1 Librairies sous Linux avec GCC

Prenons l'exemple des trois fichiers suivants :

- Listing de affiche.c :

```
#include <stdio.h>

void affiche(char *str)
{
     printf("String : %s\n", str);
}
```

- Listing de affiche.h :

```
void affiche(char *str);
```

- Listing de client.c :

```
#include "affiche.h"

void main()
{
     affiche("coucou");
}
```

1.1 Projet multi-fichiers simple

Compilation multi-fichiers simple : **cc *.c -o client**

Détermination du type de fichier : **file client**

Pour connaître les symboles contenus dans l'exécutable : **nm client**

Pour connaître les symboles non définis contenus dans des librairies dynamiques externes : **objdump -T client**

Pour savoir dans quelles libraires externes sont contenues ces fonctions, il faut déterminer les dépendances dynamiques : **ldd client**

Qu'est-ce qu'il y a dans les librairies dynamiques ? **objdump -T /lib/libc.so.6|more**. Dans la section .text, on retrouve toutes les fonctions disponibles.

Comment savoir si une fonction particulière existe ? **objdump -T /lib/libc.so.6|grep fgets**

Exemple : voir toutes les fonctions de la librairie mathématique : **objdump -T /lib/libm.so.6|more**

Taille de l'exécutable lié dynamiquement avec la libc : **ll client**

On peut lier statiquement le programme image : **cc -static *.c -o client**

Vérification : **file client**

L'exécutable ne dépend plus de rien : **ldd client**

Les symboles dynamiques ont disparus : **objdump -T client**

car toutes les fonctions sont maintenant incorporées dans l'exécutable : **nm client|grep printf**

Inconvénient, la taille de l'exécutable augmente de manière considérable : **ll client**

Le linker a utilisé la version statique de la bibliothèque (archive) libc : **ll /usr/lib/libc.a**

Pour voir le contenu d'une archive : **ar -t /usr/lib/libc.a|more**

Pour rechercher un fichier objet : **ar -t /usr/lib/libc.a|grep fgets**

On ne voit que les fichiers objets contenus dans l'archive. Pour voir les symboles dans les objets, il faut utiliser : **objdump -t /usr/lib/libc.a|grep fgets**

1.2 Projet multi-fichiers avec une librairie statique

Pour créer votre librairie statique, il faut :

- compiler les sources sans linker pour obtenir uniquement des objets : **cc -c *.c**
- créer l'archive : **ar -rv libAFF.a affiche.o**

Bien sur, il ne faut pas mettre **client.o** dans la liste des objets à insérer dans l'archive.

Les objets sont au format ELF (ex : **file affiche.o**). On peut voir les symboles définis (T) et non définis (U) avec : **nm affiche.o**

Le fichier **libAFF.a** est une archive (**file libAFF.a**) qui contient les objets que nous y avons mis : **ar -t libAFF.a**

Pour lier cette librairie avec notre programme principal **client.o**, il faut écrire : **cc client.o libAFF.a -o client**

Dans ce cas, l'exécutable dépend toujours des librairies dynamiques externes (**ldd client**). On peut aussi linker en statique avec : **cc -static client.o libAFF.a -o client**

La taille des exécutables ne change pas par rapport à celles constatées au §1.1.

1.3 Création d'une librairie dynamique

Pour créer votre librairie dynamique, il faut :

- compiler les sources sans linker (avec du code relogeable) : **cc -c *.c -fPIC**
- créer la bibliothèque : **cc -shared -o libAFF.so affiche.o -nostartfiles**

Bien sur, il ne faut pas mettre **client.o** dans la liste des objets à insérer dans la librairie. Vous pouvez utiliser deux fonctions spéciales `_init()` et `_fini()` qui seront activées au chargement et au déchargement de la DLL. Le listing de affiche.c peut être modifié de la manière suivante :

```
#include <stdio.h>

void affiche(char *str)
{
     printf("String : %s\n", str);
}

void _init()
{
     printf("chargement DLL\n");
}

void _fini()
{
     printf("dechargement DLL\n");
}
```

1.4 Projet multi-fichiers utilisant l'édition de lien implicite avec une DLL.

Il ne reste plus qu'à compiler le programme principal : **cc client.c –o client –L. –lAFF**

Avant de lancez le programme : **./client**, il faut lui indiquer le répertoire où se trouve la librairie. Pour cela, il faut modifier une variable d'environnement :

export LD_LIBRARY_PATH=. :$LD_LIBRARY_PATH

Pour vérifier que la variable a été modifiée : **echo $LD_LIBRARY_PATH**

Vous devez finalement obtenir la sortie suivante :

```
chargement DLL
String : coucou
dechargement DLL
```

L'édition de lien implicite avec la librairie standard du c (/lib/libc.so.6) est la méthode utilisée par défaut quand vous compilez un programme sous linux. Depuis 1995, Unix utilise le format d'objet ELF et les exécutables sont dynamiques par défaut. Quand vous compilez votre programme essai.c avec ***cc essai.c –o essai****, il y a un lien automatiquement créé avec la libc.*

1.5 Projet multi-fichiers utilisant l'édition de lien explicite avec une DLL.

Il faut modifier le client pour charger explicitement la DLL et copier l'adresse de la fonction `affiche` dans un pointeur de fonction. Le listing de client.c devient :

```
#include "affiche.h"
#include <dlfcn.h>

void main()
{
     void *handle;
     void (*affiche_dyn)(char *);

     handle=dlopen("./libAFF.so", RTLD_LAZY);
     affiche_dyn=dlsym(handle, "affiche");

     affiche_dyn("coucou");
}
```

Pour lier la DLL avec notre programme principal, il faut écrire : **cc -rdynamic client.c -o client -ldl**

Vous devez obtenir la sortie suivante :

```
chargement DLL
String : coucou
dechargement DLL
```

2 Librairies sous Windows avec Visual C++ 6.0

Prenons l'exemple des trois fichiers suivants :

- Listing de affiche.c :

```
#include <stdio.h>

void affiche(char *str)
{
     printf("String : %s\n", str);
}
```

- Listing de affiche.h :

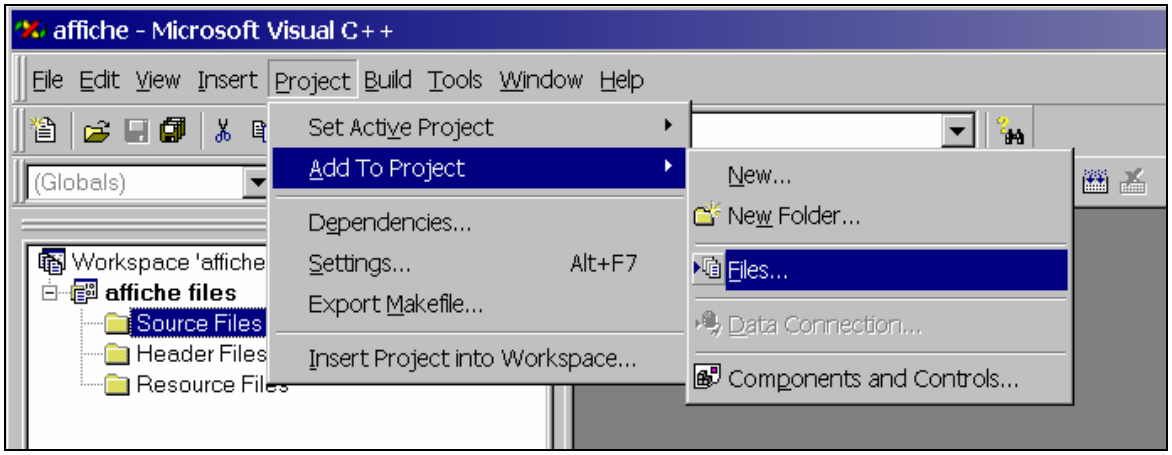
```
void affiche(char *str);
```

- Listing de client.c :

