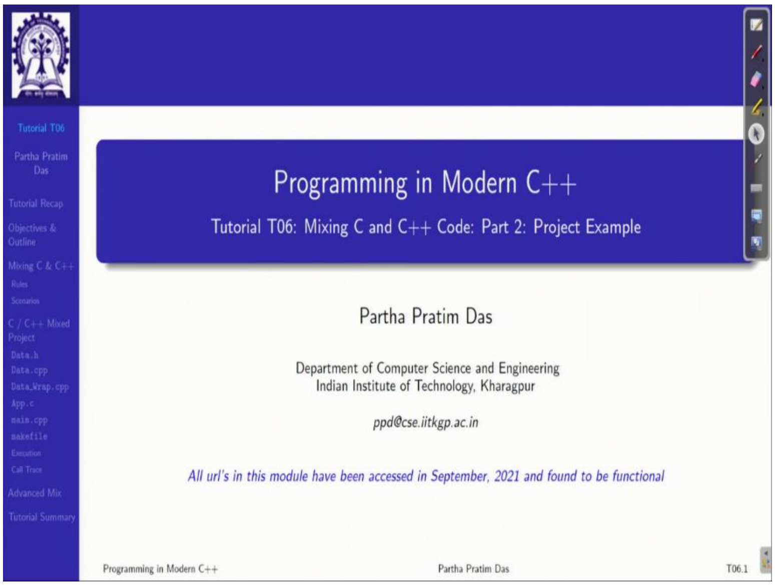
Programming in Modern C++
Professor Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Tutorial 06 - Mixing C and C++
Code: Part 2: Project Examples

(Refer Slide Time: 00:39)



Programming in Modern C++

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

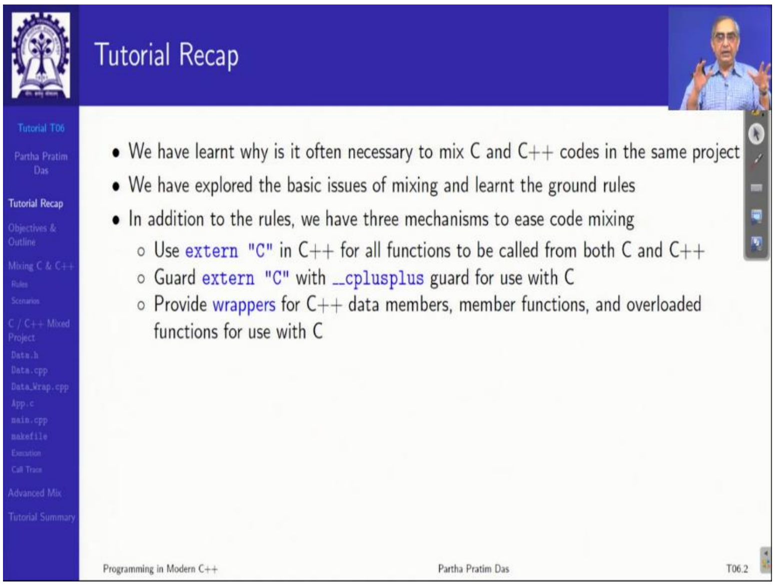
Partha Pratim Das

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

ppd@cse.iitkgp.ac.in

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

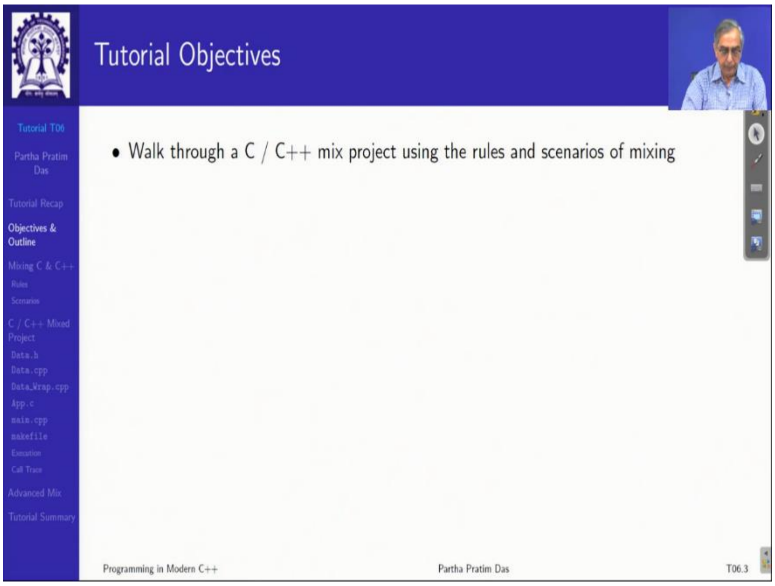
Programming in Modern C++ Partha Pratim Das T06.1



Tutorial Recap

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

Programming in Modern C++ Partha Pratim Das T06.2



Tutorial Objectives

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

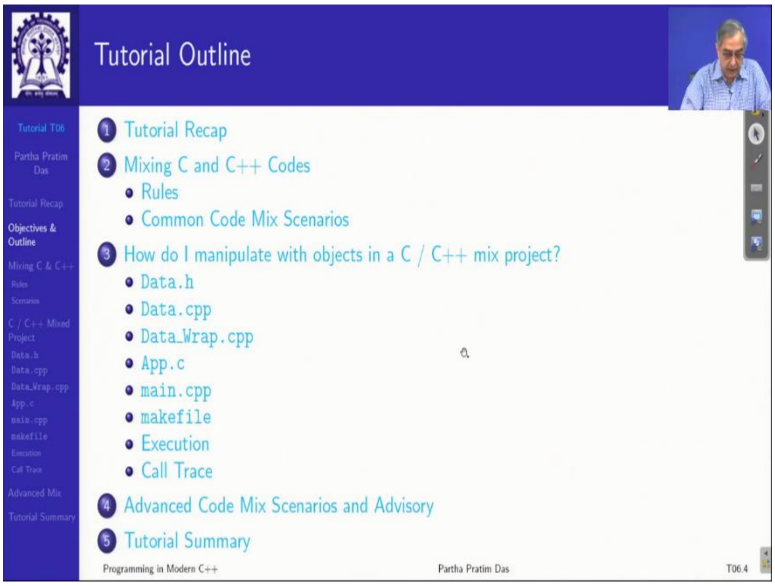
Welcome to Programming in Modern C++. We are going to discuss a tutorial 06, which is a continuation second part of the tutorial on mixing C and C++ code. In the previous tutorial, in 05, we learned the basic issues of why it is often necessary to mix C and C++ codes in the same project, and we have explored the basic issues of mixing and we have also learned a few ground rules.

In addition to those rules for actually mixing or migrating C, C++ codes into a single project, we have seen three mechanisms, which facilitate the mixing process. One is to use the extern “C” in C++ for all functions to be called from both C and C++. This will tell C++ linker to treat these functions as C functions and link without doing the mangling of the names.

Then this extern “C” often needs to be guarded with __cplusplus macro for use with C, so that if it happens to be compiled by a C compiler then this will be ignored. And finally, the actual trick lies in providing wrappers for C++ data members, member functions, overloaded functions for use with C. So, this generic paradigm of mixing approach we have seen in the previous tutorial.

What we will do in the current tutorial is to using these rules and the tricks that are that we have learned we will take a single project and build it with C and C++ code mixed in a certain way. So, this will be more like a hands-on in terms of how you mix how you do mix language project.

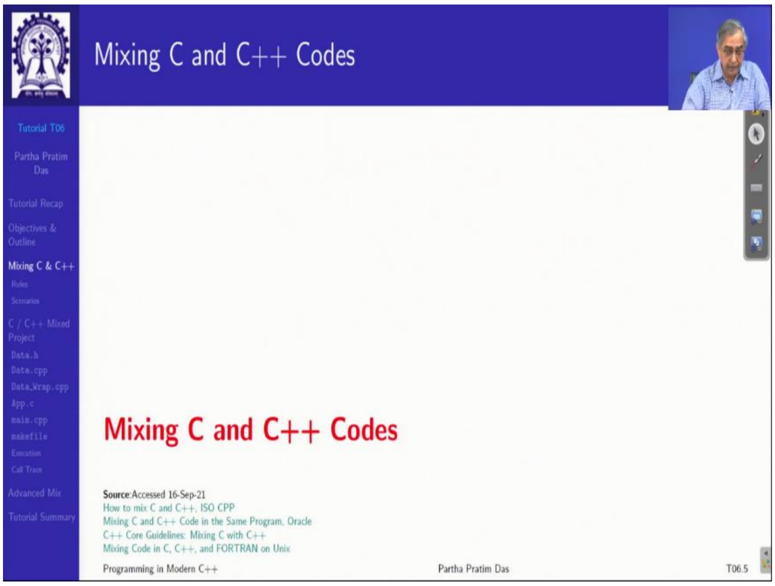
(Refer Slide Time: 02:41)



