




Functions and Conditional Statements

~ Yash Chaphekar, Anshul Shah and Vrishti Godhwani



**Let's start with a simple question and trace it
line by line to predict the output**



```
#include <stdio.h>

void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
    printf("Values of a and b inside the swap function: a = %d, b = %d\n", a, b);
    return;
}

int main()
{
    int a = 5;
    int b = 7;
    printf("Values of a and b before swapping: a = %d, b = %d\n", a, b);
    swap(a,b);
    printf("Values of a and b after swapping: a = %d, b = %d\n", a, b);
    return 0;
}
```

Predict
the
output



drumroll

The output

```
Values of a and b before swapping: a = 5, b = 7
```

```
Values of a and b inside the swap function: a = 7, b = 5
```

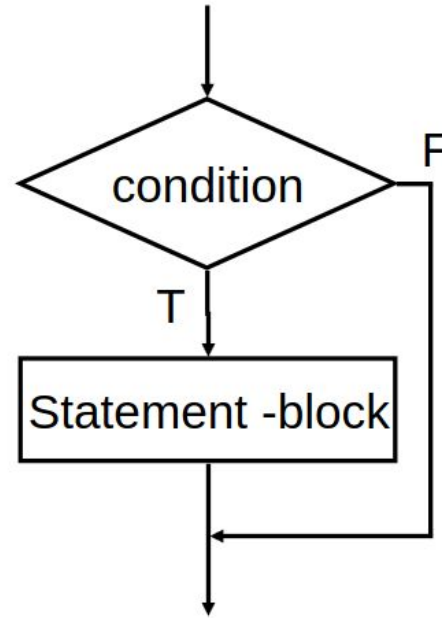
```
Values of a and b after swapping: a = 5, b = 7
```



Brief to conditional (if-else) statements

If Statement

```
if (condition)
{
    statement-block;
}
next-statement;
```

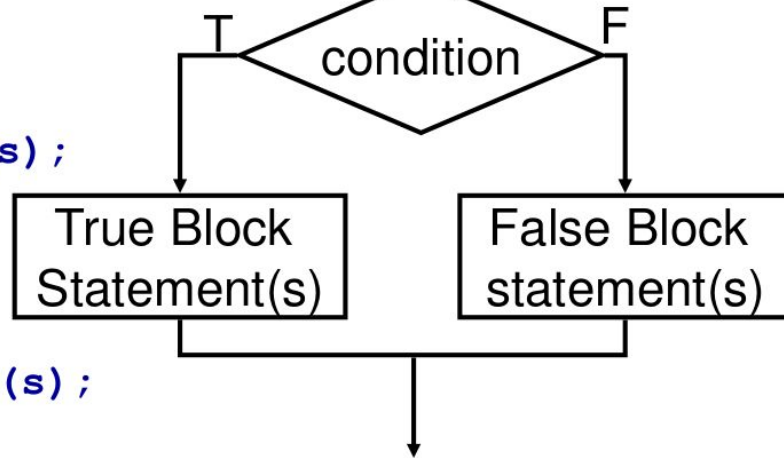


Condition is a C expression, which evaluates to **TRUE (non-zero) or FALSE (zero)**.

Statement-Block can be a single statement

If - else Statement

```
if (condition)
{
    true block statement(s);
}
Else
{
    false block statement(s);
}
```



Else allows choice between two mutually exclusive actions without re-testing condition.

The placement of the braces is a matter of personal choice.

EXAMPLES

```
if(x<=10)
{
    y = x*x + 5;
    z = (2*y)/3;
}
```

Here both statements will be executed if and only if $x \leq 10$


```
if(x<=10)  
y = x*x + 5;  
z = (2*y)/3;
```

Here only the 1st statement is conditional and depends on the if statement

The 2nd statement is executed always

```
if(x=2)  
y=5;
```

Common mistake of
using '=' instead of '==' as
the conditional
statement

The assignment
statement `x=2` is always
true, so the statement
`y=5` will always be
executed

