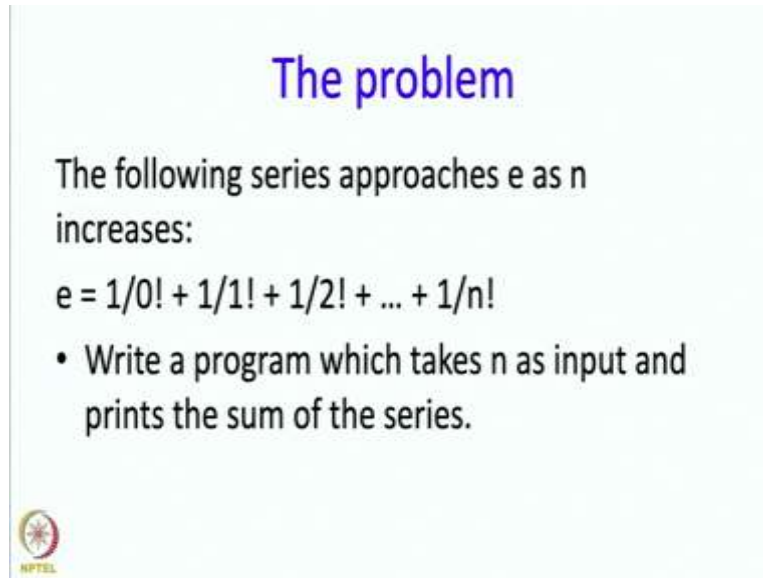


Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 4 Part - 2
Program Design
Translate manual algorithm to a program

(Refer Slide Time: 0:22)




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.



Welcome back, in the previous segment, we discussed the specification and the construction of test cases. Next, we are going to look at how the problem is solved manually and what can be derived from it. How do you sum this series manually? Let me remind you what the series was, very quickly, this was the series, $1/0! + 1/1! + 1/2!$ all the way till $1/n!$.


(Refer Slide Time: 0:47)

How do you sum the series manually?

- Step 0: Calculate the zeroth term, $1/0!$, which is just 1.
- Step 1: Calculate the first term, $1/1!$ which is just 1. Add to 1.
- Step 2: Calculate the second term, $1/2!$, add to sum so far.
- Step 3: Calculate the third term $1/3!$, add the sum so far. And so on.

Will you calculate each term independently?

- You will calculate third term by dividing the second term by 3
 $1/3! = 1/2! / 3$
- The **general pattern** for step i :
 - Calculate i th term, $1/i! = i-1$ th term $1/(i-1)! / i$. Add to sum.
- Pattern applicable to steps $1..n$. No previous term for step 0



Okay, so you might say look, this is fairly straightforward in step 0, you may calculate 0 term, or $1/0!$ and that is just 1. In step 1, you calculate the first-term $1/1!$, which is also 1 and you add that to the preceding sum that you had, so the sum until this point was 1, so you add this new one to that sum, then you calculate the second term, that is $1/2!$ and you add that to the sum so far and so on, you calculate the third term $1/3!$ again add it and you do this until n . Now, here is an interesting question, will you calculate each term independently? If you actually try to do it, you will very quickly realize that you do not need to, very likely you will calculate the third term by dividing the second term by 3, so the third term is $1/3!$, the second term is $1/2!$, so that just needs to be divided by 3, in order to get $1/3!$ and this is a general pattern, so in step I we need to calculate $1/I!$ and that is simply $1/(I-1)!$ divided by I . So in step I you will take the term which was calculated in step I minus 1, divide that by I and add it to do sum which you are keeping track of. Now, notice that general pattern is applicable to steps 1 through N , not the step 0, because in step 0, there was no previous sum, nor was there a previous term, so this means, for example, that you might think of having a loop to do the work of steps 1 through N and step 0 might be something that you do before that.

(Refer Slide Time: 2:53)

Going from manual computation towards a program

- The **general pattern** for step i :
 - Calculate i th term, $1/i! = (i-1)! / i$. Add to sum.
- Pattern applicable to steps 1..n. No $i-1$ th term for step 0

Overall program structure:


- Step 0: perform separately.
- Steps 1 to n: put in a repeat loop.

What variables to use?

What do we need to have with us at the beginning of iteration i ?

- Sum calculated in previous iteration
- Term calculated in previous iteration
- Value of i , since we divide previous term by i to get current term

So we need 3 variables: S, T, I.

