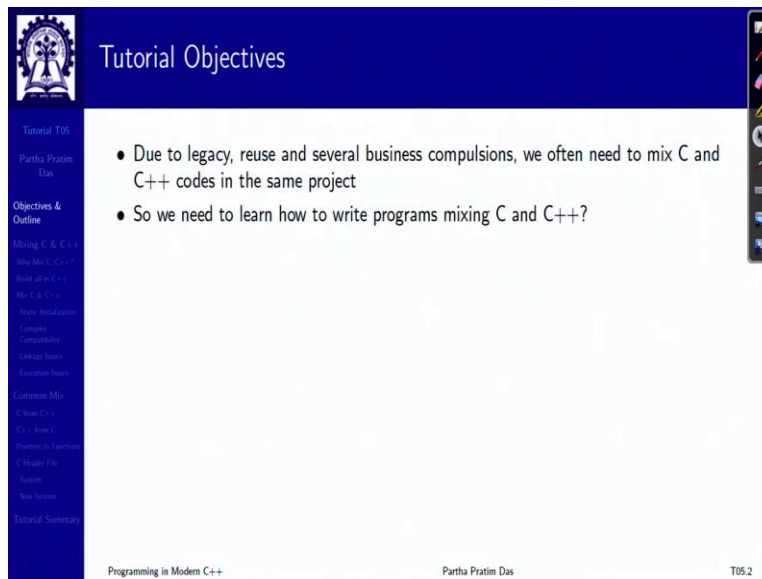


**Programming in Modern C++**  
**Professor Partha Pratim Das**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology Kharagpur**  
**Tutorial 05**  
**Mixing C and C++ Code Part: 1: Issues and Resolutions**

Welcome to Programming in Modern C++. This is another two-part series of tutorial to give you an idea about how to mix C and C++ ports.

(Refer Slide Time: 00:47)

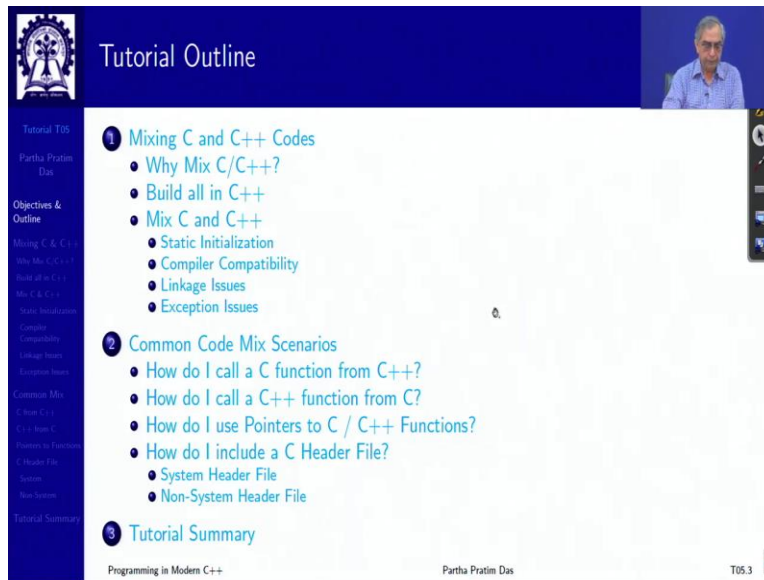


The screenshot shows a presentation slide with a dark blue header and a white content area. The header contains the text 'Tutorial Objectives' and a small logo on the left. The content area lists two bullet points: '• Due to legacy, reuse and several business compulsions, we often need to mix C and C++ codes in the same project' and '• So we need to learn how to write programs mixing C and C++?'. A vertical navigation menu is visible on the left side of the slide, and a toolbar is on the right. The footer of the slide includes 'Programming in Modern C++', 'Partha Pratim Das', and 'T05.2'.

Due to legacy and reuse and several business compulsions, we often need to mix C and C++ codes in the same project. So, what we are primarily studying in the course modules is either how you do a C project or how you do a C++ project. But in reality, you may need to do a mixed language project.

So, in this tutorial and the next, we are going to discuss what are the issues of mixing C and C++ programCs or C++ codes in the same project and what are the ways to handle that what are the resolutions and we will also in the next tutorial work out a detailed example for your complete understanding.

(Refer Slide Time: 01:50)



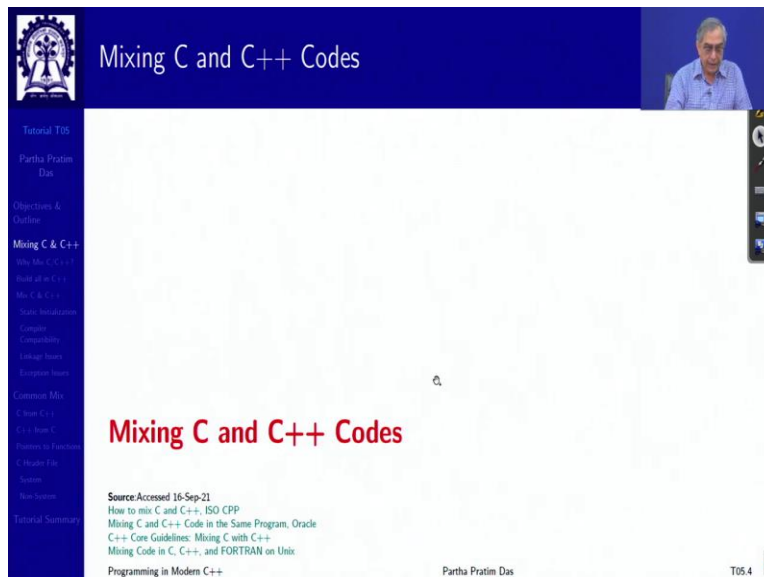
**Tutorial Outline**

- 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**

Programming in Modern C++ Partha Pratim Das T05.3

So, this is the outline will be available on the left as you know.

(Refer Slide Time: 01:56)



**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

Programming in Modern C++ Partha Pratim Das T05.4

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

- 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 is 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 Quora
  - 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**

All links Accessed on 21-Sep-21  
Programming in Modern C++  
Partha Pratim Das  
T05.5

So, first let us look at what are the different aspects of mixing C and C++ code starting with why at all we need to mix. Now, the primary reason for mixing C and C++ codes is legacy and reuse. As you know that C has been around from much earlier than C++ and also till date C is the most widely used language in the community.

So, there are there is no exact known quantum of how much of well tested C codes exist. But it is estimated to be over a trillion lines, trillion you understand million, thousand million is a billion thousand billion is a trillion. So, reusing them and or with C++ needs mixing of code because certainly if there is a well tested C library, I would not like to reimplement that if it serves my purpose.