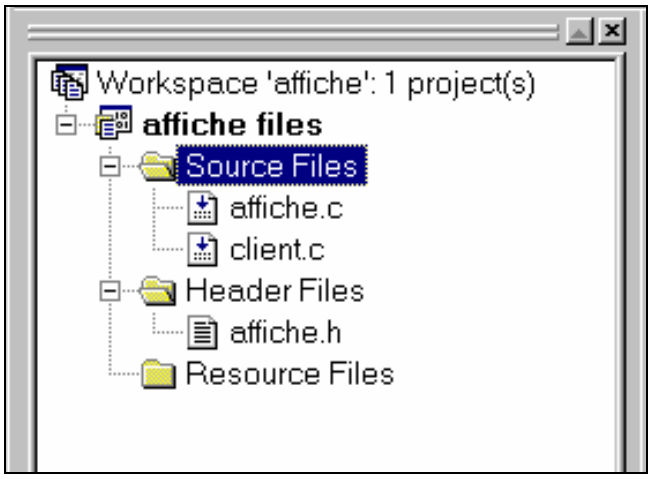
```
#include "affiche.h"

void main()
{
     affiche("coucou");
}
```

2.1 Projet multi-fichiers simple.

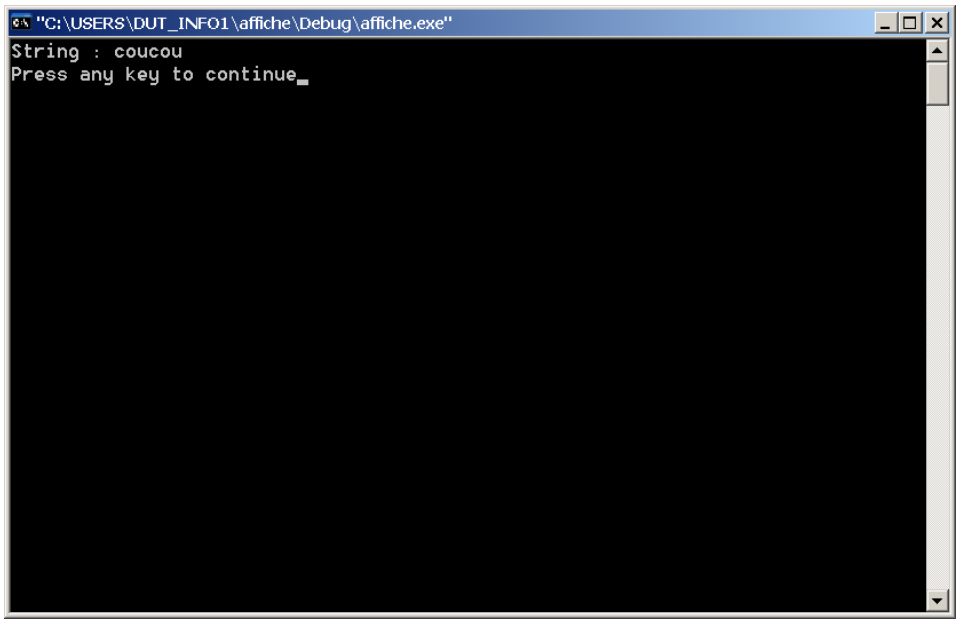
- Créez un projet console win32 sous Windows (par exemple nom de projet « `affiche` » dans le répertoire c:\users\dut_info1). Vous trouverez une documentation sur l'utilisation de Visual C++ 6.0 sur : http://cristale.free.fr/index_fichiers/support/VisualC++_6.0.pdf.
- Copiez vos fichiers .c et .h dans le répertoire `c:\users\dut_info1\affiche`.
- Ajoutez les fichiers .c et .h au projet en cliquant sur :



Vous devez obtenir :

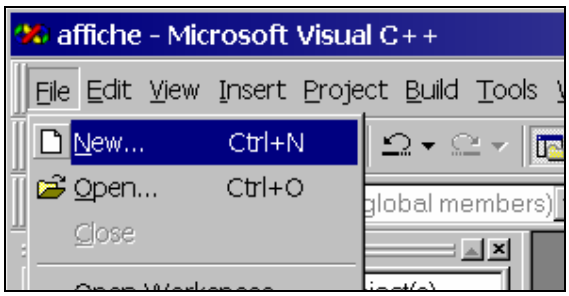


Compilez le projet puis lancez le programme (Ctrl+F5) et vérifiez que tout fonctionne bien :

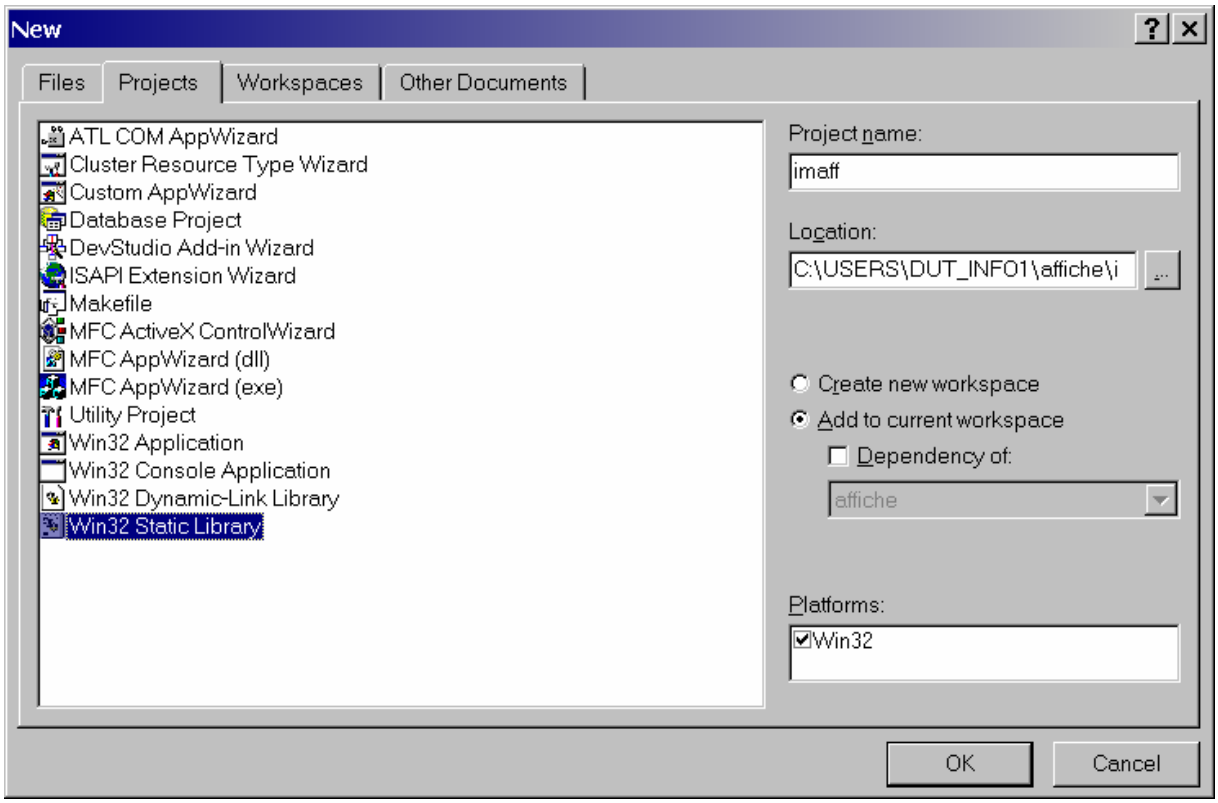


2.2 Projet multi-fichiers avec une librairie statique.

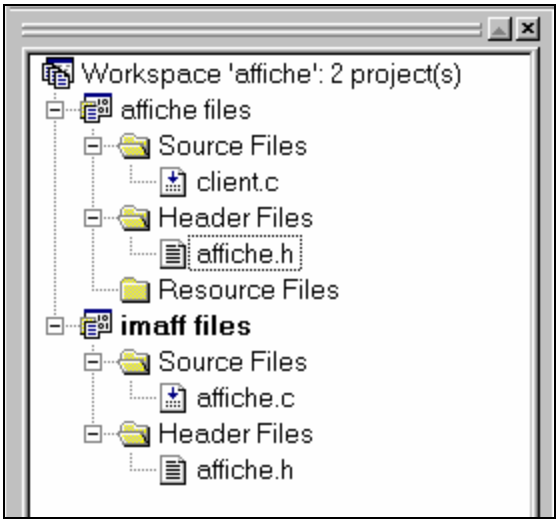
Insérez dans l'espace de travail (workspace) une librairie statique en cliquant sur :



