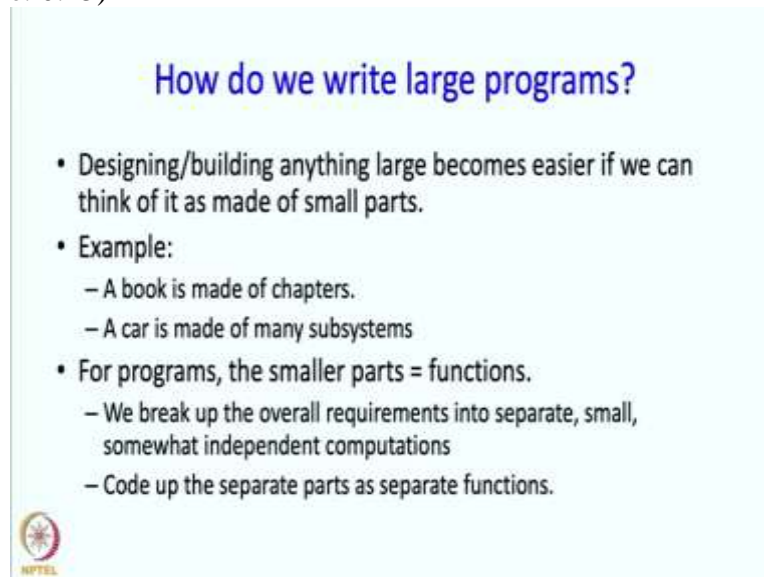


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture 13 Part 1 – Program Organization and Functions (Introduction)


Hello and welcome to the NPTEL course on An Introduction to Programming through C++. I am Abhiram Ranade, the topic for this lecture is program organization and functions and the reading is from the chapter 11 of the text. So the central question that we are going to answer in this lecture sequence is how do we write large programs.

(Refer Slide Time: 0:45)



How do we write large programs?

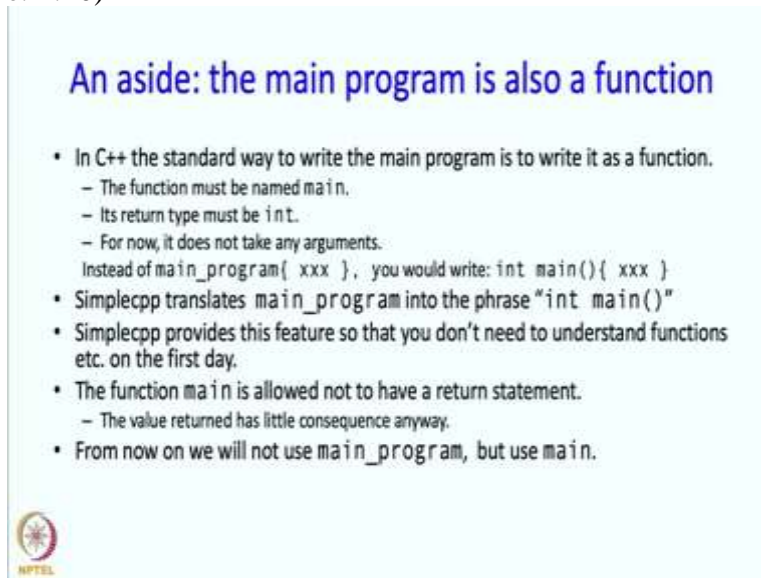
- Designing/building anything large becomes easier if we can think of it as made of small parts.
- Example:
 - A book is made of chapters.
 - A car is made of many subsystems
- For programs, the smaller parts = functions.
 - We break up the overall requirements into separate, small, somewhat independent computations
 - Code up the separate parts as separate functions.



Now designing or building anything large becomes easier if we can think of it as being made of small parts. So for example, a book is made of chapters and if we say that look, let us concentrate on chapter 1 now then concentrate on chapter 2, then the whole thing becomes easier to think about.


Say you are designing some object like a car, then a car has many subsystems and you do not design everything together, you do not have everything in your mind at the same time. But you say that look, let me now design the engine, let me now design the electrical subsystem and we sort of break it up into small pieces and work on the pieces one at a time or at least one person works on a piece at a time. For programs such smaller units are functions. We break up the overall requirement into separate, small, somewhat independent computations and code up these separate parts as separate functions.

