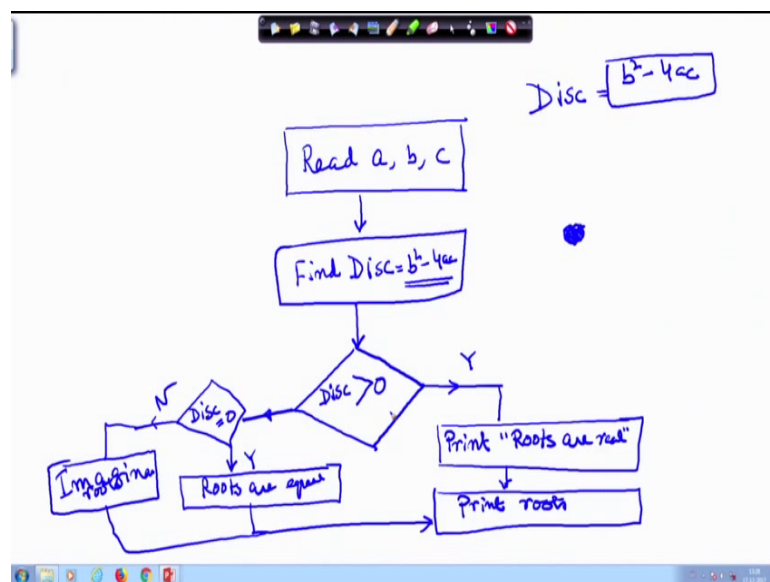


**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 23**  
**Example of Loops**

So, we were looking at solving a quadratic equation and we have seen that there are three cases. One is if the discriminant is 0 we print the roots that their roots are real, sorry if the discriminant is greater than 0 then the roots are real, otherwise if it is equal to 0 then the roots are equal and if the discriminant is neither greater than 0 nor equal to 0; that means, it is less than 0 then the roots are imaginary.

(Refer Slide Time: 00:19)



So, next what we did is we started with a program like this where we declared the coefficients here and then the discriminant and the intermediate variable d and 2 roots right root 1 and root 2.

(Refer Slide Time: 00:45)

```
#include <stdio.h>
#include <math.h>
main ( )
{ /* Program to find roots of a Quad. Eqn */
  int a, b, c ;
  float Disc, d ;
  float root1, root2 ;
```

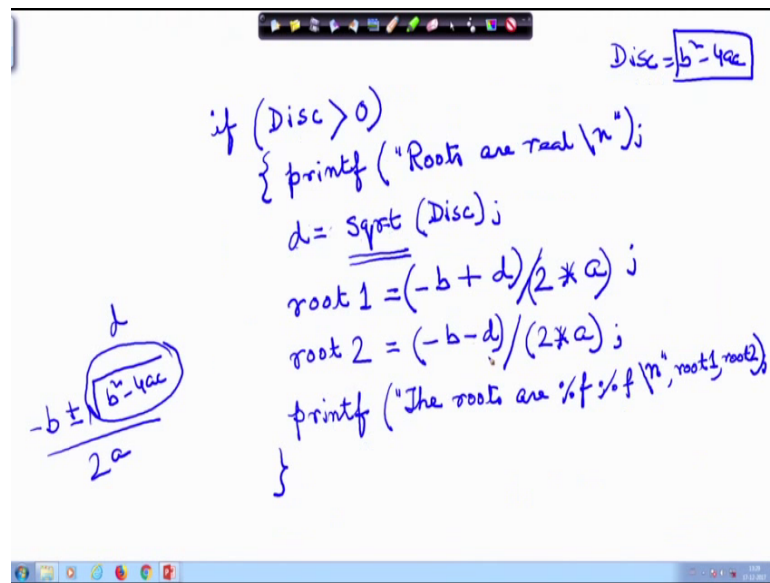
Next we proceeded with this and we modified this part.