Nesting of if – else statements with more than one variable

```
if(test-1)
{ if(test-2)
{
    statement(s)-1;
}
else
{
    statement(s)-2;
}
}
else
{
    statement(s)-3;
}
next_statement;
```

```
/*  
USE OF NESTED IF ELSE STATEMENTS  
  
Your parents have asked you to eat more fruits in college  
You go to the supermakret to buy fruits  
The vendor has only apples and oranges  
You have to pick the cheaper fruit  
And you have a budget constraint of 100Rs  
*/
```

```
if(p_apple<p_orange)  
{  
    if(p_apple<100)  
        printf("Buy apples");  
    else  
        printf("Overbudget");  
}  
else //(p_orange<p_apple)  
{  
    if(p_orange<100)  
        printf("Buy oranges");  
    else  
        printf("Overbudget");  
}
```

Understanding nested if else statements

EXAMPLE

```
//Find max of 3 numbers inputted by user using nested if else statements
int a,b,c;
scanf("%d %d %d",&a,&b,&c);

if(a>=b)
{
    if(a>=c)
    printf("%d is the maximum\n",a);
    else //(c>a)
    printf("%d is the maximum\n",c);
}
else //(b>a)
{
    if(b>=c)
    printf("%d is the maximum\n",b);
    else //(c>b)
    printf("%d is the maximum\n",c);
}
```

What if I want to use multiple if else statements instead of nested if else statements?

Use of logical operators in if-else statements

1. LOGICAL AND (&&) OPERATOR

```
if (condition-1 && condition-2)
{
    statement-1;
}
else
{
    statement-2;
}
```

X	Y	X && Y
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	FALSE	FALSE

If and only if both the conditions,
i.e.

Condition-1 and Condition-2
are true

Statement-1 will be executed
else statement-2 will be
executed

Use of logical operators in if-else statements

2. LOGICAL OR (||) OPERATOR

```
if (condition-1 || condition-2)
{
    statement-1;
}
else
{
    statement-2;
}
```

X	Y	X Y
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
TRUE	TRUE	TRUE
FALSE	FALSE	FALSE

If either one of the conditions, i.e. Condition-1 and Condition-2 is true

Statement-1 will be executed

When both conditions are false
statement-2 will be executed

Here's the implementation

```
// can be also written as
if(a>=b && a>=c)
{
    printf("%d is the maximum\n",a);
}
else if(b>=a && b>=c)
{
    printf("%d is the maximum\n",b);
}
else if (c>=a && c>=b)
{
    printf("%d is the maximum\n",c);
}
```

Using logical operators and multiple if-else statements to find maximum of 3 numbers

Another Method to ponder upon

```
// efficient method:  
int max = a;  
if(b>max)  
max = b;  
if(c>max)  
max = c;  
printf("Max =%d",max);  
  
// can be used in loops, arrays and saves time
```

Using a max variable and altering it every time another variable is greater than it

Time to code with us

BASICS TO CP 101

Question : Create a calculator to perform addition, multiplication and find the minimum of 3 numbers

Step 1 : Create add function

Step 2 : Create multiply function

Step 3: Create minimum function

Step 4:: Ask user which operation to perform

Step 5 : Take input of 3 numbers

Step 6 : Decide which function to use



Step 1 : Create add function

```
#include <stdio.h>

int add(int a, int b, int c)
{
    printf("Inside add function: %d+%d+%d\n", a, b, c);
    return a + b + c;
}
```



Step 2: Create Multiply Function

```
int multiply(int a, int b, int c)
{
    printf("Inside multiply function: %dx%dx%d\n", a, b, c);
    return a*b*c;
}
```



Step 3: Create Minimum Function

```
int minimum(int a, int b, int c)
{
    printf("Inside minimum function: min{%d,%d,%d}\n", a, b, c);
    int min;
    if(a<=b && a<=c)
        min = a;
    else if(b<=a && b<=c)
        min = b;
    else if(c<=a && c<=b)
        min = c;
    return min;
}
```



Step 4: User input for choice/operation

```
int main()
{
    int num1, num2, num3, result;

    int option;
    printf("Enter a number:\n'1' for addition\n'2' for multiplication\n'3' for finding their minimum\n");
    scanf("%d", &option);

    if (option != 1 && option != 2 && option != 3)
    {
        // Invalid number
        printf("Invalid number\n");
    }
}
```



Step 5: Take inputs

```
else
{
    // Input first number
    printf("Enter the first number:\n");
    scanf("%d", &num1);

    // Input second number
    printf("Enter the second number:\n");
    scanf("%d", &num2);

    // Input third number
    printf("Enter the third number:\n");
    scanf("%d", &num3);
}
```



