Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

Programming in Modern C++

Tutorial T05: Mixing C and C++ Code: Part 1: Issues and Resolutions

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 Objectives

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- Due to legacy, reuse and several business compulsions, we often need to mix C and C++ codes in the same project
- So we need to learn how to write programs mixing C and C++?

Tutorial Outline

Tutorial T05

Partha Pratim Das

Objectives & Outline

Mixing C & C++

Why Mix C/C++?

Build all in C++

Mix C & C++

Static Initialization

Compiler Compatibility

Linkage Issues

Exception Issues

Common Mix

C from C++

C++ from C

Pointers to Functions

C Header File

System

Non-System

Tutorial Summary

1. Mixing C and C++ Codes
   - Why Mix C/C++?
   - Build all in C++
   - Mix C and C++
     - Static Initialization
     - Compiler Compatibility
     - Linkage Issues
     - Exception Issues
2. Common Code Mix Scenarios
   - How do I call a C function from C++?
   - How do I call a C++ function from C?
   - How do I use Pointers to C / C++ Functions?
   - How do I include a C Header File?
     - System Header File
     - Non-System Header File
3. Tutorial Summary

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

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

Why do we need to mix C and C++ Codes?

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- Primary reason is **legacy and reuse**. There are possibly trillion lines of well-tested C code available. So reusing them in and / or with C++ needs mixing of codes
- Mixing of codes is actually often needed not only across C and C++, it may be used across C and Python, C# and Python, C++ and Java, and so on to get the **best of both languages** (C/C++ is efficient, C if lightweight for embedded programming, Python has rich libraries and good for web, Java is good for applications with GUI etc.) and be able **to use the available proven libraries**. Actually, we may mix *more than two languages*
- Here are some informative articles on projects using multiple languages:
  - Polyglot programming – development in multiple languages, Computer World, 2009
  - How do you use different coding languages in one program?, Quora, 2015 and several other in Ouora
  - A Large Scale Study of Multiple Programming Languages and Code Quality, IEEE, 2016
  - On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers, Springer, 2017
- **We restrict the discussions here on mixing C and C++ only**

Why do we need to mix C and C++ Codes?

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- **There are two typical situations**:
  - *Both C header and C source files are available and editable*: An existing project that needs to be migrated to C++ in full or parts (needs to be reused). Two options:
    - **Mix C and C++ codes**: Compile C code with C compiler, and C++ code with C++ compiler and then link by C++ (**how?**)
    - **Build all in C++**: Build both C and C++ codes with C++ compiler and link
  - *Only C header files are available*: For third party library where source is already pre-compiled. In fact, *the header file too may not be editable*. For example,
    - To write a C++ program/library that does scientific calculations, we would possibly use GSL (GNU Scientific Library), and it is written in C
    - The C game codes called from the C++ graphics engine

    We would like to wrap all the C functions to use in nice C++ style functions, perhaps with the necessary exceptions and returning a `std::string` instead of having to pass a `char*` buffer as argument. Now we have only one option:
    - **Mix C and C++ codes**: Compile C code with C compiler, and C++ code with C++ compiler and then link by C++ (**how?**)

Build C and C++ Codes as C++

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- **Build as C++**: In a mixed code projct where all header and source files are *available and editable*, we can compile all the code (even the C-style code) using a C++ compiler. For example, using `g++` for both `.c` and `.cpp` files. That eliminates the need to mix C and C++
- However, it is not easy to build the C code by C++ compiler unless the C code strictly uses the common subset of C and C++ (Check the Tutorial on *Compatibility of C and C++* for details). For example, consider the simple C program below where difference of behavior in C and C++ compilers are marked in different colors:

```
/* cStyle.c */
#include <stdio.h>
int main() {
    double sq2=sqrt(2); // (1): math.h missing. Warning in C89. Error in C++98
    printf("sizeof('a'): %d",
        sizeof('a'));   // (2): 'a' is int in C, char in C++. Outputs 4 in C89. Outputs 1 in C++98
    char c;
    void* pv = &c;
    int* pi = pv;       // (3): Implicit conversion from void* to int*. Okay in C89. Error in C++98
    int class = 5;      // (4): class is a reserved word in C++. Okay in C89. Error in C++98
}
```