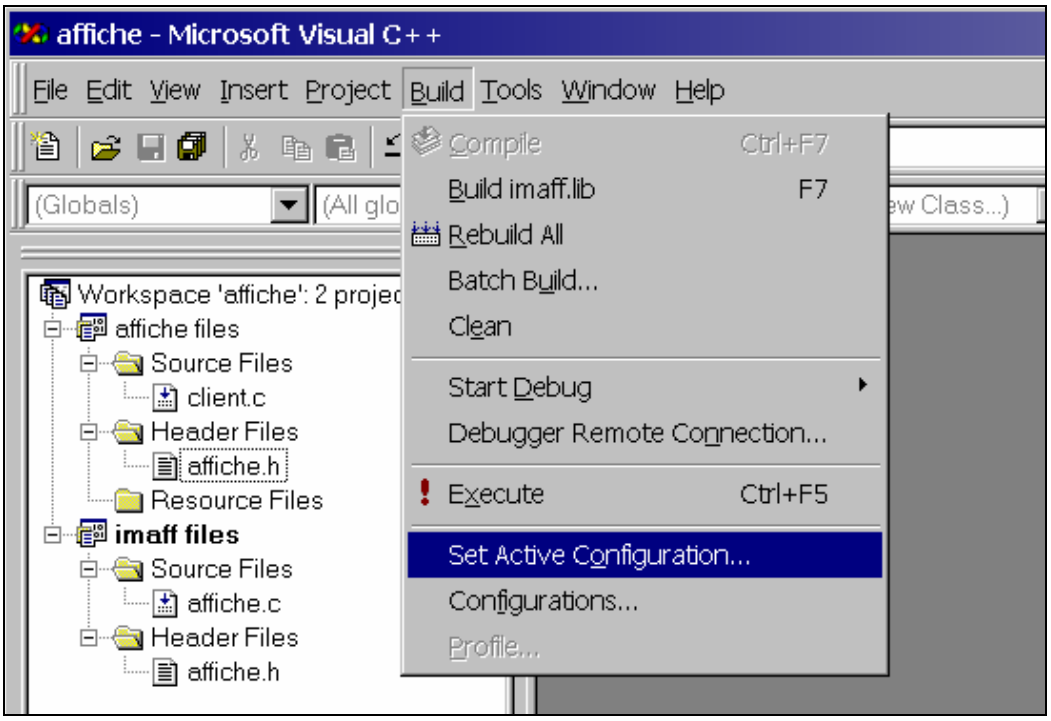
Vous devez obtenir exactement la fenêtre suivante avant de cliquer sur OK :



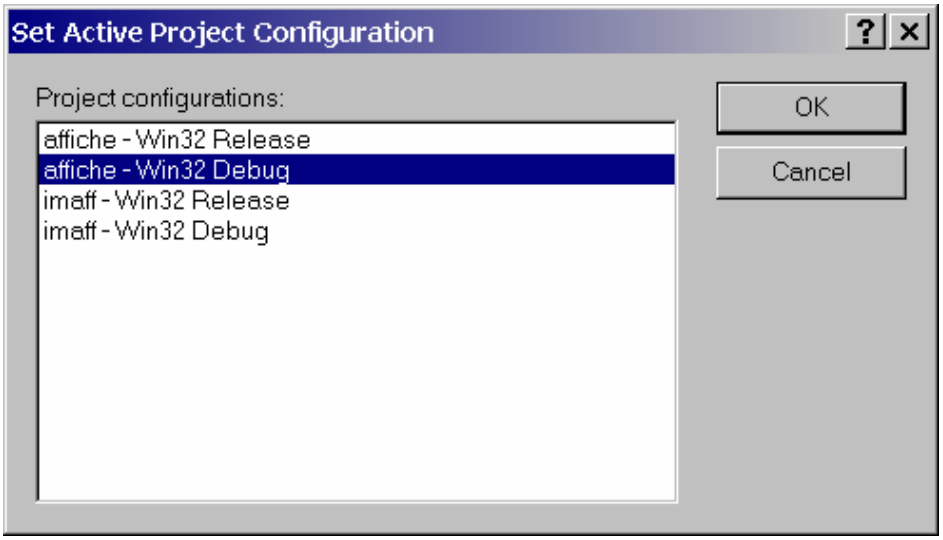
Sélectionnez les fichiers dans le projet `affiche` et copiez-les dans le projet `imaff` (pour déplacer, sélectionnez puis tirez ; pour copier, sélectionnez, appuyez sur la touche Ctrl puis tirez). Vous devez obtenir le résultat suivant (`affiche.c` a été déplacé, `affiche.h` a été copié) :



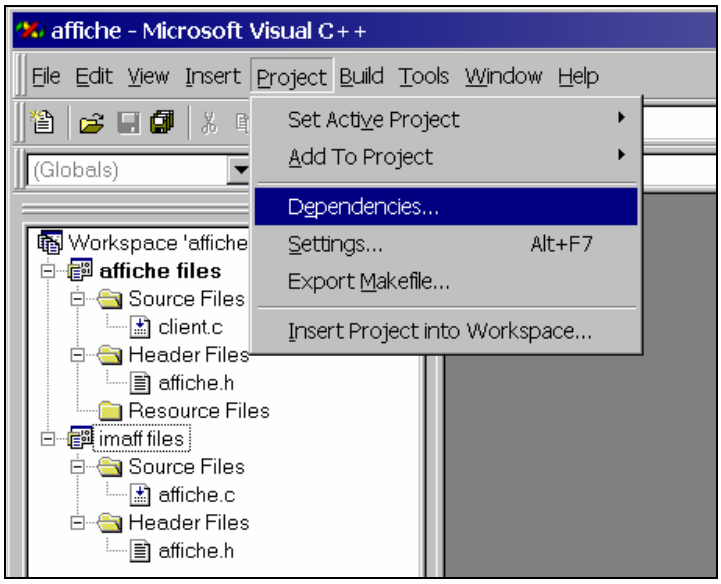
Sélectionnez la configuration active :



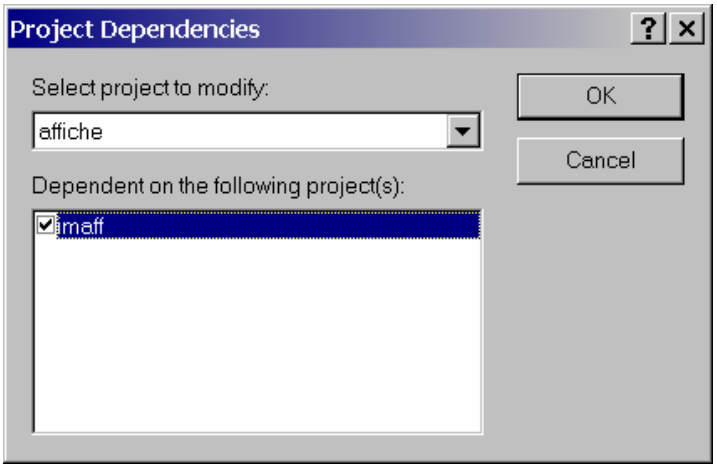
En cliquant sur :



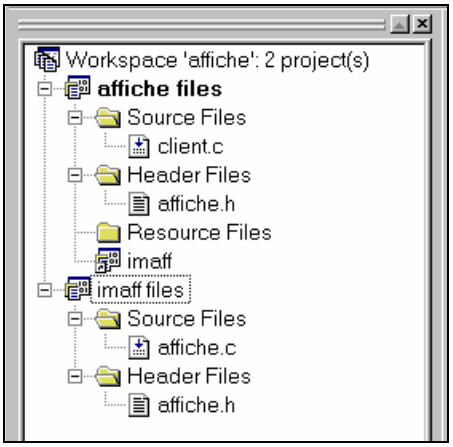
Puis indiquez à Visual C++ que le projet « `affiche` » dépend du projet « `imaff` » en cliquant sur :



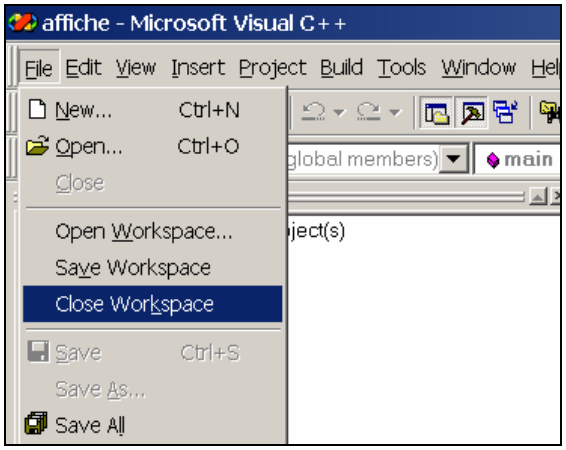
puis en sélectionnant :