So, and it is also that mixing of C and C++ is not a specific issue between these two languages because they are siblings. But it may be used across various pairs of languages like C and Python or say C# and Python or C++ and Java and so on to harvest the best features best benefits of both languages like we know C, C++ are primarily known for their efficiency.

In addition, C is lightweight, relatively and easy to use in embedded programming. We have discussed these things in the main course module. Python has reached libraries, it is good for the web, Java is good for applications with GUI and so on. So we should, in general, the community of programmers try to mix languages if it is giving a specific benefit, and use those proven libraries.

And actually in several projects, we may mix more than two languages as well. Now, there are obviously several articles on projects using multiple languages because I am sure when you hear from me that well, this kind of multi language programming often referred to as polyglot programming, are people really doing it? So, here are some references where you can see the variety of small, min, medium and large projects which use the mixes. But we will restrict our discussion only to C and C++ mix.

(Refer Slide Time: 05:10)

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

- 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 engineWe 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?)

Programming in Modern C++ Partha Pratim Das T05.6

So, if we want to mix then there are typically two situations one in which both C header and the C source file are available and editable and the other where only C header file is available. Now, if we have an existing project that needs to be migrated to C++ in full or parts to be reused, there could be two options.

One is certainly we can mix the C and C++ code that is whatever we want to reuse from C, we take that compile it by C compiler and whatever we are using from C++, we use C++ compiler and then link by C++. I have left it with a question here as to why do we link by C++ not by C.

And the discussions in this tutorial will give you a comprehensive answer to that. The other option would be to build everything in C++, we have been taking C is a subset of C++, well, it is true and it is not true at the same time. And we can so, we can build both of them C and C++ code with C++ compiler that is say G++ and link. So, this is this is a typical options where we can have the header in the source file and are editable they are editable in C.

(Refer Slide Time: 06:51)

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

- 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 engineWe 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?**)

Programming in Modern C++ Partha Pratim Das T05.6

Now, if we have C header files only, not the source file, that is third party source is already pre-compiled available as an object binary. So, the third party just does not give us the implementation of the headers, it just gives us the header file. And this is typical what we will get to see in the standard library situation also you just get the header file, not the implementations in the source file.

And further, it may also happen that the header file itself may not be editable. If I mean, header file you need otherwise, you would not be able to use that code. But the vendor may mandate that you cannot edit the header file. And which is exactly the case for the standard library and could be for any, any third party library that you use or you want to build.

So for example, I mean and why I mean what are the situations very typical situations you need that, for example, you want to write a C++ program or library that does scientific calculations. I am not sure if you know, there is a GNU library called GSL, GNU Scientific Library, which is written in C and it is as a very rich library in terms of scientific computations, very rich set of functions.

Obviously, lot more than what your `math.h` has a lot more of complex functions, different types of defined data type structures in C and so on and well tested very efficient. So, if you are writing something for scientific calculation, it is always it is most often at least advisable that

you use this GSL instead of writing the code from scratch, and going through all the pains of development

There are a lot of C games, I mean game codes efficiently written in C which you want to use with the C++ graphic engine to make the whole thing work. So, there could be I mean, I just picked up to somewhat common examples, but there could be several situations where you would have that you want to use the C code in C++ project and you have only a header available and which by itself may not even be editable.

(Refer Slide Time: 09:31)

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

- **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 engineWe 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?**)

Programming in Modern C++ Partha Pratim Das T05.6

So, we would like to often wrap the C functions for use nicely in C++ type functions. C functions have certain limited capability. For example, they do not support exceptions I mean not as richly as C++ does. They do not support very common data types like string which you have in C++.

So, you might so the function if it has to return conceptually a string it will return C string that is `char*`, `const char*`, but you may want to have a wrapper which takes that `char*` and converts it into `std::string`. So, that it can very seamlessly work with the other C++ codes or the C++ function, the same thing can be said about the parameters that you pass. So, here the option only is to mix C and C++ codes.

(Refer Slide Time: 10:35)

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

- 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 engineWe 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?**)

Programming in Modern C++ Partha Pratim Das T05.6

So, as we can see from the two scenarios here, that in both scenarios, you can mix the code that is compiled C by C compiler, compiled C++ by C++ compiler and linked by C++. And in one case, you can have the option of building everything in C++ as well. So, let us see what are the what are the different perspectives under which we will choose which pathway to take, is it only based on availability and editability of header and source or there are other factors that are involved.

(Refer Slide Time: 11:08)

### Build C and C++ Codes as C++

- **Build as C++:** In a mixed code project 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++**

Programming in Modern C++ Partha Pratim Das T05.7

So, first let me focus on what if I built the entire thing as C++. So, naturally you built everything with C++, use G++ which compiles everything by C++ rule, and naturally links by C++ linking. So, that eliminates the need for mixing.

(Refer Slide Time: 11:48)

**Build C and C++ Codes as C++**

- **Build as C++:** In a mixed code project 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; // (3): Implicit conversion from void* to int*. Okay in C89. Error in C++98
    int* pi = pv;
    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++**

Programming in Modern C++ Partha Pratim Das T05.7

Now, the question the point is, however we might think that C is a subset of C++ and so on, it is often not easy to build a C code by a C++ compiler. Unless that C code strictly uses the common subset between C and C++. This concept is somewhat, maybe sound like a jerk to you, but that is true, there are certain features in C, which does not work in C++, that is the reality.

And we have a separate tutorial on compatibility of C and C++ languages where we talk about these and give you examples to expose you with what these issues are. So, consider a simple C program here. So, you have a call to square root I mean we just ignore the fact that math.h has not been included, that is we will need to be included this is just for the illustration.

Then you print the size of character a, you define some void pointer and do implicit casting put values and so on. So, some few this program by itself is not meaningful, it is just trying to illustrate the issues.



(Refer Slide Time: 13:00)

The slide is titled "Build C and C++ Codes as C++". It contains a list of bullet points and a code snippet with annotations. The first bullet point states that in a mixed code project, all header and source files can be compiled using a C++ compiler. The second bullet point explains that it's not easy to build C code with a C++ compiler unless the C code uses the common subset of C and C++. It refers to a simple C program below where differences in behavior between C and C++ compilers are highlighted. The code snippet is as follows:

```
/* 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
}
```

The annotations explain the differences: (1) `math.h` is missing, causing a warning in C89 and an error in C++98. (2) The size of a character is 4 in C89 (because it's an `int`) and 1 in C++98 (because it's a `char`). (3) Implicit conversion from `void*` to `int*` is okay in C89 but an error in C++98. (4) `class` is a reserved word in C++, causing an error in C++98 but not in C89.

Below the code, a bullet point states: "So the C code needs to be ported for C++".