- So the **C code needs to be ported for C++**

Build C and C++ Codes as C++

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- **Build as C++**: While building a C/C++ project (both C and C++ codes) with C++ is preferable from language perspective it has a number of shortcomings from engineering viewpoint
  - The *C-style code may need porting* as the *C++ compiler is more strict* - as we have seen in the example
  - *Porting may involve substantial cost* in terms of developer effort as well as project time. This may *not be affordable from the business perspectives*
  - With porting we also need to *create new testplan for C++*, perform *extensive testing* to match the regression. This involves further cost in terms of tester effort as well as project time. This may *not be affordable from the business perspectives*
  - If we are porting a stable C code, even after regression clean and testing, it is *likely to break some of the existing functionality* in the software or even in customer's code if the C code is part of a library provided to the customer. This too may *not be affordable from the business perspectives*
  - So building as C++ is *feasible only in some select situations* though it is *preferred*

Mixing C and C++ Codes: Issues and Remedies

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- When we mix codes, that is, compile C code by C Compiler and C++ code by C++ Compiler - all into `.o` files - we expect to be free from language-specific issues in *individual translation units*
- However, issues arise as we work with different versions of compilers, link the `.o` files of translation units, and as control flows across units during execution, and so on:
  - *Static Initialization Issues*: While compiling `main()`, static initialization is handled differently in C and C++
  - *Compiler Incompatibility Issues*: The compilers may have incompatibility in calling conventions, definitions of basic types (like `int`, pointer), runtime library, etc.
  - *C Library Incompatibility Issues*: If the C++ compiler provides its own versions of the C headers, the headers used by the C compiler must be compatible
  - *Linkage Issues*: C and C++ linkage conventions differ
  - *Exception Issues*: C and C++ use drastically different exception models
  - *Scope of* `struct` *Issue*: Scoping of *nested* `struct` differ between C and C++. Check the Tutorial on *Compatibility of C and C++* for details

Mixing C and C++ Codes: Issues and Remedies

- **Static Initialization Issues**: In C and C++ both the `static` variables are constructed and initialized before `main()`. But they have different semantics and handling for `static`
  - In C, a `static` initializer must be a constant
  - In C++, a `static` variable must be constructed – its constructor must get called

  So the following code compiles in C++, but fails in C

```
#include <stdio.h>
int init(void) { return 10; }

static int i = init(); /* Error in C: initializer element is not constant.  Okay in C++ */

int main() { printf("i = %d", i); }
```

  Hence,

  - C++ compiler generates an additional Start function, where all *global function calls* (including constructors) are executed before `main` starts
  - C compiler does not generate such Start function, `main` starts as soon as it is loaded

  So,

  - **RULE 1: Use C++ compiler when compiling `main()`**

Mixing C and C++ Codes: Issues and Remedies

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- **Compiler and C Library Incompatibility Issues**: To allevite the problems outlined, we should
  - Use compilers (preferably) from the same vendor (say, `gcc`)
  - Have same / compatible versions (for example, use the same calling conventions, define basic types such as `int`, `float` or pointer in the same way)
  - C runtime library must also be compatible with the C++ compiler
  - If the C++ compiler provides its own versions of the C headers, the versions of those headers used by the C compiler must be compatible

  So,
  - **RULE 2: C and C++ compilers must be compatible**

Mixing C and C++ Codes: Issues and Remedies

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- **Linkage Issues**: C and C++ linkage conventions differ
  - In C
    - Every function is in global scope
    - Every function name is unique

    Hence the C function name is used by the linker to identify the function across units
  - In C++
    - A function may be in multiple scopes
      - Global Scope
      - Non-Static Member Function in class Scope
      - Static Member Function in class Scope
      - namespace Scope
    - A function may be overloaded in any of these scopes

    Hence the C++ function name is not unique. The linker, therefore, uses the combination of the C++ function name and signature (parameter types) to create unique identity, called **mangled name**

Mixing C and C++ Codes: Issues and Remedies

Tutorial T05

Partha Pratim Das

Objectives & Outline

Mixing C & C++

Why Mix C/C++?