Step 6: Decide which function to use (option 1 & 2)

```
if (option == 1) // nested if else
{
    // Call the add function to calculate the result
    result = add(num1, num2, num3);
    // Display the result
    printf("Result: %d\n", result);
}

else if(option == 2)
{
    // Call the multiply function to calculate the result
    result = multiply(num1, num2, num3);
    // Display the result
    printf("Result: %d\n", result);
}
```




Step 6: Decide which function to use (option 3)

```
    else
    {
        //Call the minimum function to calculate the result
        result = minimum(num1, num2, num3);
        //Display the result
        printf("Result: %d\n", result);
    }

    return 0;
```

```
}
```



Practice Question (30 mins)

Lab03_practice.c

A student has received his marks for all 3 subjects. Your task is to tell whether the student has passed or failed.

A student passes if he satisfies 2 conditions: his marks in all subjects must be more than 35 and the average of his scores must be more than 50.



You

when do we use debugging, what is the use of debugging while coding in c language



ChatGPT

Debugging is a crucial process in software development that involves identifying and fixing errors, bugs, and unexpected behavior in a program's source code. When you're coding in the C language or any other programming language, you may encounter issues such as logical errors, syntax errors, runtime errors, or unexpected behavior. Debugging helps you locate and correct these problems, ensuring that your program works as intended.

Here are some scenarios where debugging is commonly used in C programming:

1. **Syntax Errors:** These are errors that violate the rules of the programming language. Debugging helps you find and fix these errors before you can successfully compile and run your program.
2. **Logical Errors:** These errors occur when the code does not produce the expected output due to flawed logic. Debugging helps you analyze the code execution and identify the parts where the logic is incorrect.
3. **Runtime Errors:** These errors occur during the execution of the program. Debugging helps you identify the specific line or section of code where the error occurs, allowing you to diagnose and fix the problem.
4. **Unexpected Behavior:** Sometimes, a program may not behave as expected even though there are no apparent errors. Debugging helps you trace the program's execution, inspect variables, and understand why the program is not behaving as intended.
5. **Performance Issues:** Debugging can also be used to optimize code for better performance. By analyzing the program's execution flow and identifying bottlenecks, you can make improvements to enhance its efficiency.

In C programming, you can use debugging tools like gdb (GNU Debugger) to step through your code, set breakpoints, inspect variables, and analyze the program's execution. Additionally, print statements or logging can be used for simple debugging by displaying the values of variables at different points in the code.

The goal of debugging is to create reliable and efficient software by identifying and eliminating issues that may lead to incorrect behavior or program failures. It's an integral part of the software development process and helps ensure the quality and correctness of your code.

DEBUGGING

→ (Basically when you face errors and want to create an efficient solution)

```
print("Hello World");
```

```
int n;  
scanf("%d",n);
```

```
if(7>5)  
printf("True")
```

← Some basic errors that you may have encountered

How to solve them?

One should code block by block or function by function and keep on compiling repeatedly to reduce errors

After compilation, the terminal may show some errors like these

```
ifelse.c: In function 'main':  
ifelse.c:82:4: warning: implicit declaration of function 'print'; did you mean 'printf'? [-Wimplicit-function-declaration]  
82 |     print("Hello World");  
    |     ^~~~~  
    |     printf  
ifelse.c:85:12: warning: format '%d' expects argument of type 'int *', but argument 2 has type 'int' [-Wformat=]  
85 |     scanf("%d",n);  
    |           ^~  
    |           |  
    |         int  
    |         int *  
ifelse.c:88:18: error: expected ';' before '}' token  
88 |     printf("True")  
    |                 ^  
    |                 ;  
89 |  
90 | }
```

As the terminal suggests, we need to use **printf**, use **&n** and put a **;** after the last print statement

```
printf("Hello World");
```

```
int n;
```

```
scanf("%d",&n);
```

```
if(7>5)
```

```
printf("True");
```

← **Corrected Code**

When does looking at the terminal not work/help?

When it's a very long code with multiple errors or when the errors cannot be detected with the terminal

What to do then?

SIGH DEBUGGING

How to Debug?

Methods :

1. Using printf statements
2. Using the debugger

<https://www.youtube.com/watch?v=7qZBwhSlfOo>

(Please watch it later)