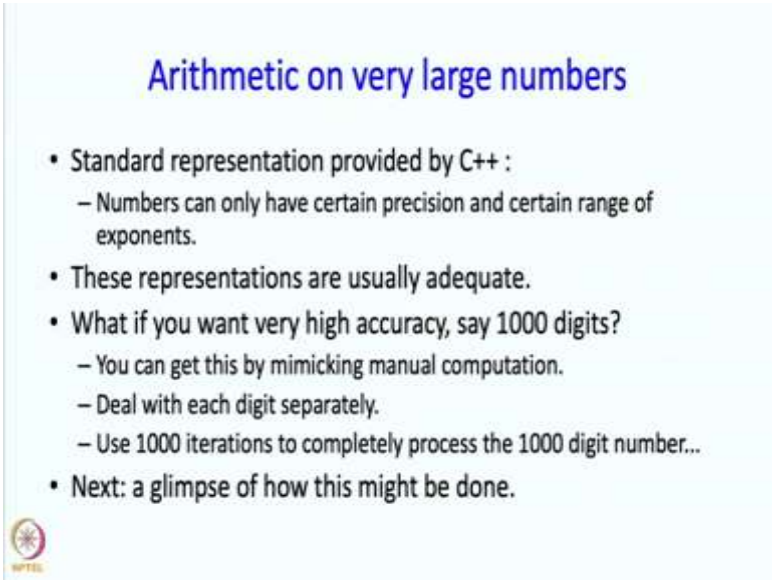
An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 09 Part - 5
Loops in various applications
Arithmetic on very large numbers

(Refer Slide Time: 00:39)



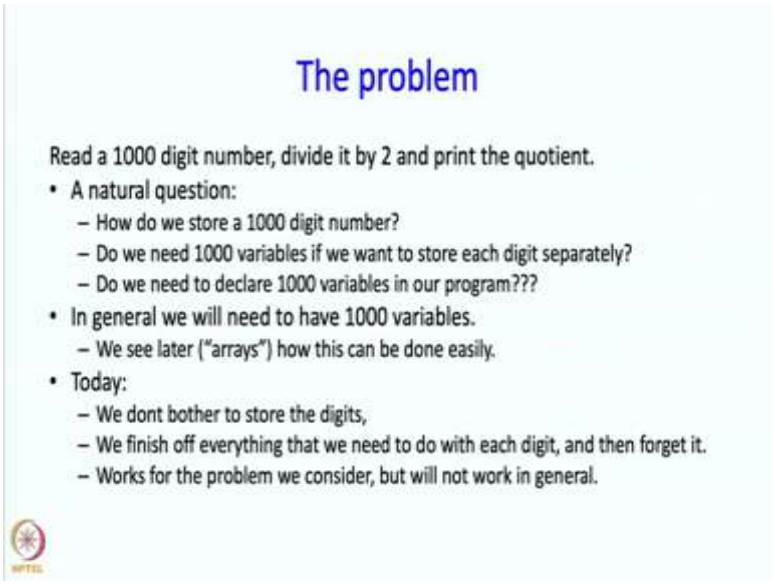
Welcome back, in the last segment we discussed how to model a real life system, system with some state using a program. In this segment we are going to discuss arithmetic on very large numbers. Now, standard representation that C++ contains, allows you to have numbers with only a certain precision and certain range of exponents.

So for example, if you are looking at floats, you will have some 23 bits of Mantissa or significand and you will have 11 bits of the exponent. And similarly, for integers, there is so say if you have unsigned int, then you will have 32 bits given for representing the integer. If you say long you may get 64 bits, but what if you want to represent very high number of digits? So these digits that C++ provides are usually very adequate. But sometimes you may want to represent numbers to very high digits, say for example, you might have seen the value of pi calculated to a 1000 places. How to they do that? So that is sort of the topic, that we want talk about right now.

Now, it turns out that sort of the central idea in this is to mimic, what you do manually, so what you do manually? You deal with each digit separately. So when you do the arithmetic you look at digit 1 at a time. So in your program as well, you typically will do something like look I am going to have 1000 iterations to process a 1000 digit number, something like that.