So, here of course, `math.h` is missing as you can see. So, in C89 this will be a warning. Because `glibc` or `libc` in general will have that and `GCC` at least and many compilers would by default include the header that is required. But it is an error in C++98. So, the first thing is well, you this is editable, but you will still have to bring in `math.h` here and edit. Now, interesting things happen when you try to print the size of a character, a character is considered a `char` is considered an `int` in C. It is represented as an `int` internally.

So, if you print it size, it will it outputs 4 if your size of `int` is 4 in C89. Whereas character is actually a separate type in C++, it is not a part of the integral type. So, in C++ it has a size 1 or something different, at least because if there unicode characters the size, maybe 2 and so on.

So, if you just run this in C++98, you will get an output which is 1. So, you can see that, the programmer has written this, expecting that the value would be 4 but when you compile it with C++ compiler, the value actually is taken to be 1. Then here I have a void pointer to pointing to a character and I am implicitly casting that to an integer pointer.

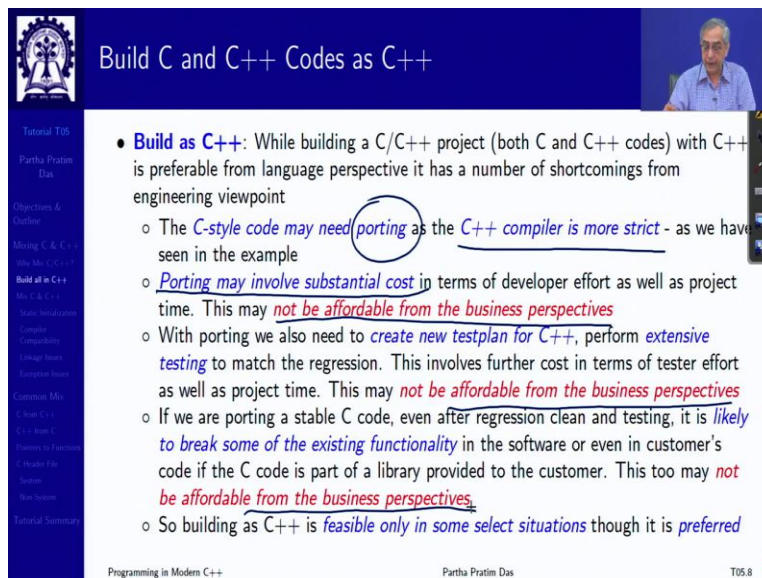
So, implicitly conversion, this is okay in C89 there is an error in C++98 is because C++ 98 does not like this implicit cast of pointer types, because that causes a lot of pain in the programming then you have `int class = 5`. Now, `class` was not there in C. So, `class` is not a key word is not a reserved word. So, in some C program somebody may have used a `class` as a variable

name, there are several keywords which have been introduced in C++, which were not reserved in C. So, the program might use any of that.

So, this will be okay in C89 and obviously, this is an error in C++98. So, these are some of the scenarios a lot more or somewhat kind of comprehensive list and discussion will be available in this tutorial of compatibility. But, my core point here was to see that even from a technical perspective of the language specification, it is not easy to build a C program as a with the C++ compiler considering it to be C++ with seamless operation.

Now, when you get compiler errors, you are better off because at least you know that this is there is some error that you have to take care of. Situations like sizeof character a is kind of disastrous, because the compiler will not say anything, it will silently build because it is correct code, but the result the logic becomes different and that is where your porting issues will be very, very difficult to handle.

(Refer Slide Time: 16:45)



The screenshot shows a presentation slide with a blue header and a white content area. The header contains the title "Build C and C++ Codes as C++" and a small video inset of a man speaking. The content area features a bulleted list of points about building C/C++ projects with C++. The text includes terms like "porting", "business perspectives", and "feasible only in some select situations". A blue circle highlights the word "porting" in the first bullet point. The slide also includes a sidebar on the left with navigation options and a footer with the text "Programming in Modern C++", "Partha Pratim Das", and "T05.8".

**Build C and C++ Codes as C++**

- **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*

Programming in Modern C++ Partha Pratim Das T05.8

Now, there are several issues that happen beyond this technical feasibility of building C code as C++. So, obviously, you have seen the difficulties of building C in C++, because C++ compiler is simply more strict. So, you cannot just build you will need what is known as porting. Porting our activities like I had to include math.h so, I had to edit.

I will need to change now the size of character a and its use to proper way we will have to somehow make that an explicitly as an integer type and then take the size of so, that C++ gives

the same thing. Now, what happens porting involves substantial cost the moment you are trying to reuse the code because it has a value proven value of having been corrected tested used by the community or your customer for a long time, but the moment you change the code all those values disappear.

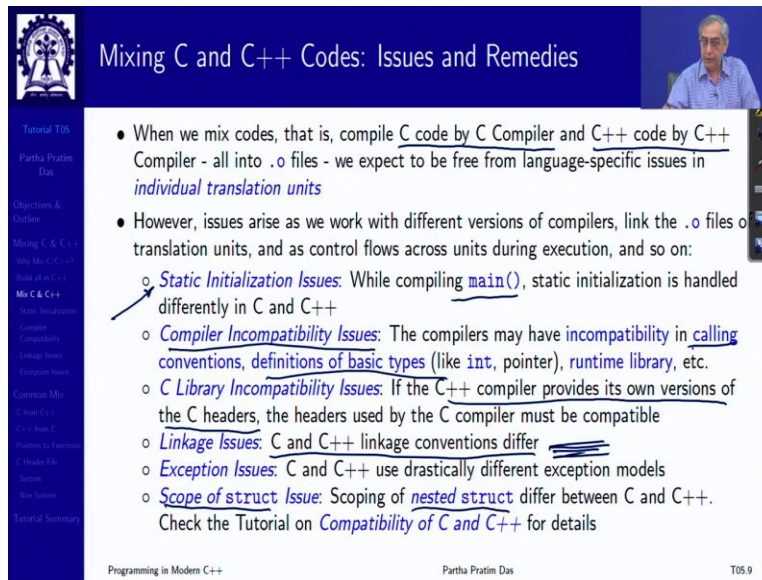
Because you have made the code different the moment I change from size of character, a to size of character a treated as int cast to int of character a, I have made the change and therefore there may be consequences of subsequent errors and all that that creep in. So, the question is it avoidable from the business perspectives?

The moment you do these kinds of porting you are changing the code you need a new test plan for the C code in C++, for the purpose of extensive testing, you will have regression test suites which exist you will have to those will have to now pass in the C++ compilation, it involves test a time further cost again, the affordability from business perspective becomes an issue.

If the C++ code is stable and it is regression tested even then, it is likely to break some existing functionality, that might even break your customers code because you may have given this library to the customer for their use in part of their code. So, the moment you convert to C++ and do a porting, then your code might break something in the customers code which you cannot even test.