Build all in C++

Mix C & C++

Static Initialization

Compiler Compatibility

Linkage Issues

Exception Issues

Common Mix

C from C++

C++ from C

Pointers to Functions

C Header File

System

Non-System

Tutorial Summary

- **Linkage Issues**: Consider C++ code with 2 overloads of `print()` in multiple scopes

```
#include <iostream>
using namespace std;
void print(int) { cout << "int" << endl; } // Global
void print(double) { cout << "double" << endl; }
class MyClass { int i; double d; const char *pc; public:
    MyClass(int _i = 1, double _d = 1.1, const char *_pc = "Hello") : i(_i), d(_d), pc(_pc) { }
    void print(int) { cout << "MyClass int " << i << endl; } // Class member
    void print(double) { cout << "MyClass double " << d << endl; }
};
class MyOtherClass { public:
    static void print(int) { cout << "MyOtherClass int " << endl; } // Class static member
    static void print(double) { cout << "MyOtherClass double " << endl; }
};
namespace MySpace {
    void print(int) { cout << "MySpace int" << endl; } // namespace member
    void print(double) { cout << "MySpace double" << endl; }
}
int main() { MyClass a;
    print(10); print(10.10);
    a.print(10); a.print(10.10);
    MyOtherClass::print(10); MyOtherClass::print(10.10);
    MySpace::print(10); MySpace::print(10.10);
}
```

Mixing C and C++ Codes: Issues and Remedies

- **Linkage Issues**: The mangled and un-mangled names of functions are:

Function	gcc 6.3.0	msvc 18.00	Mangled?
// Global: Overloaded			
print(int)	__Z5printi	?print@@YAXH@Z	Yes
print(double)	__Z5printd	?print@@YAXN@Z	Yes
// Class member: Overloaded			
MyClass::print(int)	__ZN7MyClass5printEi	?print@MyClass@@QAEXH@Z	Yes
MyClass::print(double)	__ZN7MyClass5printEd	?print@MyClass@@QAEXN@Z	Yes
// Class static member: Overloaded			
MyOtherClass::print(int)	__ZN12MyOtherClass5printEi	?print@MyOtherClass@@SAXH@Z	Yes
MyOtherClass::print(double)	__ZN12MyOtherClass5printEd	?print@MyOtherClass@@SAXN@Z	Yes
// namespace member: Overloaded			
MySpace::print(int)	__ZN7MySpace5printEi	?print@MySpace@@YAXH@Z	Yes
MySpace::print(double)	__ZN7MySpace5printEd	?print@MySpace@@YAXN@Z	Yes
// Global: Not Overloaded			
main()	_main	_main	No

Therefore C and C++ compilers need to handle the names differently. *As C compiler does not know about mangling, we need to tell the C++ compiler not to mangle the names in the C context*

Mixing C and C++ Codes: Issues and Remedies

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- **Linkage Issues**: The `extern "C"` linkage specifier can prevent the C++ compiler from mangling the names. Declaring a function within `extern "C"` in the code, we can call a C function from C++, or a C++ function from C. We may use `extern "C"`
  - for each function
    ```
    extern "C" void foo(int);
    ```
  - for each function in a scope
    ```
    extern "C" {
        void foo(int);
        double bar(double);
    };
    ```
  - or for the entire header file by using include guards
    ```
    #ifdef __cplusplus
    extern "C" {
    #endif
        void foo(int);
        double bar(double);
        ...
    #ifdef __cplusplus
    }
    #endif
    ```

  Note that the macro `__cplusplus` is defined when the C++ compiler is used. Hence, `extern "C"` is not exposed to C compiler (it does not recognize it)

Mixing C and C++ Codes: Issues and Remedies

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- **Linkage Issues**: As we have seen, using extern "C" within __cplusplus guard, we can make a header that works appropriately for C and C++ both. For an illustrative example, you may refer to the video: C Programming — Advance Topic And Mixing C with C++ Code, YouTube (Accessed 19-Sep-21)

  We now have the next rule:

  - **RULE 3: C++ compiler should direct the linking process (so it can get its special libraries)**

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- **Exception Issues**: C and C++ use different exception models
  - **Propagating Exceptions**
    - What happens if a *C++ function is called from a C function, and the C++ function throws an exception*? The C++ standard is somewhat vague this
    - In some systems like Oracle Developer Studio C++ this will work properly. In it *if a C function is active when a C++ exception is thrown, the C function is passed over in the process of handling the exception*
  - **Mixing Exceptions with `set_jmp` and `long_jmp`**
    - The C++ exception mechanism and C++ rules about *destroying objects that go out of scope* are likely to be *violated by a* `long_jmp`*, with unpredictable results*
    - Some C++ experts believe that `long_jmp` should not be integrated with exceptions, due to the difficulty of specifying exactly how it should behave.
    - If we must use `long_jmp` in C code that we are mixing with C++, we need to *ensure that a* `long_jmp` *does not cross over an active C++ function*

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

Common Code Mix Scenarios

Common Code Mix Scenarios

Tutorial T05

Partha Pratim Das

Objectives & Outline

Mixing C & C++

Why Mix C/C++?