Vous devez finalement obtenir :



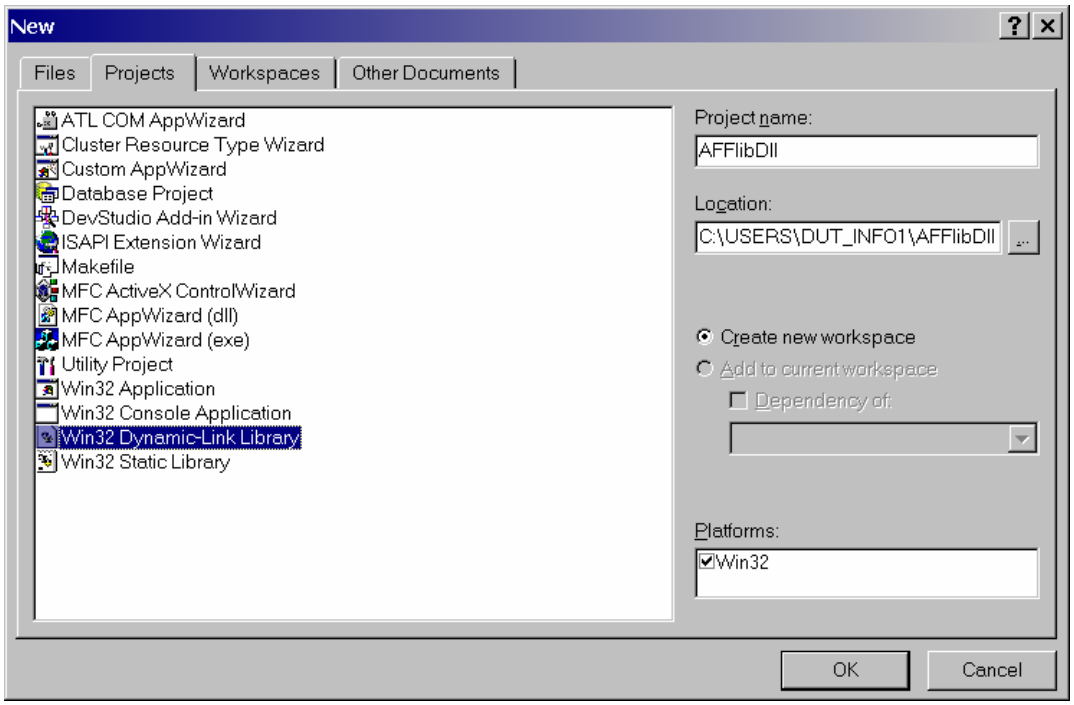
Compilez puis exécutez le programme et vérifiez que tout fonctionne bien. Fermez l'espace de travail :



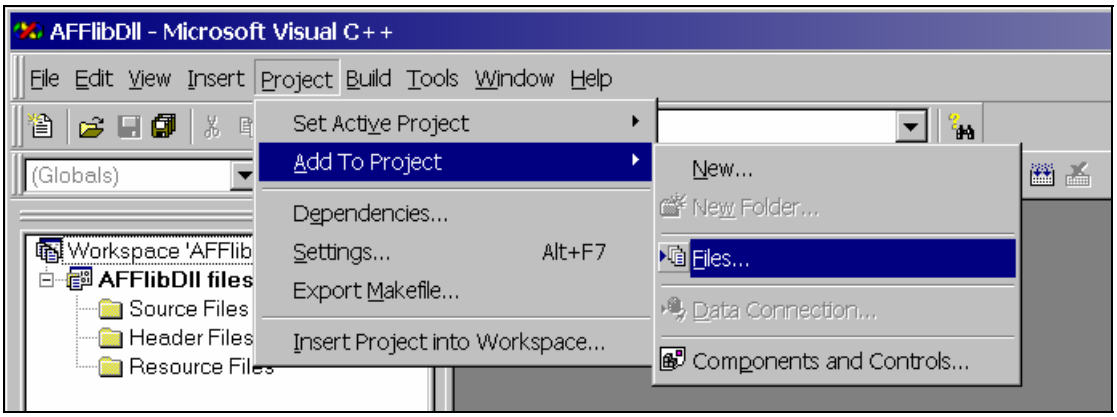
L'édition de lien statique avec la librairie standard du c (c:\Program Files\Microsoft Visual Studio\VC98\Lib\libc.lib) est la méthode utilisée par défaut quand vous compilez un programme sous Visual C++. Par contre, l'exécutable est lié dynamiquement avec le noyau du système d'exploitation kernel32.dll.

2.3 Création d'une DLL.

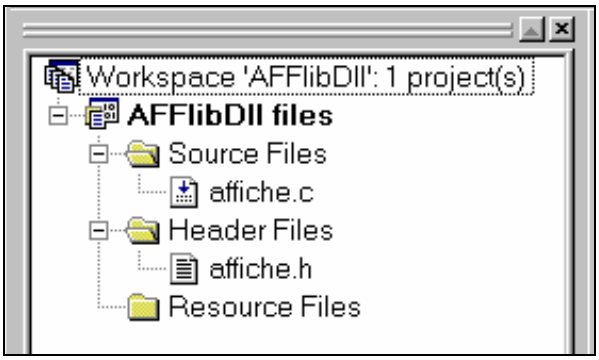
Créez le nouvel espace de travail suivant (empty DLL 32 bits nommée « `AFFlibDll` » se trouvant dans le répertoire `c:\users\dut_info1`) :



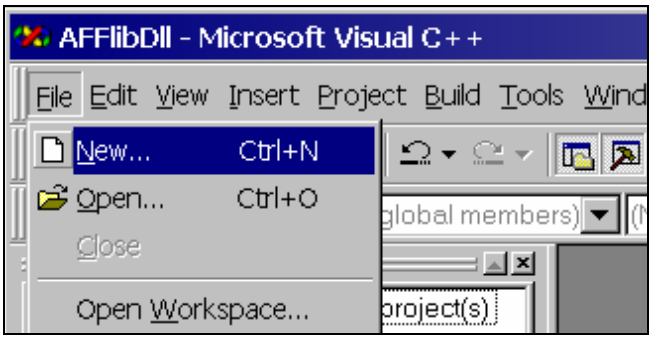
Copiez vos fichiers *.c et *.h dans le répertoire c:\users\dut_info1\AFFlibDll (sauf celui qui contint la fonction `main`). Ajoutez les fichiers *.c et *.h au projet en cliquant sur :



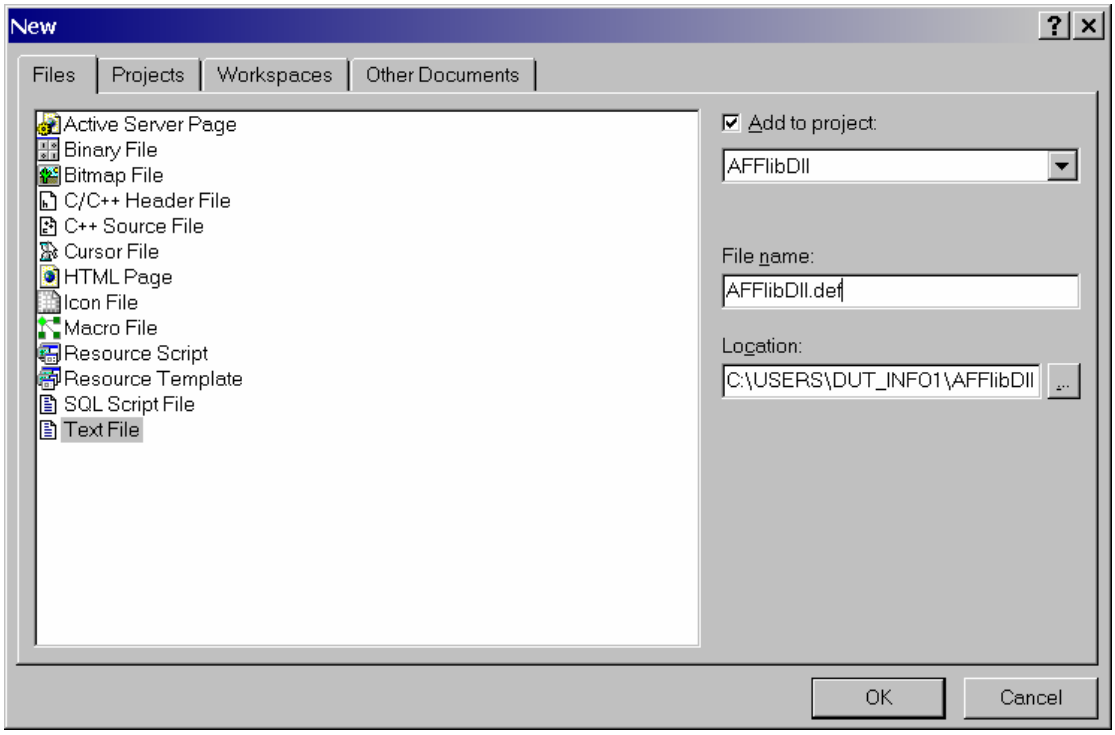
afin d'obtenir :