(Refer Slide Time: 2:10)



An aside: the main program is also a function

- In C++ the standard way to write the main program is to write it as a function.
 - The function must be named `main`.
 - Its return type must be `int`.
 - For now, it does not take any arguments.Instead of `main_program{ xxx }`, you would write: `int main(){ xxx }`
- Simplecpp translates `main_program` into the phrase "`int main()`"
- Simplecpp provides this feature so that you don't need to understand functions etc. on the first day.
- The function `main` is allowed not to have a return statement.
 - The value returned has little consequence anyway.
- From now on we will not use `main_program`, but use `main`.



Now we are going to think of a program as being made up of functions and we have already said that even if our main program which has been looking different from functions so far is actually also just a function. So let me make that clear right away. In C++, the standard way to write the main program is to write it as a function. The function must be named 'main' and its return type must be int. Why is it int? That is really for some historical reasons. The main program doesn't really returning anything but for some historical reasons it has stuck that its return type must be int.

Ok and for now the main program is not going to take any arguments. So instead of `main_program` and a body containing appropriate statements, in C++ you really should be writing `int main()`, so `main` which does not take any arguments and the body containing whatever statements.

Ok, simple CPP is just doing a translation for you, ok so when you write main program simple CPP translates this main program into this text. There is a feature in C++ called preprocessor macros which are being used over here in case you are curious but for this course you do not need to understand preprocessor macros but if you are curious they are discussed in the book.

Ok, so why did we do this? So simple CPP provides this feature so that on the very first day you do not need to understand what is a function, what are arguments to a function and why is it int or should we care that it is an int. We did not want you to ask all these questions or we did not want you to worry about all these questions and therefore we made this thing

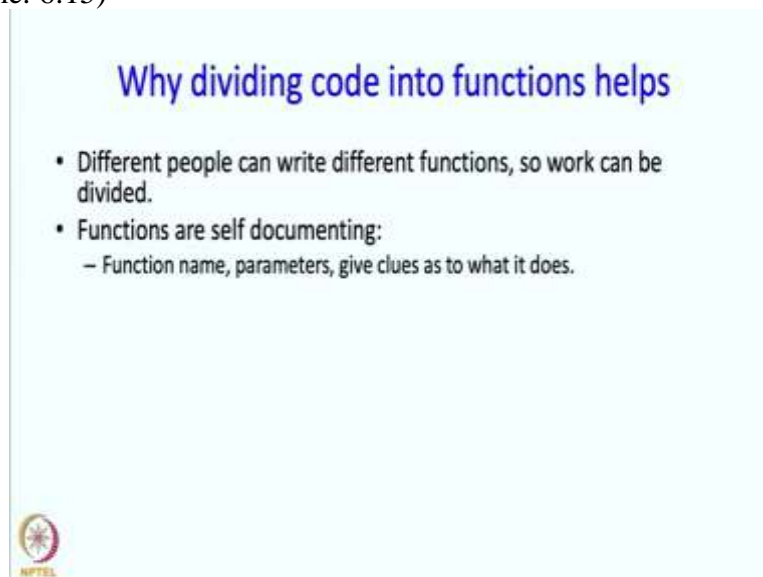
called `main_program` and which we convinced ourselves the simple CPP was translating into what is actually required.

Ok, but now that you know functions or by the way the function `main` is allowed not to have a return statement, so this is the concession that is given for `main` because the value returned is not of any consequence anyway, so we have not been returning the value inside inside the `main` program.

Now that you know what the thing is really required in C++, we are going to stop using this phrase '`main_program`', so we will write `int main` and then the body and this is how we will write our `main` program as a function whose name is `main`.


Your compilation procedure will still work if you are using an IDE. If you hit the compile and run button or just the compile button, that will work. Ok, if you are using `S++` command under Linux that will still work, ok and I will encourage you to use `int main` because after all that is sort of what will be available if you go outside simple CPP and anyway it is not really a major convenience at this point to be able to write `main` program.

