Université Joseph Fourier

GRENOBLE

UFR IM²AG

Outils Info
Tp4 : Simplified Wrapper and Interface Generator

LEFEVRE Henry - M2 MIA
SEGURET Aymeric - M2 MIA

Rapport de Travaux Pratiques

2.1 Syntax of SWIG's input

Le but est de réussir à utiliser une librairie écrite en C dans un programme python. Pour cela il faut réussir à interfacer les deux langages.

La première étape est donc d'écrire le fichier interface qui représente la libraire C que l'on voudra utiliser. C'est un fichier avec une extension .i qui se présente comme ci-dessous :

```
%module mymodule

%{
/* part A */
#include "myheader.h"
%}

/* part B */
/* declaration in myheader.h */
int foo; /* global variable */
int bar(int x); /* function */
```

La première partie (marquée Part A dans l'exemple ci-dessus) du fichier correspond aux headers de la librairie à utiliser.
La deuxième partie (marquée Part B dans l'exemple ci-dessus) correspond aux fonctions de la librairie qui sont définies et qui peuvent donc être utilisées.

2.2 Elementary Types and Functions

L'interface de la libraire tp4 est écrite dans le fichier *« main.i »*. Celle-ci définit un module appelé « mymodule » dans lequel on référence le header *'tp4.h'*. Dans ce header, seules les fonctions de la première partie de la librairie seront interfacées par swig.

Le fichier *'main.i'* est présenté ci-dessous :

```
%module mymodule
%{
#include "/home/s/segureta/Documents/M2-S1/Outils_info/Tp4/libtp4/lib/tp4.h"
%}

#define VERSION "1.0.3"

extern const double PI;
double pi();
double add_pi(double v);
void set_log(int v);
void stats();
```

Avant d'écrire le CMakeList, nous avons testé les opérations à effectuer à l'aide d'un terminal.

(1) swig -python main.i
(2) gcc -fpic -c main_wrap.c -I /usr/include/python2.7
(3) gcc -shared main_wrap.o ../libtp4/lib/tp4.o -o _mymodule.so

En considérant bien sûr que le fichier tp4.o a été généré au préalable.

La commande (1) permet ainsi de produire un fichier *'main_wrap.c'* qui sera compilé dans un module python (src : http://www.swig.org/papers/PyTutorial98/PyTutorial98.pdf).
La commande (2) permet de linker la librairie python avec notre wrapper.
La commande (3) permet de mettre en commun (de partager) le binaire du wrapper et de la librairie tp4.o afin de permettre l'utilisation avec des programmes python.

C'est cette libraire partagée *'_mymodule.so'* qui nous permettra d'inclure le module *« mymodule »* dans du code python.

Afin de tester la librairie générée (mymodule.so) nous avons traduit le fichier « test1.c » en python dans le fichier « exemple1.py ».

Le CMakeList présenté par la suite ne fonctionne pas dans toutes les salles de l'IM2AG, cependant il fonctionne sur un ordinateur personnel il se peut donc qu'il y ait quelques soucis lors de son exécution.

```
4   #commande pour obliger a lancer le cmake dans un dossier build ou autre tant que cest different du dossier source
5   if(${CMAKE_SOURCE_DIR} STREQUAL ${CMAKE_BINARY_DIR})
6     message(FATAL_ERROR "In-source builds not allowed. Please make a new directory (called a build directory) and rur
7   endif()
```

Cette partie permet de rendre plus propre l'exécution du Cmake, en effet il est nécessaire de se trouver dans un dossier différent du dossier contenant les sources avant de pourvoir l'exécuter.

```
9   #importation de la librairie SWIG
10  FIND_PACKAGE(SWIG REQUIRED)
11  INCLUDE(${SWIG_USE_FILE})
12
13  #importation de la librairie Python
14  FIND_PACKAGE(PythonLibs)
15  INCLUDE_DIRECTORIES(${PYTHON_INCLUDE_PATH})
```

```
17  #importation de tcl et de la librairie tp4
18  set(TCL_INCLUDE_PATH "/usr/include/tcl")
19  set(TP4_INCLUDE_PATH ${PROJECT_SOURCE_DIR}/../libtp4/lib)
20  INCLUDE_DIRECTORIES(${TCL_INCLUDE_PATH} ${TP4_INCLUDE_PATH})
```

