

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

Lecture – 16
IF-ELSE Statement (Contd.)

(Refer Slide Time: 00:21)

```
if e1 s1          if e1 s1
else if e2 s2      else if e2 s2

if e1 s1          if e1 s1
else if e2 s2      else if e2 s2
else s3           else s3

if e1 if e2 s1    if e1 if e2 s1
else s2           else s2
else s3           else s3

if e1 if e2 s1    if e1 if e2 s1
else s2           else s2
```

Welcome. In the last lecture we had looked at the nested if-else structure. Today we will continue with that discussion.

(Refer Slide Time: 00:30)

Example

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if (a>=b)
        if (a>=c)
            printf ("\n The largest number is: %d", a);
        else
            printf ("\n The largest number is: %d", c);
    else
        if (b>=c)
            printf ("\n The largest number is: %d", b);
        else
            printf ("\n The largest number is: %d", c);
}
```

a = 25
b = 20
c = 15

So, here is an example that uses the nested if-else structure. Here you can see again let us come to the programming fundamentals in C, we have got just for revision we start with an include stdio dot h and then we have got the main function. Inside the main function we declare 3 variables a, b and c and probably you might have guessed now that we are again trying to find out our, I mean the maximum of the 3 integers a b and c.

So, what are we doing next? We are first reading the 3 numbers from the keyboard right, scanf, percentage d, percentage d, percentage d, then a, b and c preceded with an ampersand and you know why this ampersand is used that is the address of the location a b and c.

Now, comes the main logic here if a is greater than or equal to b, a is 25 b is 20. So, if a is greater than equal to b then I check whether a is greater than c. So, this is passed then I check this. So, if a is greater than b then I proceed to do if a is greater than equal to c if that is also true, then we print the largest number is a; otherwise what we do not know as yet.

(Refer Slide Time: 02:47)

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if (a>=b)
    {
        if (a>=c)
        {
            printf ("\n The largest number is: %d", a);
        }
        else
        {
            printf ("\n The largest number is: %d", c);
        }
    }
    else
    {
        if (b>=c)
        {
            printf ("\n The largest number is: %d", b);
        }
        else
        {
            printf ("\n The largest number is: %d", c);
        }
    }
}
```

Handwritten annotations: a = 25, b = 20, c = ~~25~~ 30. A bracket groups a and b, with an arrow pointing to the first if statement. A large 'X' is drawn over the else branch of the inner if-else block.

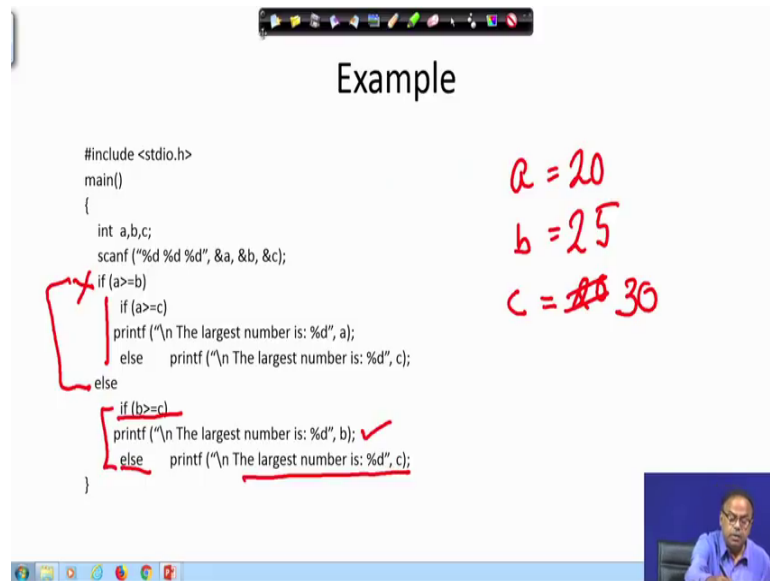
So, if a is greater than b, a is 25, b is 20, c is 15 in that case a is greater. Let us suppose c is thirty then what happens a is greater than b true then I come and compare a is greater than c this part, no false. Therefore this else is with respect to this nearest if that we learnt in the earlier lecture. So, if this fails then we print the largest number is c because a was greater than b, but a is not greater than c therefore, c must be the largest.