(Refer Slide Time: 6:15)



Why dividing code into functions helps

- Different people can write different functions, so work can be divided.
- Functions are self documenting:
 - Function name, parameters, give clues as to what it does.



```
int gcd(int m, int n){
    if (m % n == 0) return n;
    else return gcd(n, m % n);
}

/*
Precondition: m, n > 0
Post condition: GCD(m,n) returned.
*/
```


Ok, now that you understand what functions are, ok so let us come back to the question of dividing code into functions. So, why does it help? Ok, so if you if you have a lot of work to do you would like to get many people to do it. So, similarly if you want to write a big program, presumably you want to assign it to many people and certainly if you give different functions to different people then that is easier to, that makes it easier for people to collaborate.

Another very interesting feature of functions is that they are self-documenting. What do I mean by this? The function name, parameters give clues as to what the function is doing. If you just write the code inside your main program, then there is no such clue. So, going back to the functions that we had if we write `int gcd`, then the very fact that we have a name over here tells us that look something like a gcd is being computed and 'm' and 'n' are there, so it says that look, ok maybe the gcd of 2 numbers is being computed something like that.

(Refer Slide Time: 7:22)


Why dividing code into functions helps

- Different people can write different functions, so work can be divided.
- Functions are self documenting:
 - Function name, parameters, give clues as to what it does.
 - Even better if you write the specification as a comment..
 - In a long main program it is not obvious where/how to put comments.
- "Write a little bit of code, test it, write some more, ..."
- Test your program one function at a time
- A function is a good way to package and share code.
 - If you have written some code and want to give it to someone else, give them a file containing a function containing your code.



```
void tree(int L, double rx, double ry,
         double H, double W){
    // L levels, Root at (rx,ry), Height H, Width W
    if(L>0){
        Line left(rx, ry, rx-W/4, ry-H/L);
        Line right(rx, ry, rx+W/4, ry-H/L);
        right.imprint();
        left.imprint();
        tree(L-1, rx-W/4, ry-H/L, H-H/L, W/2);
        tree(L-1, rx+W/4, ry-H/L, H-H/L, W/2);
    }
}

/*
Preconditions: L >= 0
Post conditions: Drawing happens in rectangle with
top left at rx-W/2,ry-H, bottom right at 2rx,ry
*/
```



Ok and here the things are even clearer, ok that a tree is being drawn or something like something is happening to something involving a tree. Even that is better than just putting this kind of a code inside your program. Ok and of course if you are going to recurse, functions are of course essential but even if you are not going to recurse just the fact that there is a name over here forces you to say something about what is being done. Of course, all this works only if you are choosing the name nicely and we hope that you choose the name nicely, we hope that you choose the names of the parameters nicely.

So for example, you really could say over here int level not just 'l'. I have been using level over here because I want to put this inside a slide and inside a slide if you put big names then it would not fit in the slide but really good programmers will use much longer names. So, you

could even say instead of rx, you could say root x if you want or x root whatever whatever style you prefer. And when you write a function there is a natural expectation that you should write the pre-conditions and the post conditions.

So we have written pre-conditions and post conditions here in a very compact manner and here in a little bit more descriptive manner. In fact, this last comment says a little bit more as to what area what specific area the picture is going to fill out.

