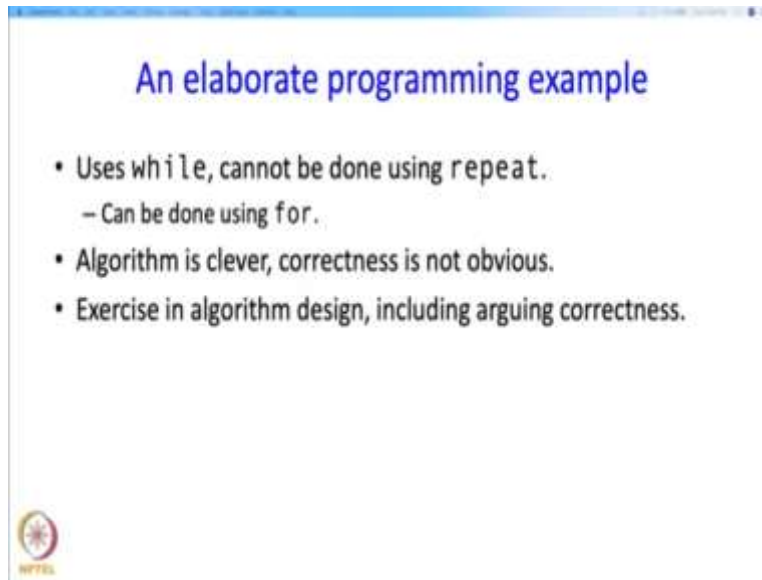


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 7 Part – 5
Looping Statements
Euclid's algorithm for GCD

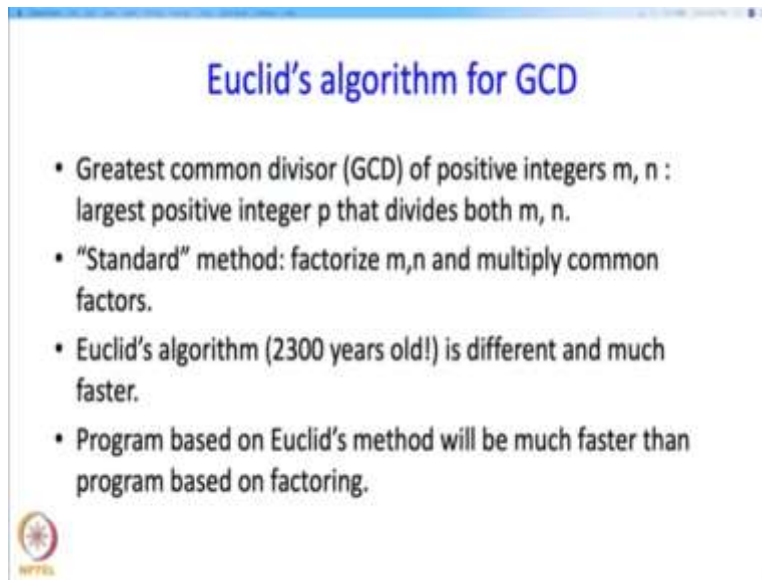
(Refer slide time: 0:43)



Hello and welcome back, in the previous segment we discussed the FOR statement. In this next segment we are going to discuss somewhat elaborate algorithm including we will derive the algorithm and it will be coded using the, the WHILE statement, but it will not be possible to coded using the repeat. Of course it can be coded the FOR statement because in some sense the FOR statement is as powerful as the WHILE statement. What I mean by that is whatever you can do using FOR you can do using WHILE and vice versa.


So here we are going to use the WHILE because in this particular programming problem we will not have anything like a single control variable and therefore, it will be much natural to use the WHILE statement. So we are going to derive the algorithm and argue its correctness because the correctness is not going to be too obvious.

