Programming in Modern C++

Tutorial T06: Mixing C and C++ Code: Part 2: Project Example

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

All url's in this module have been accessed in September, 2021 and found to be functional

Tutorial Recap

- We have learnt why is it often necessary to mix C and C++ codes in the same project
- We have explored the basic issues of mixing and learnt the ground rules
- In addition to the rules, we have three mechanisms to ease code mixing
  - Use `extern "C"` in C++ for all functions to be called from both C and C++
  - Guard `extern "C"` with `__cplusplus` guard for use with C
  - Provide wrappers for C++ data members, member functions, and overloaded functions for use with C

Tutorial Objectives

- Walk through a C / C++ mix project using the rules and scenarios of mixing

Tutorial Outline

Mixing C and C++ Codes

Mixing C and C++ Codes

Source:Accessed 16-Sep-21
How to mix C and C++, ISO CPP
Mixing C and C++ Code in the Same Program, Oracle
C++ Core Guidelines: Mixing C with C++
Mixing Code in C, C++, and FORTRAN on Unix

Mixing C and C++ Codes: Rules: Recap

- **RULE 1**: **Use C++ compiler when compiling `main()`**
- **RULE 2**: **C and C++ compilers must be compatible**
- **RULE 3**: **C++ compiler should direct the linking process**

Mixing C and C++ Codes: Scenarios: Recap

- **How do I call a C function from C++?**

```
extern "C" { void f(int); };
```

- **How do I call a C++ function from C?**
  - Non-Member

```
extern "C" { void f(int); }
```

  - Member

```
class C { /*...*/
    virtual double f(int);
};
// wrapper function
extern "C" double call_C_f(C* p, int i)
{ return p->f(i); }
```

  - Overloaded

```
void f(int);
void f(double);
// wrapper functions
extern "C" {
    void f_i(int i) { f(i); }
    void f_d(double d) { f(d); }
}
```

- **How do I include a C Header File?**
  - System / Standard Library Headers
  - Non-System Headers: Editable

```
#ifdef __cplusplus /* C compilers skip */
extern "C" {
#endif
/* Original Code of the Header */
#ifdef __cplusplus
}
#endif
```

  - Non-System Headers: Non-Editable

```
// In C++ header / source
extern "C" {
    #include "my-C-code.h" // C Header
}
```

- **How do I use Pointers to C / C++ Functions?**

```
extern "C" {
    typedef int (*pfun)(int);
    void foo(pfun);
    int g(int); // foo(g) is valid
}
```

C / C++ Mixed Project

C / C++ Mixed Project

C / C++ Mixed Project: Manipulating with Objects from C

- We present an example project comprising the following files to summarize various code mixing scenarios in an integrated manner:
  - `Data.h`: C/C++ common header containing:
    - ▷ definition of `class Data`
    - ▷ prototypes of C functions to interact with `class Data`, and
    - ▷ prototypes of C++ wrappers providing access points for C to call member functions in `class Data`
  - `Data.cpp`: Implementations of `class Data`
  - `Data_Wrap.cpp`: Implementations of C++ wrapper functions for `class Data`
  - `App.c`: Implementations of C functions for interacting with `class Data`
  - `main.cpp`: `main` to invoke the C functions
  - `makefile`: Mix build of C and C++, and C++ link script

C / C++ Mix Project: Mix Scenarios

- Calling C functions from C++ (main.cpp)
  - `Data* c_create_object(int);` /* C function to create an object */
  - `void c_access_object(Data*);` /* C function to access an object */
  - `void c_release_object(Data*);` /* C function to release an object */
- Calling C++ functions from C (App.c)
  - `Data* call_create(int);` /* C++ wrapper to create an object by new */
  - `int call_get(Data*);` /* C++ wrapper to get the state of an object by get */
  - `void call_set(Data*, int);` /* C++ wrapper to change the state of an object by set */
  - `void call_release(Data*);` /* C++ wrapper to release an object by delete */
- Passing an object from C to C++ (main.cpp)
  - `Data* c_create_object(int);` /* C function to create an object */
- Passing an object from C++ to C (main.cpp)
  - `void c_access_object(Data*);` /* C function to access an object */
  - `void c_release_object(Data*);` /* C function to release an object */
- C++ wrappers for object creations, get / set, and release (Data_Wrap.cpp)
- C functions for object creations, get / set, and release (App.c)
- Common header for C and C++ (Data.h)
  - `typedef struct Data Data;` /* Incomplete Type to access Data* in C function */
-

