

Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 4 Part - 1
Program Design

(Refer Slide Time: 0:21)




Hello and welcome to the course on an introduction to programming through C++, this is the second lecture sequence of the second week. The topic for today is a program design example and the reading for this is from chapter 4 of the book.

(Refer Slide Time: 0:34)

How to write programs

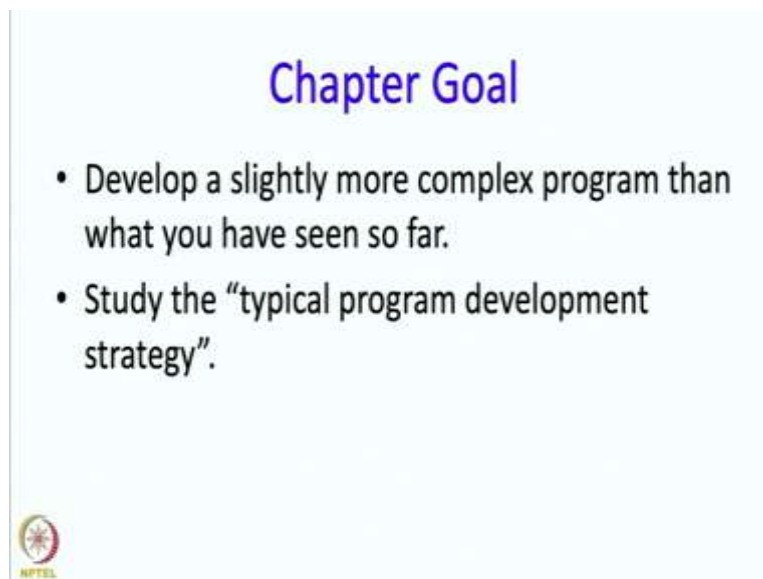
- So far, we wrote very simple programs.
- Simple programs can be written intuitively.
- Even slightly complex programs should be written with some care and planning.
- Your program could be doing very important calculations
 - controlling the flight of a plane
 - decide the amount of radiation to give to a patient
- Important to learn what can be done to ensure that a program works correctly.



So the main problem that we are going to consider in this lecture is how to write programs and we have been writing some programs, we have written some simple programs and we sort of wrote them automatically or intuitively if we will. Now, if you want to write more complex programs, even slightly more complex programs, we need to approach this with care because we might make mistakes and we might find it difficult to get out of mistakes and we really should be worried about mistakes and also about how much time we spend on the writing process.

So we should make our writing process efficient and we should also make sure that our programs do not make any mistakes. Now, that is because the program could be doing here very important calculations, maybe controlling the flight of a plane or maybe deciding how much radiation to give to a patient, so you really cannot afford to make any mistakes in this. And, so we should do sort of due diligence and it is important to learn what are the things that you should do in order to ensure that your programs work correctly.


(Refer Slide Time: 1:59)



The slide features a light blue background with the title 'Chapter Goal' in a bold, purple font at the top center. Below the title, there are two bullet points in black text. The first bullet point reads 'Develop a slightly more complex program than what you have seen so far.' The second bullet point reads 'Study the “typical program development strategy”.' In the bottom-left corner, there is a small circular logo with a red and white design, and the text 'NPTEL' is written below it.

Chapter Goal

- Develop a slightly more complex program than what you have seen so far.
- Study the “typical program development strategy”.



The “typical program development strategy”

- Specification: State what is the input data and what output is required.
 - Description or real life problems can be ambiguous or incomplete
- Construct test cases.
- Think how you would solve the problem using pencil and paper.
 - Understand the structure of the manual solution process
 - Is something repeated? How many times?
- Write the program.
 - Manual solution process structure should be reflected in the program
 - Need to decide what variables to use
- Run the test cases.
- Debug: what to do if the program does not work..