(Refer slide time: 1: 27)



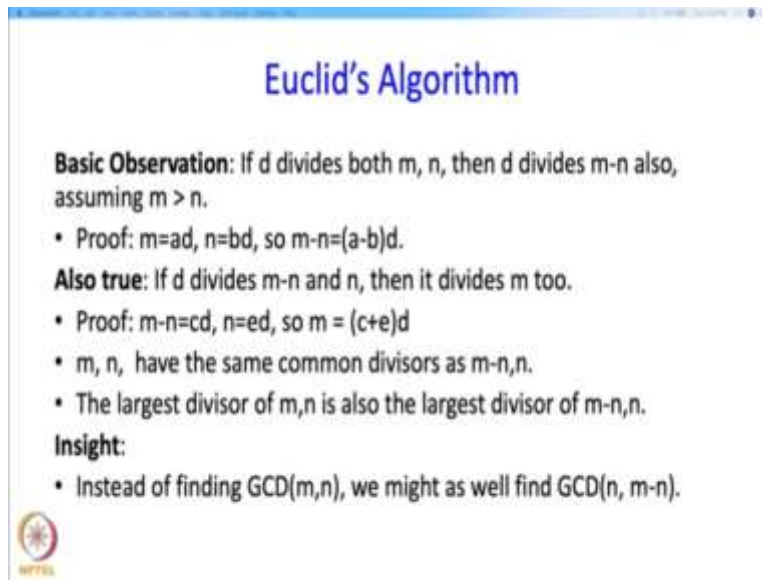
Euclid's algorithm for GCD

- Greatest common divisor (GCD) of positive integers m, n : largest positive integer p that divides both m, n .
- "Standard" method: factorize m, n and multiply common factors.
- Euclid's algorithm (2300 years old!) is different and much faster.
- Program based on Euclid's method will be much faster than program based on factoring.



Okay so, what is the problem? So we need to find the greatest common divisor of 2 numbers and we are going to see or we are going to derive Euclid's algorithm. So, what is the greatest common divisor positive integers m and n ? It is the largest positive integer p that divides both m and n . So for example the greatest common divisor of 36 and 24 is 12 because 12 divides both 36 and 24 and there is no larger number which divide both of them. There is a standard method for doing this which you probably learned in primary school which is to factorise m and n and then multiply out the common factors. Euclid's algorithm is a very old algorithm and is different and actually it runs much faster and in fact, the program based on Euclid's method will also run faster than programs based on factoring. So Euclid's algorithm might have been developed for manual computation, but it is a very commonly used method on computers.

(Refer slide time: 2:47)



Euclid's Algorithm

Basic Observation: If d divides both m, n , then d divides $m-n$ also, assuming $m > n$.


- Proof: $m=ad, n=bd$, so $m-n=(a-b)d$.

Also true: If d divides $m-n$ and n , then it divides m too.

- Proof: $m-n=cd, n=ed$, so $m = (c+e)d$
- m, n , have the same common divisors as $m-n, n$.
- The largest divisor of m, n is also the largest divisor of $m-n, n$.

Insight:

- Instead of finding $\text{GCD}(m, n)$, we might as well find $\text{GCD}(n, m-n)$.



So what is Euclid's algorithm? So there is a very basic observation which is actually fairly simple. So the basic observation says that if "d" divides both m and n, then "d" divides m minus n also and here we are assuming that m is bigger than n. So why is this? Well the proof is just a single line, so because "d" divides m, m must be 'a' times "d" for some a, "d" divides n so n must be 'b' times "d" for some b, so m minus n must be a minus b times "d" but that just shows that m minus n has "d" as a factor, done.


Also true is sort of a converse. It says that if "d" divides m minus n and n then "d" divides m as well okay. So, why is that true? Again the argument is very similar okay. So "d" divides m minus n, so m minus n must be 'c' times "d". Then "d" divides n so n must be some 'e' times "d" and so m which is m minus m plus n must be c plus e times "d". So we have written m as a multiple of c plus e times "d". So "d" is the factor of m as well.

So together what does it mean? It means that so whatever divisors m and n have m minus n, n also have so if "d" is a common divisor of m, n it is a common divisor of m minus n, n as well. So it means that the largest divisor of m, n is also the largest divisor m minus n and n, because if m minus n had an even larger divisor that would have to be a divisor of m, n so that could not be larger than the largest divisor of m, n. So which means and this is the big insight is that instead of finding GCD of m, n we might as well find GCD of n and m minus n. Now this is useful in the following sense, if you look at the numbers m, n and n, m minus n and we are assuming m is

bigger than n then we know that these two numbers are smaller than m, n . At least one of the numbers is smaller. So, so that means if you are going to this by hand we have gotten our numbers to be smaller and of course that helps when doing things by hand, but interestingly it helps also while doing things on a computer and we will see the reasons for it.


