l'édition des liens
- .dll : pour avoir accès au code au moment de l'exécution du programme

Il est possible d'utiliser des librairies dynamiques en faisant l'édition des liens au lancement du programme, ce qui permet de faire évoluer la librairie sans recompiler le programme. Il faut utiliser pour cela le principe suivant :

```
// définition du type "nomDeLaFonction" permettant de récupérer le pointeur de fonction
dans la DLL
typedef typeDeRetour (*nomDeLaFonction)(Paramètres);
nomDeLaFonction maFonctionDLL = NULL;

// Chargement de la DLL
HINSTANCE dll_handle = LoadLibrary("MaLibrairie.dll");

if(dll_handle != NULL)
{
	// récupération du pointeur de fonction dans la DLL
	maFonctionDLL = (nomDeLaFonction)GetProcAddress(dll_handle,"nomDansLaDLL");
	// il est ensuite possible d'utiliser la fonction
	// comme si elle faisait partie intégrante du programme en cours
	// en utilisant le formalism maFonctionDLL(Paramètres)
}
```

NB : typeDeRetour = le type de donnée retourné par la fonction que l'on souhaite utiliser dans la dll.

Paramètres = les paramètres (type & nom) de la fonction que l'on souhaite utiliser dans la dll.

Je vous laisse libre d'utiliser cette méthode ou de réaliser l'édition des liens directement dans le projet avec le fichier .lib (ce que je vous conseille tout de même).

La librairie fournie devra être générée en mode Release, en effet le mode Debug utilise pour compiler/éditer les liens des librairies "Debug" qui ne sont pas des librairies redistribuées au contraire des librairies Release qui elles peuvent être récupérées et installées sans problème sur l'ordinateur cible (VCRedist par exemple).

Mise en œuvre

Comment créer une librairie ?

Une librairie est une bibliothèque compilée. Une fois compilée il faut savoir quelles sont les données accessibles depuis le programme qui l'utilisera, aussi il faut indiquer aux fonctions qu'elles seront « **exportées** » avant de compiler.

Exemple de création d'une librairie

Prenons un exemple simple. Nous avons la bibliothèque lib.c/lib.h suivante :

Fichier lib.h :

```
#ifndef _LIB_H_
#define _LIB_H_

int helloworld(int p_a, int p_b);

#endif
```

Fichier lib.c :

```
#include "lib.h"

int helloworld(int p_a, int p_b)
{
     printf("Hello %d, %d", p_a, p_b);
     return p_a + p_b;
}
```

Nous souhaitons la compiler en tant que librairie. Pour cela nous allons ajouter devant le nom de la fonction à exporter « __declspec(dllexport) ».

On modifiera donc le .h et le .c de la manière suivante :

Fichier lib.h :

```
#ifndef _LIB_H_
#define _LIB_H_

__declspec(dllexport) int helloworld(int
p_a, int p_b);

#endif
```

Fichier lib.c :

```
#include "lib.h"

__declspec(dllexport) int helloworld(int
p_a, int p_b)
{
     printf("Hello %d, %d", p_a, p_b);
     return p_a + p_b;
}
```

Une fois cette modification effectuée on peut compiler la librairie (le résultat de la compilation sera composé de 2 fichiers : lib.LIB et lib.DLL).

Pour pouvoir ensuite exploiter ces fichiers dans un programme il faudra modifier le fichier .H pour indiquer que les fonctions doivent être importées de la DLL, en modifiant la déclaration « declspec(dllexport) » en « declspec(dllimport) ». Soit :

Fichier lib.h :

```
#ifndef _LIB_H_
#define _LIB_H_

__declspec(dllimport) int helloworld (int p_a, int p_b);

#endif
```

On comprend bien qu'il n'est pas très efficace de faire la modification quand la librairie contient plusieurs fonctions, donc il est souvent d'usage de créer une macro prenant en compte le fait que le .c est compilé (compilation de la librairie) ou non (utilisation de la librairie). Pour ce faire il faudra modifier le .c pour rajouter un `#define` avant l'inclusion du .h. Cela garantira que le fichier d'en-têtes saura identifier si le fichier .c est présent (compilation de la librairie) ou non (utilisation de la librairie), c'est cette opération qui sera réalisée par la macro. Cette macro sera définie en début du fichier .h et permettra de choisir automatiquement entre dllexport et dllimport :

Fichier lib.h :

```
#ifndef _LIB_H_
#define _LIB_H_

#ifdef _LIB_C_
    #define LIB_API __declspec(dllexport)
#else
    #define LIB_API __declspec(dllimport)
#endif

LIB_API int helloworld (int p_a, int p_b);

#endif
```

Fichier lib.c :

```
#define _LIB_C_

#include "lib.h"

LIB_API int helloworld (int p_a, int p_b)
{
     printf("Hello %d, %d", p_a, p_b);
     return p_a + p_b;
}
```

Statique ou dynamique ?