(Refer Slide Time: 01:04)

```
printf ("Enter integer coeff a then b then c");
scanf ("%d %d %d", &a, &b, &c);
Disc = b*b - 4*a*c ;
if (Disc > 0)
  printf ("Roots are real\n");
else
  if (Disc = 0)
    printf ("Roots are equal\n");
  else
    printf ("Roots are imaginary\n");
```

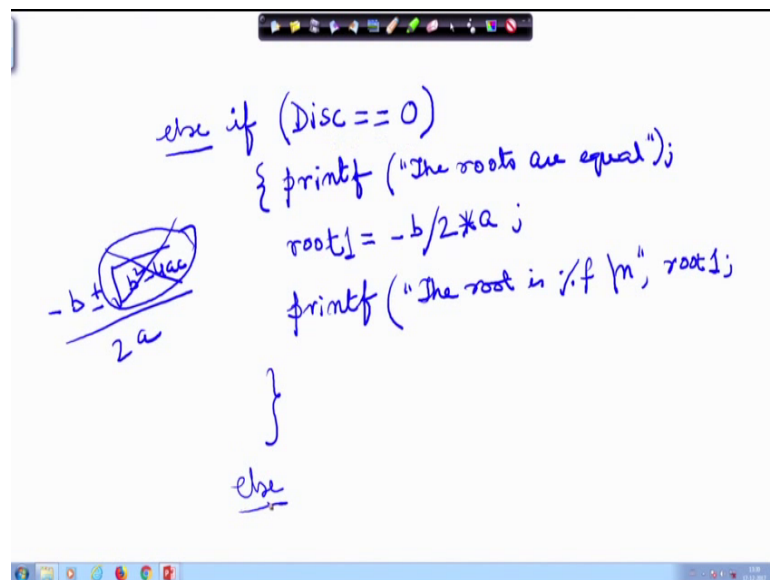
If disc is greater than 0 printf roots are real and then I modified this, this part and replaced this part with this, if discriminant is 0 prints the roots are real and also compute the root.

(Refer Slide Time: 01:16)



Now, if the discriminant is equal to 0 then prints the roots are equal compute the root and print it. Now, I asked you to see how you will deal with the third case if the discriminant is not equal to 0.

(Refer Slide Time: 01:44)



So, this if, when this if comes then we was already failed the condition the discriminant is greater than 0 and if the discriminant is not equal to 0 then we will come to just an else here right, and let me go to the next page to write the else part of this.

(Refer Slide Time: 02:07)

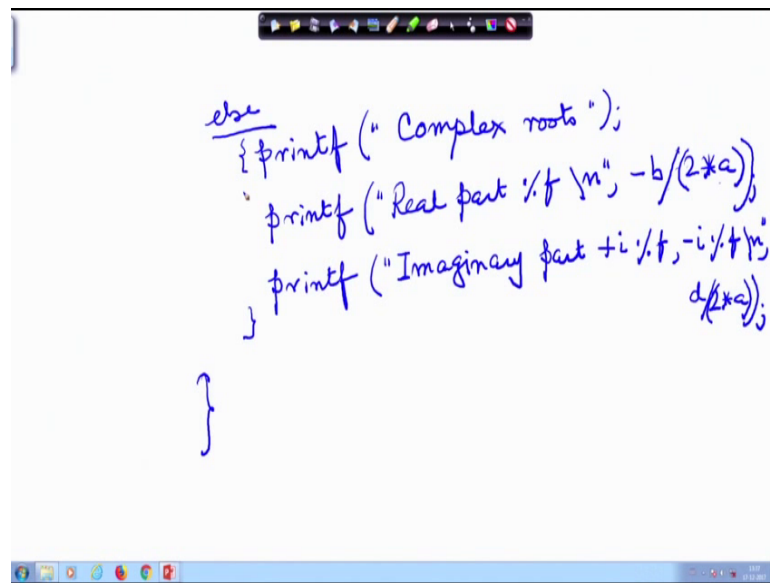
The image shows handwritten mathematical work on a whiteboard. At the top, it says "else" followed by a C-style printf statement: `{ printf("The roots are imaginary\n");`. Below this, there are two main derivations. The first shows the real and imaginary parts of a complex root:  $\frac{-b}{2a} + i \frac{d}{2a}$  and  $\frac{-b}{2a} - i \frac{d}{2a}$ . The second derivation shows the quadratic formula  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ , which is then simplified to  $x = \frac{-b \pm \sqrt{\text{Disc}}}{2a}$ . It then splits into two cases:  $\frac{-b}{2a} + i \frac{\sqrt{\text{Disc}}}{2a}$  and  $\frac{-b}{2a} - i \frac{\sqrt{\text{Disc}}}{2a}$ .