C / C++ Mix Project: Data.h

```
/* C Header and C++ Header Data.h - can be read by both C and C++ compilers */
#ifndef __DATA_H    /* include Guard */
#define __DATA_H
#ifdef __cplusplus  /* Guard for C++ */
    class Data { int d_;                            // Private data member
    public: Data(int=0); ~Data();                   // Public members: Constructor and Destructor
        int get(); void set(int);                   // get and set members
    };
#else               /* Guard for C */
    typedef struct Data Data;                       /* Incomplete Type to access Data* in C function */
#endif
#ifdef __cplusplus /* Guard for C++ */
extern "C" {       /* Linkage for C */
#endif
    extern Data* c_create_object(int);              /* C function to create an object */
    extern void c_access_object(Data*);             /* C function to access an object */
    extern void c_release_object(Data*);            /* C function to release an object */
    extern Data* call_create(int);                  /* C++ wrapper to create an object by new */
    extern int call_get(Data* data);                /* C++ wrapper to get state of an object by get */
    extern void call_set(Data* data, int d);        /* C++ wrapper to change state of an object by set */
    extern void call_release(Data*);                /* C++ wrapper to release an object by delete */
#ifdef __cplusplus /* Guard for C++ */
}
#endif
#endif /* __DATA_H */
```

C / C++ Mix Project: Data.cpp

```
// C++ code:  Data.cpp
#include <iostream>
using namespace std;
#include "Data.h"

// Class Data implementation
Data::Data(int d): d_(d)
    { cout << "Created " << d_ << endl; }

Data::~Data()
    { cout << "Released " << d_ << endl; }

int Data::get()
    { return d_; }

void Data::set(int d)
    { d_ = d; }
```

C / C++ Mix Project: Data_Wrap.cpp

```
// C++ code:  Data_Wrap.cpp
#include "Data.h"

/* C++ wrapper to create an object by new */
Data* call_create(int d)
    { return new Data(d); }

/* C++ wrapper to get state of an object by get */
int call_get(Data* data)
    { return data->get(); }

/* C++ wrapper to change state of an object by set */
void call_set(Data* data, int d)
    { return data->set(d); }

/* C++ wrapper to release an object by delete */
void call_release(Data* data)
    { delete data; }
```

C / C++ Mix Project: App.c

```
/* C code */:  App.c
#include <stdio.h>
#include "Data.h"

Data* c_create_object(int d) {                /* C function to create an object */
    return call_create(d);
}

void c_access_object(Data* data) {            /* C function to access an object */
    printf("Get data %d\n", call_get(data));
    call_set(data, 7);
    printf("Set data %d\n", call_get(data));
}

void c_release_object(Data* data) {           /* C function to release an object */
    call_release(data);
}
```

C / C++ Mix Project: main.cpp

```
// C++ code:  main.cpp
#include <iostream>
using namespace std;

#include "Data.h"

Data d(10);

int main() {
    Data* p = c_create_object(5);    /* C function to create an object */
    c_access_object(p);              /* C function to access an object */
    c_release_object(p);             /* C function to release an object */
}
```

C / C++ Mix Project: makefile

```
# Compiles .c by C and .cpp by C++. Links by C
CC=gcc
# Compiles .c and .cpp by C++. Links by C++
CPP=g++
CFLAGS=-I.
DEPS = Data.h

# Build .c by gcc (C Rules)
%.o: %.c $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)

# Build .cpp by gcc (C++ Rules). May use $(CPP)$ for g++ also
%.o: %.cpp $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)

# Link by g++ (C++ Linkage)
Data: main.o Data.o App.o Data_Wrap.o
    $(CPP) -o Data main.o Data.o App.o Data_Wrap.o

.PHONY: clean

clean:
    del *.o *.exe
```