So the goal of the chapter is to develop a slightly more complex program, then what we are seeing so far. And while doing this, we will follow a typical program development strategy, so what is this? So this begins with the description of the specifications, the specification simply and very concisely usually states what the input data is and what the output required is. Now, if you are writing a program to solve a real-life problem, then the statement that might be given to you, might be ambiguous or might be incomplete and so when you write down the specification, you should try to make it as precise and as complete as possible.

Then we construct the test cases. So for what input, what output do you expect? Then we think about how to solve the problem using pencil and paper, this is kind of the first important seriously creative step. Now, here what you are expected is to really just mimic manual computation, so the creativity is a little bit less important right now, but you really need to pay attention to how you solve the problem manually, so you need to figure out what the structure of that manual solution process is. Which means, for example, to decide whether something is repeated, if so, how many times? And after this, you get to the program writing stage and here the manual solution process will help you because the structure will get reflected in the program and you also need to decide what variables to use, you can again reason it out from the way you perform the manual calculations. Once your program is ready, you run the test cases and if the program does not work correctly as might happen from time to time, then you need to figure out what went wrong and you need to fix it.


(Refer Slide Time: 4:12)

The problem

The following series approaches e as n increases:

$$e = 1/0! + 1/1! + 1/2! + \dots + 1/n!$$

- Write a program which takes n as input and prints the sum of the series.




The problem that we are going to look at is a very simple problem, so we have a series whose value approaches e as n increases we have sum, so this sum is $1/0! + 1/1! + 1/2!$ all the way till $1/n!$, if you add the terms up, and as you take n larger and larger, you can see, you can prove in fact that, that this sum is going to get close to e , is going to tend to e . So we are supposed to take a program, we are supposed to write a program which takes n as input and prints the sum of this series, whatever the value of n might be.

(Refer Slide Time: 4:49)

The specification for our problem

- Input: an integer n , where $n \geq 0$
- Output: The sum $1/0! + \dots + 1/n!$
- This specification is obvious, but still is more than the problem statement
 - We have made explicit that n cannot be a negative number.
- Ask yourself: can something be misunderstood in this? Can it be made clearer?
- You may realize that carelessly, you may think of n as also being the number of terms to be added up.
- The number of terms being added together is $n+1$.
- The number of additions is n , however.



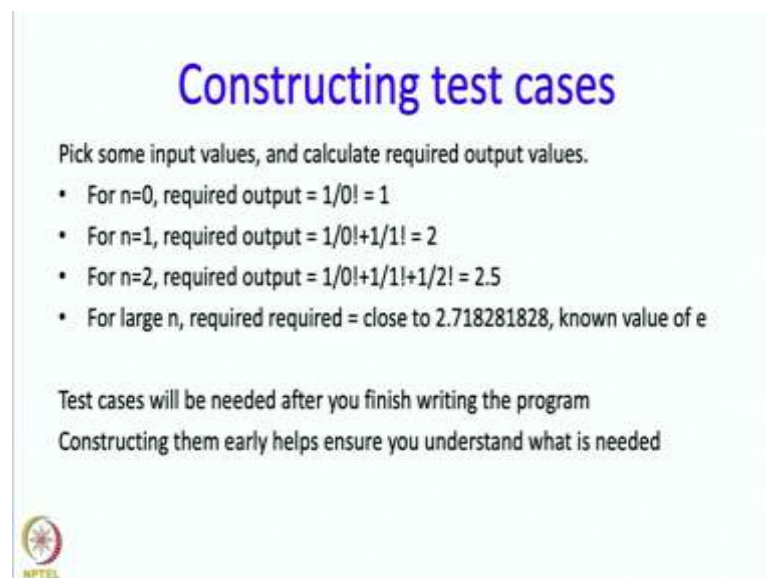
So first, let us come to the specification, well, the input there is only one, the integer n . So we have clarified that n has to be non-negative integer, the problem does not make sense if we have n negative and output is the sum $1/0!+1/1!$, all the way till $1/n!$. So notice that 0 as input

makes sense because then that is means that you get the first term, which is 1, that is, of course nowhere close to e, but we did not really promise that it would get close to e for such a small value of n in any case.