Tutorial Outline

1. Tutorial Recap
2. Mixing C and C++ Codes
   - Rules
   - Common Code Mix Scenarios
3. How do I manipulate with objects in a C / C++ mix project?
   - `Data.h`
   - `Data.cpp`
   - `Data_Wrap.cpp`
   - `App.c`
   - `main.cpp`
   - `makefile`
   - Execution
   - Call Trace
4. Advanced Code Mix Scenarios and Advisory
5. Tutorial Summary

Programming in Modern C++ Partha Pratim Das T06.4

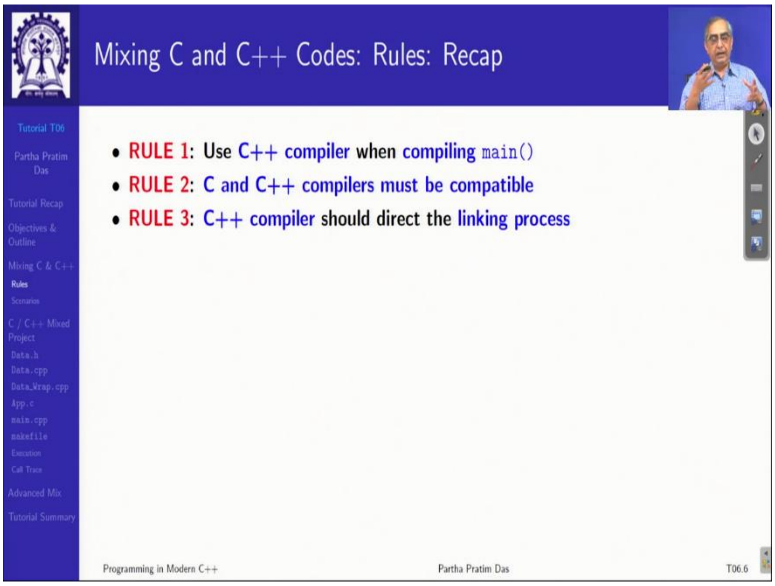


Mixing C and C++ Codes

Mixing C and C++ Codes

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

Programming in Modern C++ Partha Pratim Das T06.5



Mixing C and C++ Codes: Rules: Recap

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

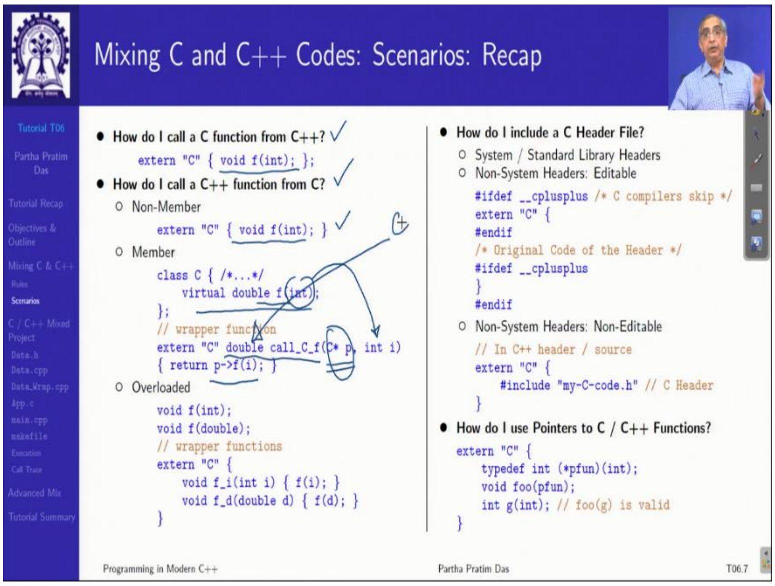
Programming in Modern C++ Partha Pratim Das T06.6

So, this is the outline which as ever available on the left. So, just a quick recap of what is the summary of rules and scenarios that we have learned so far. So, we have learned three rules. One is rule one is to use C++ compiler when compiling main. This is required as we said, because we need the C++ compiler to treat the static initialization process for which before main starts there is a separate wrapper function from the system where the initialization codes run, which is not the case for C, so it is not enough to compile it is not right to compile main with C compiler must compile it with a C++ compiler.

The second rule is that C and C++ compiler must be compatible. Preferably they must come from the same vendor, and it is best if they are of the same version like GCC. It has it provides compiled with GCC for C codes with .c and for C++ with .cpp or you can use C++ for that matter also. So, the compatibility of compiler and the C library across C and C++ compiler, C standard library across C and C++ compiler is a necessity to make sure that we do not go through several rough ages.

Third rule is C++ compiler should direct the linking process. We have discussed it at length that because of function overloading that is allowed in C++, which is not there in C the function names need to be mangled, because just the name does not tell uniquely the function, you need to know the parameter types their order and so on, so that is mangled into a different name by the linker. So, C linker does not know about this mangling, so C linker cannot handle this. So, final linking process must be done by the C++ linker.

(Refer Slide Time: 04:57)



Mixing C and C++ Codes: Scenarios: Recap

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

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

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

```
extern "C" { void f(int); } ✓
```

  - Member

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

  - Overloaded

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

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

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

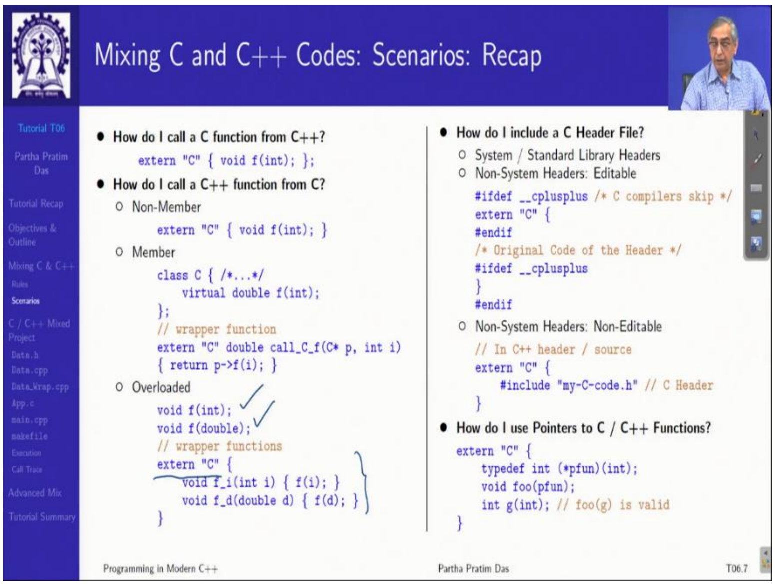
  - Non-System Headers: Non-Editable

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

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

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

Programming in Modern C++ Partha Pratim Das T06.7



Mixing C and C++ Codes: Scenarios: Recap

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

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

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

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

  - Member

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

  - Overloaded

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

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

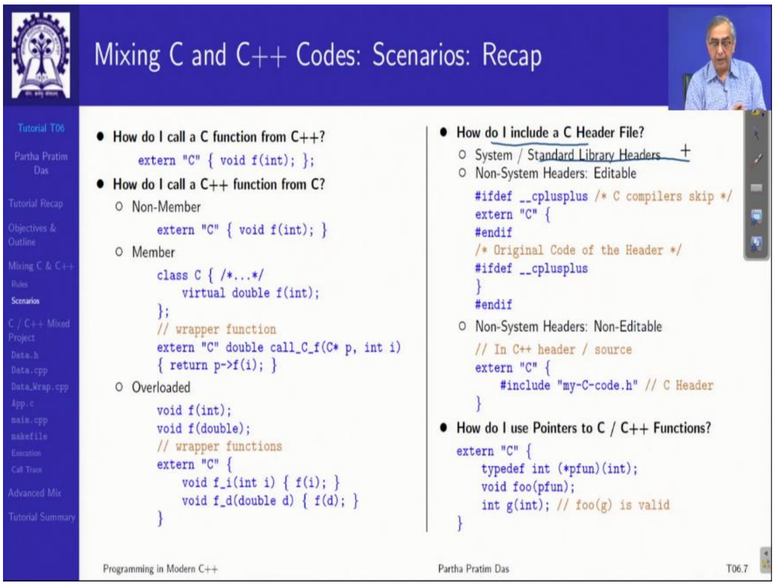
```
#ifdef __cplusplus /* C compilers skip */
extern "C" {
#endif
/* Original Code of the Header */
#ifdef __cplusplus
}
#endif
```

  - Non-System Headers: Non-Editable

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

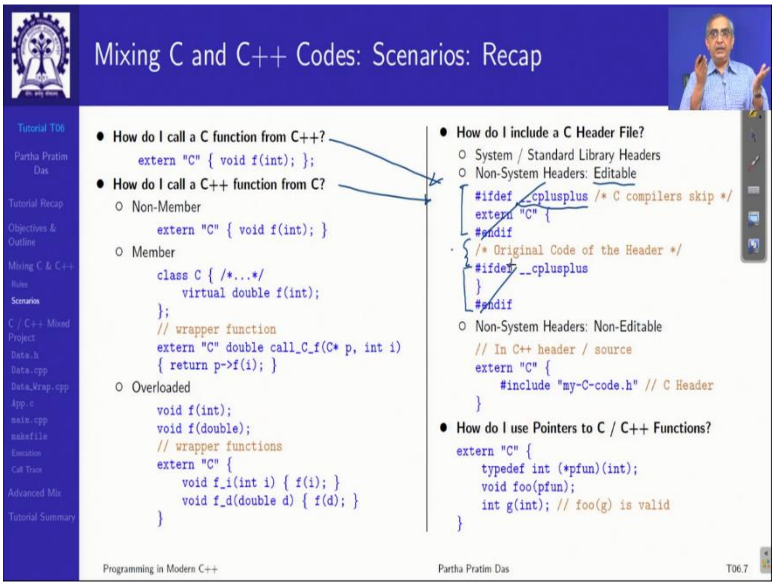
- How do I use Pointers to C / C++ Functions?