Build all in C++

Mix C & C++

Static Initialization

Compiler Compatibility

Linkage Issues

Exception Issues

Common Mix

C from C++

C++ from C

Pointers to Functions

C Header File

System

Non-System

Tutorial Summary

- How do I call a C function from C++?
- How do I call a C++ function from C?
  - Non-Member
    - Global / namespace
    - static
  - Member
    - Non-static
      - Non-virtual
      - virtual
    - static
  - Overloaded

- How do I include a C Header File?
  - System / Standard Library Headers
  - Non-System / User-defined Headers
    - Editable Headers
    - Non-Editable Headers
- How do I use Pointers to C / C++ Functions?
- How do I manipulate with objects in a C / C++ mix project?

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

Tutorial T05

Partha Pratim Das

Objectives & Outline

Mixing C & C++

Why Mix C/C++?

Build all in C++

Mix C & C++

Static Initialization

Compiler Compatibility

Linkage Issues

Exception Issues

Common Mix

C from C++

C++ from C

Pointers to Functions

C Header File

System

Non-System

Tutorial Summary

- Declare the C function `extern "C"` (in the C++ code) and call it (from the C or C++ code):

```
// C++ code
extern "C" void f(int);    // one way
extern "C" {               // another way
    int g(double);
    double h();
};
void code(int i, double d) {
    f(i);
    int ii = g(d);
    double dd = h();
    // ...
}
```

```
/* C code: */
void f(int i) {
    /* ... */
}
int g(double d) {
    /* ... */
}
double h() {
    /* ... */
}

Note that C code does not need to be edited
So pre-compiled .o file may also be used
```


- Note that C++ type rules, not C rules, are used. So a function declared `extern "C"` cannot be called with the wrong number of arguments. For example:

```
// C++ code
void more_code(int i, double d) {
    double dd = h(i,d); // error: unexpected arguments
    // ...
}
```

How do I call a C++ function from C?: Non-Member and Member Functions

Tutorial T05

Partha Pratim Das

Objectives & Outline

Mixing C & C++

Why Mix C/C++?

Build all in C++

Mix C & C++

Static Initialization

Compiler Compatibility

Linkage Issues

Exception Issues

Common Mix

C from C++

C++ from C

Pointers to Functions

C Header File

System

Non-System

Tutorial Summary

- Declare the C+ `extern "C"` (in the C++ code) and call it (from the C or C++ code):

```
// C++ code
extern "C" void f(int);
void f(int i) {
    // ...
}
// This works only for non-member functions
```

```
/* C code: */
void f(int);
void cc(int i) {
    f(i);
    /* ... */
}
```

- To call member functions (including virtual functions) from C, a simple wrapper needs to be provided. For example:

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

```
/* C code: */
double call_C_f(struct C* p, int i);
void ccc(struct C* p, int i) {
    double d = call_C_f(p,i);
    /* ... */
}
```

How do I call a C++ function from C?: Overloaded Functions

Tutorial T05

Partha Pratim Das

Objectives & Outline

Mixing C & C++

Why Mix C/C++?

Build all in C++

Mix C & C++

Static Initialization

Compiler Compatibility

Linkage Issues

Exception Issues

Common Mix

C from C++

C++ from C

Pointers to Functions

C Header File

System

Non-System

Tutorial Summary

- To call overloaded functions from C, wrappers, with distinct names for the C code to use, must be provided. For example:

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

```
/* C code: */
void f_i(int);
void f_d(double);
void cccc(int i,double d) {
    f_i(i);
    f_d(d);
    /* ... */
}
```


Note that these techniques can be used to call a C++ library from C code even if it is not desirable or possible to modify the C++ headers

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

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- A pointer to a function must specify whether it points to a C function or to a C++ function:

```
typedef int (*pfun)(int);  // pfun points to a C++ function
extern "C" void foo(pfun); // foo is a C function taking a pointer to a C++ function
extern "C" int g(int)      // g is a C function
...
// Type Mismatch Error!
foo(g);                    // foo called with a pointer to g, a C function
```

- To match the linkage of a pointer-to-function with the functions to which it will point, all declarations in this example needs to be inside `extern "C"` brackets, ensuring that the types match

```
extern "C" {
    typedef int (*pfun)(int);
    void foo(pfun);
    int g(int);
}
...
foo(g); // Types Match. Now OK
```

How can I include a standard C header file in my C++ code?