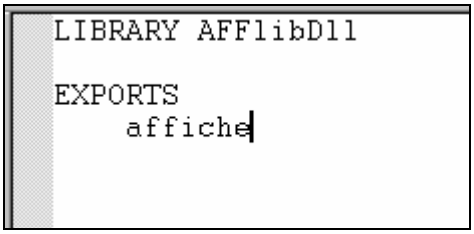
Pour créer une DLL, il faut ajouter au projet un fichier texte que l'on appellera `AFFlibDll.def`. Pour cela, cliquez sur :



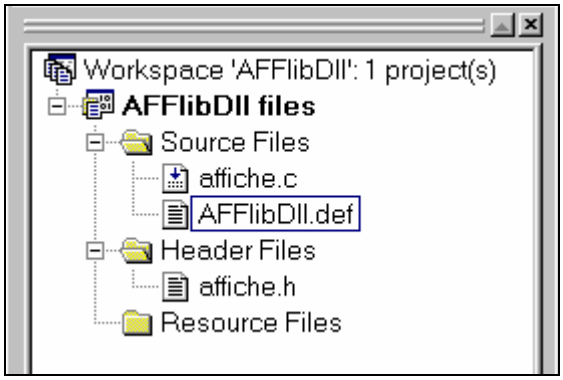
Et sélectionnez :



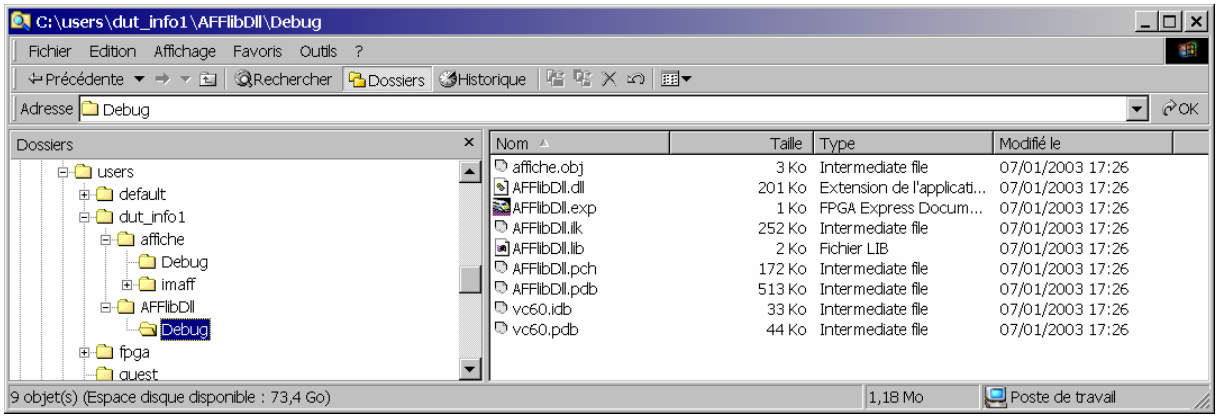
Ce fichier contient la liste des fonctions exportées par la DLL.



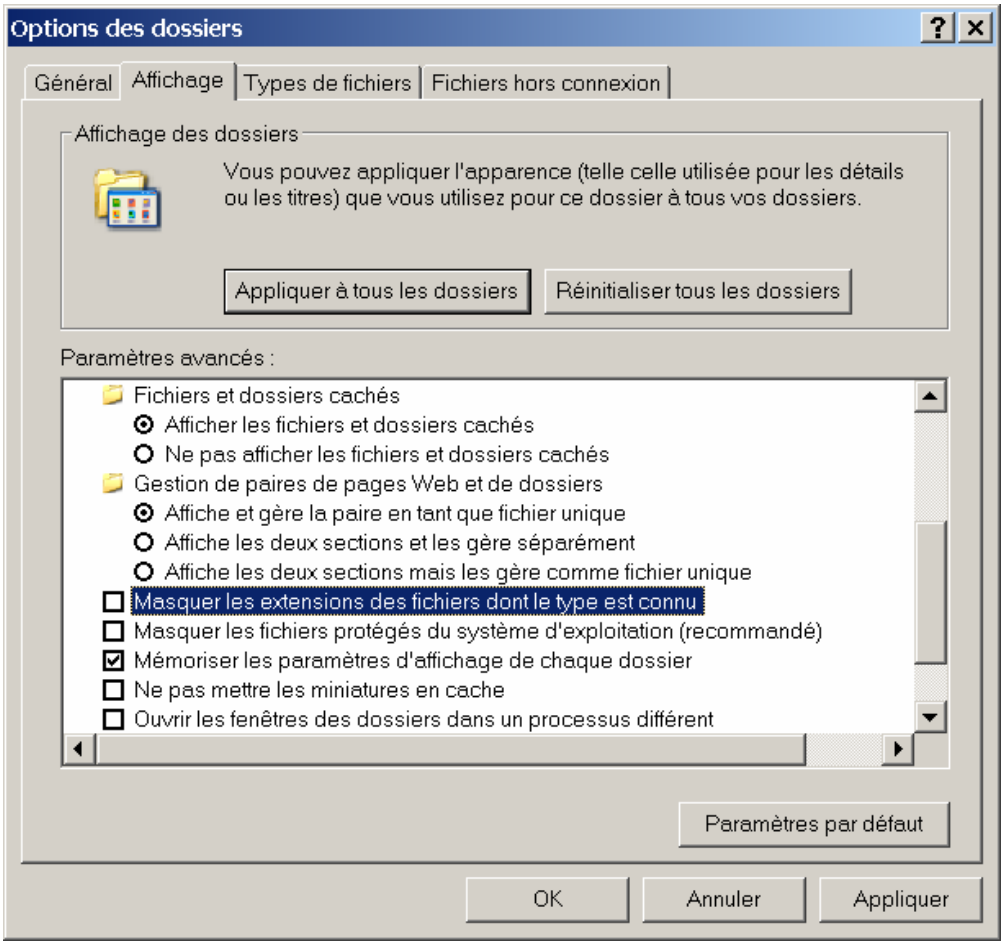
Vous devez finalement obtenir :



Compilez la DLL. Vous pouvez faire une première vérification du résultat en utilisant l'explorateur de fichiers et en sélectionnant le fichier DLL :