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


Tutorial T06
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Mixing C & C++
Rules
Scenarios
C / C++ Mixed Project
Data.h
Data.cpp
Data_Wrap.cpp
App.c
main.cpp
makefile
Execution
Call Trace
Advanced Mix
Tutorial Summary

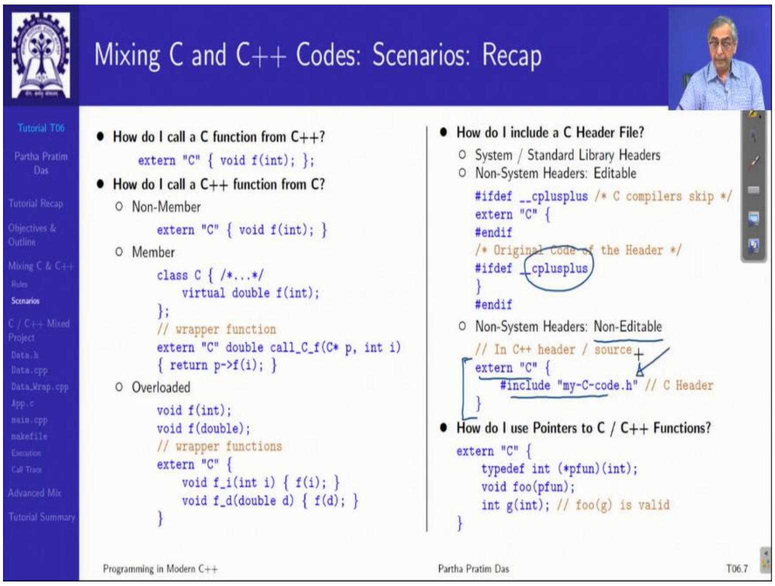
Mixing C and C++ Codes: Scenarios: Recap

- How do I call a C function from C++?
extern "C" { void f(int); };
- How do I call a C++ function from C?
  - Non-Member
extern "C" { void f(int); }
  - Member
class C { /*...*/
    virtual double f(int);
};
// wrapper function
extern "C" double call_C_f(C* p, int i)
{ return p->f(i); }
  - Overloaded
void f(int);
void f(double);
// wrapper functions
extern "C" {
    void f_i(int i) { f(i); }
    void f_d(double d) { f(d); }
}
- How do I include a C Header File?
  - System / Standard Library Headers +
  - Non-System Headers: Editable
#ifdef __cplusplus /* C compilers skip */
extern "C" {
#endif
/* Original Code of the Header */
#ifdef __cplusplus
}
#endif
  - Non-System Headers: Non-Editable
// In C++ header / source
extern "C" {
    #include "my-C-code.h" // C Header
}
- How do I use Pointers to C / C++ Functions?
extern "C" {
    typedef int (*pfun)(int);
    void foo(pfun);
    int g(int); // foo(g) is valid
}


Tutorial T06
Partha Pratim Das
Tutorial Recap
Objectives & Outline
Mixing C & C++
Rules
Scenarios
C / C++ Mixed Project
Data.h
Data.cpp
Data_Wrap.cpp
App.c
main.cpp
makefile
Execution
Call Trace
Advanced Mix
Tutorial Summary

Mixing C and C++ Codes: Scenarios: Recap

- How do I call a C function from C++?
extern "C" { void f(int); };
- How do I call a C++ function from C?
  - Non-Member
extern "C" { void f(int); }
  - Member
class C { /*...*/
    virtual double f(int);
};
// wrapper function
extern "C" double call_C_f(C* p, int i)
{ return p->f(i); }
  - Overloaded
void f(int);
void f(double);
// wrapper functions
extern "C" {
    void f_i(int i) { f(i); }
    void f_d(double d) { f(d); }
}
- How do I include a C Header File?
  - System / Standard Library Headers
  - Non-System Headers: Editable
#ifdef __cplusplus /* C compilers skip */
extern "C" {
#endif
/* Original Code of the Header */
#ifdef __cplusplus
}
#endif
  - Non-System Headers: Non-Editable
// In C++ header / source
extern "C" {
    #include "my-C-code.h" // C Header
}
- How do I use Pointers to C / C++ Functions?
extern "C" {
    typedef int (*pfun)(int);
    void foo(pfun);
    int g(int); // foo(g) is valid
}



Mixing C and C++ Codes: Scenarios: Recap

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

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

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

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

  - Member

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

  - Overloaded

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

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

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

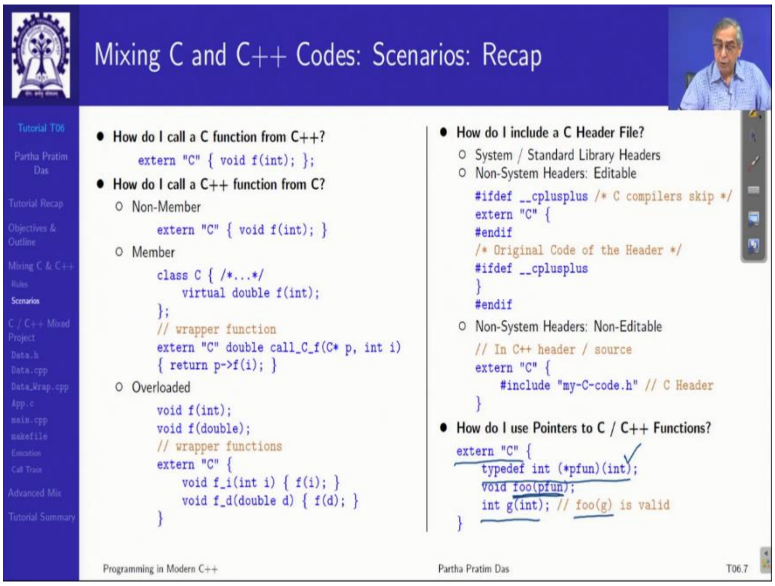
  - Non-System Headers: Non-Editable

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

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

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

Programming in Modern C++ Partha Pratim Das T06.7



Mixing C and C++ Codes: Scenarios: Recap

How do I call a C function from C++? extern "C" { void f(int); }; How do I call a C++ function from C? Non-Member extern "C" { void f(int); } Member class C { /*...*/ virtual double f(int); }; // wrapper function extern "C" double call_C_f(C* p, int i) { return p->f(i); } Overloaded void f(int); void f(double); // wrapper functions extern "C" { void f_i(int i) { f(i); } void f_d(double d) { f(d); } }

How do I include a C Header File? System / Standard Library Headers Non-System Headers: Editable #ifdef __cplusplus /* C compilers skip */ extern "C" { #endif /* Original Code of the Header */ #ifdef __cplusplus } #endif Non-System Headers: Non-Editable // In C++ header / source extern "C" { #include "my-C-code.h" // C Header } How do I use Pointers to C / C++ Functions? extern "C" { typedef int (*pfun)(int); void foo(pfun); int g(int); // foo(g) is valid }

Programming in Modern C++ Partha Pratim Das T06.7

So, these are the basic rules as we have seen, and then these are common scenarios, which certainly, will be required to start with. One is, how to call a C function from C++. Simple thing to do is to put that function in C++ code put that function within extern "C", which will tell the C++ compiler that this is a C function and the name will not be mangled and you will be able to call it.

How do I call a C++ function from C? The other way? So, a C++ function could be of many types. It could be non-member function, it could be member function, it could be overloaded function, and so on. So, the basic approach is the same that for a non-member function which is a global function, so, to say, you just again have to put that function within extern "C" to make sure that you have the you do not have the mangling of the name, because then the C code will not be able to call it because the C compiler will not understand.

For a member function naturally for a member function the things get a little difficult because for a member function you do not know. You know that any member function needs a hidden parameter, which is the identity of the object on which that member is being called, and that parameter is at this point it, naturally C does not know about this.

So, what we will have to do is to call a member function like say, in class C, if I have to call a member function f(), then I have to create a wrapper function for this and this wrapper is a basic technique of doing various function calls. So, we create a wrapper function, which besides taking the member function's original parameter will also take a parameter which is a pointer to that object.

