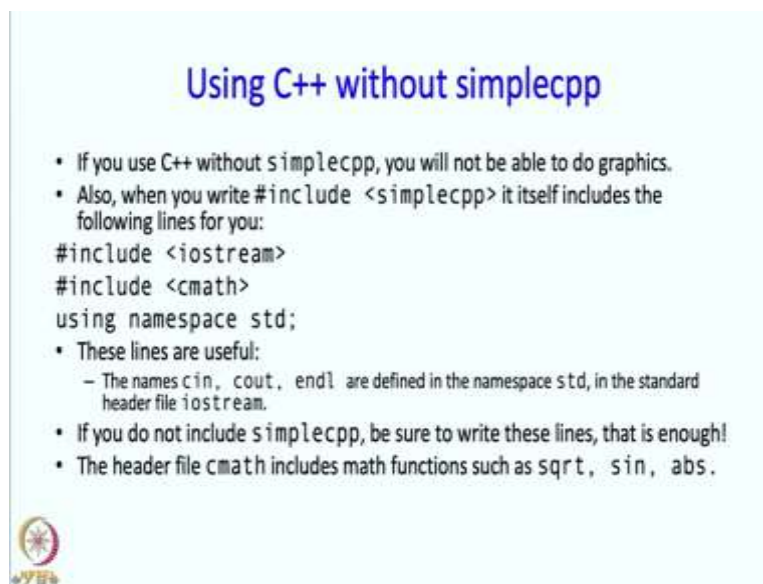


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture 13 Part – 4: Program Organization and Functions
(How to use C++ without simple CPP)

Welcome back, in previous segment we discussed namespaces and the using directive. In this segment we are going to talk about how we can use C++ without simple CPP and we will also conclude this entire lecture sequence. So, how do you use C++ without simple CPP?

(Refer Slide Time: 0:44)



Using C++ without simplecpp

- If you use C++ without simplecpp, you will not be able to do graphics.
- Also, when you write `#include <simplecpp>` it itself includes the following lines for you:
`#include <iostream>`
`#include <cmath>`
`using namespace std;`
- These lines are useful:
 - The names `cin`, `cout`, `endl` are defined in the namespace `std`, in the standard header file `iostream`.
- If you do not include simplecpp, be sure to write these lines, that is enough!
- The header file `cmath` includes math functions such as `sqrt`, `sin`, `abs`.

So, if you want to use it without simple CPP, first of all you will not be able to do graphics, but that is not all. When you write hash include simple CPP that itself includes the following lines for you. So it includes a line called, line with goes `#include<iostream>` and also a line which goes `#include<cmath>` and it also includes this line `using namespace std`, these lines are useful. The name in the `cin` the `cout` and `endl` all related to input output are defined in the namespace `std` and this definition happens in the standard header file `iostream`.

So therefore, you need to include `iostream` so that you get these names. And if you want to use them directly without having to write `std::cin`, you also need to have this line `using namespace std`. So, if you do not include CPP, if you do not have this line then you should have all these lines and you should have the line `cmath` because if you want to use functions like `sqrt` and all of that, `sqrt`, `sin`, `abs`, whatever.


(Refer Slide Time: 2:22)

Simple example: Using C++ without simplecpp

```
#include <iostream>      #include <iostream>
using namespace std;

int main(){
    int n;
    cin >> n;
    cout << n*n*n <<
        endl;
}

int main(){
    int n;
    std::cin >> n;
    std::cout << n*n*n <<
        std::endl;
}
```



So let me give you a very simple example of how you can use C++ without simple CPP. So you might say `#include<iostream> using namespace std, int main, int n, cin n` so read `n` and say print out it is cube. So, this could be really simple program which you could write without having to use the simple CPP features. But because simple CPP was including these things and because you need to use `cin`, and `endl`, you need to have these lines. Here is another way of writing the same thing, so you could write `iostream, int main, int n`.

And now because you did not write `namespace std`, you will have to write `std` using `namespace std`, you would have to write `std::cin` and not just `cin` and likewise `std::cout` and even `std::endl`. So that is basically how you are going to use C++ without using simple CPP. And of course had you needed to you use any math functions, you would also have to include `#include<cmath>`.