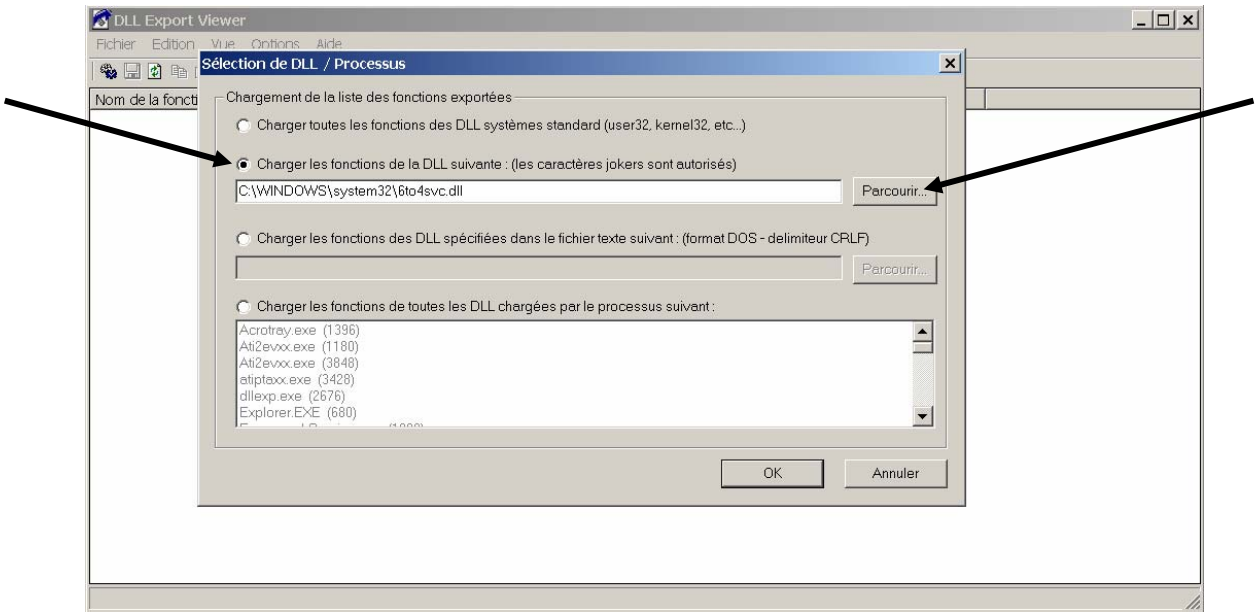


Si vous ne voyez pas la DLL, cliquez sur « Outils », « Options des dossiers… » et faites apparaître l'onglet « Affichage ». Vérifiez que les options sont les mêmes que dans la fenêtre suivante (l'explorateur cache les DLL par défaut).

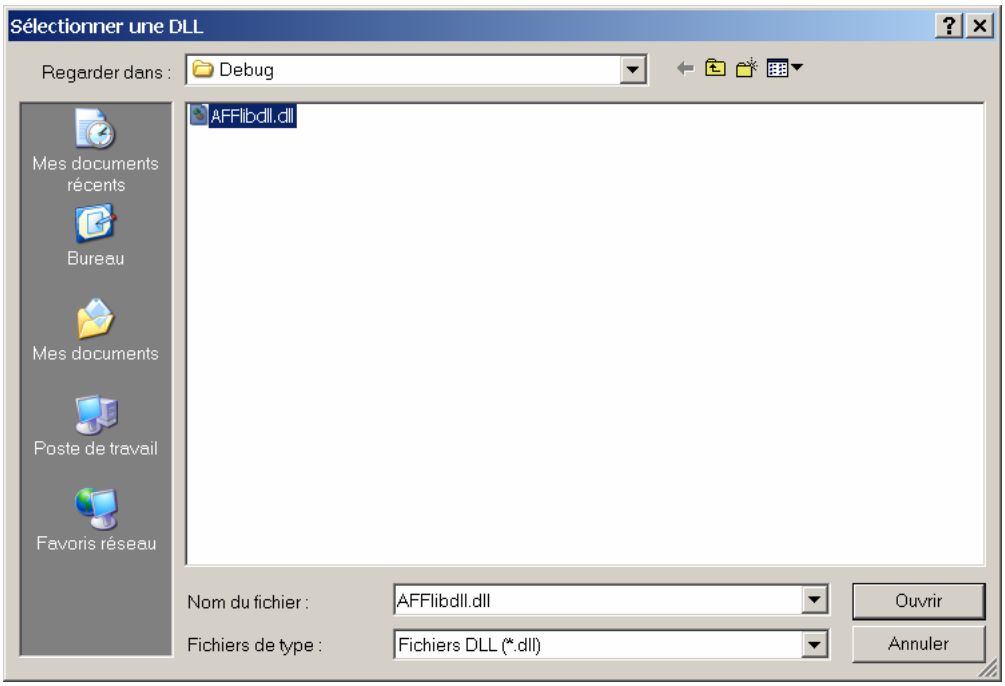


Une fois que vous voyez la DLL dans l'explorateur de fichiers, exécutez le programme « DLL Export Viewer » (`C:\Program Files\dllexp_fr\dllexp.exe` sur les machines du laboratoire). Si ce programme n'existe pas sur votre PC, allez le chercher sur

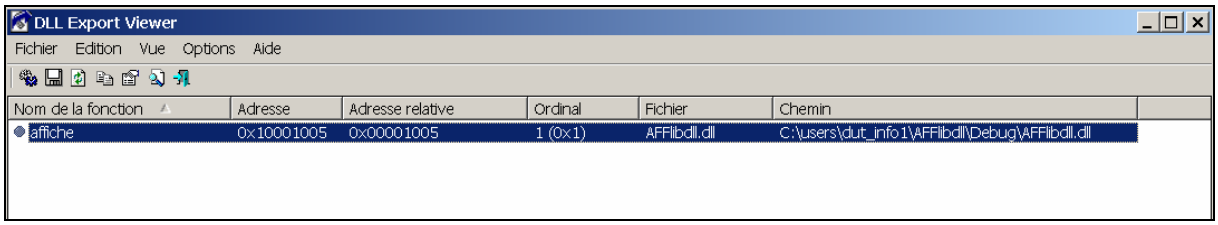
http://cristale.free.fr/index_fichiers/soft/dllexp_fr.zip, décompactez-le où vous le souhaitez sur votre machine puis lancez-le. Cochez la case « Charger les fonctions de la DLL suivante… » puis cliquez sur le bouton « Parcourir » :



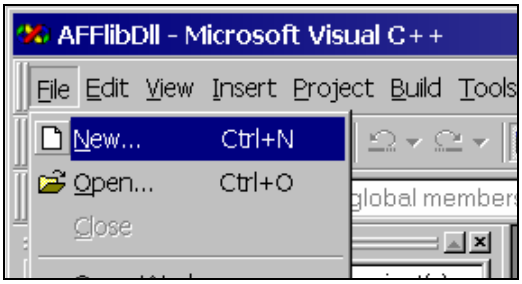
Sélectionnez votre DLL puis cliquez sur « Ouvrir » :



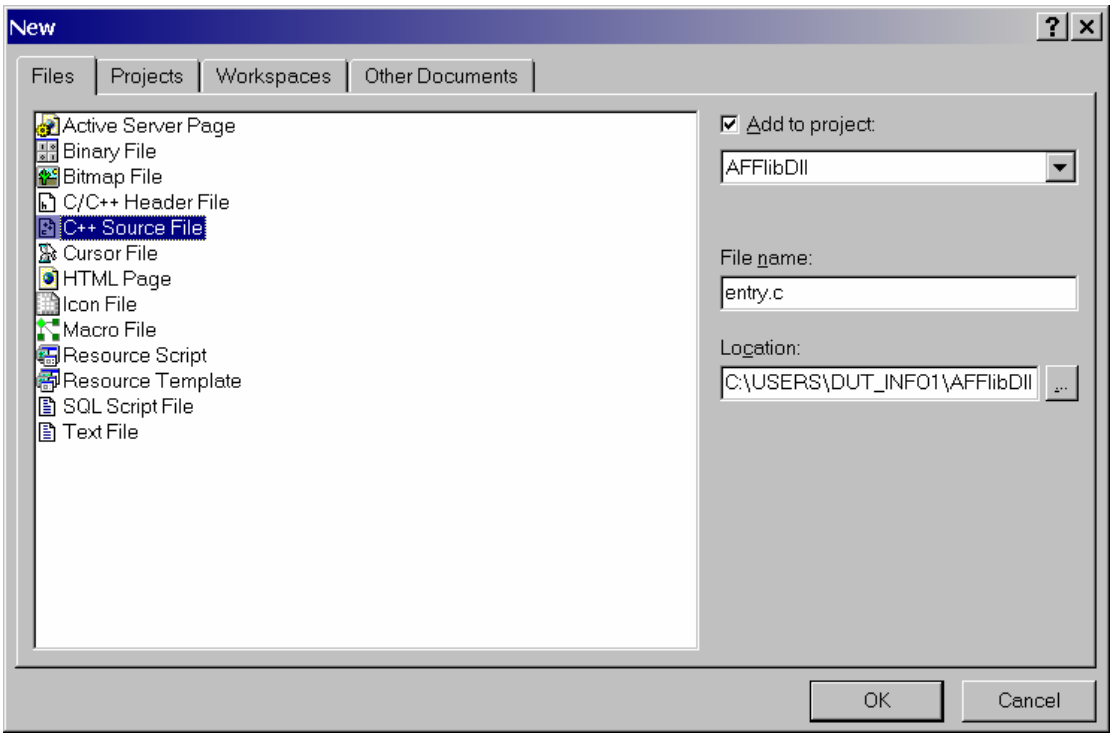
Cliquez sur « OK » dans la fenêtre de sélection de DLL. Les fonctions exportées par la DLL apparaissent dans une fenêtre :