So, business wise, it is not liked by the top management. So, building C as C++ is feasible from technical as well as business perspectives only in some select situation, though it is preferred.

(Refer Slide Time: 19:40)



The slide is titled "Mixing C and C++ Codes: Issues and Remedies" and features a small video inset of the presenter in the top right corner. The main content is a bulleted list of issues and remedies. The first bullet point states that when mixing C and C++ code, each is compiled by its respective compiler into .o files, which are expected to be free from language-specific issues in individual translation units. The second bullet point notes that issues arise from using different compiler versions and linking .o files. It lists several specific issues: Static Initialization Issues (main() handling), Compiler Incompatibility Issues (calling conventions, basic types like int and pointer, runtime library), C Library Incompatibility Issues (C++ compiler's own headers vs C compiler's headers), Linkage Issues (different conventions), Exception Issues (different models), and Scope of struct Issue (nested struct scoping). A reference is made to a tutorial on the compatibility of C and C++.

- 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 o 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

So, the actual engineering practice will much less do building C as C++ rather it will like to do a mixing which means the meaning of mixing is compile C by C compiler, compile C++ by C++ compiler and then link the whole thing together all these .o's. So, basically what you are saying that we will rather instead of using G++ for doing this, we will rather use GCC for doing this which selectively can do the build.

So, now we are getting another sense of why we have two different compiler target names like GCC and G++ G++ is building everything in C++, GCC is targeted towards doing a mix. Now, if you do that, the issues will arise as we work with different versions of compilers and link .o files in the translation units and control flows, because the control will not be limited to the C code that you have built by your C compiler, it is it will be in the C++ will come to C will go to C++ will come to C and so on. So, as you do that, you the question is what will be where will be main.

Because if main is one function, so it will either be built a C or be built a C++, that makes a significant difference because there are a lot of static initialization issues that are handled differently in C and in C++, the static initialization issues are how initialise your static objects defined in their static scope, whether they are global, whether they are class static members or their function local static and so on.

How do you initialise them before main? And how do you deal initialise them after main? So, this is this is a big issue area, which we will have to keep in mind. There could be compiler compatibility issues, so do not ever use different compilers. For C, C++ use the same compiler that is what is preferred.

So, if you use GCC to do the both, then you are safe that they are not incompatible, because it is the same compiler two parts of that, but if you use some version of GCC to build your C code and some other version of C++ to build your C++ code or some other compiler, then there may be several issues in terms of calling conventions definition of basic types and so on.

There may inherently be compatibility issues between C library. The C++ compiler provides its own version of C headers. So, the headers used by C compiler must be compatible with the header provided by the C++ compiler for the standard libraries at least. The linkage convention and this is a big heartburn area, I raise this question earlier as to why do you always link by C++.

The linkage issues are a big thing which need to be handled. There could be exception issues because C and C++ use drastically different exception models. If you are not on top of exceptions, you may not be able to realise all of that very well, C either you does not use anything, most programs do not use anything just they do a you know code involved error checking or they may use a paradigm known as setjump, longjump kind of whereas C++ as a full grown exception model.

So, how to integrate them is a big question in the mix. Then, there are some specific issues like scope of struct, the nested structure, if you have nested structure, structure within another structure, their meaning differ between C and C++ I mean and you can look at the compatibility tutorial go through that and you will get to know what those things are.

(Refer Slide Time: 24:07)

**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()**

Programming in Modern C++ Partha Pratim Das T05.10

**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()** ✓

Programming in Modern C++ Partha Pratim Das T05.10

So for the static initialization issue, say they both initialise C and C++ both initialised static variables. There constructed and initialised before main, but they have different semantics. In C, the static initializer necessarily is a constant that is, if I want to initialise i, I cannot call a function.

I mean this is just a just to show you there is a trivial function which returns a value 10. If I write it as 10 a constant then it is okay in the C compiler. So, C compiler will not compile this kind of a code. So, if I want to compile main as C, then we will not be able to write this which often may be required. I have written this as a placeholder.

But if you instantiate any object as a static, then the constructor will need to get called which is equivalent to the init that I am putting here just to show you obviously, this is okay in C++. So, C++ what it does the basic difference of static initialization is it generates an additional start function, where all global function calls constructors and so on are executed before the main starts.

And the C compiler does not generate this kind of a start function. So, the main starts as soon as it is loaded. So, the lesson that we get from here is obviously C++ does more in terms of static initialization. So, we form the rule number one, which is use C++ compiler when compiling main, never compile main by C compiler. This is the first rule we must be followed following when you are mixing.

(Refer Slide Time: 26:07)

**Mixing C and C++ Codes: Issues and Remedies**

Tutorial T05  
Partha Pratim Das

- **Compiler and C Library Incompatibility Issues:** To alleviate 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,

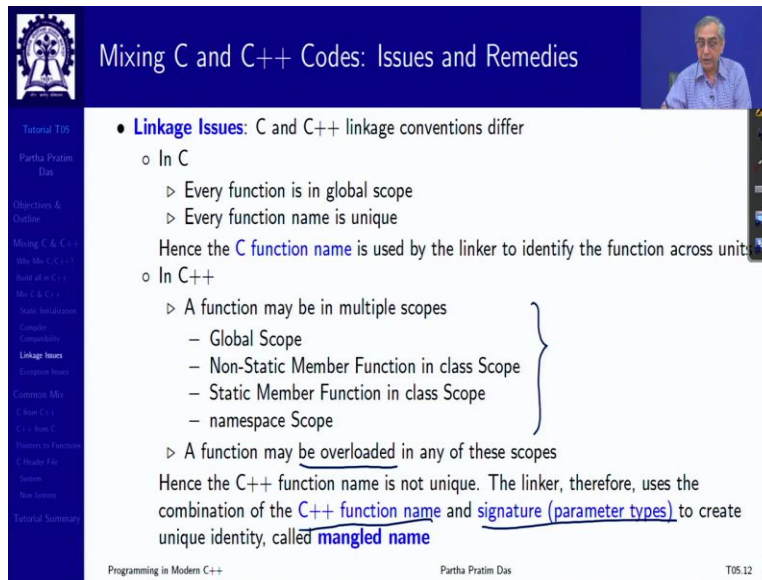
- **RULE 2: C and C++ compilers must be compatible** ✓

Programming in Modern C++ Partha Pratim Das T05.11

Then there are compiler and C library compatibility issues. So, preferably use library from the same vendor like GCC and use same or compatible versions, if it is preferable if you use the same version, so that you do not have issues of mismatch of data type size and so on, runtime library we must of C must also be compatible with the C++ compiler.