- `#include`ing a standard header file such as `<cstdio>`, is nothing unusual. For example:

```
// C++ code

// Nothing unusual in #include line
#include <cstdio>
int main() {
    // Nothing unusual in the call either
    std::printf("Hello world");
    // ...
}
```

```
/* C code */
/* Compiled using a C++ compiler */
/* Nothing unusual in #include line */
#include <stdio.h>
int main() {
    /* Nothing unusual in the call either */
    printf("Hello world");
    // ...
}
```


  - The `std::` part of the `std::printf()` is for the `std` `namespace` where C++ Standard Library components like `<cstdio>` belongs
  - So a C code using the C++ compiler can still use just `printf()` from `<stdio.h>` as C Standard Library belongs to global
  - Care is needed if there is difference between the C header and its corresponding C++ version

How can I include a standard C header file in my C++ code?

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
Tutorial Summary

- To summarize

	C Header	C++ Header
C Program	Use `.h`. Example: `#include <stdio.h>` *Names in global namespace*	Not applicable
C++ Program	Prefix `c`, no `.h`. Example: `#include <cstdio>` *Names in `std` namespace*	No `.h`. Example: `#include <iostream>`

- A C std. library header is used in C++ with prefix 'c' and without the `.h`. These are in `std` namespace:

```
#include <cmath>  // In C it is <math.h>
...
std::sqrt(5.0);   // Use with std::
```

It is possible that a C++ program include a C header as in C. Like:

```
#include <math.h> // Not in std namespace
...
sqrt(5.0);        // Use without std::
```

This, however, is not preferred

- **Using `.h` with C++ header files, like `iostream.h`, is disastrous. These are deprecated. It is dangerous, yet true, that some compilers do not error out on such use. Exercise caution.**

How can I include a non-system C header file in my C++ code?: Editable Case

Tutorial T05

Partha Pratim Das

Objectives & Outline

Mixing C & C++

Why Mix C/C++?

Build all in C++

Mix C & C++

Static Initialization

Compiler Compatibility

Linkage Issues

Exception Issues

Common Mix

C from C++

C++ from C

Pointers to Functions

C Header File

System

Non-System

Tutorial Summary

- *If the C header is editable*, add the `extern "C" {...}` logic inside the header and guard it by `__cplusplus` to let C compiler to ignore it

```
#ifdef __cplusplus /* C++ compiler notes, C compiler ignores */
extern "C" {
#endif

void f(int i, char c, float x) /* Original Code of the Header */

#ifdef __cplusplus
}
#endif
```

Now the C header may be #included without `extern "C"` in the C++ code:

```
// C++ code
// Get declaration for f(int i, char c, float x)
#include "my-C-code.h" // Note: nothing unusual in #include line
int main() {
    f(7, 'x', 3.14); // Note: nothing unusual in the call
    // ...
}
```

How can I include a non-system C header file in my C++ code?: Non-Editable Case

Tutorial T05

Partha Pratim Das

Objectives & Outline

Mixing C & C++

Why Mix C/C++?

Build all in C++

Mix C & C++

Static Initialization

Compiler Compatibility

Linkage Issues

Exception Issues

Common Mix

C from C++

C++ from C

Pointers to Functions

C Header File

System

Non-System

Tutorial Summary

- *If the C header is* **not** *editable*, the #include line must be wrapped in an extern "C" { /*...*/ } construct. This tells the C++ compiler that the functions declared in the header file are C functions

```
// C++ code
extern "C" {
    // Get declaration for f(int i, char c, float x)
    #include "my-C-code.h"
}

int main() {
    f(7, 'x', 3.14); // Note: nothing unusual in the call
    // ...
}
```

Tutorial Summary

Tutorial T05
Partha Pratim Das
Objectives & Outline
Mixing C & C++
Why Mix C/C++?
Build all in C++
Mix C & C++
Static Initialization
Compiler Compatibility
Linkage Issues
Exception Issues
Common Mix
C from C++
C++ from C
Pointers to Functions
C Header File
System
Non-System
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 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