Il existe d'autres possibilités lorsque l'on crée une DLL. Vous pouvez ajouter au projet un fichier source en C que l'on appellera `entry.c`. Pour cela, cliquez sur :



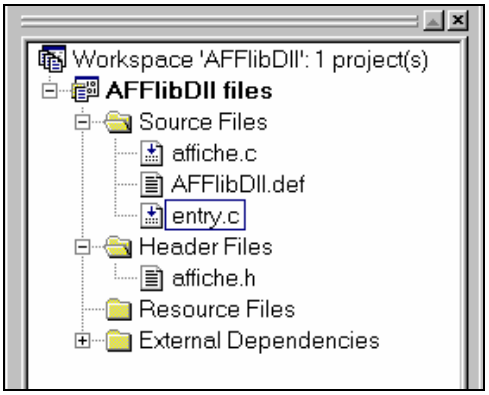
et sélectionnez :



Ce fichier n'est pas obligatoire. Il permet d'initialiser éventuellement des processus au moment du chargement ou du déchargement de la DLL. Dans cet exemple, il ne réalise rien :

```
#include <windows.h>

BOOL WINAPI DllEntryPoint(
    HINSTANCE hinstDLL,  // handle to DLL module
    DWORD fdwReason,     // reason for calling function
    LPVOID lpReserved )  // reserved
{
    // Perform actions based on the reason for calling.
    switch(fdwReason)
    {
        case DLL_PROCESS_ATTACH:
         // Initialize once for each new process.
         // Return FALSE to fail DLL load.
            break;

        case DLL_PROCESS_DETACH:
         // Perform any necessary cleanup.
            break;
    }
    return TRUE;  // Successful DLL_PROCESS_ATTACH.
}
```

Vous devez finalement obtenir :



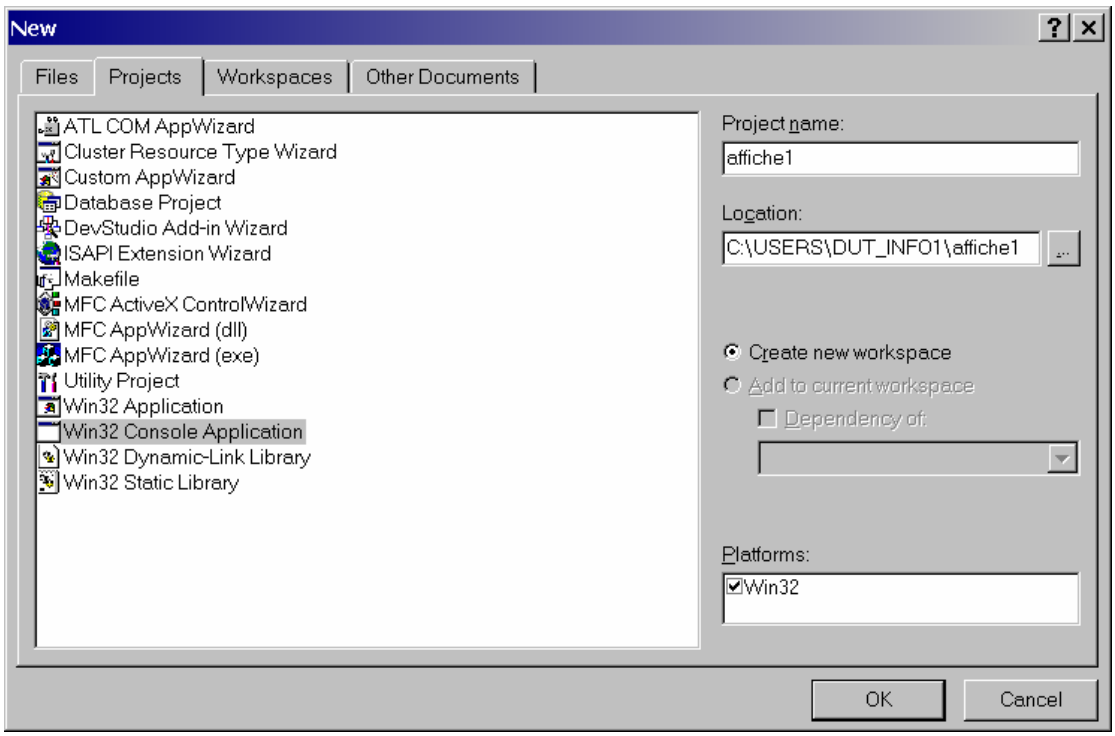
Recompilez la DLL et testez-la avec DLL Export Viewer. Il est aussi possible de ne pas utiliser le fichier .def pour exporter les fonctions mais d'utiliser la fonction `dllexport`. Supprimer le fichier `AFFlibDll.def` du projet en le sélectionnant et en cliquant sur la touche « Suppr » du clavier. Modifiez le fichier `affiche.c` de la manière suivante :

```
#include <stdio.h>

__declspec( dllexport ) void affiche(char *str);

void affiche(char *str)
{
    printf("String : %s\n", str);
}
```

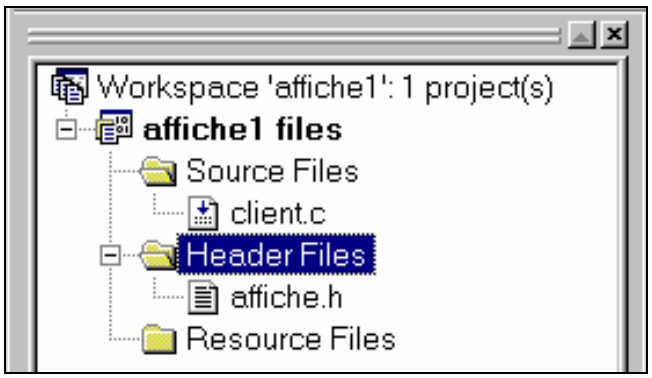
Faites bien attention aux espaces et au double __. Recompilez la DLL et testez-la avec DLL Export Viewer. Vous pouvez maintenant fermer le projet.

2.4 Projet multi-fichiers utilisant l'édition de lien implicite avec une DLL.

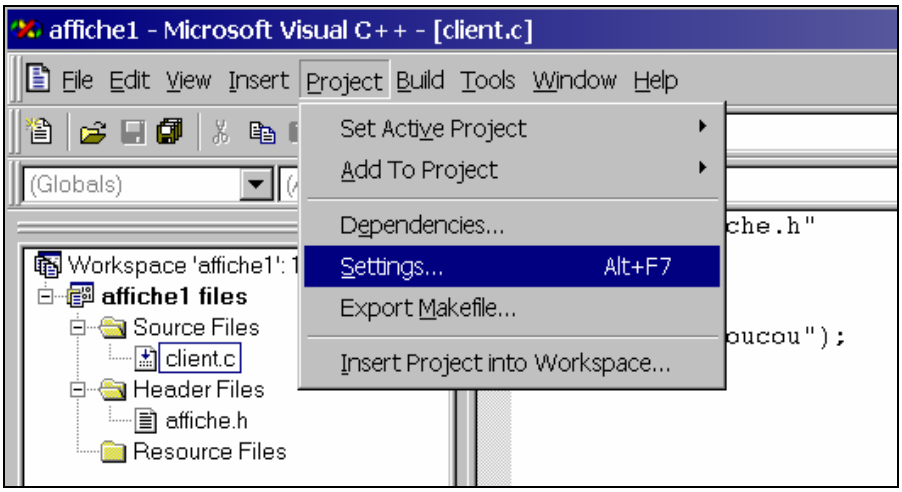
Créez un nouveau projet console win32 qui s'appellera `affiche1`.