Ces deux parties du CMakeList permettent d'importer les différentes librairies nécessaires à la compilation. Dans un premier temps on importe les librairies python et swig avec des « find_package » et dans un second temps on définit le chemin avec des « set ».

```
28  #commande perso pour creer le wrapper
29  add_custom_command(OUTPUT main_wrap.c COMMAND ${SWIG_EXECUTABLE} ARGS -python ${PROJECT_SOURCE_DIR}/main.i)
```

Ici on génère le wrapper à l'aide d'une commande personnalisée. C'est cette commande qui ne passe pas sur toutes les machines, il sera donc peut être nécessaire de le générer à la main.

```
31  #commande pour la creation du .so à partir du wrapper et des librairies
32  SET_SOURCE_FILES_PROPERTIES(main.i PROPERTIES CC ON)
33  SET_SOURCE_FILES_PROPERTIES(main.i PROPERTIES SWIG_FLAGS "-includeall")
34  SWIG_ADD_MODULE(myModuleCmake python  ../src/main_wrap.c)
35  SWIG_LINK_LIBRARIES(myModuleCmake ${PYTHON_LIBRARIES})
```

Et enfin ces quelques lignes, qui permettent de finir la création de la libraire avec les sources, leur ajout au module et l'ajout de la librairie python.

```
1   #!/usr/bin/python
2
3   import mymodule
4
5   def print_part1():
6     print("VERSION="+str(mymodule.VERSION))
7     print("PI="+str(mymodule.PI))
8     print("pi()="+str(mymodule.pi()))
9     print("PI+5="+str(mymodule.add_pi(5)))
10    return
11
12  ########### MAIN ############
13  mymodule.stats()
14  print_part1()
15  mymodule.set_log(1);
16  print_part1();
17  mymodule.set_log(0);
18  mymodule.stats();
19
```

Exemple1.py

Le code présenté ci-dessus est un code équivalent au code test1.c, et permet de tester la librairie générée précédemment. Après test cela fonctionne correctement :

```
[lefevreh src 16:55]$ ./exemple1.py
Nombre de Vecteur créés : 0
Nombre de Vecteur détruits : 0
VERSION=1.0.3
PI=3.14
pi()=3.14
PI+5=8.14
VERSION=1.0.3
PI=3.14
        LOG: Invocation de pi()
pi()=3.14
        LOG: Invocation de add_pi(5)
        LOG: Invocation de pi()
PI+5=8.14
        LOG: Invocation de set_log(0)
Nombre de Vecteur créés : 0
Nombre de Vecteur détruits : 0
```

2.2 Structures, pointers and objects

Le ficher ‘main.i’ a été complété avec la deuxième partie de la libraire :

```
%module mymodule
%{
#include "/home/s/segureta/Documents/M2-S1/Outils_info/Tp4/libtp4/lib/tp4.h"
%}

#define VERSION "1.0.3"

extern const double PI;
double pi();
double add_pi(double v);
void set_log(int v);
void stats();

Vecteur* Vecteur_create(double a, double b, double c);
Vecteur* Vecteur_add(Vecteur *v1, Vecteur* v2);
double Vecteur_elem(Vecteur *v, int coord);
char* Vecteur_str(Vecteur *v);
void Vecteur_destroy(Vecteur *v);

```

La manipulation des vecteurs dans le programme « exemple .py » se fait par l’intermédiaire des fonctions définies dans la libraire tp4 :

```
#!/usr/bin/python

import mymodule

def f() :
    v1 = mymodule.Vecteur_create(1, 2, 3)
    v2 = mymodule.Vecteur_create(4, 5, 6)
    v = mymodule.Vecteur_add(v1, v2)
    mymodule.stats()
    val_v = mymodule.Vecteur_str(v)
    print("Vecteur v:" + val_v + "\n")
    mymodule.Vecteur_destroy(v2)
    mymodule.Vecteur_destroy(v1)
    mymodule.Vecteur_destroy(v)
    return v

########### MAIN ############
mymodule.stats()
vecteur = f()
mymodule.stats()
val_v = mymodule.Vecteur_str(vecteur)
# Erreur normalement a la ligne precedente : v a ete libere #
```

Et comme précédemment on exécute ce programme python afin de vérifier la librairie :