(Refer slide time: 5:44)

Example



$$\begin{aligned} & \text{GCD}(3977, 943) \\ &= \text{GCD}(3977-943, 943) = \text{GCD}(3034, 943) \\ &= \text{GCD}(3034-943, 943) = \text{GCD}(2091, 943) \\ &= \text{GCD}(2091-943, 943) = \text{GCD}(1148, 943) \\ &= \text{GCD}(1148-943, 943) = \text{GCD}(205, 943) \end{aligned}$$

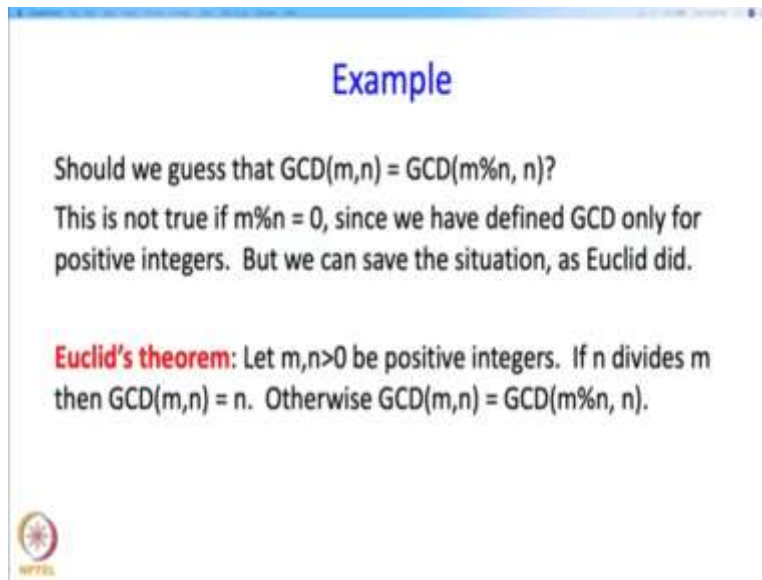
We should realize at this point that 205 is just $3977 \% 943$.
So we could have got to this point just in one shot by writing
 $\text{GCD}(3977, 943) = \text{GCD}(3977 \% 943, 943)$



So let us take an example, so suppose I want the GCD of 3977 and 943 So if we apply the previous idea then we know that this must be the GCD 3977 minus 943 or m minus n and n . But I can apply the idea and so if will do the subtraction I get GCD of n 3034 and 943, but I can apply this idea one more time, who is stopping me? So again I subtract 943 from that so I get 2091 but I can apply one more time and I supply 943 one more time so I get 1148 and 943. If I apply one more time, I am going to get 1148 minus 943 GCD of 205, 943.

Now, if you think about this you will realize that 205 is just $3977 \bmod 943$. So we did not have to go through subtracting 943 one at a time we could have just subtracted as many times in one shot and just taken the remainder okay. So, so that what exactly we should do instead of going in these several steps. So what should we do? We should write GCD of 3977, 943 is equal to GCD of $3977 \bmod 943$ and 943 because that is what this process is going to get us to.

(Refer slide time: 7:21)




Example

Should we guess that $\text{GCD}(m,n) = \text{GCD}(m\%n, n)$?

This is not true if $m\%n = 0$, since we have defined GCD only for positive integers. But we can save the situation, as Euclid did.



Euclid's theorem: Let $m,n>0$ be positive integers. If n divides m then $\text{GCD}(m,n) = n$. Otherwise $\text{GCD}(m,n) = \text{GCD}(m\%n, n)$.



Alright, so does that mean that GCD of m and n is GCD of $m \bmod n$ and n . Well we have to be a little careful because if $m \bmod n$ is 0, and since we have defined GCD only for positive integers then we cannot write this. But we could we could say that look lets define GCD for 0 as well or we could do something less dramatic and we could say maybe like what reasonably Euclid did. So we will write a statement which we could call Euclid's theorem, which is that let m and n be positive integers they have written as 0. If n divides m then GCD of m and n is n , if n divides m than clearly the largest number dividing both m and n must be n , But, if n does not divides m , then GCD of m and n must be equal to GCD of $m \bmod n$, n . So yes, so what we wrote at the top of the slide is almost true except when n divides m and I am not going to prove this, you can prove it I mean we almost have proved it but if you really want you can prove it again by writing m as some k time t and so on.