So, it will be else simply I am sorry here did I give the parenthesis, yes I did. So, I am ok. This is please note these are the points where we often make the mistakes. Else printf "the roots are imaginary backslash n" fine, and what are the roots? So, if we if the roots are imaginary then the real part say one, one root is minus b by 2a is the real part and plus i that is what we, i d by 2a is it clear another this is one root another root is b by 2a minus i d by 2 a.

What does, so this d, basically you see what we are doing is x is minus b plus minus root over b squared minus 4 ac by 2a. So, this is my discriminant and I take the square root of that. So, plus minus root over d by 2a right. In my program what did I write d or discriminant square root of disc sorry disc. So, I should not use different name. So, it should be disc by 2a and that square root of disc can be either imaginary, so negative if this is negative then I will find it is its absolute value and, this can be written as minus b by 2a plus square root of disc by 2a is one root another root is b by 2a minus root over disc by 2a and since its imaginary it should be i, the imaginary part.

So, I can print it as a real part to be b minus 2a and imaginary part is square root of disc by 2a and the square root of disc is nothing but d. So, minus b by 2a plus i d by 2a that is what I am writing here all right. So, this is the real part this is the imaginary part. So, I can print it in different ways.

(Refer Slide Time: 05:28)

A screenshot of a whiteboard with handwritten C code. The code is written in blue ink and shows an 'else' block with three printf statements. The first printf prints "Complex roots". The second printf prints "Real part %f\n", with the argument being -b/(2\*a). The third printf prints "Imaginary part +i %f, -i %f\n", with the argument being d/(2\*a). The code is enclosed in curly braces, with a closing brace for the 'else' block and a closing brace for the entire block.

```
else
{
    printf("Complex roots");
    printf("Real part %f\n", -b/(2*a));
    printf("Imaginary part +i %f, -i %f\n", d/(2*a));
}
```

I can just say else printf roots are imaginary or rather I should have said this is wrong to say. What should I have said, you should have I mean I am doing a mistake here all through, starting from this point, you must have observed it.

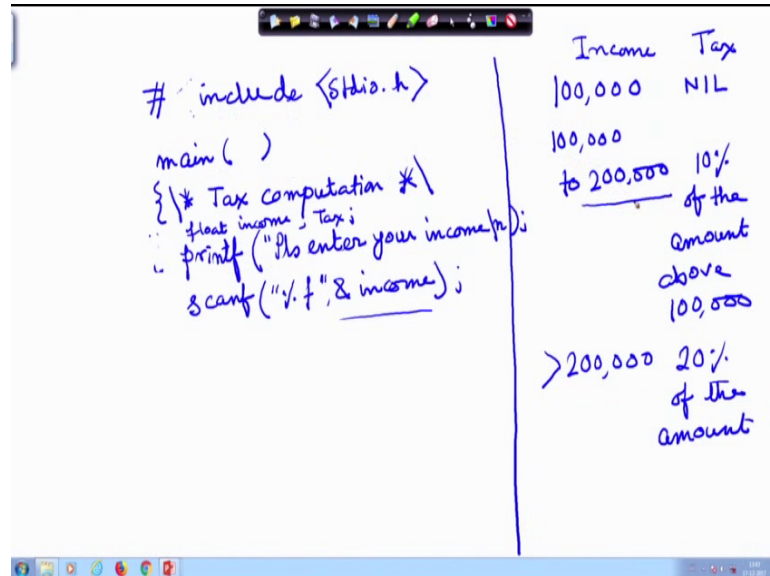
Here the roots are complex roots then in that case, not the imaginary roots because it has got a real part and the imaginary part. So, the roots are complex. I should have written roots are I should have written complex roots and then I could print real part percentage f backslash n, I can put an expression here minus b divided by 2a. We can make it a little cleaner, looks very nasty here, b divided by 2a that is the imaginary part that is a real part all right.