Now, we will come to that as to how the pointer to an object be available in C because C does not have classes and so, on, so we will have to do another trick there, which will be exposed when developing the project. So, you have that pointer. So, this is basically the value of the this pointer so, using that in C++ you can call that function you can call that member function easily you can call virtual functions also in that way. So, with this wrapper, which a C function will be able to call, it will be able to invoke member functions in C++ classes.

Now, for overloaded functions if we have overloaded functions either in the non-member or in the member all kinds of things are possible. Again, the same trick will have to be done, we will create wrappers for them. And all of these will again be in extern "C" so that C can recognize it. So, that is the overall way to handle function calls across C and C++. To include the header files, the system header files, you do not have any special action because C++ standard library has already given the mechanism to include the C standard library, which you have already seen number of times.

For non-system header files, that is the header files that are written by the user all that we will need to do is to make sure that in C++ that header file, here functions in that header file must be within extern "C", otherwise C++ will not be able to handle, so both of these rules will apply.

So, you can do it in two ways. One is, if your header is editable then within the header itself, so this is the header code, you can just do a wrapper of the header, you can wrap the header in this extern "C" code, which you make guarded by __cplusplus macro, which means that this header can be uniformly used in C as well as in C++.

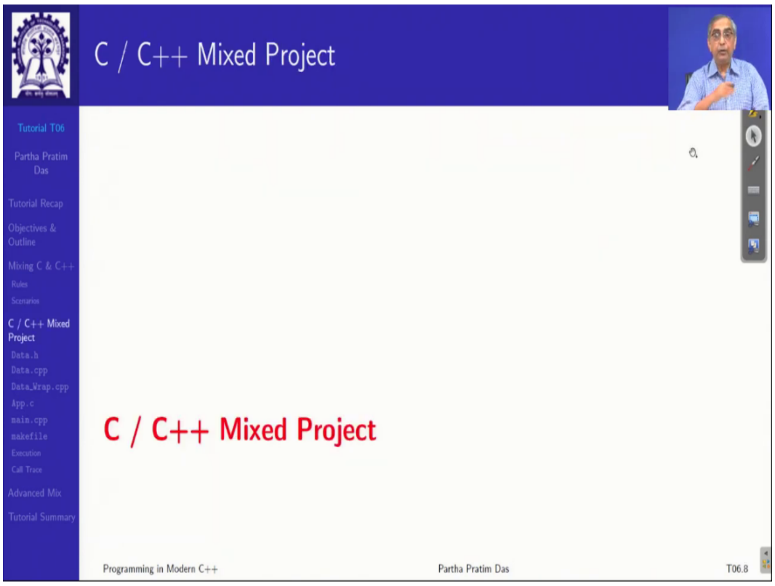
In C, these two do not will be skipped out by the CPP so you have the original C header of the library, and in C++ they will come in and put all your functions in the within extern “C” as you need. It is a very nice solution. Now, in case your header is not editable, then you cannot do such changes in the header.

So, in that case in the C++ code itself, you will have to wrap that header as a whole within extern “C”. In that case naturally, then there is no question of guard because this code will only go in the C++ in the C code obviously you will only have the simple header included.

Now, finally, to use pointers to functions, you have to make sure that again the everything including the typedef of the function pointer everything must be within extern “C”, unless you put this typedef also inside the extern “C”, the compiler will not be able to understand that this is going to be a function pointed to a C function so that the mangling rules should not be applied and therefore, you will have type errors, and such calls like when foo takes a function this function pointed and eventually takes a C function g() as a parameter will not work.

So, these are the scenarios that we had seen. The only scenario which we have not seen, which we will illustrate through the workout example is how do you basically interact with C++ objects from C, and how do you deal with that mixing.

(Refer Slide Time: 11:20)



C / C++ Mixed Project

Tutorial T06

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Mixing C & C++

Rules

Scenarios

C / C++ Mixed Project

Data.h

Data.cpp

Data_Wrap.cpp

App.c

main.cpp

makefile

Execution

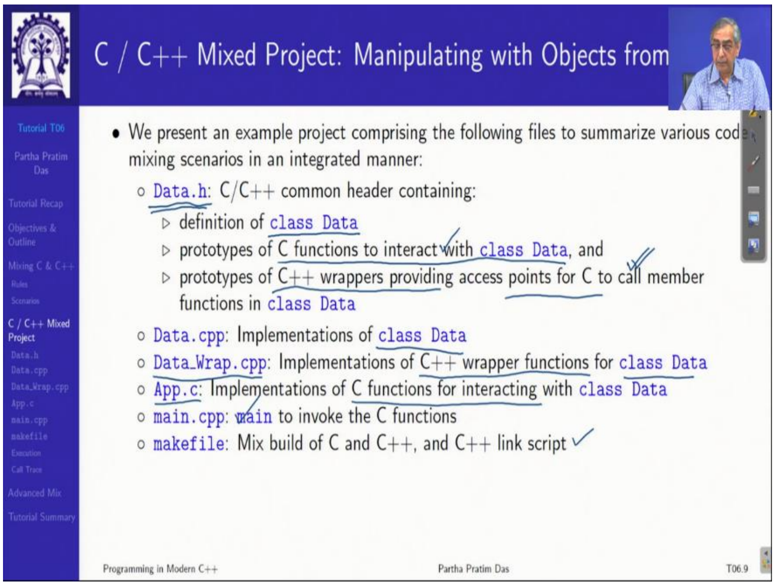
Call Trace

Advanced Mix

Tutorial Summary

C / C++ Mixed Project

Programming in Modern C++ Partha Pratim Das T06.8



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

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

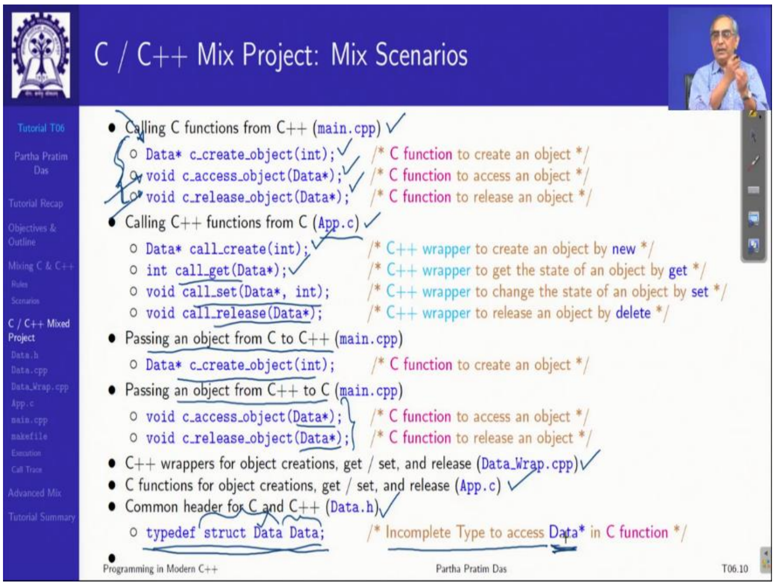
So, we will illustrate and put all these together in terms of a C, C++ mixed project. So, we present a project which comprises of a number of files. One is, this does not do anything meaningful, this is just to show the structure of the whole code C side as well as C++ side and how did the bridge and how does the calls across C to C++ and C++ to C work, and how does references made to objects in C++ from the C code will work.

So, Data.h is assumed to be a common header. This common header which will, which is kind of the bridge between the C part of the project as well as C++ part of the project this common header will be included in on both sides. So, which will include for C++ it will include the definition of a C++ class data. So, it will also have prototype function prototypes of C functions to interact with this data it cannot directly interact, but there are prototypes of interaction with the class data its members and so on.

It will also have prototypes of C++ wrappers to be provided to give access points for C to call the different members. Call the different member functions. These are the three kinds of things. We will see the actual details. Data.cpp will have the implementation of the class data which is understandable Data_Wrap.cpp will have the implementation of the C++ wrapper function for class data, these functions for class data.

App.c is implementation of C functions for interaction with class data, so this is conceived as if this is an application in C which you are integrating with C++. Main.cpp will invoke the C functions. And finally, we will show how to write a makefile to build make to build this mix of C and C++ codes with respective compilers and then linked with that C++ linker a complete demonstration.

(Refer Slide Time: 13:53)



C / C++ Mix Project: Mix Scenarios

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

Programming in Modern C++ Partha Pratim Das T06.10

So, here are the different main contents of different files. So, main.cpp calls C functions from C++ because main has been compiled by C++ so, it is a C++. So, what are the C functions, it can call. So, I am taking the basic object usage model that is you need to create an object you need to access an object and finally, the third is you need to release an object if you can do this then you can do everything else. So, three representative functions in C which main will be calling.

For App.c, App.c is written in C so, it will be calling C++ functions to get C++ resources equivalent of that C++ accesses, so it will have a function to create an object which is will be called in C++ to create an object. It will have a wrapper - C++ wrapper that will be called for the getting data then similar for setting data and finally for releasing it.