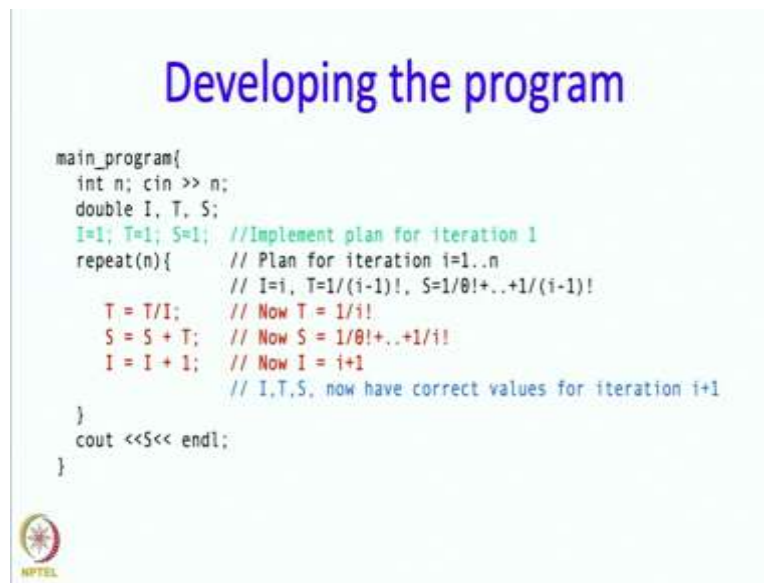


Okay, so we have a reasonable understanding of what the manual computation is, now we are going to move towards a program. So as we said the general pattern for step I is calculate the I th term by dividing the $(I-1)$ th term by I and then adding it to sum. Now, we also said that this pattern is applicable to steps 1 through N and not to step 0, alright, so this suggests an overall program structure which is we perform step 0 separately and steps 1 to N put in the repeat loop.

So that is good we have made progress towards the structure of the program and now the question is what variable should we will using? So here you should ask a very standard question, which is what do we need to have with us at the beginning of each iteration or in particular iteration I . So these are the things you need, so you need the sum calculated in the previous iteration. You also need the term calculated in the previous iteration, why? Because we said that we will take that term and divide it by I okay, this is the number of the iteration and then get what we want to add during this iteration and of course we need to know the value of I , since we divide by I .

So these are the things we need to have and we need to have them from the previous iteration, so which means we need to remember them, we need some mechanism to remember them, so we are going to have 3 variables respectively to remember the sum, term and I and we are going to call the variable capital S, capital T and capital I.

(Refer Slide Time: 4:53)



Okay, so now we are going to develop the program. So, what we have said so far allows us to write this skeleton, so we have the main program, first we are going to read the input, of course we are going to define the variable N, then we are going to define the variables we said we need I, T and S and then we also said that we need the repeat loop and so we are going to have repeat with count of N, and then finally we are going to have to print out the sum, which we are hoping will be in the variable S.

And we also said that for the Ith iteration we would like capital I to have the value little i, we would like capital T to be the value of the term added in the I minus 1 iteration or in other words $1/(I-1)!$ and we also want S to be the sum calculated in the first I-1 iterations or the value $1/0!+1/1!$ all the way till $1/(I-1)!$, so this is our skeleton and now we are going to build it up.

So, what we really need to do to make sure that the plan that we have written down. Okay, can be realized. So, let us talk about the first iteration itself, can we make sure that the values of I, T and S are as we have asked for? So for example, in the first iteration that is little i equal to 1, we want capital I also to be little i or we want capital I to be 1. Likewise we want T to be $1/(I-1)!$ or $1/(1-1)!$ or $1/0!$ or 1 again and S must be to 0 term of this series, so it is also desirable to have S equal 1.

Well, the only way we can make sure that this happens at the beginning of the loop of the very first iteration is, if we initialize the variables to the values, so that is what we are going to do, so we are going to write an assignment statement I equal to 1, T equal to 1, S equal to 1

or separate statements and this implements our plan for iteration 1. How do you implement it for the subsequent iterations?


Well, one thing you said was that in the I th iteration we need to divide the previous term by I and we have the value of the previous term, we are expecting to find it in the variable capital T , so we should be dividing T by capital I . Now, the question is where should we put it? So we are going to put it in T itself, and here is a reason for it, the value that T originally had was $1/(I-1)!$ and this value we are not going to need, we are going to need the value that we just calculated T upon I , which is 1 upon I factorial and since, T is going to be the variable which in the next iteration is anyway expected to hold $1/I!$, we might as well store that value in T itself.

Next, we said that we are going to add the newly calculated term to the sum and in fact the right statement used for that is just to say $S=S+T$, so note that this now means that $S=1/0!+1/1!$ all the way till $1/I!$. And finally to prepare for the next step, we really should be changing capital I to I plus 1, so that is what we are doing over here. So with these 3 assignment you can see that I , T , S now have the correct values for the next iteration.