And printf imaginary part is plus i percentage f comma minus i percentage f backslash n followed by. What is my imaginary part? Imaginary part is d by 2a. So, you see I have put this i part plus i and i here comma b divided by again d divided by twice a all right. So, this and then of course, everything is done so I come to the end of my program. Before that here there should be a parenthesis. So, this is the complex part.

Now, as we will become little more conversant with programming you can see that I have repeated many things right. So, instead of doing it there are some common you can think of how can write it in a much more elegant and shorter way. But this is an example which gives you an exposure to the use of if then else statements, if else statements in C.

Next, well another example we can take up. Say for example, we are going to compute the income tax of a person.

(Refer Slide Time: 09:56)



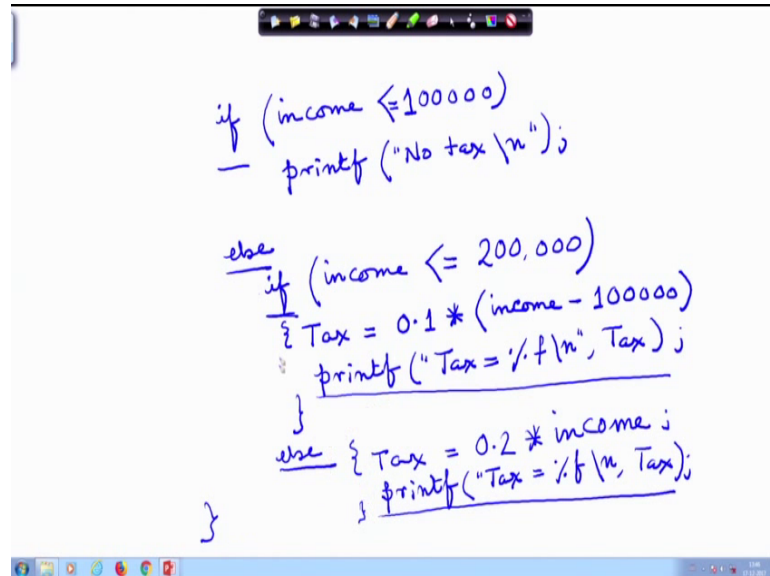
Suppose the if the income tax is less than 100,000 rupees then income tax is nil, if it is between 100,000 to 200,000 rupees then you pay 10 percent of the amount above 1 lakh, 100,000. For income above this 200,000 if the income is above 200,000 then you pay 20 percent of the amount whole amount not how much is exceeding 10,000 suppose this is our income tax policy. How can we do this?

So, what is the input that I need from the user? I need this income from the user right; and what is the other variable that I want to compute? The tax. So, here earlier I needed math dot leave in this case probably I will not need that, but, I will not need that if needed I will can always add it later main and then you can say put a comment computation of income tax, tax computation all right.

Now, I start my program all right I can put the parenthesis above, does not matter . Then printf its always better to do this, "Please enter your income" because if you do that then you make your program interactive; that means, the user can see what you are doing right, please enter your income. Well I have not yet declared the variables at all. So, here I should write float income and tax. Then here scanf, percentage f and income, so I read the income here fine. Then what should I do? Next let me come to the next page then or

because there will be number of if conditions. So, this part is ok, I have read then come here.

(Refer Slide Time: 14:11)



```
if (income <= 100000)
    printf("No tax\n");
else
    if (income <= 200000)
    {
        Tax = 0.1 * (income - 100000);
        printf("Tax = %f\n", Tax);
    }
    else
    {
        Tax = 0.2 * income;
        printf("Tax = %f\n", Tax);
    }
}
```

Now, if income is less than or equal to 100,000 printf no tax, else what is my principle; 1 lakh to 2 lakhs, 10 percent of the income, 10 percent of the income above 100,000, so else if income is less than equal to 200,000. So, it is not when it is coming here it is already greater than 100,000, if it is less than 200,000 then tax will be 10 percent; that means, 0.1 times the amount that the amount exceeding 100,000, so income minus 100,000. Printf tax equals percentage f backslash n tax, is this part clear.

