Lab 6: SWIG and Ruby

Learning Objectives:

1. What is SWIG?
2. Compatibility and Features of SWIG
3. Running SWIG

Supporting files:

1. SWIG_example.zip

---

1. What is SWIG?

SWIG (Simplified Wrapper and Interface Generator) is an interface compiler that connects programs that are originally written in C and C++ to scripting languages such as Perl, Python, Ruby, and Tcl. SWIG takes the declarations found in the header files of these programs written in C/C++ and uses them to generate the wrapper code that scripting languages need to access the underlying C/C++ code. There are several benefits of using SWIG, they are:

- **More powerful C/C++ programs:** SWIG enables the us to replace the main() function of a C program with a scripting interpreter from which we can control the application. This adds a lot of flexibility to the program as, the scripting interface allows users and developers to easily modify the behavior of the program without having to modify low-level C/C++ code. This can be beneficial for the large software packages that are used on daily basis.

- **Easy prototyping and debugging:** SWIG eases the testing and debugging of C/C++ programs that are placed in a scripting environment. For example, a library can be tested with a group of scripts only. Moreover, as SWIG does not perform any modifications to the underlying C/C++ code, it can be used even if the final product does not rely upon scripting.

- **Enables system integration:** SWIG assists in gluing loosely-coupled software components together.

- **Constructs scripting language extension modules:** SWIG can turn the common C/C++ libraries into components that can be used in scripting languages like Ruby, Python etc.

A detailed documentation on SWIG can be found here.

---

2. Compatibility and Features of SWIG

SWIG is compatible with Unix, 32 bit version of Windows, and Macintosh environments. As for programming language support, ever since SWIG 1.1 programming languages such as Tcl, Python, Perl, and Guile were supported. From SWIG 1.3.6 onwards, SWIG also supports Java, Ruby, and Mzscheme. Currently SWIG supports twenty three different programming languages, as listed here.
SWIG provides a full C preprocessor and control over most aspects of wrapper generation. A detailed documentation of the same can be found here.

---

3. Running SWIG

For running SWIG, the first step is to download and install the software from here.

Next, we start with a simple C++ code as follows:

```
#include "mylib.h"
void foo()
{
   cout << "Hello World!" << endl;
}
int sum(int a, int b)
{
   return a+b;
}
```

An example header file for the above C++ program can be as follows:

```
#include < iostream>
using namespace std;
void foo();
int  sum(int a, int b);
```

Now we need to wrap our C++ library. Interface files are the input to SWIG. To tell SWIG to wrap the above function, we create the following interface file.

```
%module mylib
%{
#include "mylib.h"
%}
void   foo();
int    sum(int a, int b);
```

This tells SWIG to create a Ruby module called *mylib*, and all the functions listed will be wrapped. To run swig against the interface file, type:

```
$ swig -ruby example.i
```

In the above piece of code, the example.i is the corresponding interface file for the C code written in example.c file. In this case, to build a C++ extension, add the -c++ option:

```
$ swig -c++ -ruby mylib.i
```

Later, run the following code in order to generate a makefile:

```
$ ruby gen_makefile.rb
```

Then a `Makefile` will be generated in the present directory. Run `make` to build `mylib.so` which is a shared library, and can be used from Ruby scripts.

Now, let us create a Ruby program in the same directory in order to test the generated wrapper. The test Ruby program can be written as follows:

```
require_relative 'mylib'

# void method call
Mylib::foo

# method that takes arguments and returns integer
sum = Mylib::add(30, 12)
print sum
```

Let us name our test file as swig_test.rb. Once run, the test file will produce the following output:

```
Hello World!
42
```

Here, ‘mylib’ is used as a Ruby module that is generated from the C++ code mentioned in the previous page. The examples discussed in this tutorial are attached as the supporting files.