(Refer slide time: 8:56)

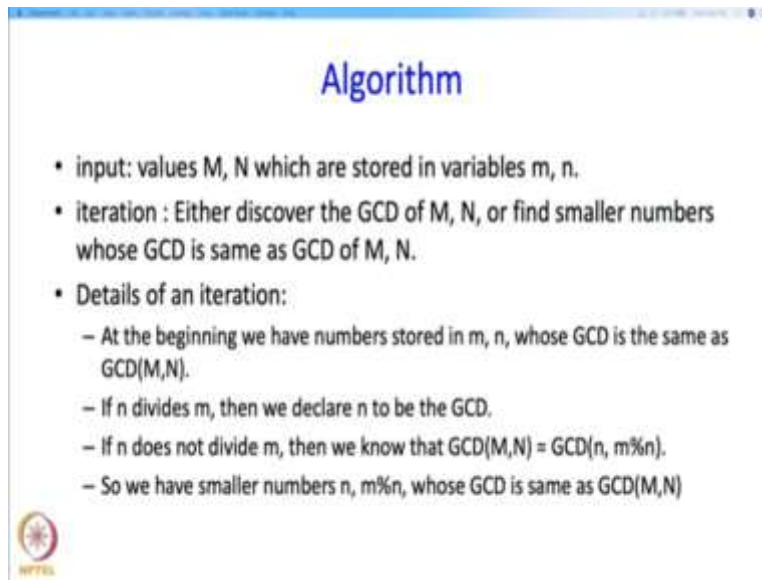
Example continue

$$\begin{aligned} & \text{GCD}(3977, 943) \\ &= \text{GCD}(3977 \% 943, 943) \\ &= \text{GCD}(205, 943) = \text{GCD}(205, 943 \% 205) \\ &= \text{GCD}(205, 123) = \text{GCD}(205 \% 123, 123) \\ &= \text{GCD}(82, 123) = \text{GCD}(82, 123 \% 82) \\ &= \text{GCD}(82, 41) \\ &= 41 \quad \text{because 41 divides 82.} \end{aligned}$$


Okay, so now we can go and do that problem again. So which is $\text{GCD}(3977, 943)$ is equal to $3977 \bmod 943$. Okay, so we have got GCD of 205 and 943, but now we can use the same idea again so GCD of 205 and $943 \bmod 205$. So that is what 205, 123 but again the same idea $205 \bmod 123$, 123 so that is 82, 123; 82, 123 is 82 and $123 \bmod 82$ and that is GCD of 82, 41. But now, if we take $82 \bmod 41$ we see that 82 is a multiple, 41 divides 82 so that means 41 must be the GCD . Because 41 divides 82 so 41 must be the GCD of this but the GCD of this is the GCD of this, this all the way till this. So we have found the GCD of 3977 and 943 and that happens to be 41.

Now let me point it out as an aside that we have done some number of divisions over here, but it will turn out that the number of divisions we do in this method is much smaller than the number of divisions you would have done if you would have actually done the factoring, forget the multiplications here. So that we are not going to prove, but you can take examples and you can persuade yourself. So, so we have manual algorithm now, are we ready to write a program? Well before that I would like you to make sure that you understand this and I would like you to find Euclid's GCD of say 26 and 42.

(Refer slide time: 10:47)



The slide is titled "Algorithm" in blue text. It contains a bulleted list of steps for finding the GCD of two numbers M and N. The steps are: 1. input: values M, N which are stored in variables m, n. 2. iteration : Either discover the GCD of M, N, or find smaller numbers whose GCD is same as GCD of M, N. 3. Details of an iteration: - At the beginning we have numbers stored in m, n, whose GCD is the same as GCD(M,N). - If n divides m, then we declare n to be the GCD. - If n does not divide m, then we know that $GCD(M,N) = GCD(n, m\%n)$. - So we have smaller numbers n, $m\%n$, whose GCD is same as GCD(M,N). There is a small logo in the bottom left corner of the slide.