Now in this particular example, for this particular problem, the specification is fairly straightforward. However, notice that even here we have added a bit of value, so when we said that input is n, we also wrote down that it had better be larger than or equal to 0, in general, when you write the specification or maybe say after you write the specification, you should ask yourself, well, I have written this down, but are there any tricky points to it? Can something be misunderstood? Even by mistake, because you do not really want any misunderstanding because those misunderstandings may creep into your program as well.

So in this case, you may realise that, if you are careless you may think of n as also being the number of terms to be added up, but that is not correct, $1/0!$ might be regarded as 0th term, than $1/1!$ is the first term, $1/N!$ is the nth term, so they really are n plus 1 terms to be added up, however, the number of additions you perform is actually n, so n does have some significance, but not as the number of terms.

(Refer Slide Time: 6:54)




Constructing test cases

Pick some input values, and calculate required output values.

- For $n=0$, required output = $1/0! = 1$
- For $n=1$, required output = $1/0!+1/1! = 2$
- For $n=2$, required output = $1/0!+1/1!+1/2! = 2.5$
- For large n, required required = close to 2.718281828, known value of e

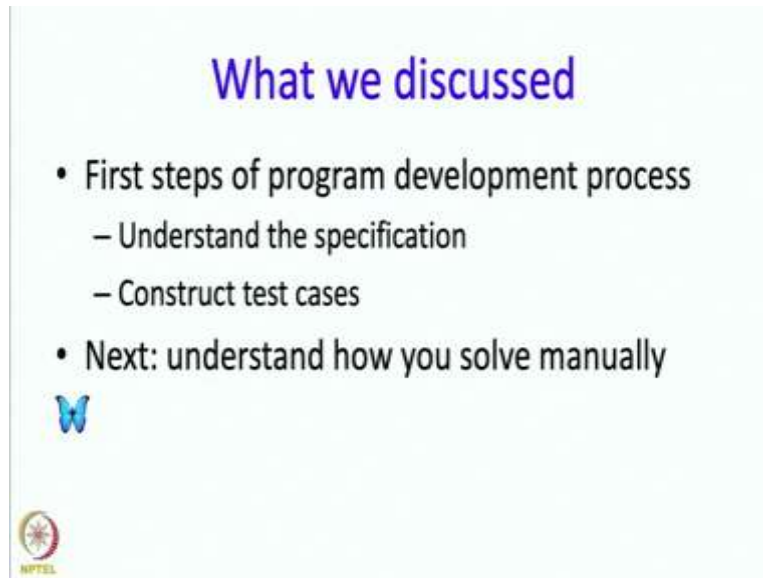
Test cases will be needed after you finish writing the program
Constructing them early helps ensure you understand what is needed



The next step is to construct the test cases, so here you just pick some input values, and you calculate the required output values So for example, if we say n is 0, then the required is $1/0!$ or 1, if we say n is 1, the required output is $1/0!+1/1!$, which is equal to 2, if n is 2 than the required output is $1/0!+1/1!+1/2!$, which is 2.5 and this program really is meant for calculating e and that happens at some large enough value.


So maybe we will see that if you pick say n equal to 10 over something like that, you would like the answer to be close to the known value of e , which to so many decimals it is 2.718281828. Now, test cases will be really needed after you finish writing the program, however, if you construct them early it is just the confirmation to yourself that look, I understand what is expected of me, I have not missed out any point.

(Refer Slide Time: 8:17)



What we discussed

- First steps of program development process
 - Understand the specification
 - Construct test cases
- Next: understand how you solve manually



Okay, so what have we discussed so far? So we said that the first steps of the program development process consist of the specification, writing the specification and constructing the test cases. Next, we are going to turn to how to solve the problem manually, we will take a break.