So, all that we are saying that from C perspective all that you need to do with the object is to instruct the appropriate C++ wrapper code to actually create an object and give you then call tell the wrapper code to do a get operation or a set. If you can do get and set you can do anything else because everything else is just an algorithm. Get and set is the fundamental as to whether you can read or you can write and so on. And finally, you should be able to instruct that now release this object.

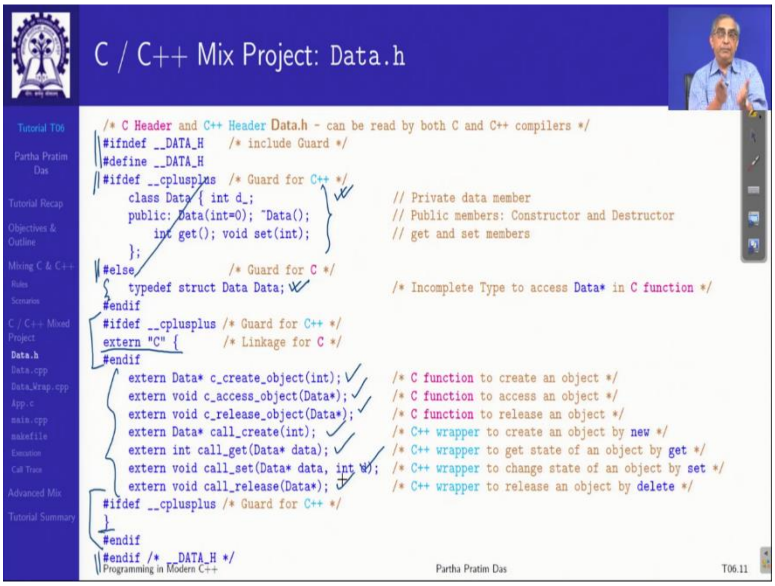
Now, you will have to pass an object from a from C to C++. So, we will see how we are doing that using this function in main. We have to pass objects from C++ to C which will again we will see how we do this in main. We have seen these functions. So, the first function creates, gets the object created by the C code, obviously, with a call back into the C++ and then gets it so, it is passed as a return value, whereas, in the other two in these two functions, you are passing an existing object pointer to C, so that C part can continue can do manipulations with that, do the release of that and so, on.

So, the C++ wrapper required for creation, release get, set will be in data wrap C function for creations release and get set is in App.c common header is Data.h, and in Data.h we will have a very peculiar type declaration called typedef struct Data Data. So, this first data is alias. So, we say that we are saying struct data is called data. So, if you have a struct data.

Now, if you define a structure, you need its fields and so on so only then it is a full definition complete definition. So, when you do not have that, then it is an incomplete type that is you do not have the fields or anything. So, you cannot instantiate a structure of this data type in C because it is an incomplete type it does not know the members or anything. The only thing you can do with the incomplete type is to have a pointer to that type.

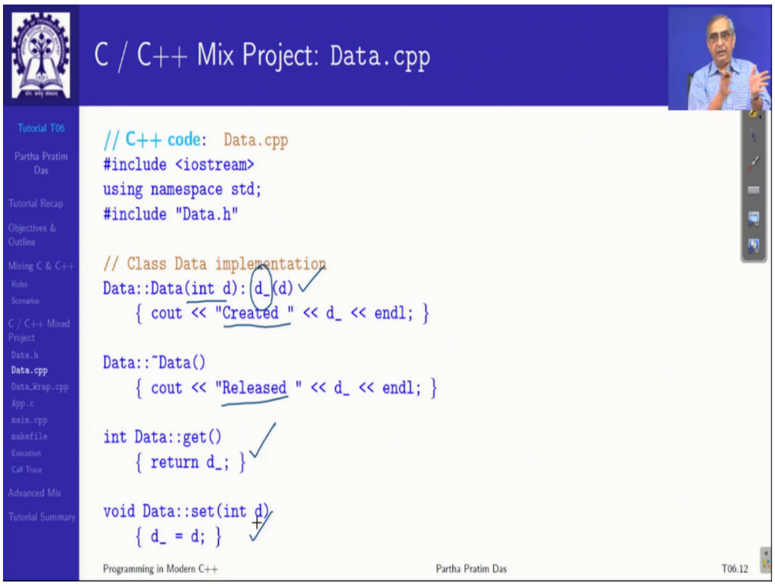
So, you can only deal with Data* pointed to this. So, the basic mechanism of communicating objects between C and C++ is in C++ it is class data, so it has a pointer of type Data*, which is nothing but an address. In C we deal that as a pointer to an incomplete struct data type. So, that the name remains same, but all that you are passing is just a pointer not the actual object because C does not understand object. So, that is the trick to actually sharing the object.

(Refer Slide Time: 18:44)



C / C++ Mix Project: Data.h

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



C / C++ Mix Project: Data.cpp

Tutorial T06

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Mixing C & C++

Rules

Scenarios

C / C++ Mixed Project

Data.h

Data.cpp

Data_Wrap.cpp

App.c

main.cpp

makefile

Execution

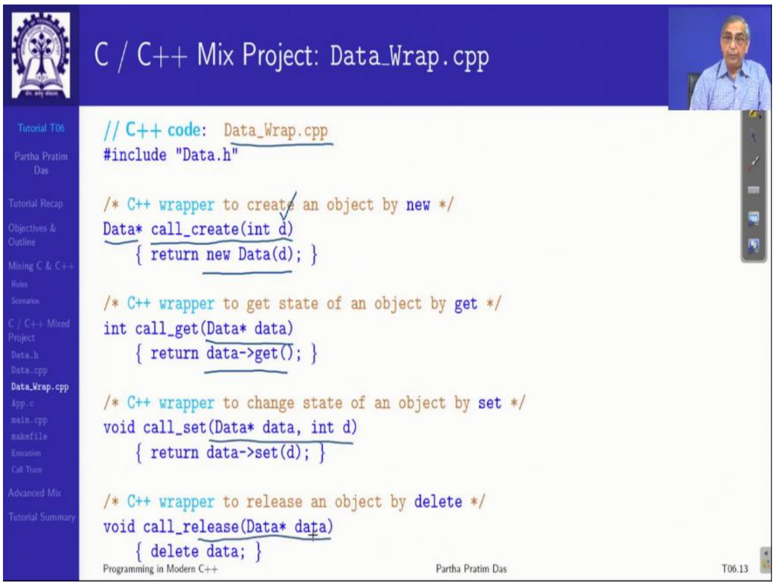
Call Trace

Advanced Mix

Tutorial Summary

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

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

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

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

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

Programming in Modern C++ Partha Pratim Das T06.12



C / C++ Mix Project: Data_Wrap.cpp

Tutorial T06

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Mixing C & C++

Rules

Scenarios

C / C++ Mixed Project

Data.h

Data.cpp

Data_Wrap.cpp

App.c

main.cpp

makefile

Execution

Call Trace

Advanced Mix

Tutorial Summary

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

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

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

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

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

Programming in Modern C++ Partha Pratim Das T06.13

So, with this let us look into the codes quickly. So, this is Data.h, I want to use it at both C and C++. So, firstly about the include guard then have a guard for C++ part. So, the only the C++ compiler will see this which is the definition of the class data, C part will not see this at all. Then what C else we have this.

So, this part only is the C compiler will see, C++ compiler will not see anything. So, C++ compiler sees class data C compiler sees this incomplete type. So, both of them know there is something of a type called Data. C in C++ it can build the object it can instantiate objects in C it cannot instantiate the object, but both of them can create a pointer to that type. So, that is the basic trick.

Then again you have you want to put everything else under extern “C”, so these are guards for that so that C++ compiler will see that C compiler will see nothing. And then it says that these are the extern functions, all C functions and the C++ wrappers, all of them are put here so that both C and C++ will know that these functions exist, both, all of them treat these as if they are un-mangled C names.

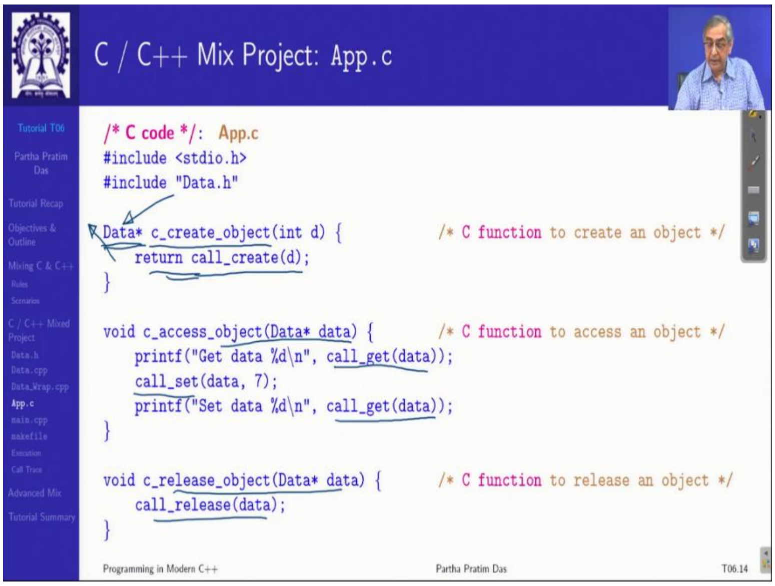
What is Data.cpp very simply, it is the implementation of the data class. So, you have a constructor I am assuming one parameter to that as a parameter is constructor. And instead of actually having an action I am just putting cout messages so that we can we can trace that. Get returns that only data member set will set it to header. This is the, if this functionality is there, then you can do anything in this class. Everything else is just writing extra codes for the different algorithms, so that is Data.cpp.