Now this turns at to be somewhat involved, but what we are going to do in this segment is, we are going to give you a glimpse. A very brief introduction to how this is possibly done, it is going to be a very simplified example, but it will give you some idea.

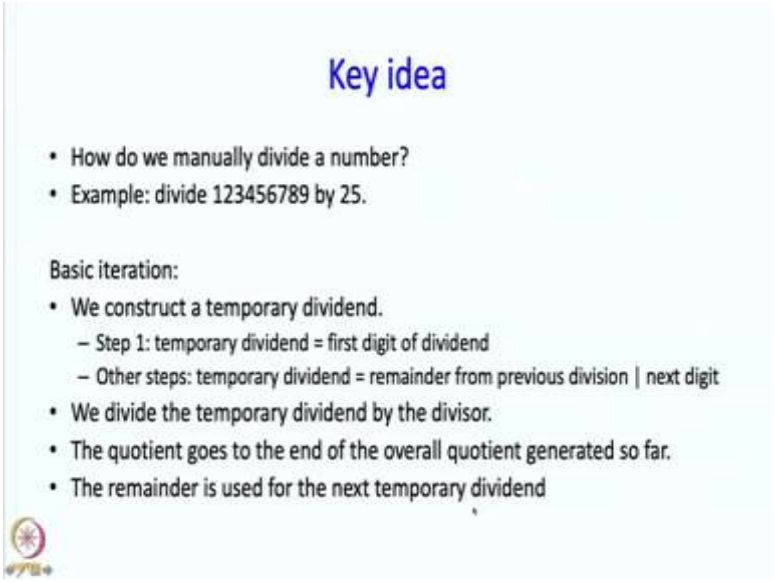
(Refer Slide Time: 02:51)

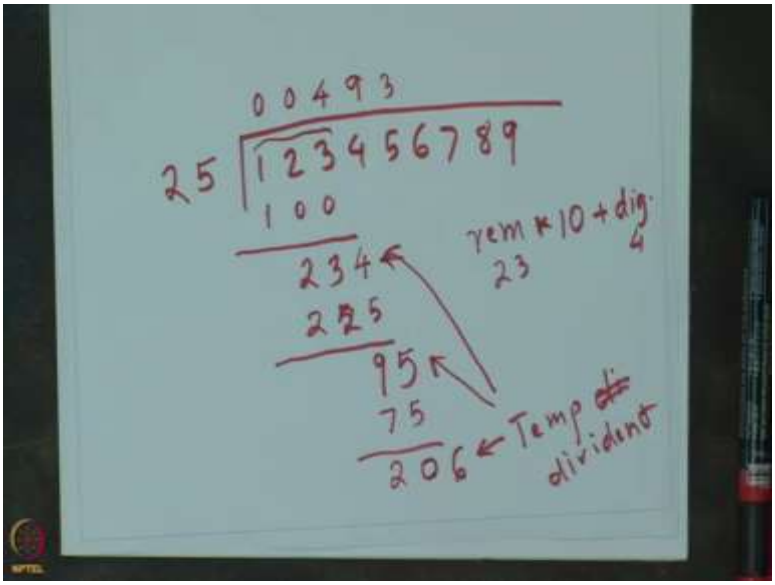


The specific problem that we are going to consider is the following, so you are asked to read a 1000-digit number divided by 2 and print the quotient. Now, the moment you talk about 1000 digits, there is very natural question. How do we store a 1000 digit number? Do we need 1000 variables? Especially, if we say that we want have, we want to separately store each digit, well seems like there is no getting away from having the 1000 variables. And do we have to write int digit 1, digit 2, digit 3 and so on 1000 times? Even that sounds just completely painful.

Well, in general, you will need to do that, not right now, not with the program that I am going to show you, thank god, you will need to do that. And very soon we are going to see some, we are going to see a C++ feature, called an array, which will enable you to do this very easily. But right now, we do not know it, today what we are going to do is, we are going to not bother storing all the digits, instead we read 1 digit, we do everything that needs to be done with that digit and then we forget it. This may not always be possible, but it is possible for the problem that we have on our hands today. It will not work in general. In general, you will need to remember all the 1000 digits, and we will do, you will see how that can be done when you have learned arrays, but the basic principle can be understood even right now.