This part is so when am I coming here this if condition as failed. So, I am coming here; that means, it is not less than 100,000 so it is more than 100,000, but if it is less than 200 less than equal to 200,000 then I am the second bracket and I will be paying 10 percent of the amount that is exceeding 100,000 that is why I computed this and then I am printing this. So, here it is more than one statement I put a bracket. Here you see under if I had only one statement. So, putting a bracket is not mandatory, but here it is mandatory otherwise it will mean something else.

So, else this, now if this is also not true, that is if this if does not satisfy then I am in the third bracket that is here what should I do else tax is 20 percent of the income and printf tax assigned percentage f backslash n tax. Now, here then I will come to the end. So, I come to this beginning this, this beginning point and I end the program here.

Now, you can see that I can reduce this program a little bit. How? I have written this printing the text this thing the same thing twice, I am sorry here they should be another bracket for completing this. This should be completed and then the next bracket. Now, this I could have done later after these 2 use all right, but under this else under this else. So, you can also try to reduce it and as an assignment you should run this program and get yourself satisfied all right. So, this is another classical example of if then else usage all right.

Given this we will. Now, move to some more examples, some more examples of the other construct that is while and do while type of constructs. So, here is an example to show if a number is prime or not.

(Refer Slide Time: 19:23)

Example 1: Test if a number is prime or not

```
#include <stdio.h>
main()
{
    int n, i=2;
    scanf ("%d", &n);
    while (i < n) {
        if (n % i == 0) {
            printf ("%d is not a prime \n", n);
            exit;
        }
        i++;
    }
    printf ("%d is a prime \n", n);
}
```

Handwritten annotations on the slide:

- Next to the code,  $n$  is written with ~~13~~ and 14 below it.
- To the right,  $i$  is written with 2, 3, 4, 5 listed vertically.
- At the bottom right, the number 12 is circled.
- Blue arrows indicate the flow of the while loop: from the condition  $i < n$  to the if statement, then to the printf and exit statements, then to the i++ statement, and finally back to the condition.

Now, how do I go about it let us first think of the algorithm for finding out whether a number is prime or not.



(Refer Slide Time: 19:45)

The whiteboard contains the following handwritten content:

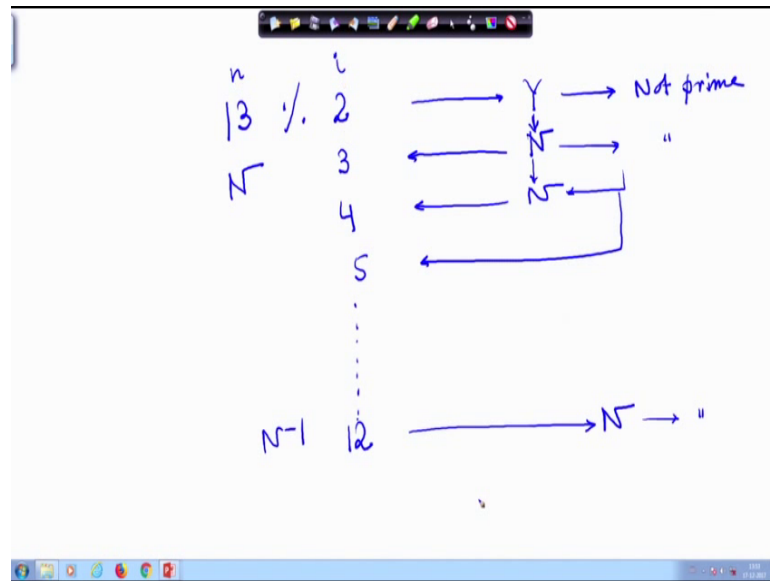
- A circled number 13 with an arrow pointing to the number 2.
- Equation:  $14 \% 2 = 0$
- Equation:  $26 \% 2 = 0$
- Equation:  $13 \% 2 = 1$  (boxed)
- Equation:  $15 \% 3 = 0$  (underlined)

A small video inset in the bottom right corner shows a man with glasses and a green shirt.