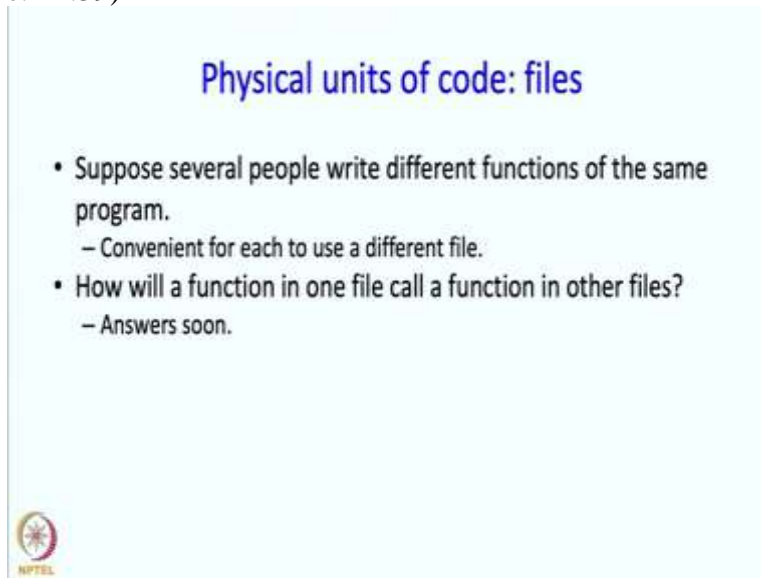
So what happens is that in the long main program it may not be obvious as to where as to where and how to put the comments, whereas in a function there is a proper place for it, there is a natural place for it, ok and I cannot overemphasize the need to document your code. As I said earlier, your code will be written will be read by you later on and you would want to know what it does and believe me you will forget why you wrote some piece of code and of course if somebody else is going to read the code, they will need to know why you are writing, what you are writing and so functions say that in this small piece of code this is what is going to happen, ok. The names implicitly say what is going to happen and furthermore if put in comments giving the pre-conditions and post conditions, great.

Functions also allow you to adopt a strategy. Write a little bit of code, test it, write some more, so basically you can test your program one function at a time and you do not have to write the entire program and then test the whole thing. That is really tricky to do because you will not easily know where the error is. If you write a small function and it makes an error, it is much easier to fix the error. And finally a function is a good way to package and share a code.

If you have written some code and you want to give it to someone else, you can give them a file containing a function which contains the code. Ok, so a function is sort of is a nice wrapper into which you can put code. So in some sense a function is a logical unit of code, the physical unit of code. The physical units are files. If you want to give anything like code to anyone, you have to give a file and of course if that has to make sense it had better it is nice to put a function in it rather than sort of code by itself does not make sense.


Code just a loop by itself or just a declaration by itself does not make sense whereas a function is kind of a nicely packaged piece of code but that is sort of logical packaging. Physically you have to give a file. So, that raises a bunch of questions.

(Refer Slide Time: 11:39)



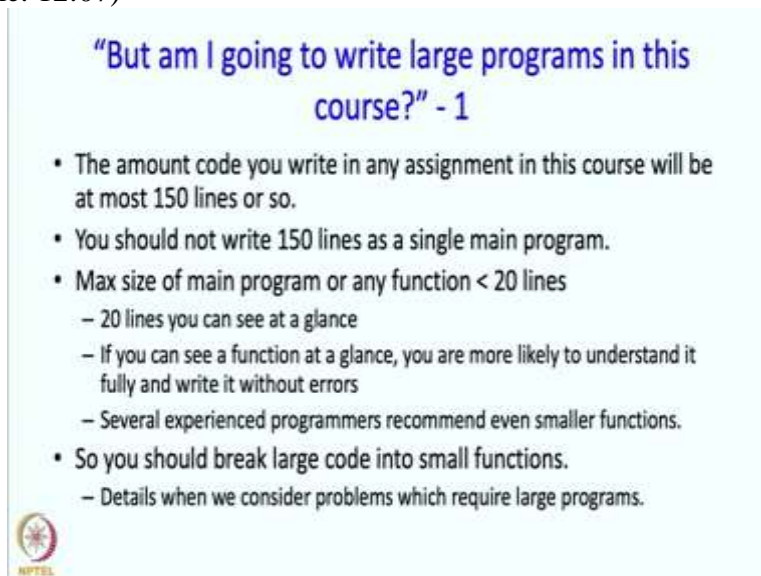
Physical units of code: files

- Suppose several people write different functions of the same program.
 - Convenient for each to use a different file.
- How will a function in one file call a function in other files?
 - Answers soon.




So, suppose several people write different functions of the same program. It is convenient for each person to use a different file and so the question that is raised is, how will a function in one file call a function in other files? So, we are going to answer these questions fairly soon.