(Refer Slide Time: 3:52)

Demo

- `withoutSimplecpp.cpp`
 - Uses using directive
- `withoutSimplecpp2.cpp`
 - Does not use the using directive
- These may be compiled using the basic C++ compiler, without loading any of the simplecpp header files and libraries.



```
File Edit Options Buffers Tools C++ Help
#include <iostream>
using namespace std;

int main(){
    int n;
    cin >> n;
    cout << n*n*n << endl;
}
```

U:-----F1 withoutSimplecpp.cpp All L1 (C++/L Abbrev) 12:31PM 1.23

```
~/Desktop/nptel/week5 : %
emacs -nw recursiveGcd.cpp

[1]+ Stopped emacs -nw recursiveGcd.cpp
~/Desktop/nptel/week5 : s++ tree.cpp
+ g++ tree.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week5 : s++ tree.cpp
+ g++ tree.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week5 : ./a.out
~/Desktop/nptel/week5 : cd ../week6
~/Desktop/nptel/week6 : ls
Lec6.2.pptx          withoutSimplecpp.cpp  withoutSimplecpp2.cpp
a.out*              withoutSimplecpp.cpp~
~/Desktop/nptel/week6 : open Lec6.2.pptx
~/Desktop/nptel/week6 : %
emacs -nw recursiveGcd.cpp (wd: ~/Desktop/nptel/week5)

[1]+ Stopped emacs -nw recursiveGcd.cpp (wd: ~/Desktop/nptel/w
eek5)
(wd now: ~/Desktop/nptel/week6)
~/Desktop/nptel/week6 : g++ withoutSimplecpp.cpp
~/Desktop/nptel/week6 : ./a.out
~/Desktop/nptel/week6 : %
```

So just to make sure just to persuade you that this works, I am going to show you both of these programs and run them. So this is the first program that we saw where you are using, you are not including simple CPP, but you are including iostream and you are using the using namespace std and you are putting and you are writing the program.

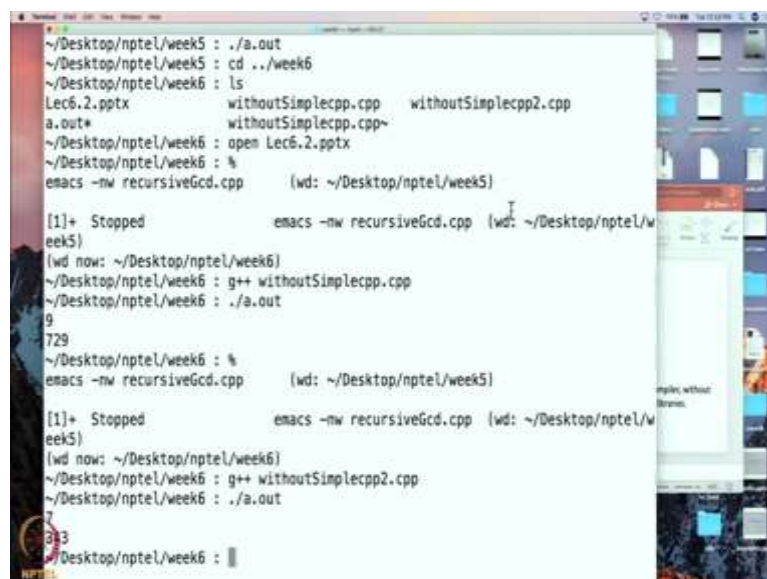
So, let us compile it, we can compile it with s++ still, but we can also compile it with the basic C++ compiler and that is called g++ on Unix. So, I can compile without simple CPP. So it has done its job, so it expects me to type in a number, so let us say I type (n) 9 and indeed it is printing out the cube of 9 which is 729.

(Refer Slide Time: 5:06)



```
File Edit Options Buffers Tools C++ Help
#include <iostream>

int main(){
    int n;
    std::cin >> n;
    std::cout << n*n*n <<
    std::endl;
}
```