So what is algorithm? So input “R” value is capital M and capital N which are stored in variables m and n, little m and little n. And in each iteration we will either discover the GCD of M, N capital M and N, so that will happen if we have the division and we have the larger number is perfectly divided by the smaller number. Or we will find the smaller number whose GCD is the same as GCD of M and N.

So that is, that is basically the idea of the iteration, so what are the details of the iteration? At the beginning we have the numbers stored in little m and little n, whose GCD is the same as GCD of M and N, well at the beginning of we will store capital M and capital N in m and n okay. So this statement is obviously true, but in each iteration we will try to maintain the statement. So if n divides m well we have we declare n to be the GCD, okay so that is by Euclid’s theorem. But if it is not, then we know that GCD of M and N or the GCD of m and n which is equal to the GCD of M and N must be equal to GCD of n and $m \bmod n$. So as a result we have changed the numbers n and $m \bmod n$, we have changed the numbers which we have in our hand. But we know that their GCD is still the same as the GCD what we wanted. So what we can do is we can now store the n and $m \bmod n$ in m and n. So here we are effectively assuming that the larger number is always in m and the smaller number is in n okay. So that is what is the algorithm is, okay.

(Refer slide time: 12:48)

Program for GCD

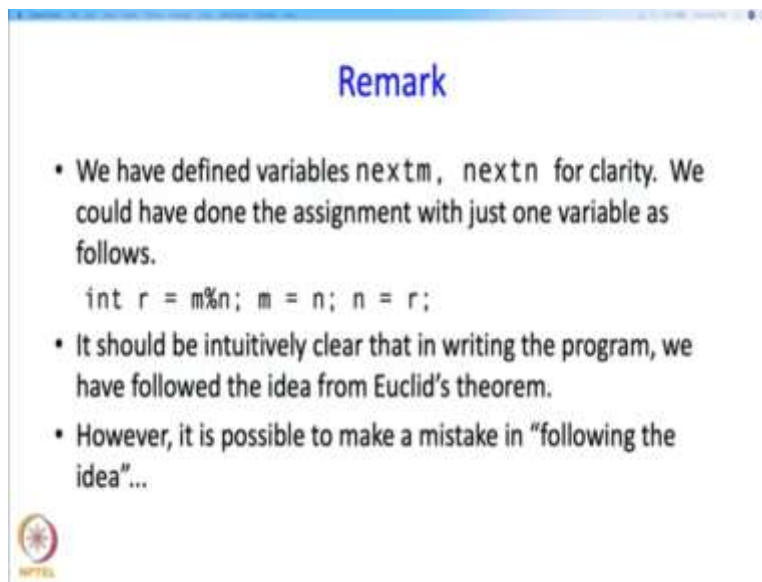
```
main_program{
    int m, n; cin >> m >> n;
    while(m % n != 0){
        int nextm = n;
        int nextn = m % n;
        m = nextm;
        n = nextn;
    }
    cout << n << endl;
}
// To store n, m%n into m,n, we cannot
// just write m=n; n=m%n;
// Can you say why? Hint: take an example!
```



So what is the program so the main program is `int m` and `n` and we read values of `m` and `n`. So the values read are capital `M` and capital `N`. Now, so we will check whether `m mod n` is equal to 0. So if it not equal to 0 then we have something to do. So what is it that we have to do? We will calculate the next value of `m` which is going to be `n`, the next value of `n` which is going to be `m mod n` and then we simply said `m` and `n` to be the next values. And if this iteration finishes that is if at some point we discover `m mod n` to be equal to 0 okay then we know that `n` must be the divisor. So we print out `n`.

So now there are some settle points over here which I should note, so if I want to store `n` and `m mod n` into `m` and `n` I cannot just write `m equal to n`, `n equal to m mod n`. Why? Because if I write `m equal to n` then when I try to do this `m mod n` this `m` would already have the value `n` so then I would be doing `n mod n`, so that would just give me 1. So that is not good. So that is why we need to do something elaborate like this just to be careful we are saying what is the next value of `m` and of `n` and we are doing that okay. Alright, I guess I will resolve that for you okay.