What is the wrapper? Wrapper is required to create to, get to, set to, release. So, what does the wrapper do is, if the wrapper is called to create then C actually is called this is this wrapper, so C is, C is a C++ wrapper, and C will call it as if it is a C function. So, it passes the parameter that you want for the object construction internally you construct that object, new Data(d).

Minute this is a C++ code, so it can use the operator new to construct the new object get that pointer of Data* type, returns that. Here when it is returning it is returning actually a pointer to the class data type. In C it is being treated as a pointer to the incomplete struct Data type, but pointer is nothing but just an address value, so everything falls in place. In C actually does nothing with that pointer C just keeps on you know passing it around, as an identity card.

So, similarly, forget data C gives you the pointer of the struct Data type, and here you think it is a pointer to the class Data type, and using that pointer you invoke the member function. Same thing you do for set. For release, you get the pointer and you call delete, so this is how the wrapper functions will work. And so, everything that you normally do in C++ now, you can do from C also.

(Refer Slide Time: 22:42)



C / C++ Mix Project: App.c

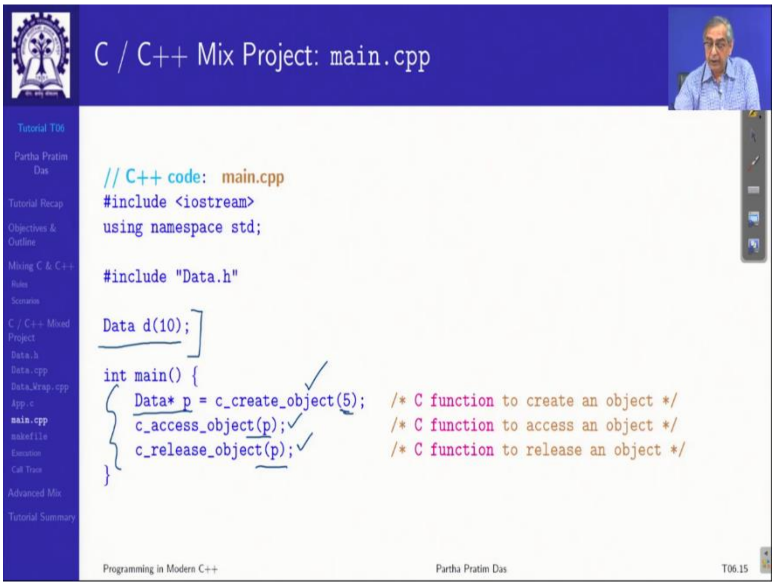
Tutorial T06

Partha Pratim Das

Tutorial Recap
Objectives & Outline
Mixing C & C++
Rules
Scenarios
C / C++ Mixed Project
Data.h
Data.cpp
Data_Wrap.cpp
App.c
main.cpp
makefile
Execution
Call Trace
Advanced Mix
Tutorial Summary

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

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

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

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

Programming in Modern C++ Partha Pratim Das T06.14



C / C++ Mix Project: main.cpp

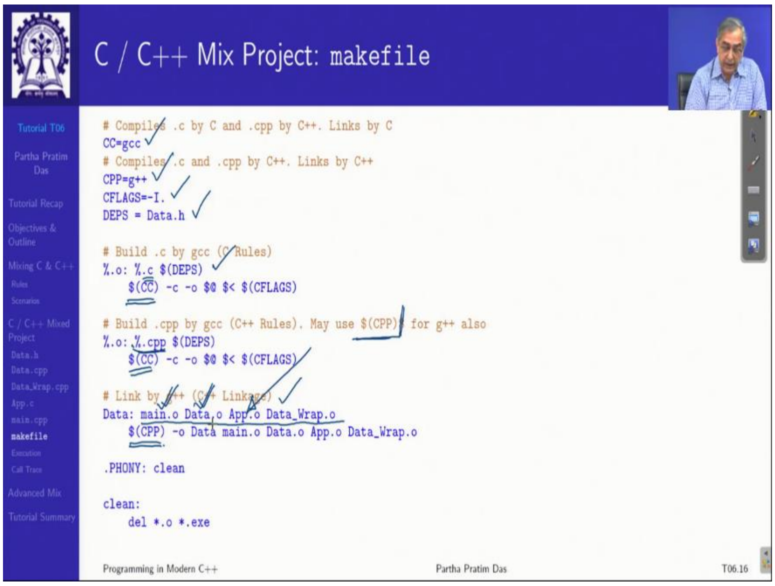
Tutorial T06

Partha Pratim Das

Tutorial Recap
Objectives & Outline
Mixing C & C++
Rules
Scenarios
C / C++ Mixed Project
Data.h
Data.cpp
Data_Wrap.cpp
App.c
main.cpp
makefile
Execution
Call Trace
Advanced Mix
Tutorial Summary

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

#include "Data.h"

Data d(10);

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

Programming in Modern C++ Partha Pratim Das T06.15



C / C++ Mix Project: makefile

Tutorial T06

Partha Pratim Das

Tutorial Recap
Objectives & Outline
Mixing C & C++
Rules
Scenarios
C / C++ Mixed Project
Data.h
Data.cpp
Data_Wrap.cpp
App.c
main.cpp
makefile
Execution
Call Trace
Advanced Mix
Tutorial Summary

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

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

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

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

.PHONY: clean

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

Programming in Modern C++ Partha Pratim Das T06.16

Now, in C how will you call it so app gives you those functions, so that you can actually use them in C. So, the C function here it is a C function of create object, which calls the C++ wrapper. It calls a C++ wrapper, it goes to C++, creates the object by new returns to that pointer and that pointer is given as Data*, this Data* remind you, what it has returned is a pointer to the class data type. What you return from here is a pointer to the struct data incomplete type, but is a pointer there.

Similarly, for object access, I have, here I have shown both calls to get as well as set, so you can access and manipulate objects in any form cin such a function. Finally, to release the object, you just call release data wrapper in C++, and that wrapper will do your job of doing the delete inside it.

What is main? In main, we have perceived only the simple things that you create an object, do something with it, and release that object. So, creating the object, it is calling the C function in App.c, passing the parameter, access, it calls so it that does get set whatever manipulations you want to do, and then you release.

So, it is calling the functions in App.c in the C domain, you have the algorithms for doing all this. So, you are doing all this and the in this case, the result is returned in these two cases, nothing is returned, but you work with the actual object pointer, this pointer of the instantiated object of data type.

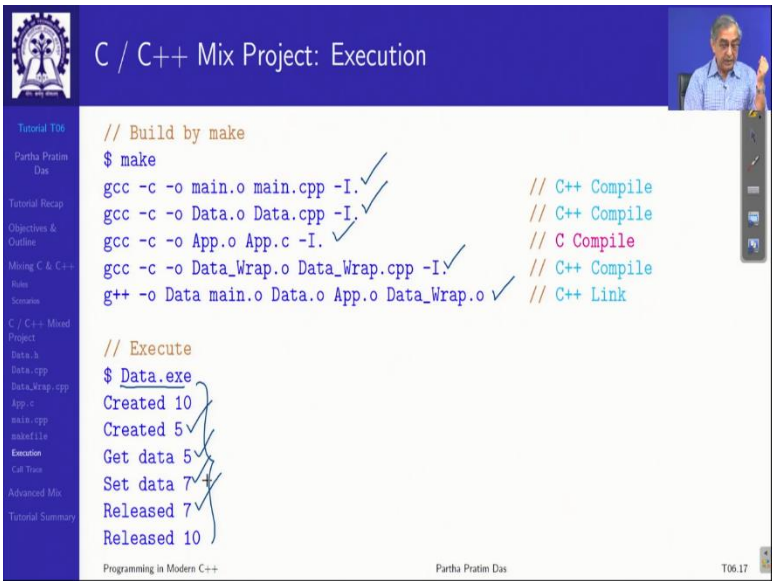
What I’ve included for that just for illustration to show you as to what is the speciality of the static initialization is, I have also shown a Data object d in the global scope outside of main. This is not only for interaction, but we will see how does it fall in place along with all the other interactions that are happening.

Coming to the makefile. So, we define two compilers now, because I want to compile C codes by gcc and C++ code by g++. I can only use GCC and just depend on the extent filename extension also, then, my compile flag is -i . that is the current directory, my dependencies Data.h, the same header being used on both sides. So, this is my building rule for object files, which you already know. I am using cc, so that the .c files will be built by gcc by the C compiler rules.