(Refer Slide Time: 03:40)

Example

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if (a>=b)
    {
        if (a>=c)
            printf ("\n The largest number is: %d", a);
        else
            printf ("\n The largest number is: %d", c);
    }
    else
    {
        if (b>=c)
            printf ("\n The largest number is: %d", b);
        else
            printf ("\n The largest number is: %d", c);
    }
}
```

$a = 20$
 $b = 25$
 $c = \del{20} 30$



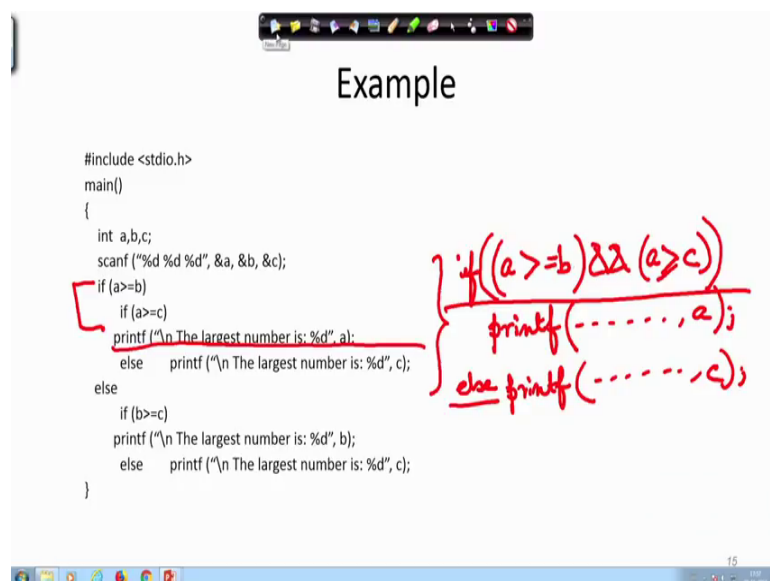
Now, if a is not greater than b. Suppose a is 20 and b is 25, then a greater than b this fails, this fails then I am not entering this block at all I am straightway going to this else then I am checking a is not greater than b, but therefore, b is must be greater than or equal to a or not even equal to b must be greater, greater than a then I check if b is greater than c suppose b is greater than c then the largest number is b otherwise if that is not. So, b is not greater than c suppose c was thirty then this else will come because this if we will fail therefore, I will print gain the largest number is c.

(Refer Slide Time: 04:46)

Example

```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if (a>=b)
    {
        if (a>=c)
            printf ("\n The largest number is: %d", a);
        else
            printf ("\n The largest number is: %d", c);
    }
    else
    {
        if (b>=c)
            printf ("\n The largest number is: %d", b);
        else
            printf ("\n The largest number is: %d", c);
    }
}
```

$\left. \begin{array}{l} \text{if } (a >= b) \ \&\& \ (a > c) \\ \text{printf } (\dots, a); \\ \text{else printf } (\dots, c); \end{array} \right\}$



So, I would suggest you to work it out yourself and I can also write the same thing. If you consider that this if a is greater than b then if a is greater than equal to c, tell me if I write it in this way if a is greater than equal to c and and; that means, and a is sorry here I right a is greater than equal to b and a is greater than equal to c printf this line largest is a else printf largest is c. So, do you think that these two are equivalent? These two nested ifs and this logical operator.

Now, here I should have put a parenthesis here. Please note I need a parenthesis here it is better that if I put it it becomes much clearer. Are these two equivalent? If you just think a while you will find yes they are equivalent because I am computing this and then I am computing this if this any one of this is false I will not be executing this statement right. So, this is an example of nested if.

We can have many such examples. Let me give you one example. Let us workout right now.

(Refer Slide Time: 06:45)