(Refer Slide Time: 04:49)





So, suppose we have some way of getting to the digits, how do we manually divide, how do we divide? So we said we do it manually, so let us try to understand how that happens. So, let us say we want to divide this large number by 25, how do we divide it? So let me write the down.

So, our number is 123456789, how do we divide this? We want to divide this by 25, then probably we write something like this, then we look at the very first number so can this number be divided by 25, or is it, is this too small, if it is too small we put a 0 over here. Then we go on to the next number 12, can it be divided? Well no, so we put a 0 over here, again. Then we bringing the third number, can it be divided? Well yes, now it can be divided. So what kind of a quotient can we try out, well we can try out a quotient 4 in which we get case we get 100 over here, and then we subtract. So, we get a 23 over here.

Alright, so, what is going on? So we are going to look at some most significant digits of our dividend. And this part is what I am going to call as the temporary dividend. So we are going to divide the temporary dividend by 25. Actually we are going to divide even this 1, this 1 was also a temporary dividend. But say in this case, this is the temporary dividend we are going to divide it by the divisor and we are going to get a quotient. So we are going to subtract the product of the quotient and the divisor from over temporary dividend. And we are going to get the remainder.

What do we do with this remainder? So we bring in the next digit, so 234, so we bring in the next digit, and attach it to the end of the remainder. But what does it mean, attached to the end? So this simply means we are taking the remainder, multiplying it by 10 and then we are adding the new digit. So the remainder was 23, the new digit was 4, so we got 234, so this is our new temporary dividend.

So again we are going to try and divide this temporary dividend by the quotient. So what do we get? So this time we will get a 9 over here. So we will put a 225 over here, and we will get 9 as the remainder over here. Again we continue, so we are going to get a 95 again we use this step, remainder times 10 plus digit. As this is over new temporary dividend. So now this, we are going to have 3 as over quotient, so here not we get a 75, and then we are going to get as remainder 20.

So then again we are going to bring the 6 down over here, and that is going to be our temporary dividend. So, this was the temporary dividend, this was the temporary dividend, this was the temporary dividend and these were also the temporary dividends. How do we find temporary dividend? We have a remainder from the previous iteration, so at this point we can think of the previous remainder being 0. But if there was a previous remainder, we multiply it by 10. Then we add the new digit, and that gives us our temporary dividend. Then we tried to divide the temporary dividend by our divisor. And the quotient goes into our solution.

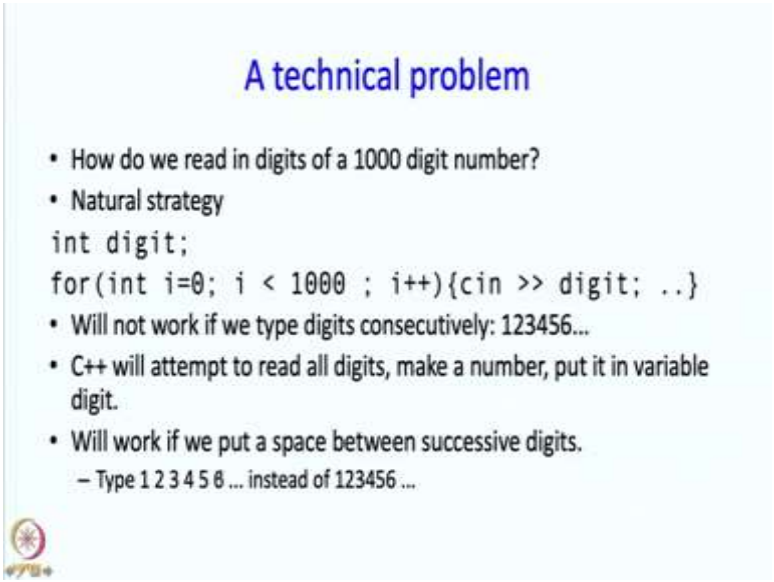
So, you will notice that we are processing the digits a left to right order. Once we process 123, then this 1 is completely useless, we do not have a remember it, so after we process 234, these digits are completely useless. So, we are at any time we are at most keeping 1 extra digit beyond the size of this divisor. So, all the previous digits we do not have to worry about. So that is have we are going to get by without storing everything we are not even reading these digits, we are initially. We are going to read them only when we actually want to look

at them. So, that is the idea, that is how we can get by without maintaining too much state, without remembering too much stuff.