Pour linker en statique ou en dynamique une librairie, rien de plus simple, il suffit d'indiquer au linker la librairie à utiliser (.lib) et à inclure le header correspondant pour la compilation (.h). A la compilation, les fonctions seront reconnues puisque présentes dans le fichier header (.h) et l'édition des liens sera faite avec le fichier .lib correspondant. Dans le cas d'une librairie dynamique il faudra mettre la DLL à côté du fichier EXE généré et le programme pourra exploiter les fonctions de la librairie.

La librairie utilisée dans notre cas pourrait être statique (un seul programme exploitera la librairie) mais pour que vous soyez dans une configuration la plus proche de ce que vous pourrez rencontrer par la suite en entreprise nous vous demandons de générer une librairie dynamique. En effet, il n'est pas rare qu'un constructeur de périphériques fournisse un driver et/ou un SDK contenant les 3 fichiers .h/.lib/.dll.

Doxygen

Comme indiqué précédemment, votre librairie devra être fourni documentée. Pour cela vous utiliserez l'outil Doxygen pour générer une documentation technique à partir des commentaires de votre code.

Pour beaucoup d'entre vous vous allez découvrir l'outil Doxygen très utilisé en entreprise pour générer automatiquement la documentation des codes produits dans un projet. L'intérêt de cet outil est qu'il permet de renseigner la documentation directement dans le code via un formatage spécifique des commentaires. Plusieurs formats sont disponibles (pour le java, le C++, …). Nous allons en imposer un (format C++) pour que tous les codes produits soient homogènes. La documentation produite ressemble très fortement à ce que vous pouvez trouver sur MSDN ou JavaDoc.

Les formats à utiliser sont décrits à cette adresse :
http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html
(*section « Comment blocks for C-like languages (C/C++/C#/Objective-C/PHP/Java) »*).

Pour résumer rapidement quelques possibilités de Doxygen voici un bref récapitulatif.

Pour documenter dans le .h et que les commentaires soient interprétés par Doxygen il faut encadrer les commentaires par les symboles :

```
/*!
      ...
*/
```

Dans ce bloc il est possible de créer des « balises » pour renseigner divers paramètres. Il est possible de donner des infos sur :

- Le fichier courant (.h) :

```
/*!   \file       monFichier.h
      \brief      Brève description.

      Détails.
*/
```

- Une variable :

```
/*!   Commentaire */
int variable;
```

Ou si l'on souhaite commenter la variable après l'avoir déclaré (il suffit de rajouter le sigle "<" pour indiquer que le commentaire se réfère à la déclaration précédente) :

```
int variable;          /*!<  Commentaire */
```

- Une fonction :

```
/*!   \brief           Brève description.

      Détails.

      \param[in]       param1     Description du paramètre 1
      \param[in/out]   param2     Description du paramètre 2 (il sera
modifié dans la fonction)

      \retrun          description de la valeur retournée
*/
int fonction(float p_param1, char * p_param2);
```

Ici on décrira la fonction (son utilité, ses paramètres et sa valeur de retour). On peut aussi utiliser la balise « \see » pour faire référence à une fonction ou à une variable afin de mieux renseigner le développeur.

Nous utiliserons uniquement ces balises qui permettront via l'outil Doxywizard installé sur vos machines de générer une documentation HTML, documentation qui devra permettre aux autres utilisateurs (développeurs) d'exploiter la bibliothèque sans souci.

Modalité de livraison

Le projet est à réaliser en binôme. Les binômes ont été générés de manière pseudo-aléatoire, de manière à ce qu'il n'y ait pas de groupe composés de deux personnes nouvellement arrivé à l'ESIEA.

Concernant le date de livraison de votre librairie, celle-ci est fixée au ***mardi 22 septembre à 23h59*** au maximum. La livraison se fera par mail en me mettant en copie. Pour chaque échange de mail (livraison comme SAV), vous devrez me mettre en copie et l'objet du mail devra commencer par ***LAB3040***.

Lors de la livraison vous fournirez une archive au format ZIP contenant les fichiers DLL/LIB/H ainsi que la documentation Doxygen associée dans un dossier DOC.

L'objet du mail devra être ***LAB3040 FOURNITURE LIB***.

Merci de respecter **scrupuleusement** les consignes de rendu. Un projet ne respectant pas les consignes de rendu se verra attribué un malus. Un projet rendu hors délai se verra lui aussi attribué un malus fonction du retard.