Création d'une solution gérant une bibliothèque statique, une application et une DLL, avec interfaçage avec Excel

1 Solution

Le but est de créer une solution contenant trois projets :
- un projet pour créer une bibliothèque statique, nommé **rational_stat**,
- un projet pour créer une application, nommé **rational_app**,
- un projet pour créer une bibliothèque dynamique (DLL), nommé **rational_dll**, pour l'interfaçage avec Excel.

L'application et la DLL utiliseront le code de la bibliothèque statique. Ainsi, le code du projet **rational_stat** ne sera pas *recopié* dans les projets **rational_app** et **rational_dll**, mais *utilisé*.

1.1 Bibliothèque statique

Pour créer le projet de la bibliothèque statique :

- File → New → Project
- Choisir Win32 et Win32 Console Application (à droite)
- nom projet (en bas) : *rational_stat*
- nom solution (en bas) : *Rational*
- Click sur OK. Dans la nouvelle fenêtre, cliquer sur Application Settings, puis à droite sur Static library, et décocher **Precompiled header**.
- Cliquer sur Finish.

Maintenant que le projet est crée, il faut rajouter les fichiers. On commence par le fichier d'en-tête :

- Dans **rational_stat**, click droit sur Header Files
- Add → New Item
- Choisir Code et Header File (.h)
- nom fichier(en bas) : rational

Maintenant on remplit le fichier rational.h :

```
#ifndef RATIONAL_H_
#define RATIONAL_H_

typedef struct _Rat Rat;

struct _Rat
{
  int num_;
  int den_;
};

Rat rat_new(int num, int den);
double rat_double_get(Rat r);

#endif /* RATIONAL_H_ */
```

On ajoute maintenant le fichier source :

- Dans **rational_stat**, click droit sur Source Files
- Add → New Item
- Choisir Code et C++ File (.cpp)
- nom fichier(en bas) : rational.c

Maintenant on remplit le fichier rational.c :

```
#include "rational.h"

Rat rat_new(int num, int den)
{
  Rat r;

  r.num_ = num;
  r.den_ = den;

  return r;
}

double rat_double_get(Rat r)
{
  return (double)r.num_ / r.den_;
}
```

Pour compiler et créer la bibliothèque statique, appuyer sur F7. Le fichier **rational_stat.lib** est crée dans *dossier_projets/Rational/Debug*.

1.2 Application

Pour créer le projet de l'application :

- Click droit sur la solution **Rational**
- Add → New Project
- Choisir Win32 et Win32 Console Application (à droite)
- nom projet (en bas) : *rational_app*
- Click sur OK. Dans la nouvelle fenêtre, cliquer sur Application Settings, puis à droite sur cocher Empty Project
- Cliquer sur Finish.

Maintenant que le projet est créé, il faut rajouter le fichier du programme :

- Dans **rational_app**, click droit sur Source Files
- Add → New Item
- Choisir Code et C++ File (.cpp)
- nom fichier(en bas) : rational_app.c

Maintenant on remplit le fichier rational_app.c :

```c
#include <stdio.h>

#include <rational.h>

int
main()
{
  Rat r;

  r = rat_new(2, 3);

  printf("%f\n", rat_double_get(r));

  return 0;
}
```

Il faut ensuite configurer le projet pour qu'il utilise la bibliothèque statique que l'on a créée :

- Click droit sur **rational_app** et choisir Properties. Une nouvelle fenêtre apparaît avec à gauche un ensemble d'options.
  - C/C++
    - General
      - à droite, Additional Include Directories
      - clicker sur **...** et une nouvelle fenêtre apparaît.
      - Cliquer sur l'icône jaune du dossier, puis sur **...**. Sélectionner le répertoire où se trouve le fichier **rational.h** : *dossier_projets/Rational/rational_stat*.
      - Cliquer sur OK.
  - Linker
    - General
      - Additional Library Directories
      - clicker sur **...** et une nouvelle fenêtre apparaît.
      - Mettre *dossier_projets/Rational/Debug*.
    - Input
      - Cliquer sur Additional Dependencies
      - clicker sur **...** et une nouvelle fenêtre apparaît.
      - Mettre **rational_stat.lib**
    - On clique sur OK
- On clique sur OK

Pour compiler et créer l'application, il faut commencer par dire à Visual Studio que le projet à compiler (le projet par défaut) est **rational_app**. En effet, appuyer maintenant sur F7 compilera **rational_stat**. Pour cela, click droit sur **rational_app**, puis choisir Set As Startup Project. Appuyer maintenant sur F7. Pour l'exécuter, appuyer sur Ctrl - F5.

1.3 Bibliothèque dynamique

Pour créer le projet de la bibliothèque dynamique :

- Click droit sur la solution **Rational**
- Add → New Project
- Choisir Win32 et Win32 Console Application (à droite)
- nom projet (en bas) : **rational_dll**
- Click sur OK. Dans la nouvelle fenêtre, cliquer sur Application Settings, puis à droite

sélectionner DLL et cocher Empty Project
- Cliquer sur Finish.