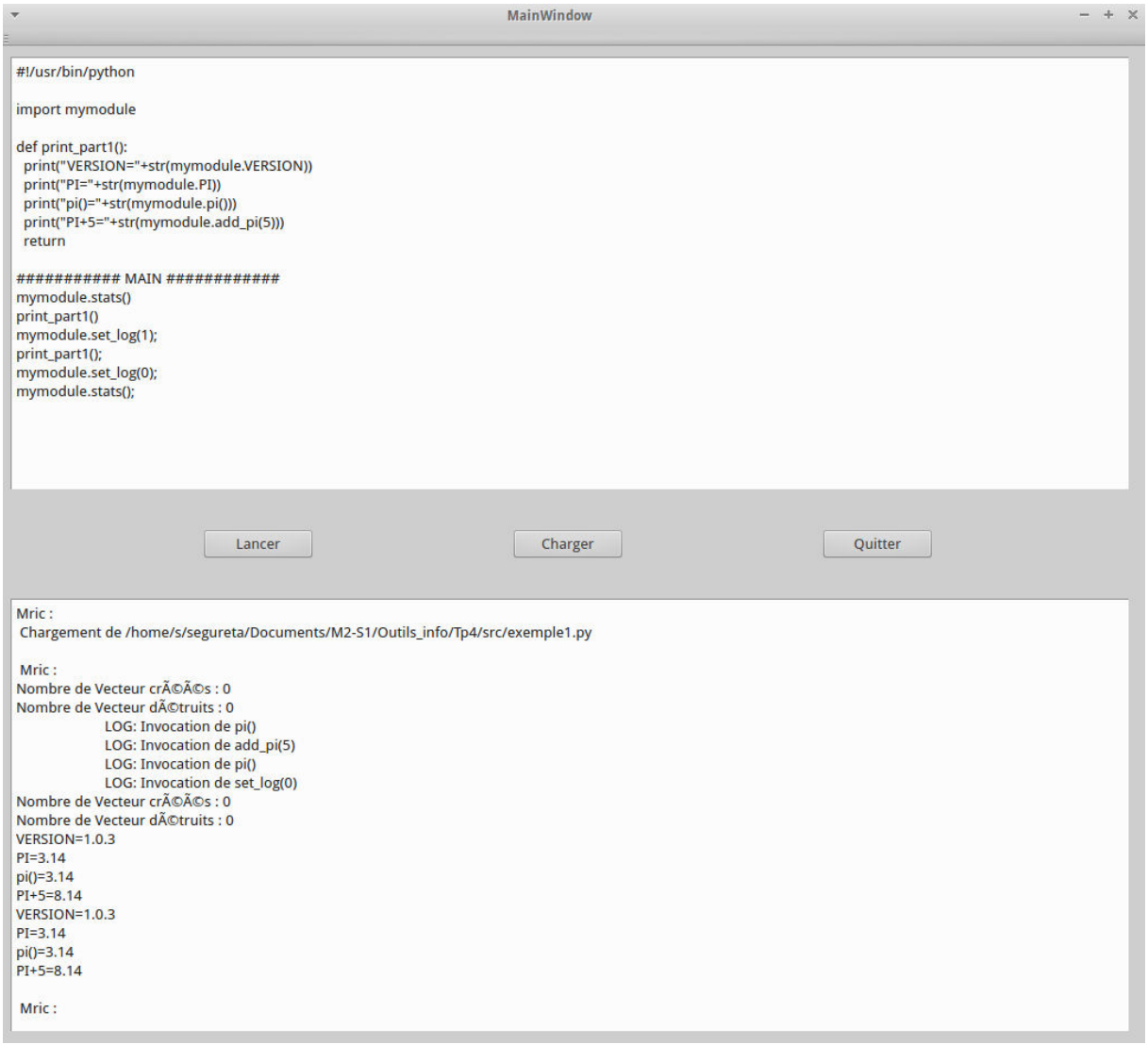
```
[lefevreh src 16:55]$ ./exemple2.py
Nombre de Vecteur créés : 0
Nombre de Vecteur détruits : 0
Nombre de Vecteur créés : 3
Nombre de Vecteur détruits : 0
Vecteur v:[5,7,9]

Nombre de Vecteur créés : 3
Nombre de Vecteur détruits : 3
python: tp4.c:64: check: Assertion `v->valid == 0xBEEF' failed.
Abandon (core dumped)
```

3 Graphical User Interface

Pour cette troisième partie, nous avons créé une petite interface graphique à l'aide de Qtcreator qui permet de charger un programme C, python ou bien simplement un fichier texte et d'afficher la sortie de son exécution (dans le cas d'un fichier python).

Voici un aperçu :



On note juste un petit problème d'affichage des accents mais surement un simple problème d'UTF8.