(Refer Slide Time: 14:25)



Remark

- We have defined variables `nextm`, `nextn` for clarity. We could have done the assignment with just one variable as follows.

```
int r = m%n; m = n; n = r;
```
- It should be intuitively clear that in writing the program, we have followed the idea from Euclid's theorem.
- However, it is possible to make a mistake in "following the idea"...

Alright, so `next m` and `next n` have been defined for clarity we could have just used 1 variable instead of both and that is what is shown over here, but you do not have to really worry about these details you can just use two variables, who is counting. Now, intuitively it should be clear that we have followed Euclid's theorem or the idea from the Euclid's theorem. But, we already said that there can be one mistake that we could make we could just write `m` and `n` the new values of `m` and `n` directly into `m` and `n` and that could be one mistake. Then you might also worry is `m` always going to be larger than `n`, are we taking care of that, do we need to take care of that okay so these are things that we need to worry about because if those things are not working out then maybe our program will not run correctly.

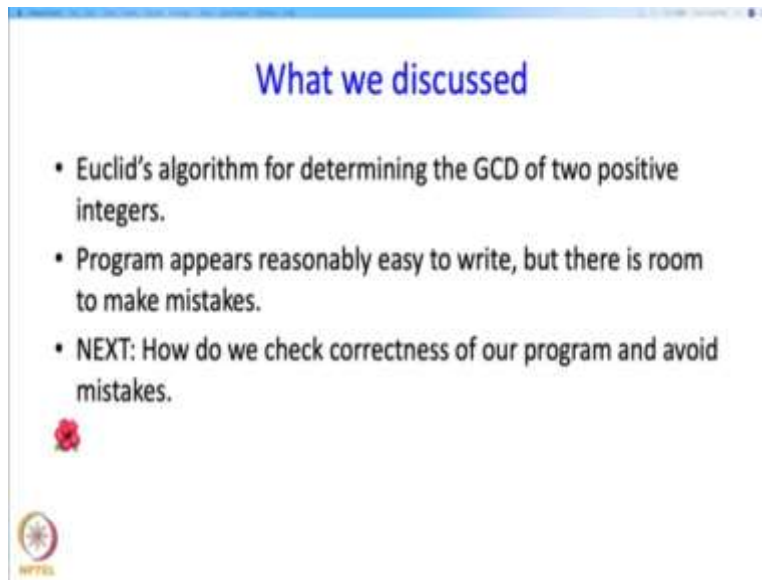
(Refer slide time: 15:27)

```
Exercise: Find the mistake in this program

main_program{
  int m, n; cin >> m >> n;
  while(m % n != 0){
    int nextm = n;
    int nextn = m % n;
    m = nextn;
    n = nextm;
  }
  cout << n << endl;
}
```

So here is a wrong program which seems to be following exactly the idea that we have discussed. So it does all of those things, but in this update mistakenly we put what we intended for m into n and what we intended for n into m, small mistake, right? But as a result of this the program will not run. So you are asked to figure out what is the mistake or whether there is a mistake. And maybe you can check that by exchanging the values assigned to m and n we get the right answer. But this exercise makes the following point that once we write the program, we claim that we are following Euclid's idea, but have we, while doing this, have we introduce some mistakes into it? So we need some ways of checking that okay and so we will look at that in the next segment.

(Refer slide time: 16:21)



The slide is titled "What we discussed" in blue text. It contains three bullet points: "Euclid's algorithm for determining the GCD of two positive integers.", "Program appears reasonably easy to write, but there is room to make mistakes.", and "NEXT: How do we check correctness of our program and avoid mistakes." There is a small red flower icon and a circular logo with a star in the bottom left corner.

What we discussed

- Euclid's algorithm for determining the GCD of two positive integers.
- Program appears reasonably easy to write, but there is room to make mistakes.
- NEXT: How do we check correctness of our program and avoid mistakes.

So what did we discussed in this segment? We discussed Euclid's algorithm for determining the GCD of two positive integers and we said that the program appears easy to write, but there is room to make mistakes and in the next segment we will check for correctness of our program and make sure that we have not made any mistakes. We will stop here.