So, if you take the from the same vendor and the same version compiler, you will not have a problem if you have to, if you are mixing in that, then you have to make sure that rule number two is satisfied that is C and C++ compilers must be compatible.

(Refer Slide Time: 26:55)



The slide is titled "Mixing C and C++ Codes: Issues and Remedies" and features a small video inset of the presenter in the top right corner. The main content is as follows:

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

At the bottom of the slide, it says "Programming in Modern C++" on the left, "Partha Pratim Das" in the center, and "T05.12" on the right.

Now, linkage is a big problem, because C and C++ linkage conventions differ in the basic philosophy, because in C every function is global, it is in the global space. So, every function name is unique. Whereas as you have learned in C++ , a function may have multiple scope it can be in the global scope, it could be a non static member function in the class scope, it could be a static member function in the class scope, it could be in the namespace scope, a function may be overloaded and so on so forth.

The name is just an indicator of the function and then there are several other factors, which decide exactly which function is that which include the name besides a name, it includes the complete signature. So, what does C++ compiler do? In contrast to the C compiler is that they take all this information mangle them, kind of encode them into a new name called the mangled name of the functions. Whereas C does not have that kind of feature.



(Refer Slide Time: 28:03)

**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);
}
    
```

So, just to show you I mean just to show how this is what kind of mangling you get here is I show that there are overloaded functions in the global overloaded functions as class member, non static, overloaded function as class member, which has static and namespace overloads. So, I have all these overloads, which are perfectly valid, all of these different print can exist in the same programme, actually in the same code at the same time C++ is perfectly okay with that. So, how does it handle this issue? It does handle this issue by mangling.

(Refer Slide Time: 28:49)

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

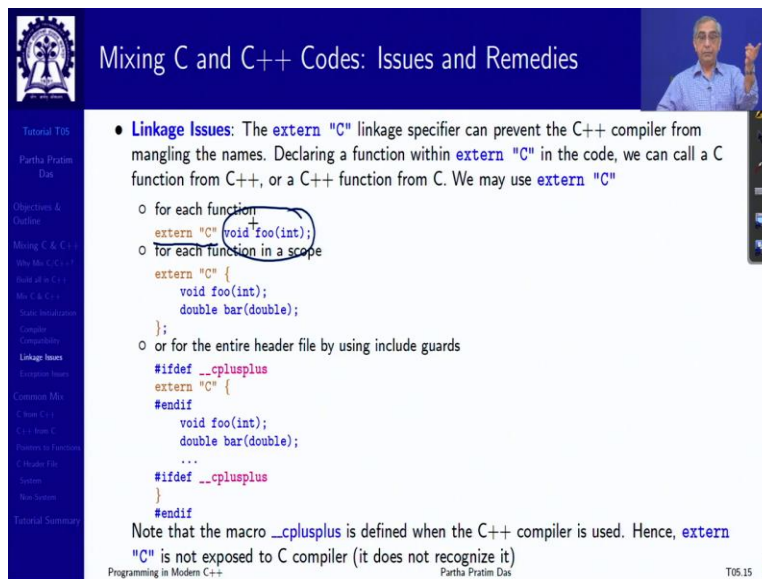
Function	gcc 6.3.0	msvc 18.00	Mangled?
// Global: Overloaded			
<code>print(int)</code>	<code>_Z5printi</code>	<code>?print@YAXH0Z</code>	Yes
<code>print(double)</code>	<code>_Z5printd</code>	<code>?print@YAXN0Z</code>	Yes
// Class member: Overloaded			
<code>MyClass::print(int)</code>	<code>_ZN7MyClass5printEi</code>	<code>?print@MyClass@QAEXH0Z</code>	Yes
<code>MyClass::print(double)</code>	<code>_ZN7MyClass5printEd</code>	<code>?print@MyClass@QAEXN0Z</code>	Yes
// Class static member: Overloaded			
<code>MyOtherClass::print(int)</code>	<code>_ZN12MyOtherClass5printEi</code>	<code>?print@MyOtherClass@QSAXH0Z</code>	Yes
<code>MyOtherClass::print(double)</code>	<code>_ZN12MyOtherClass5printEd</code>	<code>?print@MyOtherClass@QSAXN0Z</code>	Yes
// namespace member: Overloaded			
<code>MySpace::print(int)</code>	<code>_ZN7MySpace5printEi</code>	<code>?print@MySpace@YAXH0Z</code>	Yes
<code>MySpace::print(double)</code>	<code>_ZN7MySpace5printEd</code>	<code>?print@MySpace@YAXN0Z</code>	Yes
// Global: Not Overloaded			
<code>main()</code>	<code>_main</code>	<code>_main</code>	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

So, if you look at the names they actually generate, there are ways to ways to access these names by looking at the compilation by looking at the assembly. So, here is the original function and if you look at these are, I have given it two different I mean just to give you an example, I have given you the list from GCC as well as from Microsoft, visual C++ and you can see how the names have been mangled.

So, that they can be actually distinct in terms of the functions called by the compiler. So, you can see all of these are mangled, except for main. Because main, even in C++ is a unique function, you cannot have another main function or anything by that name. So, this mangling really makes it difficult because your C linker does not know about mangling, there is no mangling in C. So, C linker does not know about mangling whereas you C++ linker will always have to deal with mangling.

(Refer Slide Time: 30:00)



The slide is titled "Mixing C and C++ Codes: Issues and Remedies" and features a small video inset of the presenter in the top right corner. The main content is a list of bullet points explaining linkage issues and providing code examples. The first bullet point states that the `extern "C"` linkage specifier prevents the C++ compiler from mangling names. It then provides two examples: one for a single function and one for an entire header file using include guards. The code examples show how to declare and define functions in a C++-friendly way for C linkage. A note at the bottom explains that the `__cplusplus` macro is defined when the C++ compiler is used, which is why `extern "C"` works in C++ but not in C.

- **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:

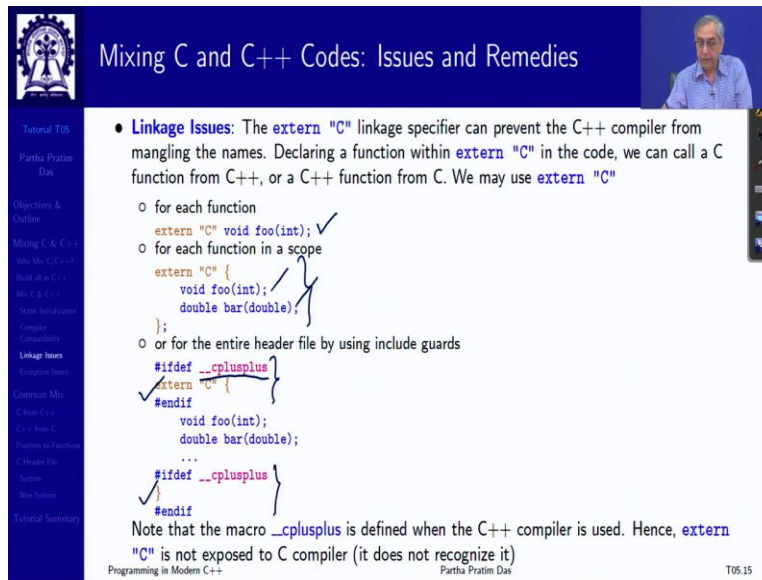
```
#ifndef __cplusplus  
extern "C" {  
#endif  
    void foo(int);  
    double bar(double);  
    ...  
#ifdef __cplusplus  
}  
#endif
```

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)

Programming in Modern C++ Partha Pratim Das T05.15

So, when you want to use a C function in C++ or other ways, you will have to instruct the C++ compiler not to mangle and also tell the C++ compiler if you are using a C function that is not a mangled name, it is a pure name, which means, if you stop this mangling, that means that you are going back to the C style of function naming which means that you cannot have overloaded names for those functions. So, this is handled by using a specific phrase linkage specifier as we say `extern` within double quote C, which says that this is the what we write inside this is pure C link this as C.

(Refer Slide Time: 30:09)



**Mixing C and C++ Codes: Issues and Remedies**

- **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  

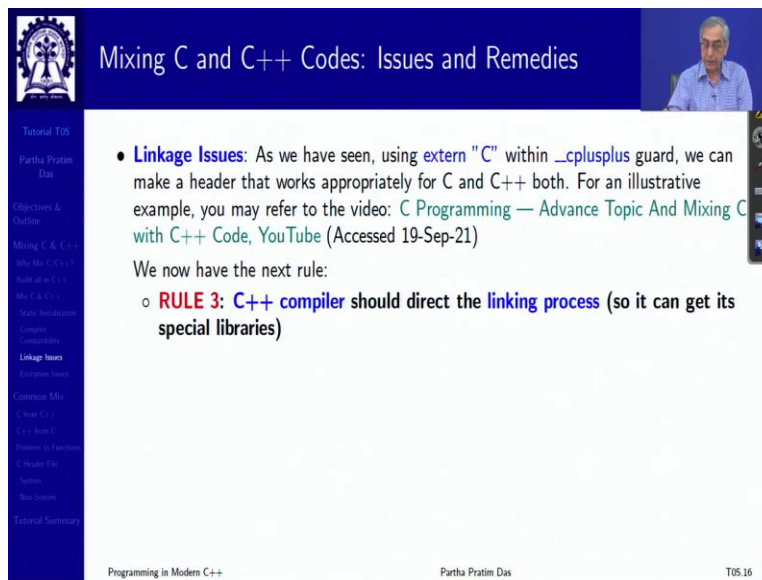
```
extern "C" void foo(int); ✓
```
  - for each function in a scope  