So our work in this iteration is done, but not only that T , S and I we have changed, so that they have the correct value for the next iteration, why is that? Well as we can see capital T has the value $1/I!$ now, in the next iteration, which is $(I+1)$ th iteration, it really should have this value, then sum in the $(I+1)$ th iteration should be the sum of all the terms from 1 over 0 factorial to $1/(I+1-1)!$ or $1/I!$ and that is exactly what we have here as we have written down in that comment. And finally capital I should take the value I plus 1 at the beginning of the $(I+1)$ th iteration and that exactly is what we have done.

So we have followed our plan, not only that we have done exactly what we used to do in the manual computation and therefore, we have finished writing the program. Now in this program we put down some comments about what plan we are going to follow. So this is, these are the black lines over here. So these two lines tell the user what our plan is, this is absolutely necessary.

(Refer Slide Time: 10:43)



Remarks

- The program has comments about the plan being followed.
 - Very essential to explain this
- The order in which we updated T,S,I is important.
 - Result will be different if you change the order
 - “Do we want to divide T by the old value of I or the new value?”
 - “Do we want to add new T to S or old T?”

If somebody else is reading a program, they should know this and if they know this it will be much easier for them to follow what is going on. Now, one more important point - the order in which we updated T, S and I is very important. So for example you cannot update I first, that would not really work, you cannot exchange the order of this statements, so since in these statements each variable is going to have an old value and new value, we had better be very careful as to taking into account what value it is that we want to add to which variable or multiply each variable or whatever that case might be, and then make sure that the update to that variable happens at the right time, in other words statement like these will have to be handled carefully and the order of the statements is going to be really important.

So yes, so the considerations will be we want to divide T by the old value of I or the new value and depending upon that the ordering should be done, the ordering of the statements should be done or similarly, do we want to (see it, we want to, you want to ask) we want to add the new T to S or old T?

(Refer Slide Time: 12:05)

Next step: execute the program

- Type the program into a file or the IDE
- Compile it
- Run.
 - Use the test cases constructed earlier.
- Demo

A screenshot of a C++ IDE window. The code defines a function 'main_program' that takes an integer 'n' as input. It initializes variables 'i', 'T', and 'S'. 'i' is set to 1, 'T' to 1, and 'S' to 0. A 'repeat' loop runs from 'i=1' to 'i=n'. Inside the loop, 'T' is updated to '1/i', 'S' is updated to 'S + T', and 'i' is incremented. After the loop, 'S' is printed. The code is as follows:

```
File Edit Options Buffers Tools C++ Help
#include <iostream>
main_program
{
    int n; cin >> n;
    double T, S;
    i=1; T=1; S=0; //Implement plan for iteration i
    repeat(n){
        // Plan for iteration i=1..n
        // i=1, T=1/(1-1), S=0/0!+...+1/(1-1)!
        T = T/i; // Now T = 1/i!
        S = S + T; // Now S = 1/0!+...+1/i!
        i = i + 1; // Now i = i+1
        // T,T,S, now have correct values for iteration i+1
    }
    cout <<S<< endl;
}
```



```

~C
~/Desktop/nptel/week3 : %
emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)

[1]+ Stopped emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)
(wd now: ~/Desktop/nptel/week3)
~/Desktop/nptel/week3 : ls
Lec3.1.pptx      a.out      turtleController.cpp
Lec3.1old.pptx   buttonController.cpp
~/Desktop/nptel/week3 : mv Lec3.1.pptx Lec3.1old.pptx
~/Desktop/nptel/week3 : mv Lec3.1old.pptx Lec3.1.pptx
~/Desktop/nptel/week3 : ls -l
total 992
-rw-r--r-- 1 abhiram staff 173557 Mar 28 11:47 Lec3.1old.pptx
-rw-r--r-- 1 abhiram staff 182557 Mar 28 14:28 Lec3.1.pptx
-rwxr-xr-x 1 abhiram staff 137588 Mar 28 14:08 a.out
-rw-r--r-- 1 abhiram staff 989 Mar 28 14:08 buttonController.cpp
-rw-r--r-- 1 abhiram staff 298 Mar 28 14:08 turtleController.cpp
~/Desktop/nptel/week3 : cd ../week2
~/Desktop/nptel/week2 : ls
e.cpp      octagon.cpp      temperature.cpp
Lec2.2.pptx ebuggy.cpp      octagon.cpp~    temperature.cpp~
Lec2.2old.pptx ebuggy.cpp~    projectile.cpp~  test.cpp
Lec2.2.pptx eDebug.cpp~    projectile.cpp~ ~$Lec2.2.pptx
a.out      eDebug.cpp~    spiral.cpp
e.cpp      factorial.cpp   spiral.cpp~
~/Desktop/nptel/week2 : open Lec2.2.pptx
~/Desktop/nptel/week2 : %
emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)

[1]+ Stopped emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)
(wd now: ~/Desktop/nptel/week2)
~/Desktop/nptel/week2 : s++ e.cpp]