Then I have the built for cpp that is the C++ files. And I'm also using cc that is gcc because gcc builds cpp extension files as CPP compiler I can I may alternately use $(CPP) also for this $ is extra for G++ build directly it will have the same effect. Now, finally, to link so, I have .o files created from C built .o files created from C++ build now we have to put them together linked in a single executable.

So, I put everything main.o, data.o, app.o. So, of this, this, this, this are generated from the C++ compilation and this has generated from C compilation. We put all of that here, and now it is critical to do $(CPP) because I want to do a CPP linkage. So, if I do $(CC) here, then the linkage will not work it will give you error because it has, it will try to link, it does not know what to link with because it has got a mix. Then the clean, cleaning stuff and so on.

(Refer Slide Time: 27:36)



C / C++ Mix Project: Execution

Tutorial T06

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Mixing C & C++

Rules

Scenarios

C / C++ Mixed Project

Data.h

Data.cpp

Data_Wrap.cpp

App.c

main.cpp

makefile

Execution

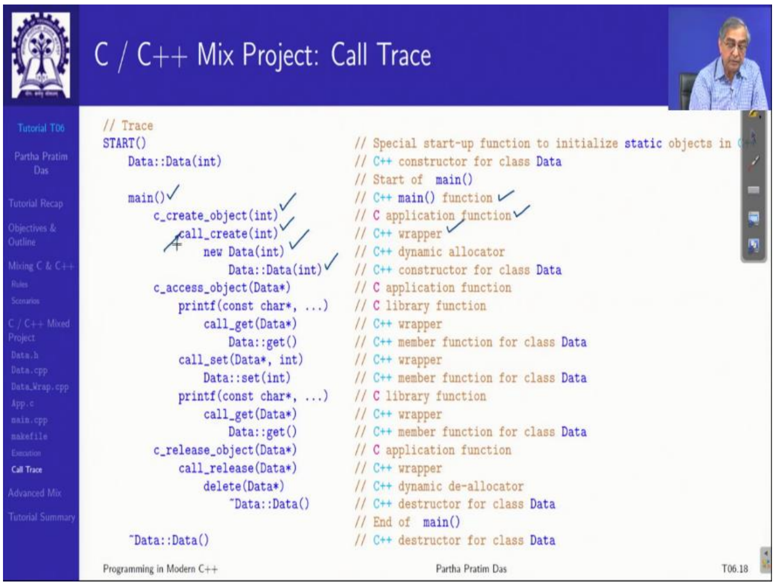
Call Trace

Advanced Mix

Tutorial Summary

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

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

Programming in Modern C++ Partha Pratim Das T06.17



C / C++ Mix Project: Call Trace

Tutorial T06

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Mixing C & C++

Rules

Scenarios

C / C++ Mixed Project

Data.h

Data.cpp

Data_Wrap.cpp

App.c

main.cpp

makefile

Execution

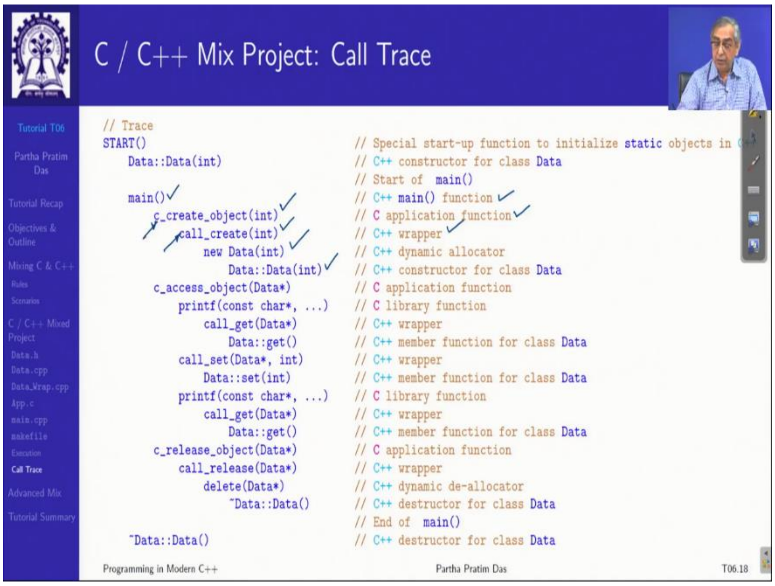
Call Trace

Advanced Mix

Tutorial Summary

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

Programming in Modern C++ Partha Pratim Das T06.18



C / C++ Mix Project: Call Trace

Tutorial T06

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Mixing C & C++

Rules

Scenarios

C / C++ Mixed Project

Data.h

Data.cpp

Data_Wrap.cpp

App.c

main.cpp

makefile

Execution

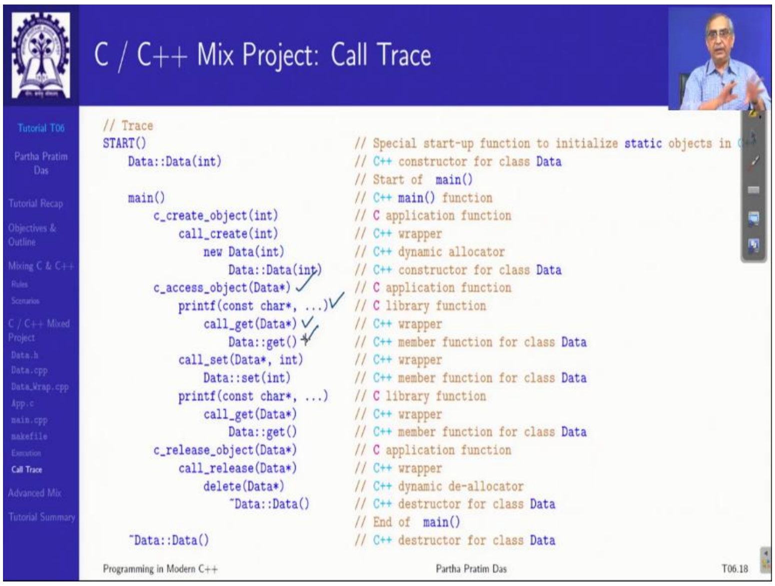
Call Trace

Advanced Mix

Tutorial Summary

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

Programming in Modern C++ Partha Pratim Das T06.18



C / C++ Mix Project: Call Trace

Tutorial T06

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Mixing C & C++

Rules

Scenarios

C / C++ Mixed Project

Data.h

Data.cpp

Data_Wrap.cpp

App.c

main.cpp

makefile

Execution

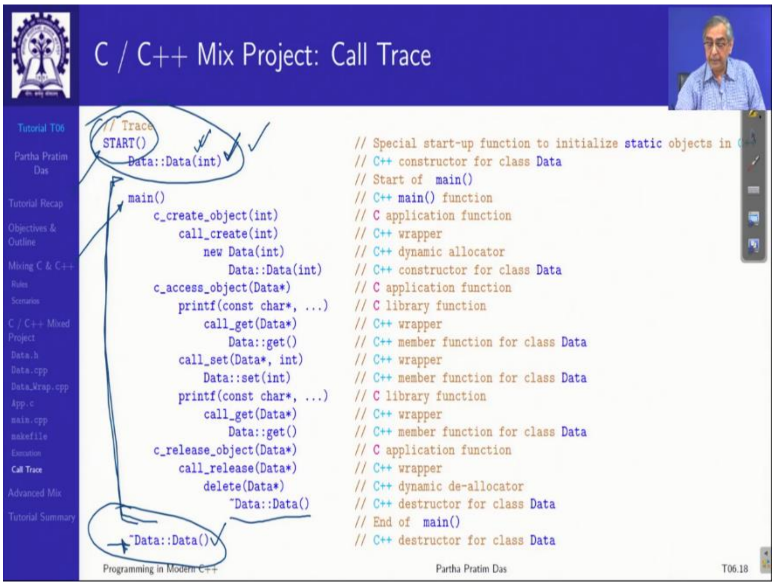
Call Trace

Advanced Mix

Tutorial Summary

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

Programming in Modern C++ Partha Pratim Das T06.18



C / C++ Mix Project: Call Trace

Tutorial T06

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Mixing C & C++

Rules

Scenarios

C / C++ Mixed Project

Data.h

Data.cpp

Data_Wrap.cpp

App.c

main.cpp

makefile

Execution

Call Trace

Advanced Mix

Tutorial Summary

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

Programming in Modern C++ Partha Pratim Das T06.18

Now, if you execute this and if you ask your make to tell you what it is doing, it does the C++ build of main.cpp then Data.cpp C build of App.c, C++ build of wrap Data_Wrap.o and the g++, C++ linkage of the entire code your Data.exe is ready.