```

scanf ("%d", &no_of_sides);
if (no_of_sides == 3)
{
scanf ("%d %d %d", &a, &b, &c);
if (a == b || a == c || b == c)
printf ("It is isosceles\n");
else
printf ("Not isosceles\n");
}
else
printf ("Not a triangle\n");

```

Suppose I am trying to see if a figure is a triangle or a square or a rectangle or rectal I mean 4 sided figure whether it is a triangular figure or a quadric quadrilateral figure. And if it is a triangular figure then I want to see whether it is isosceles; that means, two sides are equal or it is not isosceles. Suppose that is what I want to do. So, how can I proceed to do that; how would I write the logic? Let me just only write the relevant part. I have done some the declarations are here and then I have done scanf percentage d, number of

sides all right number of sides. So, number of sides are variable. And then I also then; what I can do? I check if number of sides is equal to 3 then I will be doing something all right because in that case it will be a triangle.

Then what do I do I? Then read that is scanf the 3 sides of the triangle percentage d, first and suppose all the sides are integer has got all the sides have integer values all right 5, 6, 5 something of that sort. That is why I am putting percentage d then I am reading the 3 sides and a and b and c semicolon.

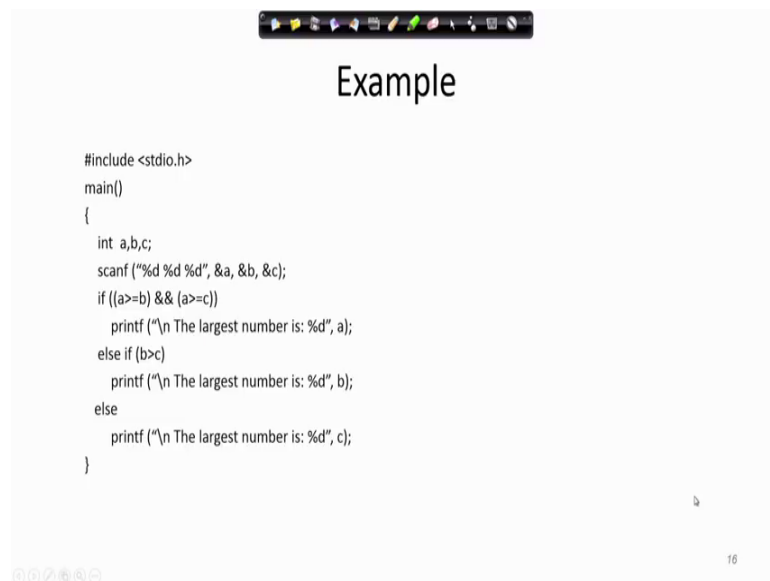
If a is same as b or you remember this is the logical OR, or a is the same equal to c or b is equal to c then what can I say, printf "it is isosceles\n", as our common practices. If any of these conditions are true either this is equal to this or this is equal to this or this is equal to this then is isosceles. Else I will printf say I am writing in brief "not isosceles". You can write in a much better way. Remember this double quotes we complete this. Now, that is the, I complete this. Now, this is the if part. If the number of sides is 3 if the number of sides is not 3 then I do not do anything I will come at this point because this if you will fail I will come at this point. I can say here can I write here printf not a triangle, can I do this, just think because here I have checked that the number of sites is 3. If it is number of sites is 3 I will do this and print and otherwise I write it here is it all right. If you look at it you will see that there is a problem here, that if the number of sites is 3 and then it checks whether it is isosceles or not and then it comes out and again prints is not a triangle I do not want that. So, what should I do, how can I avoid this problem?

Here you must have guessed now, here I have to put an else that goes with this if all right and then if this is not true then only this will be executed otherwise some point down the line here will be executed. Another thing that I would like to mention in this case look at the use of this parenthesis why was this parenthesis required because I had more than one statements for this if condition if this condition is true then there are more than one statements.

Now, here is a puzzle. How many statements are there inside this block? How many statements are there? 1, 2, 3, 4, let us see. Here is one statement scanf so one and then this if statement you see goes up to this. So, that is one statement, but here I am using I said that if felt is a structure. So, this entire thing is a statement. So, actually I have got

two statements. Since I have got more than one statement I had to put these braces. Now, suppose of course, then the program will not work here, suppose I had not written this statement this statement was not there all right this statement was not there in that case I could have done away with these braces because then it is within this if statement there is one statement only this brace was not essential. So, you could see this and you can practice and in the assignment we will also give few programs that you have to do using this sort of if-else type of structure.

(Refer Slide Time: 14:52)