So again, let me summarize the basic iteration, so the ideas we are going to construct a temporary dividend. So the temporary dividend in the very first step is simply the first digit of the dividend. So this is the huge dividend that was given to us, we just pick up the very first digit. In other steps, the temporary dividend is formed in this manner, so which is what have a written down it here. So it is formed by taking the remainder from the previous division and attaching it the next digit, on the right. So that means multiplying this previous remainder by 10, and adding the next digit to it. So that is how we get the temporary dividend. Then we divide the temporary dividend by the divisor. The quotient goes to the end of the overall quotient generated so far. So 4 went here, the 9 went here, 3 went here and so on. Are we keeping track of the quotient? Are we remembering the quotient? No, we do not have to as soon as we know that this is a digit of the quotient, we just print it. And we know that we are going to be printing digits in this order, so they will get, they will come out in the right order.

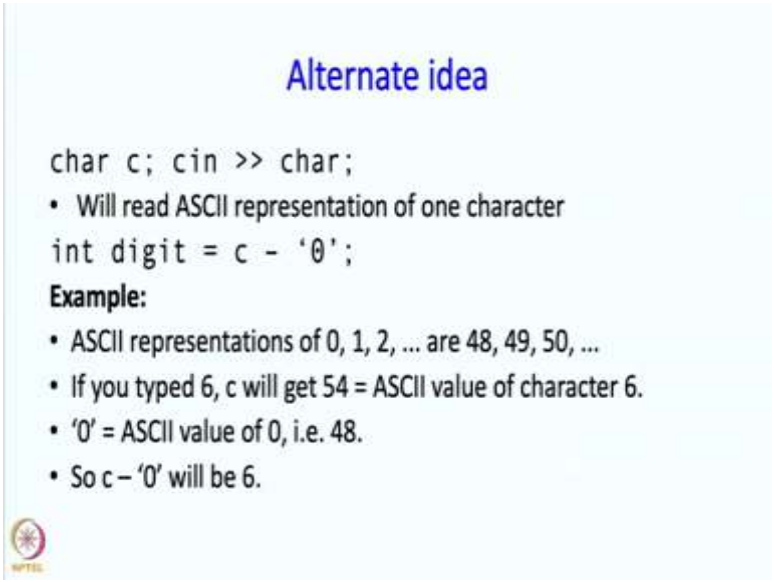
And the remainder from this division, so we did a division over here, so from this division we got a quotient, which goes to the end of the overall quotient generated so far. And the remainder is used in the next temporary dividend. So I just written down what I have shown you in this division over here and that is exactly, what we are going to mimic in the program that I am going to show you next. Well, before I get to the program, there is a small technical problem.

(Refer Slide Time: 12:11)



So we said we want to read in the digits of a 1000 digit number one at a time, but how do we read it? Does C++ even allow us to do that? So here is what might seem like the natural strategy. So we might say int digit and then we might read each of those 1000 digits one at a time. Will this work? Well, if we type in the digits consecutively in this manner, then it will not work, we will not get 1 the first time we read it. The first time this command executes C++ will try to construct this whole thing into a single number and put that into the digits. So that is not going to might work. If we put a space in between the 1 and 2 then, C++ will get a 1 for us. So that is one possibility, so you instead of typing in 123456 whatever the digits are we put spaces in between, then each time we read, we will get exactly 1 digit.