(Refer Slide Time: 12:07)



"But am I going to write large programs in this course?" - 1

- The amount code you write in any assignment in this course will be at most 150 lines or so.
- You should not write 150 lines as a single main program.
- Max size of main program or any function < 20 lines
 - 20 lines you can see at a glance
 - If you can see a function at a glance, you are more likely to understand it fully and write it without errors
 - Several experienced programmers recommend even smaller functions.
- So you should break large code into small functions.
 - Details when we consider problems which require large programs.



Now you may be thinking at this point, look in this course am I going to be writing large programs? Or is this discussion about large programs really not relevant for this course? Ok, so some answers are due. So it is true that you are not going to be writing very large programs in this course. So any program that we ask you to write will be at most 150 lines. Most of the time they will be fairly short, so maybe there will be 20 lines. A 20 line program does not, can fit in the file and does not, it probably does not need to be divided into small functions. Ok, on the other hand a 150 line program should be divided.

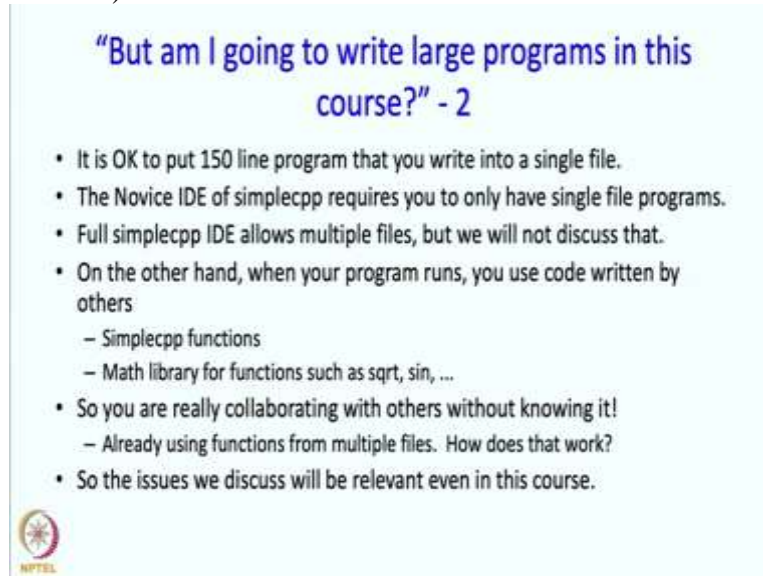
Ok. So, you do not write 150 lines as a main program, it is too long, it is too hard to figure out what is going on. So you break it up into pieces and the maximum size recommended for the main program or for a re-function is less than 20 lines. Why 20 lines? Because 20 lines you can sort of see at a glance. Anything bigger you sort of have to move your head around or you have to at least move your eyes around and if you are not seeing things at a glance then it is harder for you to sort of keep everything in your mind.

The bigger the chunk that you have to keep in your mind the harder it is for you to work with it. So if you can break things into pieces which are about 20 lines then that is great. Yeah, so you are more likely to understand that code fully and write it without errors. So again, you may notice that I am really worried about making errors when I write programs. And that is for two reasons, the first reason is that if I want to write program that I write a program that I give to someone else, I want it to run correctly because I know that that person might be using that program for something really important and I would be failing in my duty if I if my program does not work correctly.

The second reason I am worried about errors is that experience has thought me that I will make errors when I write programs and I should be prepared for that, I should be I should take as many precautions as possible and I should not become proud that oh, I am going to write this program without errors in the first shot. Sorry, it does not work like that for most people, so be a little humble, write the program, write it very carefully, write it by cultivating the style which prevents you from making errors as much as possible. Ok but anyway be willing to test it and so yes, I will be pre-occupied about making errors and I will be suggesting strategy so that you do not make errors and breaking things into small pieces is one such strategy.


Actually, many experienced programers even recommend smaller size for functions longer than 20 lines. Ok, yeah, so now how do you break code into small functions? So, that is not quite what we are going to talk about here, ok that is something that we will consider when we write large programs, when we ask you to write programs. At that point we will suggest how to break things into small pieces. Ok, the consideration this time is a little bit different. Ok, so this is this is the second consideration as to why we are talking about large programs.