Say for example, I take a number 13. I want to find out whether 13 is prime or not. So, when is a number called prime? When the number is not divisible by any other number other than the number itself and 1. So, in order to find out whether 13 is prime or not, what should I do? I will start with 2 and I will try to see whether the 2 is dividing 13. How do I know where the 2 is dividing 13? By the modulus operator, if there will be 2 integers say 14 modulus 2 the modulus leads gives me the remind, sorry remainder. So, if 14 is divisible by 2 the remainder will be 0. 26 divided by 2 the remainder will be 0, but 13 divided by 2 the remainder will be 1.

So, similarly, suppose I want to see where there are number 15 is divisible by 3 I will do the modulus operation of 15 with 3 and the remainder will be 0. So, 15 is divisible by three right.

(Refer Slide Time: 21:32)



So, now if I want to find out whether 13 is a prime or not; I will first try to divide see where that is divisible by 2. If it is divisible yes, will tell me not prime right it will tell me not prime, but no then I will try to divide it by 3, if this is divisible by 3 then again not prime. No, I will divide by 4 then; obviously, not divisible then I will divide it by 5 so on so forth I will go till I divide it by 12. Since of course, by 13 it will be divisible, if the answer continuously up to 12 is no then we say it is not a prime right, not a prime.

So, here you see there are 2 things happening. One is I am taking a check branching here I am dividing it by 2, if it is divisible is it is not a prime, if no then I am taking another path. What is that path? That path is again trying with the next successor of this 2, yes; if the result is yes I take this path otherwise I will follow this path with a successor of this. So, the same thing trying with the successor I am doing continuously repeatedly if the divisibility result is false; that means, if it is not divisible I will continuously go on till I reach if this number is N, till I reach N minus 1 right, up to that I will try up to N minus 1 I will try. So, this is an example where there is a branching as well as a looping, another real life mathematically important application. So, let us look at the program now.

Let us try to understand the program here. This include stdio dot h is known to you mean. I am declaring 2 variables one is n another is i; n is an integer, i is also an integer, initialize to 2. I am reading the number here say 13 while i is less than n; that means, i

was 2 if we look at this page we started with 2 we started with 2 and then went on right, this is the value of  $i$  and this is the value of  $n$ .

Now, while  $i$  is less than  $n$ , I had 13 and  $i$  is 2 this is  $n$ , I will go on up to 13 up to while  $i$  is less than  $n$  that means up to 12, I will go on trying this thing what; if  $n$  is divisible by  $i$ . Is  $n$  divisible by  $i$ , yes then print that the number  $n$  is not a prime and then exit we come out of this loop. Just like break we come out of this loop, but break is not applicable in if statement exit we can write. So, I come out of the loop.

Otherwise what I am doing still in the while loop, the while loop is starting from this point and going up to this point. I am incrementing  $i$ , so  $i$  becomes 3 and again I go and try to see if it is divisible with 3 know then I come make it 4, make it 4, I will again try this one fails, it is not divisible so I do not do this part, I do not do this part. I come here make it 5 and go on in this way while  $i$  is 12, up to 12 I check and then it becomes 13. So, I come here and ultimately I come out of the loop and till 13 I could see that no, I could find no number no integer that is dividing it. So, we say that this is a prime number.

Say for example, what would have happened if I had given the instead of  $n$  13 if I had given it 14. What would be the change in this flow? I would start with the 2 and again I divide it by 2, try dividing it by 2 it is divisible I print that it is not a prime and I exit and come out of this entire while loop. So, in the while loop here this I am exiting from the while loop all together, not from the if loop I am exiting from the while loop if is not a loop by the way. I am exiting from the while loop completely. So, this example also illustrates the use of if here and the while loop here all right.

We will see a few more examples in the future lectures.