(Refer Slide Time: 13:25)

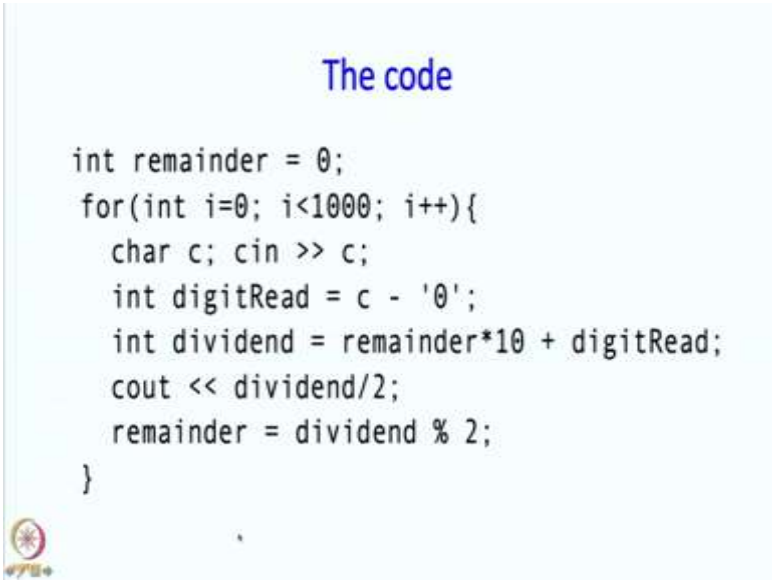


But there is an alternate idea which is slightly nicer, which I think you should know about and therefore, I am going to discuss it. So I am going to not read integers at all, I am going to read in characters. So I am going to have a character variable c and I am going to read in into that character variable. Ohh I am sorry, this should have been c here, not char, just a c over here. So we are going to read in into c. So it will read the ASCII representation of 1 character.

Now, I have what was typed into this variable c. So, what is in the c? So suppose I typed 6, it contains the ASCII representation of that character 6. So I am going to subtract from that ASCII representation of character 6 the ASCII representation of character 0 and put the result int digit. I claim that this is going to be the digit that you really want it to read. Let us see why? So the point to note first is that the ASCII representations of 0, 1, 2 the digits, are in order and they in particular they are 48, 49, 50 and so on.

The 0 has 48, 1 has 49, 2 has 50, 3 has 51, 4 has 52, 5 has 53, and 6 has 54. So at this point when I have just read in, and they should be c and not char. At this point c would get the number 54 inside it. This 0 is the ASCII value of the character 0. Which is 48. So when you perform this, this result will in fact be 6 and so digit will in fact become 6. So, this is how you extract digits from characters. You subtract the character representation of 0.

(Refer Slide Time: 15:51)



So now we are ready to write the code, so let me remind you, we have this notion of a temporary dividend that we are going to form. And for that we need to know what the remainder from the previous time is. So, we are starting with this at this point there is nothing previous so the remainder is 0.

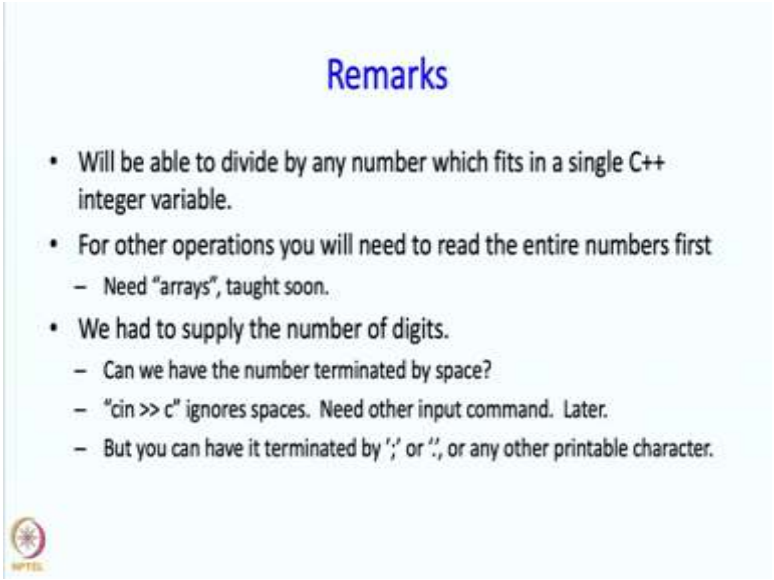
Then, we are going to process each digit. So, there are 1000 digits let us say, and we are going to process them, and so what happens? Well, we are going to read in each digit, then we are going to extract the actual digit value. So as we just said it is going to be c minus 0 the ASCII for 0. Then this is the temporary dividend that I am going to form. So what was that? We said that earlier it is going to be the remainder times 10 plus the digit that we just read.