```
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)

Programming in Modern C++ Partha Pratim Das T05.15



**Mixing C and C++ Codes: Issues and Remedies**

- **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)

Programming in Modern C++ Partha Pratim Das T05.16

So, if we do that here, you can you can do it for a single function or you can define extern C as a scope. And so, both of these in the scope will be treated as C function or you can actually put an entire header as extern "C". And, again, the problem is this we are doing for to help the linker. Now, the problem is C compiler does not understand extern "C", because C does not have anything like that.

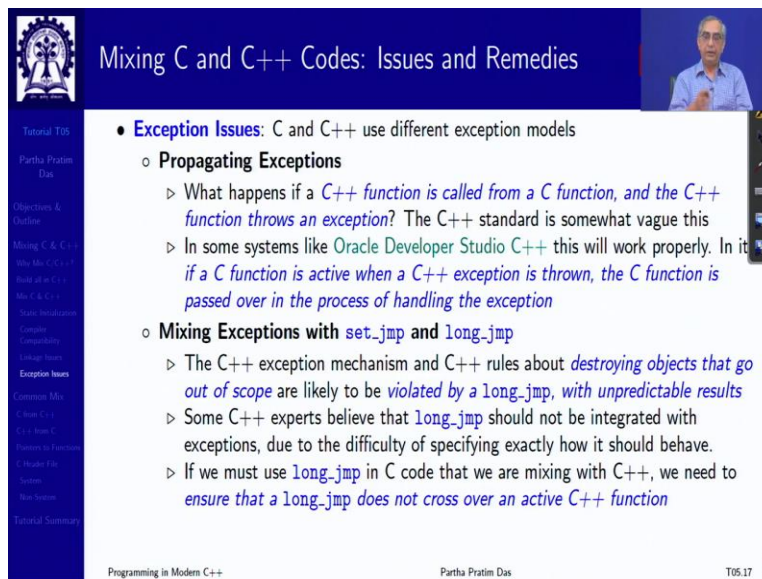
So, you have to make sure that the C compiler does not get to see it, only the C++ compiler gets to see it because that needs to know that C++ linker needs to know that the mangling is not there

or mangling is not being preferred. So, you have to make use of this particular macro from the CPP we discussed this which is true only if you are compiling as C++.

So, if you are compiling as C++ this to comment in the entire scope and you take cognizance of that, whereas, if we are compiling by C, these are just skipped, because `__cplusplus` does not is not defined and therefore, everything is treated as C and is not mangled anyway.

So, this is the kind of, solution that you can have for this, mixed type of linkage. And after that you naturally with this C++ linker has more information, and it has to deal with the decision of whether something is mangled whether something is not. So, we will leave it to the C++ compiler to complete the linking process do the linking process.

(Refer Slide Time: 32:55)



The screenshot shows a presentation slide with a blue header and a white content area. The header contains the title 'Mixing C and C++ Codes: Issues and Remedies' and a small video inset of a speaker. The content area lists 'Exception Issues' and details about exception propagation and mixing with `setjmp` and `longjmp`. A sidebar on the left shows a table of contents, and the bottom of the slide has footer text.

**Mixing C and C++ Codes: Issues and Remedies**

- **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 with 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 `setjmp` and `longjmp`**
    - ▷ The C++ exception mechanism and C++ rules about *destroying objects that go out of scope* are likely to be *violated by a `longjmp`, with unpredictable results*
    - ▷ Some C++ experts believe that `longjmp` should not be integrated with exceptions, due to the difficulty of specifying exactly how it should behave.
    - ▷ If we must use `longjmp` in C code that we are mixing with C++, we need to *ensure that a `longjmp` does not cross over an active C++ function*

Programming in Modern C++ Partha Pratim Das T05.17

There are other issues I put them as advanced because unless you know these in the in your language, well, you will not be able to understand, but there are issues relating to propagation of exceptions and mixing the exception models. So, exception is a big problem. So, for now, I am just assuming that well, you are not I mean we are not discussing about mixing codes, where the exceptions are used either in C or in C++. The basic points I have told here, but in a different tutorial, I will come back with talking about exceptions where I will deal with this particular aspect.