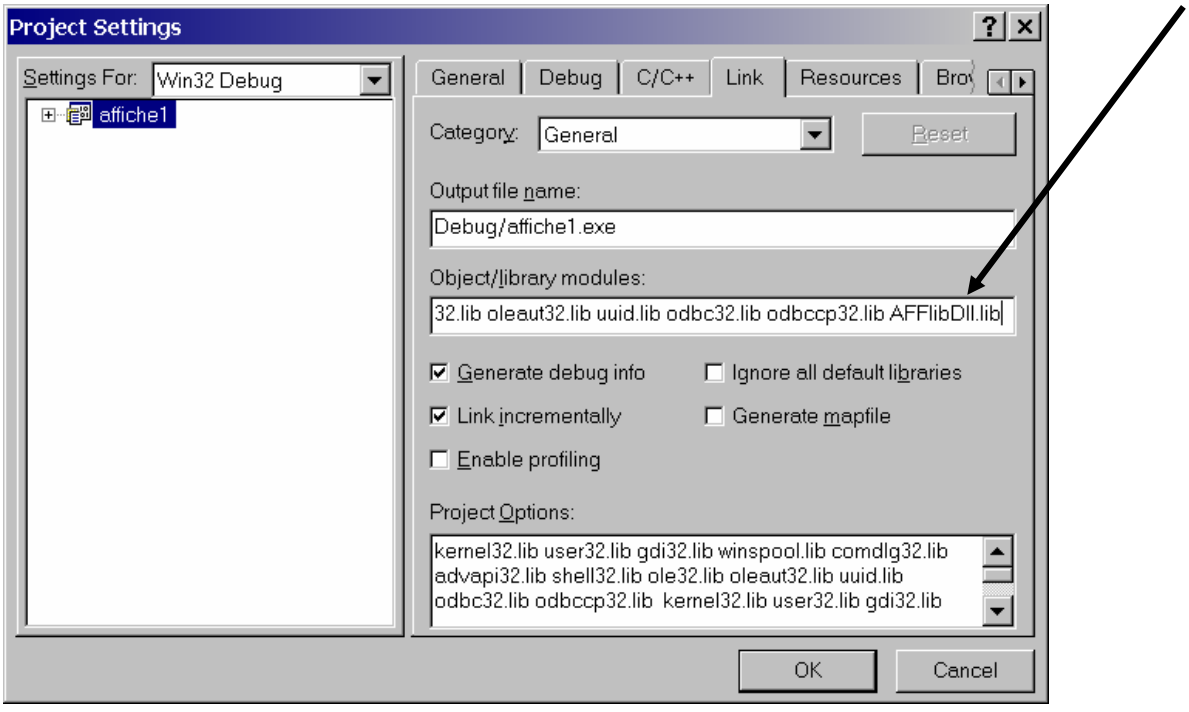
Copiez vos fichiers `client.c` et `affiche.h` dans le répertoire c:\users\dut_info1\affiche1. Copiez au même endroit le fichier `AFFlibDll.lib` se trouvant dans le répertoire de création de la DLL (c:\users\dut_info1\AFFlibDll\Debug). Ajoutez les fichiers `client.c` et `affiche.h` au projet. Vous devez finalement obtenir :



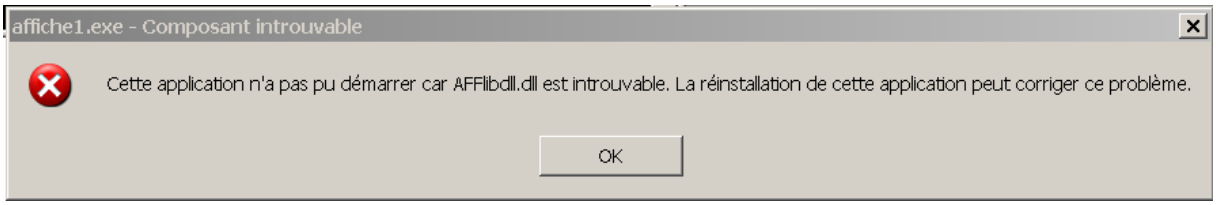
Il faut maintenant indiquer à Visual C++ le nom de la librairie d'importation à utiliser pour l'édition de lien. Il s'agit de `AFFlibDll.lib` :



Sélectionnez l'onglet « Link » puis tapez le nom de la librairie à la fin de la ligne « Object/library Modules : » :



Compilez le projet puis exécutez-le. Un message d'erreur apparaît car la DLL n'a pas été trouvée. Ce message est émis par le système d'exploitation puisque c'est lui qui gère le chargement de la DLL.

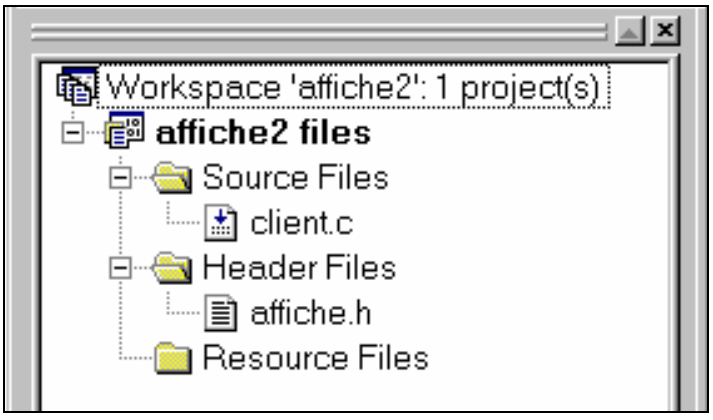


Copiez la DLL `AFFlibDll.dll` dans le répertoire `c:\users\dut_info1\affiche1` et relancez le programme qui doit maintenant marcher correctement. Bien sur, en temps normal, on copie les DLL dans le répertoire `c:\winnt\system32` pour qu'elles soient accessibles par tous les programmes. Vous noterez que le programme ne connaît pas le nom de la DLL.

L'édition de lien implicite est la manière la plus simple pour utiliser une DLL. Mais elle vous oblige à lier l'exécutable avec la librairie d'importation `.lib` qui est créée par défaut en même temps que la DLL. Pour que l'exécutable soit totalement indépendant de la DLL, il faut utiliser l'édition de lien explicite.

2.5 Projet multi-fichiers utilisant l'édition de lien explicite avec une DLL.

Créez un nouveau projet console win32 qui s'appellera `affiche2`. Copiez vos fichiers `client.c` et `affiche.h` dans le répertoire c:\users\dut_info1\affiche2. Ajoutez les fichiers `client.c` et `affiche.h` au projet. Vous devez finalement obtenir :



La méthode préconisée par Microsoft pour appeler explicitement les fonctions contenues dans une DLL est la suivante. A l'aide d'un `typedef`, créez un nouveau type correspondant à un pointeur sur une fonction, par exemple :

```
typedef void (*AFF)(char *str);
```

puis créez un pointeur de fonction en utilisant ce type :

```
AFF affiche_dll;
```

Ces deux déclarations peuvent être placées dans le fichier d'entête. Voici un exemple de fichier `affiche.h`. Le pointeur opaque `maDLL` permet de récupérer l'adresse de la DLL.

```
#include <windows.h>

HINSTANCE maDLL = NULL;

typedef void (*AFF)(char *str);

AFF affiche_dll;
```

L'appel de la DLL s'effectue de la façon suivante (fichier `client.c`) :

```
#include "affiche.h"

void main()
{
    maDLL=LoadLibrary("AFFlibDll.dll");

    if (maDLL == NULL) {
        MessageBox(0,"La DLL AFFlibDll.dll n'a pas été chargée.","ERROR",0);
        exit(0);
    }

    affiche_dll = (AFF)GetProcAddress(maDLL, "affiche");

    affiche_dll("coucou");
}
```

La fonction `LoadLibrary` charge la DLL en mémoire et la fonction `GetProcAddress` récupère l'adresse d'une fonction dans la DLL. Une fois l'adresse copiée dans le pointeur de fonction, celui-ci s'utilise à la place de la fonction.

Compilez le projet `affiche2` puis exécutez-le. Un message d'erreur apparaît car la DLL n'a pas été chargée. Ce message n'est plus émis par le système d'exploitation puisque c'est votre programme qui gère le chargement de la DLL. En cas de problème de chargement, vous

pouvez gérer l'erreur et demander par exemple à l'utilisateur d'indiquer un autre répertoire pour chercher cette DLL.

Copiez la DLL `AFFlibDll.dll` dans le répertoire `c:\users\dut_info1\affiche2` et relancez le programme qui doit maintenant marcher correctement. Bien sur, en temps normal, on copie les DLL dans le répertoire `c:\winnt\system32` pour qu'elles soient accessibles par tous les programmes.