And now this is going to be divided by the divisor. So, the divisor in this case let us say we have fixed it to 2. So we just want, in the problem statement we said that we want to have it. So this is the quotient dividend upon 2 and if you remember, coming back to this picture we divided this temporary dividend by the divisor, and whatever the quotient was 9 we just declared as a digit of the final answer. So that is exactly what you are doing over here, this is a digit of the final answer and we are printing it out right away. We do not have to do anything to it further.

So we are just going to print it, we are going to print it out. So every time we figure out the digit of the final answer, we are just going to print it out. So that is what is being done. But that is not enough, we need the remainder as well, so what is the remainder? The remainder is simply dividend mod 2. So, if you want it to divide by 25, we could just have 25 over here in both of this places that is it, that is the only change, and that is it, we just repeat this so many

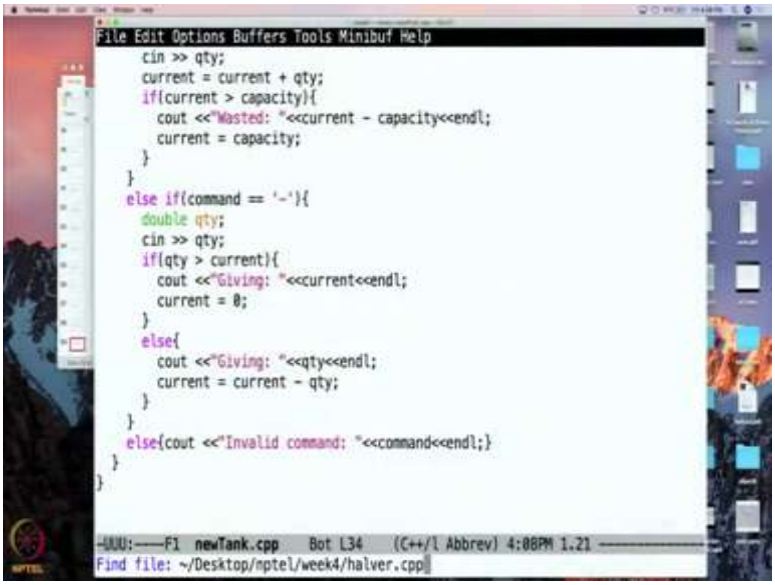
times and at the end of it we will have quotient as we want. We will also have the remainder, the variable remainder will contain the remainder at the end of the division. So, we could print it out if we want, but in this code we have not printed it.

(Refer Slide Time: 18:24)



Alright, so in this manner we will be able to divide any number, which fits in a single C++ integer variable. So as I said instead of the 2 over there, you put in whatever number you want. But not you cannot say I want to divide a 1000 number, a 1000 digit number by another 1000 digit number. So for other operations you will need to read the entire numbers. First you need arrays which will be taught soon and in this code we supplied how many digits there are in our number. You might say look why cannot we have, why do we need to do that is in that inconvenient? So can we have the number we terminated by space, unfortunately this arrow arrow command, this greater than, greater than command, this operator ignore spaces, so there is some other kind of command which we need to learn, which we will learn a little bit later, but for now it can be done. However, if you want to terminate it by a semicolon or stop or comma or any other printable character, that is possible.