Maintenant que le projet est créé, il faut rajouter les fichiers d'interfaçage de la DLL :

- Dans **rational_dll**, click droit sur Header Files
- Add → New Item
- Choisir Code et Header File (.h)
- nom fichier(en bas) : rational_dll.h
- Dans **rational_dll**, click droit sur Source Files
- Add → New Item
- Choisir Code et C++ File (.cpp)
- nom fichier(en bas) : rational_dll.c

Pour remplir ces deux fichiers, voir la section 2.

On configure le projet exactement comme pour l'application (voir 1.2).
Il suffit de compiler la bibliothèque avec F7. Ne pas oublier au préalable de choisir ce projet comme projet par défaut.

1.4 Arborescence de la solution

Maintenant que tout est configuré proprement, voici l'arborescence de la solution, avec les principaux fichiers :

- Rational/
  - Debug/
    - rational_stat.lib
    - rational_app.exe
    - rational_dll.dll
  - rational_stat/
    - rational.h
    - rational.c
    - Debug
      - rational.obj
  - rational_app/
    - rational_app.c
    - Debug
      - rational_app.obj
  - rational_dll/
    - rational_dll.h
    - rational_dll.c
    - Debug
      - rational_dll.obj

2 Interfaçage avec Excel

Le but de cette section est de décrire les étapes pour que le code de la bibliothèque statique soit utilisé avec VBA.

2.1 Fichier source

La définition des fonctions qui doivent être utilisées dans VBA doivent avoir le prototype suivant :

```
type_retour __stdcall nom_fct (parametres)
{...}
```

c'est-à-dire, juste après le type de retour, l'ajout de **__stdcall**. Voici un exemple pour rational_dll.c :

```
#include <stdio.h>

#include <rational_stat.h>
#include "rational_dll.h"

double __stdcall rat_double_get (int num, int den)
{
  Rat r;

  r = rat_new(num, den);

  return r.double_get();
}
```

Ainsi, pour récupérer la valeur en tant que double d'un rationnel, le programme VBA pourra utiliser la fonction **rat_double_get**.

2.2 Fichier d'en-tête

Dans le fichier d'en-tête **rational_dll.h**, il faut mettre la déclaration des fonctions qui vont être utilisées dans Excel. En plus de **__stdcall**, il faut rajouter en début de déclaration **__declspec(dllexport)**. On peut utiliser une macro pour faciliter l'écriture. On doit, de plus englober ces déclarations dans un bloc **extern "C"**.

Voici le fichier d'en-tête pour *rational_dll.c*, nommé *rational_dll.h* :

```
#ifndef __RATIONAL_DLL_H__
#define __RATIONAL_DLL_H__

#define DLL_EXPORT __declspec(dllexport)

extern "C"
{
  DLL_EXPORT double __stdcall rat_double_get (int num, int den);
}

#endif /* __RATIONAL_DLL_H__ */
```

2.3 Utilisation dans Excel

- Dans Excel, ouvrir une nouvelle feuille et la sauvegarder sous le nom **rational_dll.xls**, par exemple.
- Appuyer sur Alt-F11 pour ouvrir VBA.
- Ajouter un module (click droit sur **VBA Project (rational_dll.xls)**, puis Insertion → Module).
- Au début du programme en VBA, il faut indiquer les fonctions à utiliser dans le programme en VBA. La syntaxe est la suivante :
  `Declare Function nom_fct_vba "chemin_dll" Alias "_nom_fct_c@N" (parametres_vba) type_retour_vba`
  où :
  - *nom_fct_vba* est le nom de la fonction utilisée dans le programme en VBA.
  - *chemin_dll* est le nom de la DLL avec son chemin complet.
  - *_nom_fct_c@N* est le nom de la fonction en C correspondante, préfixée par un underscore, et à laquelle on ajoute le symbole arrobe (@), suivi de *N*, le nombre d'octets de ses arguments. Voir ci-dessous pour le calcul de *N*.
  - *parametres_vba* sont les paramètres de la fonction en C, mais avec la syntaxe du VBA.
  - *type_retour_vba* est le type de retour de la fonction en C, mais avec la syntaxe du VBA.

  Voici un exemple d'utilisation de la DLL *rational_dll.dll* créée ci-dessus :

```
Declare Function rat_double_get Lib "c:chemin\rational_dll.dll" _
 Alias "_rat_double_get@8" (ByVal num As Long, ByVal den As Long) _
 As Double

Sub test()

    Feuil1.Cells(1, 1).Value = rat_double_get (2, 7)

End Sub
```

Le calcul de N est fait ainsi : on regarde les types de chacun des paramètres des fonctions en C et on fait la somme de la taille en octet des types apparaissant dans le prototype. Ici, la fonction **rat_double_get** a deux paramètres, tous des *int*. Un *int* étant codé sur 4 octets, $N = 4 + 4 = 8$ pour **rat_double_get**.
Si le prototype de **rat_double_get** avait été :

```
double rat_double_get (int i, double x)
```

sachant que un **double** est codé sur 8 octets, $N = 4 + 8 = 12$ pour cette nouvelle fonction **rat_double_get**.

Il suffit maintenant d'exécuter la macro dans la feuille Excel créée au début.