```
#include <stdio.h>
main()
{
    int a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>=b) && (a>=c))
        printf ("\n The largest number is: %d", a);
    else if (b>c)
        printf ("\n The largest number is: %d", b);
    else
        printf ("\n The largest number is: %d", c);
}
```

So now, here is the example that we did.

(Refer Slide Time: 15:02)

The slide is titled "Confusing Equality (==) and Assignment (=) Operators". It contains the following text:

- Dangerous error
 - Does not ordinarily cause syntax errors
 - Any expression that produces a value can be used in control structures
 - Nonzero values are true, zero values are false

Handwritten annotations in red ink include:

- An arrow pointing to `if (a = b)` with a line underneath it.
- Below that, the text `a = b` with a line underneath it.
- To the right, an arrow pointing to `a ← b`.
- Further right, an `if (a == b)` statement with a line underneath the condition, and a block of code below it: `{`, `↑`, `}`.

Now, there is a problem here there is a danger I would rather say. You have seen earlier when I was writing I was writing if a is equal to b, then I was doing something right. Now, this is very important that is a common point of source of, common source of error that often because we have learnt to say equal to like this it will we can write in this way. Now, the problem is if I write it in this way when a compiler looks at this what will it assume it will see that is an if often you will not find that it will cause an error because what the compiler will think is well, what is the meaning of this; a this means a is assigned the value of b. So, it will try to successfully, it will try to transfer copy the value of b to a all right and this operation will be executed successfully.

Now, if x in any expression is computed successfully; what does it return? If you recall it returns a one and one means true; that means, this condition will evaluate to true. I once again repeat. Now, we can use any expression that produces a value in the control structure. Now, if it be a nonzero value then is true and 0 value is false.

(Refer Slide Time: 17:03)

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
 - Does not ordinarily cause syntax errors
 - Any expression that produces a value can be used in control structures
 - Nonzero values are true, zero values are false

Handwritten code examples:

```
if (z = a + b + c) → if (1)
    printf("A");
else printf("B");
```

The slide also features a small video inset of a speaker in the bottom right corner.

So, if I write `if a plus b plus c printf "A" else printf "B"` what will happen? `a plus b plus c` is if they will be just be added, but there will be no assignment suppose I make it more meaningful. Suppose I make it `z` sorry `z` assigned (Refer Time: 17:53) I write `z` assigned `a plus b plus c` all right then this there is a problem here because there is no semicolon here right. So, this will become suppose I do this.

Now, this will be computed and this assignment will be done and this assignment will be done successfully. So, anything that is done successfully will return this will be equivalent to `if 1`, `if 1` means it is true and so what will be printed, `printf A` will be executed, this will be printed.

(Refer Slide Time: 18:48)

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
 - Does not ordinarily cause syntax errors
 - Any expression that produces a value can be used in control structures
 - Nonzero values are true, zero values are false

Handwritten notes in red ink:

```
if (i)
{
}

null
Statement
;
if (1)
{
}
}
```

The slide also shows a Windows taskbar at the bottom with various application icons and a system tray.

Now, similarly I can say let us see I just write if this and something. What does this mean? Semicolon nothing before this we call it a null statement; that means, I have not state anything meaningful, but it is still a statement. Remaining silent is sometimes a set statement. Now, this means that this is always true this will always be true. So, this is equivalent to if 1 this; that means, always you do this statement this, whatever is here you read do it always all right. So, this is a interesting thing in C.

(Refer Slide Time: 19:43)

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
 - Does not ordinarily cause syntax errors
 - Any expression that produces a value can be used in control structures
 - Nonzero values are true, zero values are false
- Example:

```
if ( payCode == 4 )
printf( "You get a bonus!\n" );
```

Checks paycode, if it is 4 then a bonus is awarded

Equality check improper

```
if ( payCode = 4 )
printf( "You get a bonus!\n" );
```

Handwritten notes in red ink:

```
} if (i)
printf( "You get a bonus!\n" );
```

A flowchart diagram shows a diamond-shaped decision box labeled "paycode = 4?". An arrow labeled "N" points left from the diamond, and an arrow labeled "Y" points down from the diamond.

The slide also shows a Windows taskbar at the bottom with various application icons and a system tray.

However sometimes you know it varies all with the compilers also some compilers take care of such situations and may give you a warning some very few compilers can also give any error. Now, let us look at this example sorry this example.

If pay code is equal to 4, what does this mean? This means if the pay code is 4 this is a logical equality if this is true then will print you will get a bonus. Now, instead of, I want to do like this, this means that I am coming here I am checking pay code in non C language you speak out 4 yes I do something no I do something else. But if I had written it in this way what would it mean?

If pay code assigned 4. Now, pay code being out suppose pay code is an integer type of variable then this can be assigned the value 4 successfully and anything that is done successfully this will result in if done successfully one do this. That means, it will be successfully executed it is not doing what you are intending. So, sometimes we are saved by the compiler when it points out that note here, here is something some syntax error that you are committing all right, but there are situations when we unintentionally can do such mistakes which will go unnoticed by the compiler and the result that will be getting may not be what we desired.

So, this is a very critical point you should keep it in mind for all programming languages we will find there such nuances some specialties which you will have to keep in mind. We are just mentioning here the ones for C. But if I had done it in this way it would be ok.

(Refer Slide Time: 22:15)

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
 - Does not ordinarily cause syntax errors
 - Any expression that produces a value can be used in control structures
 - Nonzero values are true, zero values are false
- Example:

```
if ( payCode == 4 )
    printf( "You get a bonus!\n" );
```

Checks paycode, if it is 4 then a bonus is awarded

Equality check proper

```
if ( payCode == 4 )
    printf( "You get a bonus!\n" );
```

18

So, generalization of expression evaluation in C is assignment operated is also a part of expression that we know.

(Refer Slide Time: 22:24)

Generalization of expression evaluation in C

- Assignment (=) operation is also a part of expression.

`i=3;` → Returns the value 3 after assigning it to i.

if (0/num3)
if (1)
if (3)

19

Now, let us see here i is assigned 3. So, typically some compilers what they do till now, I was saying that if an instruction is executed correctly it will return 1, but generally speaking it will return the value that has been assigned.

So, typically here i assign 3 will return the value 3 after assigning it to it, but for us when we do what we are bothered about if the condition this part is open sorry I would rather.

Now, say is part is 0 or nonzero. If it is nonzero then this true I was saying if 1 it is true, if 3 that is also true the only falsity will come if it is 0. So, this returns the value 3 after assigning it to i.

(Refer Slide Time: 23:48)

Generalization of expression evaluation in C

- Assignment (=) operation is also a part of expression.

```
int i=4, j;  
if(i=3)  
j=0;  
else  
j=1;
```

Diagram illustrating the evaluation of the expression `i=3` in the `if` statement. The expression `i=3` is evaluated, returning the value 3 after assigning it to `i`. The code snippet shows `int i=4, j;` followed by `if(i=3) j=0; else j=1;`. The diagram shows the state of variables `i` and `j` after the `if` statement is executed: `i` is 3 and `j` is 0.

And then, so let us look at this code. I give you 1 minute to look at this code and think what this code will do, what will be the result of this. Look at this code carefully then we will analyze it.

You can see that here we have initialized a variable `i` to 4 and `j` has just been declared, but not initialized. So, here the picture is something like this, `i` has been initialized to 4 and `j` can have something, but I have not initialized at. Now, here if this statement, what will this statement do? First the condition part; that means, within the bracket this part will be evaluated and what will happen, what is this `i` is being assigned 3 so this will be 3 and the value that will be returned by this execution of this statement will be 3, then `j` will be 0 then `j` will be 0. Else `j` equal to 1 that will never come because this will be done successfully, this will never come. So, the output will be, output I have not given any `printf` here, but this `j`, value of `j` will be 0.

(Refer Slide Time: 25:41)

Generalization of expression evaluation in C

- Assignment (=) operation is also a part of expression.