C / C++ Mix Project: Execution

```
// Build by make
$ make
gcc -c -o main.o main.cpp -I.                             // C++ Compile
gcc -c -o Data.o Data.cpp -I.                             // C++ Compile
gcc -c -o App.o App.c -I.                                 // C Compile
gcc -c -o Data_Wrap.o Data_Wrap.cpp -I.                   // C++ Compile
g++ -o Data main.o Data.o App.o Data_Wrap.o               // C++ Link

// Execute
$ Data.exe
Created 10
Created 5
Get data 5
Set data 7
Released 7
Released 10
```

C / C++ Mix Project: Call Trace

```
// Trace
START()                                   // Special start-up function to initialize static objects in C++
    Data::Data(int)                       // C++ constructor for class Data
                                          // Start of  main()
    main()                                // C++ main() function
        c_create_object(int)              // C application function
            call_create(int)              // C++ wrapper
                new Data(int)             // C++ dynamic allocator
                    Data::Data(int)       // C++ constructor for class Data
        c_access_object(Data*)            // C application function
            printf(const char*, ...)      // C library function
                call_get(Data*)           // C++ wrapper
                    Data::get()           // C++ member function for class Data
            call_set(Data*, int)          // C++ wrapper
                Data::set(int)            // C++ member function for class Data
            printf(const char*, ...)      // C library function
                call_get(Data*)           // C++ wrapper
                    Data::get()           // C++ member function for class Data
        c_release_object(Data*)           // C application function
            call_release(Data*)           // C++ wrapper
                delete(Data*)             // C++ dynamic de-allocator
                    ~Data::Data()         // C++ destructor for class Data
                                          // End of  main()
    ~Data::Data()                         // C++ destructor for class Data
```

Advanced Code Mix Scenarios

Advanced Code Mix Scenarios and Advisory

Advanced Code Mix Scenarios and Advisory

- The common code mix scenarios as described:
  - Are *simple to code* and *easy to debug*
  - Covers *most situations* in several projects
  - Are *stable*, *portable*, and *recommended*
- Beyond this, however, several other scenarios may need resolution from time to time:
  - Using **exceptions** in C++ code
    - ▷ C++ exception mechanism and rules about *destroying objects that go out of scope* are likely to be *violated* by a C `long_jmp`, with *unpredictable results*
    - ▷ **ADVISORY: Do not to use `long_jmp` in programs that contain C++ code**
    - ▷ *Stack unwinding* while the control passes through a C function, is *not portable or stable*
    - ▷ **ADVISORY: Avoid exception path going through any C function**
  - Manipulating objects in a **polymorphic hierarchy** should be carefully handled as
    - ▷ *C does not support dynamic dispatch*
    - ▷ **ADVISORY: Use appropriate C++ wrappers to avoid getting into C's `if-else` type switch**
  - Directly **accessing data members** in classes from C. This can be really tricky because
    - ▷ *Layout of objects in a hierarchy is not portable*
    - ▷ **ADVISORY: Access data members only through appropriately designed C++ wrappers**
  - Several project / software specific scenarios

Tutorial Summary

- We have learnt why is it often necessary to mix C and C++ codes in the same project
- We have explored the basic issues of mixing and learnt the ground rules
- In addition to the rules, we have four mechanisms to ease code mixing
  - Use `extern "C"` in C++ for all functions to be called from both C and C++
  - Guard `extern "C"` with `__cplusplus` guard for use with C
  - Provide wrappers for C++ data members, member functions, and overloaded functions for use with C
  - Incomplete struct type (with the same name as a C++ class) to allow pointers of C++ UDT objects in C
- We have also noted a few advanced mix scenarios and learnt the advisory of do's and don'ts