I am talking on Windows platform so the name is Data.exe, your platform. And now we can see that this is what it prints. It says created 10. What was 10? 10 was the static object outside of main, so that is first created. Then the object that main asked to be created then get, set all this has happened, then the object created from main is released through the release and finally the static object is released. So, this is what goes on in the actual execution.

To get a better hold on this, and I would request you to spend time on this particular trace, and really understand this what we are saying is, if this is main, then main calls this function so indentation every indentation is a further function call. So, this is a main function then it calls C create object which is C application function you are in the C space of C that in turn calls C call create to actually create the object through new, so this is a C++ wrapper. That wrapper calls operator new, which is remains in the C++. This once this is done then new invokes the constructor remains in the C++ returns back to C to this C function and back to the main, and it has given the object pointer.

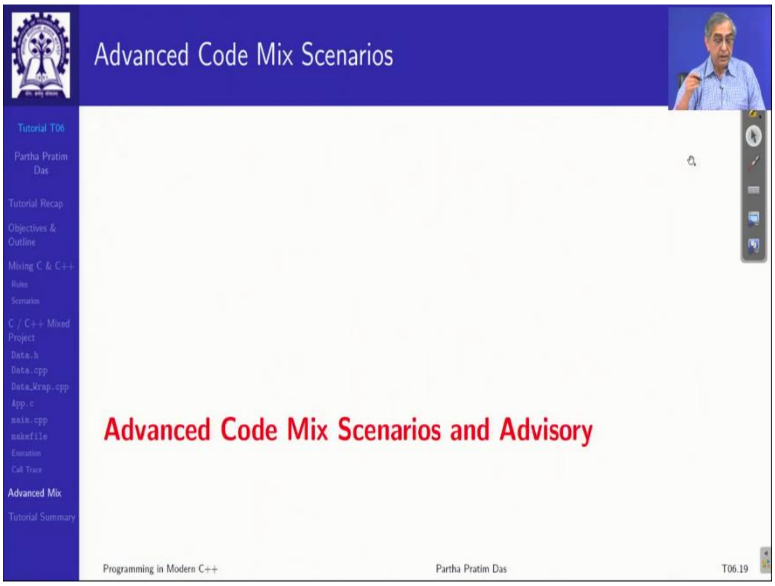
Similarly for the next call from main, you call the C wrapper it calls printf that calls get, that calls data in get, so you can see that in this way how seamlessly we are traveling between C and C++ and doing things. So, this is basically the whole idea of the mixing or the integration. So, this is the whole thing that happens in main. Interestingly, you see that this constructor is called even before main was called. So, who called that constructor?

So, basically the compiler has provided a special function for start-up. I have called it start, this is not a publicly exposed name, so the compiler will have its own name. So, I do not know what name it will give, but I have just called, as if there is a function which does this static function calls, initialization calls before main starts.

Then the main starts and does everything, the final object is released from main and the control goes back. Where does the control go back? Control goes back to the start function, which somehow remembers that it had called a constructor of Data, as it remembers in the normal scope, it does remember in the global scope also. There are some more tricks into this, which at a later point of time, I will talk of. There is some registration and stacking of function pointers that are required here, but it somehow remembers that this constructor was called, so it calls that destructor.

So, it clearly tells you as to why we need to compile you know link main with C++ because this part of the mechanism will not be supported in the C linker, C linker does not know about it. In terms of C linker, whenever main is loaded it directly starts. And here, you load your exe, what starts is a start function then this initialization then main, then deinitialization and the whole process ends.

(Refer Slide Time: 32:03)



Advanced Code Mix Scenarios

Tutorial T06

Partha Pratim Das

Tutorial Recap

Objectives & Outline

Mixing C & C++

Rules

Scenarios

C / C++ Mixed Project

Data.h

Data.cpp

Data_Wrap.cpp

App.c

main.cpp

makefile

Execution

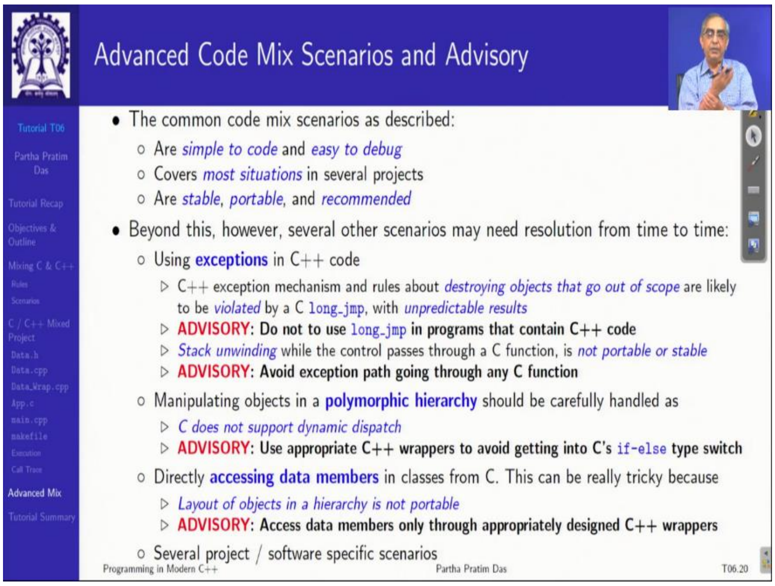
Call Trace

Advanced Mix

Tutorial Summary

Advanced Code Mix Scenarios and Advisory

Programming in Modern C++ Partha Pratim Das T06.19



Advanced Code Mix Scenarios and Advisory

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

Programming in Modern C++ Partha Pratim Das T06.20

So, this is a story about the general mixing. There are several other scenarios, I will not go through, I have just documented them. Because I cannot go through them because you may not understand much of them, you may not be familiar with it. The main issue lies with the fact that if you use exceptions then be very careful in terms of how you mix it.

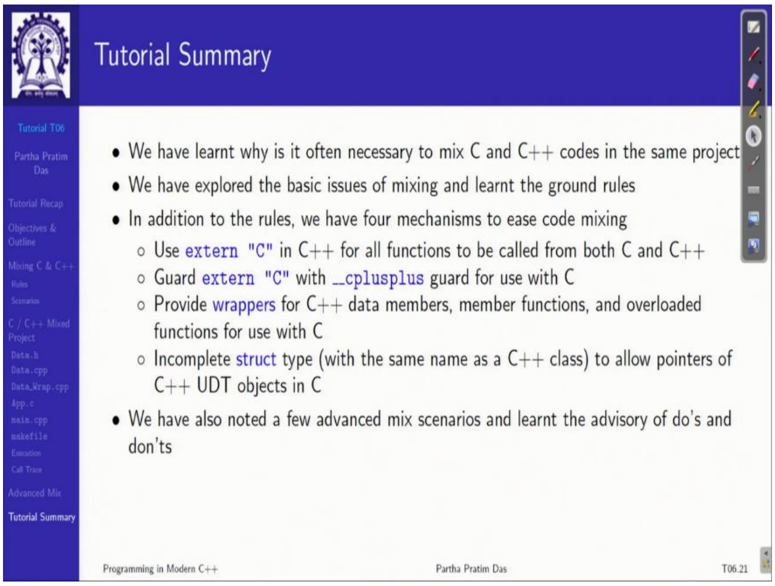
Because see, exceptions when you throw an exception the unwinding of stack involved which the C++ implements, but a C code will not implement. So, if you have, I mean, like we have seen C function calling C++ function that in turn calling C functions C++ functions like that. So, in the call stack if you have C functions coming in, then when you throw from a C function, C++ function and it has to pass through the C function, it might give you some unexpected results.

Similarly, if you do a long jump from C, it will have unexpected results because C++ does not support the long jump directly. So, the basic advice is do not use long jump, avoid exception parts that has C functions. Even for a polymorphic hierarchy, you will have to be very careful because in a polymorphic hierarchy that is when you have virtual functions then you need a dynamic dispatch, as you have learned in the module, that you need a dynamic dispatch.

So, you can handle that by C++ wrapper, but as you can see now you will have all different wrappers, for all different object types and so on. And therefore you will have to handle that in C and that will become kind of a if else switch which is what you do not want. And never access data members directly from C. You have got a pointer and you think that I will ask this the data member syntactically it will go through, but semantically it is a disaster because the object layout has known to see is different from what it actually is in C++.

So, if you directly want to in C if you want to you access the data members, your address calculation for the data members would be all wrong, so do not doing it, always use wrappers for doing these tasks.

(Refer Slide Time: 34:30)



Tutorial Summary

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

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

So, this is the closing of this tutorial on mixing C and C++ code. We have learned why you are doing this, why we need to mix, what are the ground rules. And we have by now four mechanisms use extern "C", guard extern "C" with __cplusplus. Provide wrapper for any C++ interaction, data members, member functions, overloaded functions and what we have specifically learned in this tutorial is you can navigate with the object between C and C++ using an incomplete struct type by the same name as of your C++ class in the C code.

So, that was, that brings us to the conclusion of these two-part tutorial. I hope, you enjoyed it and you will be able to use it in your professional life. Thank you very much, and see you in the next module or tutorial.