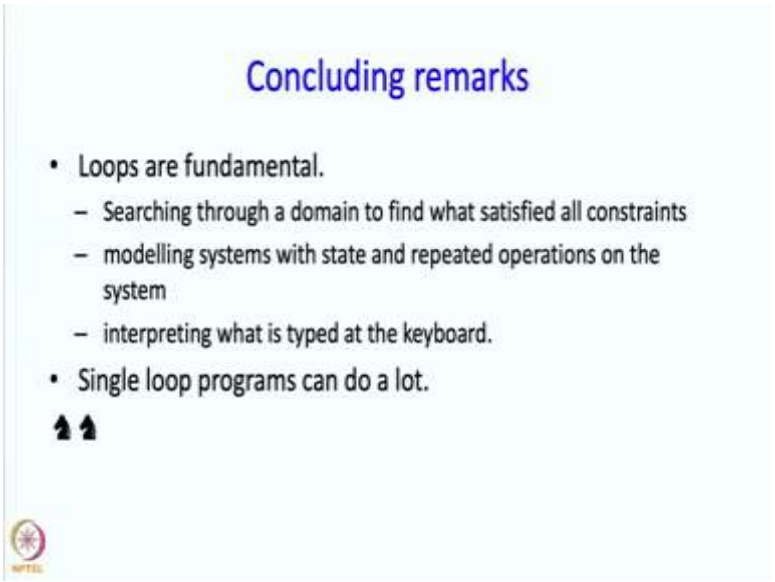
(Refer Slide Time: 19:43)



Alright, so before I conclude actually I think maybe we should take a look at the code and run it. So, let us say I look at this code called halver.cpp. So this is exactly the same program, I think I have called my character x rather than c and I have put in 20 over here. I do not want to put 1000 because I do not want to put, I do not want to have a 1000 digit, in fact, I do not even really want to have 20 digits when we are testing.

So let us say we will just put 5, of course you will say 5 we will fit, so let me put say 11. 11 certainly not fit, or let me put say 13. 13 digits will they fit? No, 15 is certainly not fit in a single int. So, let us see what happens, so let me compile that, and run it. So now I am supposed to give a 15 digit number, so let me do that. So 123456789012345. What happens? So this is half of that number. Does it seem right? Well yes, so 2 gets me 6 over here, 3 gets me 1, so 14, 7, 5, 2, 6 carry from 1 here, so 8 okay it seems reasonable. So I am getting the right answer. And well, you could say look why can we do something, so that this leading 0 is not printed? Of course we can do something, but that will require a little bit more programing and may be that is a good exercise for you to take this code and modify it. Alright, so let me get back to the presentation, actually we are at the end of not only just this segment, but the entire lecture, this segment sequence.

(Refer Slide Time: 22:13)



So, this is, these are concluding remarks on the entire sequence, so I want to observe that loops are really fundamental. You can do a lot using the loops. So we already have seen lots of things, even before today's lecture sequence. But even in today's lecture sequence, we saw lots of interesting things.

So for example, loops are sort of the primary structure that you need, the control structure you need to use, in order to go through a domain and figure out which elements in the domain satisfy all the constraints that you are interested in or a loop can allow you to step through and process the interaction with the world one at a time. So in each iteration, you process one command that the world is throwing at your system and you generate how your system as responding to it, so again a loop is absolutely a crucial control structure. And we looked at reading in from the keyboard, and we said that if you type a 15 digit number, then you have to read it out, read it in as a character, and then convert it into a digit. And then we can string those digits together to make a number.

But this whole process, which I just showed you, it is something that C++ routinely, so when you would type in a number to your program C++ is doing something like that it is extracting the digits, when it is saying ohh is this digit ending over here, or is this the next digit and the previous digits that I have accumulated so far should be multiplied by 10. And then this new digit should be added to it.

So, that code that I showed you something similar is used by C++ just to read in what you type. So, so far we have being saying that when you type 365, 365 magically goes into the variable you are ‘cin’-ing into. That is not really what happen C++ is taking each of those digits as characters and then converting them and accumulating them together and that is how the number actually goes and sits in your variable.

So all this is being done with a single loop program, and note that single loop programs do a lot. Well, sometimes you will have to nest a loop, nest loops of course. So if you have a, if your domain has two variables or in other words you think of it as a two dimensional domain then you will have to have nested loops, but it is loops. Alright, so that concludes this lecture sequence. We introduced a bunch of new application themes over here, and we saw that loops are very powerful and we got a lot of practice using loops. Thank you.