(Refer Slide Time: 33:38)

Common Code Mix Scenarios

Common Code Mix Scenarios

Programming in Modern C++ Partha Pratim Das T05.18

So, some of the common code mix scenarios. So, this was the so we have three basic rules compile main by C++ compiler, C and C++ compiler must be compatible, their libraries runtime all must be compatible and third is linked with C++ linker, write and use extern “C” as and when required to guard.

(Refer Slide Time: 34:03)

Common Code Mix Scenarios

- 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? ✓

Programming in Modern C++ Partha Pratim Das T05.19

So, with that I have listed here is some of the common scenarios obviously, how do you call a C function from C++? How do you call a C++ function from C? The this is just simple because you have only global type of function here. You will have to decide what should be done of all

different types of functions that C++ has. So, how do I call a function of each type from C? Then the question is, how do I include a C header file in a C++ project?

If it is a system header or if it is a non-system header, how do I use pointed to functions? And finally, last, but not the least, how do I manipulate objects in C, C++ project mix? Because C does not have objects, whereas C++ certainly will have objects.

(Refer Slide Time: 34:58)

The slide is titled "How do I call a C function from C++?". It contains the following content:

- 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
    // ...
}
```

Programming in Modern C++ Partha Pratim Das T05.20

So, some of the quick techniques all are primarily based on few very simple strategies. One is to call a function from C to C++ as we have seen, use `extern "C"`. So, you do not do actually need to do anything for the C code. Just in C++ code, call them as `extern "C"` so that when you compile either .o will be told that these are unmagnified names, these are global names, and you can make the call directly from here and that will perfectly work.

But you here the C++ type rules are not getting where are being used not in when you compile this you are using C++ rules only have told that these are C functions. So, a function declared `extern "C"` cannot be called with wrong number of arguments in C you can C++ you can do that. But here you will not be able to do that. So, if you call this as say, it does not have a parameter if you pass two parameters, then it will be you will have an error now, which is unexpected arguments which is actually fine, which actually makes it safer.

(Refer Slide Time: 36:18)

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

- 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 cc(struct C* p, int i) {
  double d = call_C_f(p,i);
  /* ... */
}
```

Programming in Modern C++ | Partha Pratim Das | T06.21

Doing the reverse is somewhat of a difficulty that is how do you call a C++ function from C. C does not know mangling. So, if you want to call a C++ function from C the name of that function must be explicitly unmingled. So, now, we have a situation of a C++ function, but for that, we say that `extern "C"` that function signature. Which ensures that the compiler will generate `f` as a unmingled name and then from C I can easily call that function (does not a).

So, if I do `f(i)` it will be called with no difficulties, but obviously, this will work only for non-member functions only for global functions. Now, what happens if you call want to call a member function? What is the speciality of calling a member function you would recall that a member function always gets in implicit this pointer that is the address pointer of the object. So, you need to know that and if I want to kind of make the member function globalised not having the behaviour of the member function, I will need to pass this address.

(Refer Slide Time: 38:02)

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

- 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);
}
```

*Handwritten notes: p → f(i) and a circle around p->f(i) in the wrapper function.*

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

Programming in Modern C++ | Partha Pratim Das | T05.21

So, the most common and most reliable way to do that is define a wrapper function that is if p is an object pointer and you want to call the function f which is a member function of the class of which p is pointed object is an instance then you want to make this call. Now, in C there is no way to make this call. So, what do you do you define a C unmingled function name say call cf, pass the parameter that you need for the member function, but most importantly you pass the address of the object this pointer of that object on which this call has to be made.

(Refer Slide Time: 38:59)

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

- 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 cc(struct C* p, int i) {
    double d = call_C_f(p,i);
    /* ... */
}
```

Programming in Modern C++ | Partha Pratim Das | T05.21



So, you can see what I mean the workaround that we are getting on here is the C++ code is as it is, this has been added and you are telling the compiler that link it as a unmingled name. So, treat this as a global function which it will do. And that to that global function we are passing the object and this is an in C++. So, here I can make a call to a member function and call the member function.

Whereas, in C you do not know about what these are, what is this pointer and all these are? So, in C you will have to call it simply as a global function. Now, the question is what is C\*? What will C\* p? So, there is a other feature that C/C++ boundary has provided that any class can be treated as a struct in C. Because class is finally an aggregation. The data member part of it forget about if you forget about private, public access specifiers, and all that everything is public in a struct, but it is just data collection format.

(Refer Slide Time: 40:18)

The slide is titled "How do I call a C++ function from C?: Non-Member and Member Functions". It contains two bullet points and corresponding code snippets.

- 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 cc(struct C* p, int i) {
    double d = call_C_f(p, i);
    /* ... */
}

```

Handwritten annotations on the slide include a blue circle around 'struct C' in the C code, a blue arrow pointing from the C code to the wrapper function in the C++ code, and a blue arrow pointing from the wrapper function to the member function 'f' in the C++ code.

At the bottom of the slide, it says "Programming in Modern C++", "Partha Pratim Das", and "T05.21".

So, you in terms of the C code, you declare it as struct C\*, which is a C\* pointer, but treating C as a struct which C understands which I am sorry too many C so which C language understands this class C. So, taking that you can now have a C function which calls this. So, when you call with this, then on that object, the function f will get called.

So, this is the basic addition to in addition to the extern "C", doing proper wrapper function is a basic strategy that you keep on doing to create the plethora of varied C++ function types that you need to call. Anything that you need to call from C, you will have to give a wrapper with a

unique name and C calls that wrapper, C++ treats that wrapper as a global function which who's named must not be mingled. So, it is put in the extern "C" that is the basic strategy.

(Refer Slide Time: 41:27)

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

- 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**

Programming in Modern C++ | Partha Pratim Das | T05.22

So, if you have overloaded function, I have two overloaded functions here. So, for each one of them, I will have a wrapper which we just do the same thing it just makes a call to the C++ function, but they are global and treated as unmingled name. So, in C, you get to see these global functions, the wrapper functions only you call the wrapper functions as you want because you cannot have the advantage of having the overloaded name in C.

So, you have to explicitly called f(i) with an integer parameter if you wanted to call f(d) with the double parameter and correspondingly the wrappers will be called in C++ and the overloaded functions will be invoked. So, it is strategy wise now you have got the strategy the strategy is to have a wrapper to take the function name to the global space.

(Refer Slide Time: 42:26)

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

- 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
```

Programming in Modern C++ Partha Pratim Das T05.23

Now, there is some caveat that can happen for example, you are trying to do this you are trying to define a function foo which takes a function pointer. Function pointer is another type of way to call function and that function pointer is defined here this code is in C++. And now, when you use g to call foo(g), you will have a type mismatch error. Why? This is both of these are treated as unmingled name.