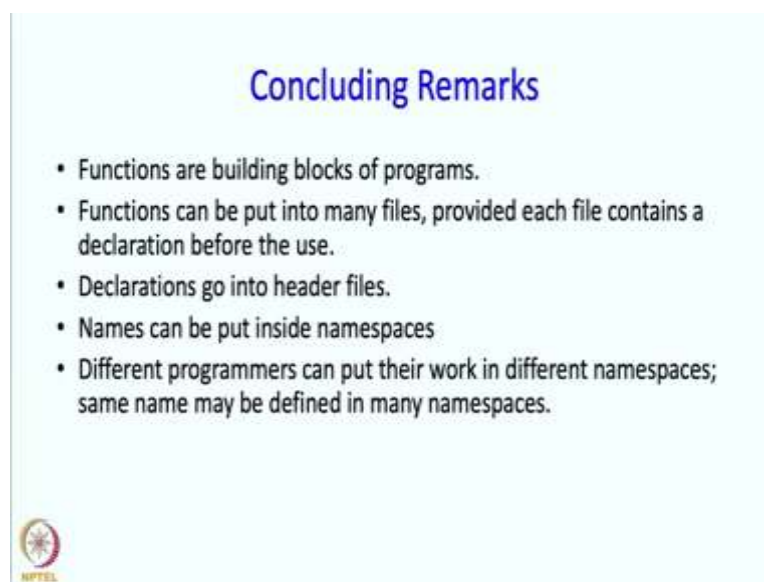
```
~/Desktop/nptel/week5 : ./a.out
~/Desktop/nptel/week5 : cd ../week6
~/Desktop/nptel/week6 : ls
Lec6.2.pptx          withoutSimplecpp.cpp  withoutSimplecpp2.cpp
a.out*              withoutSimplecpp.cpp~
~/Desktop/nptel/week6 : open Lec6.2.pptx
~/Desktop/nptel/week6 : %
emacs -nw recursiveGcd.cpp (wd: ~/Desktop/nptel/week5)
[1]+ Stopped emacs -nw recursiveGcd.cpp (wd: ~/Desktop/nptel/w
week5)
(wd now: ~/Desktop/nptel/week6)
~/Desktop/nptel/week6 : g++ withoutSimplecpp.cpp
~/Desktop/nptel/week6 : ./a.out
9
729
~/Desktop/nptel/week6 : %
emacs -nw recursiveGcd.cpp (wd: ~/Desktop/nptel/week5)
[1]+ Stopped emacs -nw recursiveGcd.cpp (wd: ~/Desktop/nptel/w
week5)
(wd now: ~/Desktop/nptel/week6)
~/Desktop/nptel/week6 : g++ withoutSimplecpp2.cpp
~/Desktop/nptel/week6 : ./a.out
7
343
~/Desktop/nptel/week6 : %
```

Let me show you other program without simple CPP. It is called and here I have not put in the line using namespace std, but instead I have included the prefix std:: before all the names

which are belonging to that namespace std. So again let me compile that, so s++ so I could compile it with s++ as well, but I could compile it with g++, let us use g++ again because that is what you would do if you are not using simple CPP.

So without Simplecpp2.cpp, so it is compiled dot slash a dot out, again it is expecting me to give a number. So let say I type 7 and yes, the cube is 343, so that is done. So, if you wish you can run your programs without using simple CPP, but then you would have to type these lines and if you are feeling tired about typing these lines, you can continue to use s++, (no) it will work in anyway. Alright, so that concludes the part about how to use C++ without simple CPP and specifically what include files you need to use.

(Refer Slide Time: 6:53)



Now, I want to conclude this entire lecture sequence on how to write large programs. So, the first point that we made are functions are building blocks of programs. So just as when you write a book, you typically break it into chapters, if you are designing a device you break it up into subsystems, systems and subsystems. Similarly, when you are writing a program you kind of (break it up) break it down into functions. There will be other things that will also come up, but we will talk about those later.

Functions can be put into many files provided each file contains a declaration before the use of the function. Declarations can go into header files, it is convenient if you put declarations into header files. And when you put declarations into header files, there are some additional details which you might want to learn if you use really if you yourself try to write really large programs. Some of these details are discussed in the book. Names can be put into namespaces

and you do have some namespaces which are defined as a matter of, as standard by default in C++ and the namespace `std` is one of those namespaces.

So different programmers can put their work in different namespaces and the same name may be defined in many namespaces. So, in fact this says that you could have a variable called `cin` in your program, I would recommend that you do not do that because you will confuse yourself, but in principle you could do it because the `cin` that you are using so far is a name defined in the namespace `std`.

So if you carefully want to differentiate it and put `cin` in your own namespace that is possible, but again as I said, not recommended. The details of all this are discussed in the book, and that concludes this lecture sequence. Thank you.