```
int i=4, j;  
if(i=3)  
j=0;  
else  
j=1;
```

```
int i=4, j;  
if(i==3)  
j=0;  
else  
j=1;
```

So, what will be the value of j? Whatever be the value of i, j will be 0 right. Now, instead if I had written it correctly, correctly means if my intention was that if i is equal to 3, j should be 0 otherwise j should be 1 then I should have written it in this way. So, that if I is equal to 3 then j will be 0 otherwise j will be 1. So, this is a very important point that you have to keep in mind and gradually with practice they will be ok with this.

(Refer Slide Time: 26:34)

More about expressions

- Increment (++) and Decrement (--) Operations

```
i++;  
i--;
```

```
i = i + 1;  
i = i - 1;
```

So, here, more about expression. Now, here we introduce two new expressions typically when we write suppose there is a variable i and the variable i has got a value 5. Now, if I

want to implement the value 5, if I want to implement the value of i I will write i assigned i plus 1 or if I want to decrement the value of i then I can write i assign i minus 1, but C allows us or gives us one special implemented another special decrement operation.

So, this thing I could have written also as instead of this, this part I could have simply written i plus plus, i plus plus means i is assigned i plus 1 and this 1 will be i minus minus all right. So, this is known as the decrement operator, this is known as the increment operator. Sometimes it becomes handy to write it in this way, but if you are new in programming i personally would discourage you because you should not get ground in the special features of a language initially better you go by the standard features because these features have got some more complications which will come to later. But you may know that this is an increment operator which is equivalent to this and this is a decrement operator which is equivalent to this all right.

(Refer Slide Time: 28:55)

More about expressions

- Increment (++) and Decrement (--) Operations

Prefix operation

```
++i;
```

```
i++
```

```
++i
```

So, this is I can write it in two ways sorry i plus plus or plus plus i. The reason why I were I discourage you to get into the usage of this initially will be evident soon. If I write i plus plus it means this is called post operation this means pre operation of prefix operation. We will soon see what that means. So, I can have plus plus i or minus minus i that is a prefix operation and postfix operation is i plus plus and i minus minus.

(Refer Slide Time: 29:55)

More about expressions

- Increment (++) and Decrement (--) Operations

Prefix operation: `++i;` `--i;`

Postfix operation: `i++;` `i--;`

First increment / decrement and then used in evaluation

Handwritten notes:
 $z = 5;$
 $x = ++i + z; \rightarrow x = 11$
 $x = i++ + z; \rightarrow$

Now, plus plus i means first increment or decrement depending on whether it is plus or minus and then use it in the evaluation. For example, suppose I have got the value i here as 5 and I write x as signed plus plus i plus z or let me make it a constant, let us say z is 5. Then what will happen, this is what, this is pre increment i first increment i and then do the operation.

Now, this has got precedence of course, over this. So, first i will be, i will be incremented, so i will be 6 and z was 5. So, the result will be x will be 11, but if I had done written it x assigned i plus plus plus z here also I will first increment this 6 and then z will be added to that. But there are situations where we will do it after computation of the whole thing. I think I will need a separate session for you to explain this clearly. We will do that in a more careful manner.

(Refer Slide Time: 31:50)

More about expressions

- Increment (++) and Decrement (--) Operations

Prefix operation: ++j; --i;
First increment / decrement and then used in evaluation

Postfix operation: j++; i--;
increment / decrement operation after being used in evaluation

int t,m=1; t=++m; → m=2; t=2;

int t,m=1; t=m++; → m=2; t=1;

(Handwritten note: t = 1)

Let us look at this here int t is an integer and the value of m is 1. t assigned plus plus m; that means, what m was 1 and m is first incremented and assigned to t therefore, t will be 2 all right. Again if we do it like this here then yes, here it makes the difference very clear I am sorry t sorry here you see m was 1 and what I have done I have put the assignment of I have done a post operation, post fix operation; that means, first I will be doing the assignment, first I will be doing the assignment and then I will increment. So, what will happen? m was 1 here, so first this 1 will go over here. So, t will become 1 right and then before I complete this operation I increment m so m becomes 2 all right. So, we will see more examples of such things in the subsequent lectures.

Thank you.