So, g is a C function to C++ which you are passing here whereas, the typedef that you have done is not in extern "C". So, typedef expects this to be mingled name typedef expects this to be a mangled name, whereas, it gets an unmingled name it cannot handle it does not know.

(Refer Slide Time: 43:52)

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

- 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
```

Programming in Modern C++ Partha Pratim Das T05.23

So, the simple thing is to take care of this all that you need is to put everything in `extern "C"`. So, that now, the typedef knows that this function pointer is an unmingled name and this is an unmingled function name. So, this will work perfectly fine now.

(Refer Slide Time: 44:09)

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

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

<pre>// C++ code // Nothing unusual in #include line #include &lt;stdio.h&gt; int main() {     // Nothing unusual in the call either     std::printf("Hello world");     // ... }</pre>	<pre>/* C code */ /* Compiled using a C++ compiler */ /* Nothing unusual in #include line */ #include &lt;stdio.h&gt; int main() {     /* Nothing unusual in the call either */     printf("Hello world");     // ... }</pre>
---	---
- The `std::` part of the `std::printf()` is for the `std` namespace where C++ Standard Library components like `<stdio.h>` 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

Programming in Modern C++ Partha Pratim Das T05.24

So, finally, to include the standard library headers, it is you have already seen that C standard library headers are available in C++ in the namespace `std`. So, that is the `stdio` that we have discussed several times you can the C code could use `printf` in C++ this simply becomes `std::printf` rest of the functionality remains the same. So, care needed if there is a difference

between the C header and the corresponding C++ version which by rule two we have said we must not try to do.

(Refer Slide Time: 44:52)

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

	C Header	C++ Header
C Program	Use .h. Example: <code>#include &lt;stdio.h&gt;</code> Names in global namespace	Not applicable
C++ Program	Prefix c, no .h. Example: <code>#include &lt;cstdio&gt;</code> Names in std namespace	No .h. Example: <code>#include &lt;iostream&gt;</code>

- To summarize
  - 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.

Programming in Modern C++ Partha Pratim Das T05.25

The header conventions I will not I mean this slide I have included in the tutorial just for your quick reference in case you have forgotten, but this we have discussed what are the different naming conventions of standard library headers and how to use but when. So, we will just follow that.

(Refer Slide Time: 45:14)

**How can I include a non-system C header file in my C-Editable Case**

- 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
 

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

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

#endif
}
#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
    // ...
}
```

Programming in Modern C++ Partha Pratim Das T05.26

Now, the question is if we are this was good for the standard library or system headers, what if there is a non-system C header, the header that say I have written so, then you have two cases when the if the header is editable. If you can edit the header, then all that you need to do is to parenthesize it in the extern “C” scope.

So, what will happen when that header is included in a C++ file if this macro is going to be true and all of that header all functions they are in will be treated as unmingled name. So, it can you can simply include that header you do not need to do anything in the C++ side. You and this header could be common between C and C++ because when you use a C compiler for the C code, this particular macro is false. So, it will not see anything, it will see a pure simple C header.

(Refer Slide Time: 46:31)

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

- 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"
}

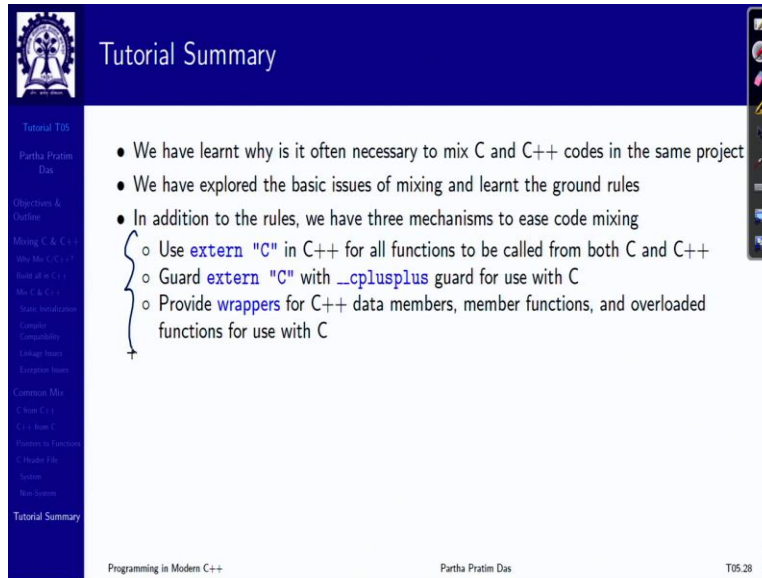
int main() {
    f(7, 'x', 3.14); // Note: nothing unusual in the call
    // ...
}
```

Programming in Modern C++ Partha Pratim Das T05.27

Now, what if the header is C header is not editable? If you are not allowed to edit the header, then you cannot do what we said. So, in that case, you do not have a choice but at your use place, which is the C++ code, instead of including it directly, you will have to include it within extern “C”. So, that the entire header becomes that is a basic thing you are trying to do that put everything of the C header into extern “C”.

If the header is editable, it is most preferred to do it in the header itself and use that same header everywhere. Otherwise, you have to do this inclusion within the extend “C” in C++ and without the extern “C” in the C code that results all the.

(Refer Slide Time: 47:21)



The image shows a presentation slide titled "Tutorial Summary" with a dark blue header. On the left, there is a vertical navigation menu with various topics listed. The main content area contains a bulleted list of points. The first two points are general observations. The third point is followed by a list of three specific mechanisms, which are grouped by a large curly brace on the left. The footer of the slide includes the text "Programming in Modern C++", the name "Partha Pratim Das", and the slide number "T05.28".

- We have learnt why it is 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

So, these were the different scenarios in which you can mix C and C++ codes function calls of varied crimes compilation linking and all that and the besides the three rules that we have given the summary is to know that you have to use `extern C++` properly. So, that all functions can be called between C and C++ as needed. You have to guard `extern "C"` with the C++ macro `__cplusplus` macro for C code.

And everywhere else when you have the any context of mineral names overloading you have to use wrappers to specifically provide global names. So, this way, the kind of so, here we have looked at our overall requirement and issues of mixing what are the different problems and talked about the resolution the remedies that are possible. Continuing in the next tutorial, I will walk you through a complete mixed code example of a project switches which you will be able to actually run on your system.

And see that you are being able to understand the whole issues of mixing and how to work with them. Thank you very much for your attention. See you in the continuation of this tutorial later.