```

```

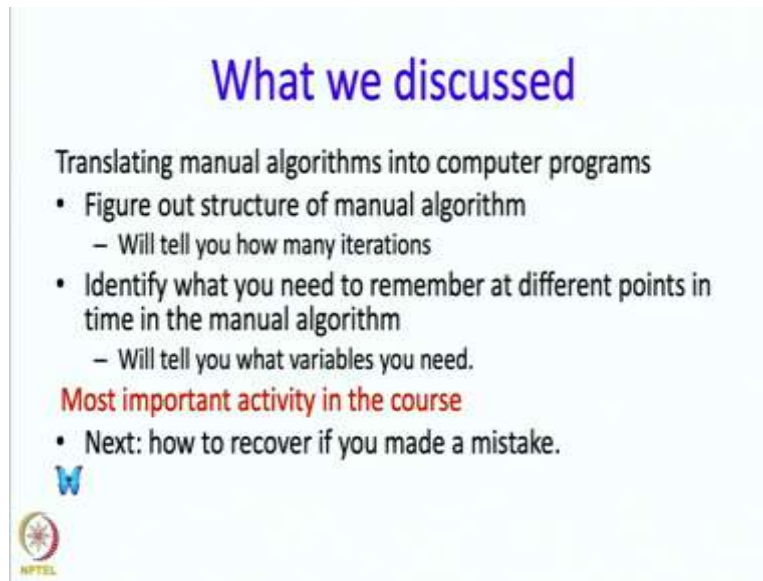
-rw-r--r-- 1 abhiram staff 182557 Mar 28 14:28 Lec3.1.pptx
-rwxr-xr-x 1 abhiram staff 137588 Mar 28 14:08 a.out
-rw-r--r-- 1 abhiram staff 989 Mar 28 14:08 buttonController.cpp
-rw-r--r-- 1 abhiram staff 298 Mar 28 14:08 turtleController.cpp
~/Desktop/nptel/week3 : cd ../week2
~/Desktop/nptel/week2 : ls
e.cpp      octagon.cpp      temperature.cpp
Lec2.2.pptx ebuggy.cpp      octagon.cpp~    temperature.cpp~
Lec2.2old.pptx ebuggy.cpp~    projectile.cpp~  test.cpp
Lec2.2.pptx eDebug.cpp~    projectile.cpp~ ~$Lec2.2.pptx
a.out      eDebug.cpp~    spiral.cpp
e.cpp      factorial.cpp   spiral.cpp~
~/Desktop/nptel/week2 : open Lec2.2.pptx
~/Desktop/nptel/week2 : %
emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)

[1]+ Stopped emacs -nw dijkstra.txt (wd: ~/dataStructures/spring18)
(wd now: ~/Desktop/nptel/week2)
~/Desktop/nptel/week2 : s++ e.cpp
g++ e.cpp -Wall -I/Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week2 : ./a.out
0
~/Desktop/nptel/week2 : ./a.out
1
~/Desktop/nptel/week2 : ./a.out
2
~/Desktop/nptel/week2 : ./a.out
2.5
~/Desktop/nptel/week2 : ./a.out
10
2.71828
~/Desktop/nptel/week2 :

```

Okay, so after this we are going to execute the program, so for this of course we should type the program into a file or into the IDE, compile it, and then run and we should use the test cases constructed earlier. So let see this. Here is the program that we just wrote. So that is compile it and let us run it and let us try our 3 or 4 test cases, so the first was 0 and we were expecting the value 1. So we got the value 1, let us try one more case, so when we type 1 we expect the value 2 yes and when we try 2 we expect the value 2.5, let see if we get that? Yes and finally, let us try some large value, say maybe 10 and we get 2.71828, which is in fact exactly the value to 5 decimal places, so our programs seems to be doing just fine okay, so here is what we discussed.

(Refer Slide Time: 13:20)




What we discussed

Translating manual algorithms into computer programs

- Figure out structure of manual algorithm
 - Will tell you how many iterations
- Identify what you need to remember at different points in time in the manual algorithm
 - Will tell you what variables you need.

Most important activity in the course

- Next: how to recover if you made a mistake.



So we said that translating manual algorithms into computer programs is important and what you are supposed to do during that is figure out the structure of the manual algorithm and this will tell you how many iterations are needed for the repeat loop that you might use in your computer program. Then you are supposed to identify what you need to remember at different points of time in the manual algorithm and this will tell you what you should be storing in what variables and this thinking about manual algorithm and figuring out its structure and so on is probably the most important activity that you will learn in the course okay, so please, please get it right. So we will stop over here and in the next segment, we will see what to do if you make a mistake.