(Refer Slide Time: 16:14)



“But am I going to write large programs in this course?” - 2

- It is OK to put 150 line program that you write into a single file.
- The Novice IDE of simplecpp requires you to only have single file programs.
- Full simplecpp IDE allows multiple files, but we will not discuss that.
- On the other hand, when your program runs, you use code written by others
 - Simplecpp functions
 - Math library for functions such as sqrt, sin, ...
- So you are really collaborating with others without knowing it!
 - Already using functions from multiple files. How does that work?
- So the issues we discuss will be relevant even in this course.

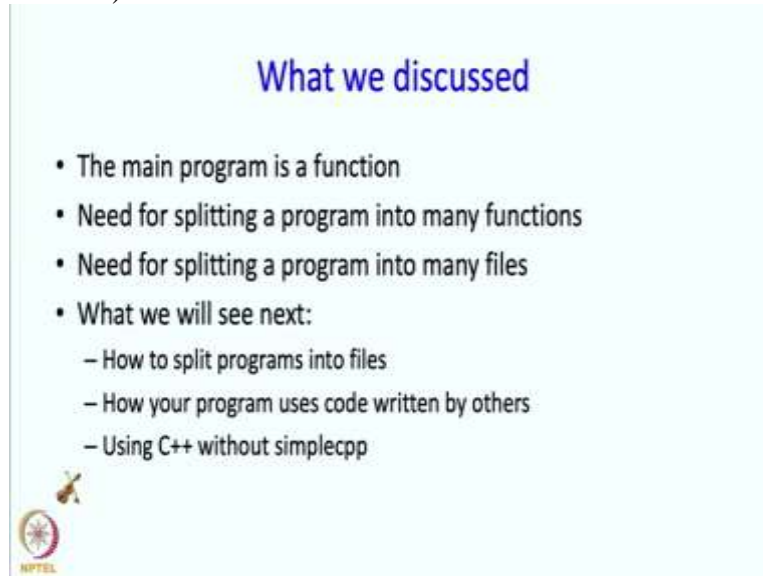


So as we said it is ok to put a 150 line program into a single file but not larger programs and again we may say that look, ok so we are not going to write bigger than 150 line program, so why are we bothering? In fact, yes, the novice IDE of simple CPP requires you to only have a single file program.

Ok, the first simple CPP IDE on the other hand does allow multiple files, but we are not going to discuss it. Ok, so then why are we talking about all this? Here is the point. When you write even your simple 15-20 line program, you will be using code which is written by others. So effectively, what you call your program is going to be a large program. Effectively, you are cooperating with others who have written the simple CPP functions for you. So your program really contains all those simple CPP functions effectively. The math library functions such as square root, sin, cos whatever that you are using have been written by somebody else and they have to be a part of your program if your program is making calls to square root. So in some sense you should know how a large program works, how a program which exists across several files, how do you put that together? And that is really what this part is about.

So you are collaborating with others without knowing it and you are using functions from many files, so you really should know how all that works. So to this extent even though you are not writing large programs what we discuss will be relevant to you.

(Refer Slide Time: 18:15)



The slide has a light blue background. At the top center, the title "What we discussed" is written in a dark blue font. Below the title is a bulleted list of five items. The first three items are "The main program is a function", "Need for splitting a program into many functions", and "Need for splitting a program into many files". The fourth item is "What we will see next:", followed by three sub-bullets: "– How to split programs into files", "– How your program uses code written by others", and "– Using C++ without simplecpp". In the bottom left corner of the slide, there is a small icon of a bee and the NPTEL logo.

- The main program is a function
- Need for splitting a program into many functions
- Need for splitting a program into many files
- What we will see next:
 - How to split programs into files
 - How your program uses code written by others
 - Using C++ without simplecpp

Ok, so what have we discussed in this last segment? We have said that the main program is a function and then we have articulated the need for splitting a program into many functions and we have articulated the need for splitting a program into many files. Next, we are going to see how to split the programs into files and how your program will effectively use code written by others and finally, we will see how you can use C++ without simple CPP. So we will take a break.