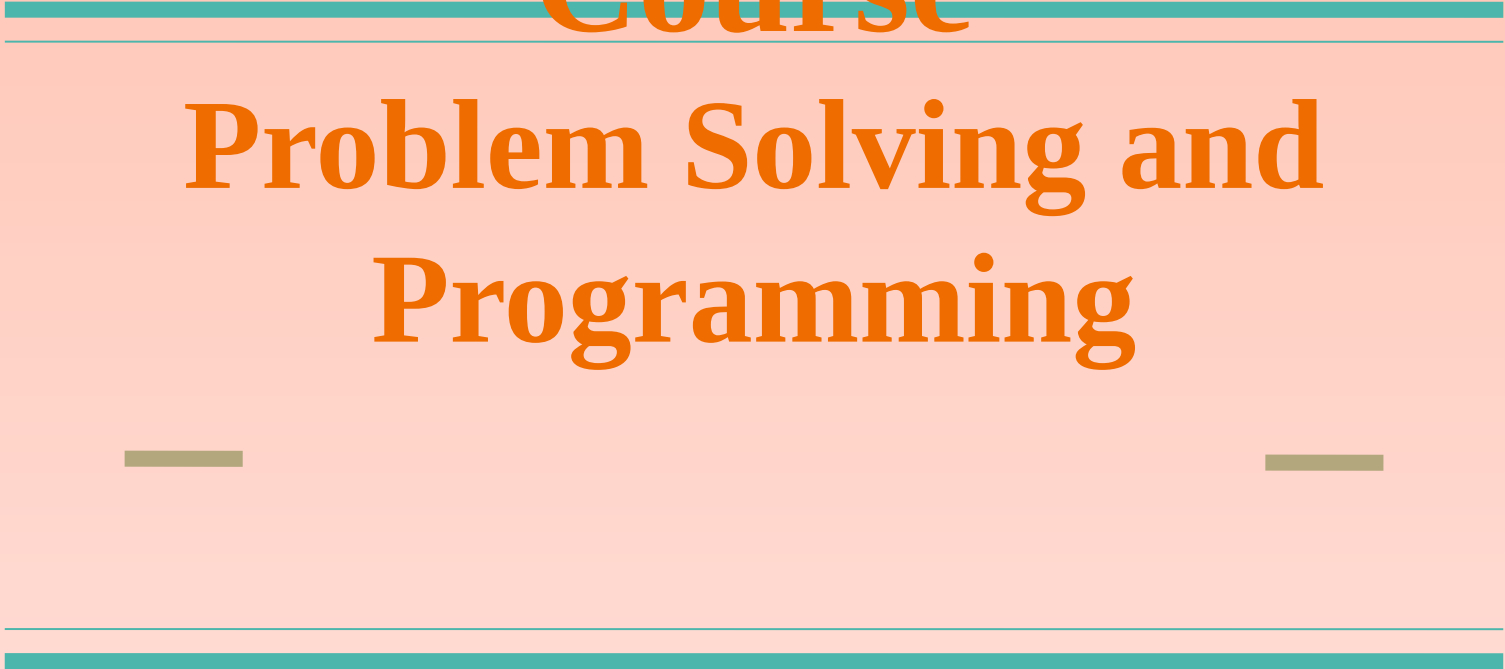


Course

Problem Solving and Programming

The slide features a light orange background. The word 'Course' is at the top in orange, followed by two horizontal lines (one teal, one light blue). Below this, 'Problem Solving and Programming' is written in orange across two lines. At the bottom, there are two small olive-green bars, followed by two more horizontal lines (one light blue, one teal).

Section 1:

- 1. Computing Fundamentals**
- 2. Overview of C**
- 3. C variables, Constants and Data Types**
- 4. Operators and Expressions in C**
- 5. C Control Statements**

Section 2:

- 1. Arrays**
- 2. Strings**
- 3. C Functions**
- 4. Pointers in C**

C Control Statements

Course Teaching Scheme:

Course Code:ES1034

Credits:5

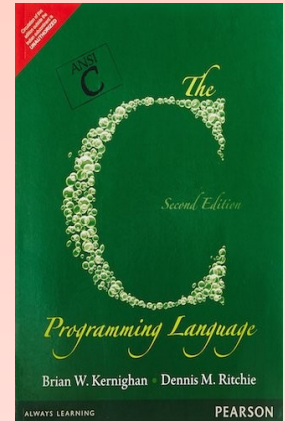
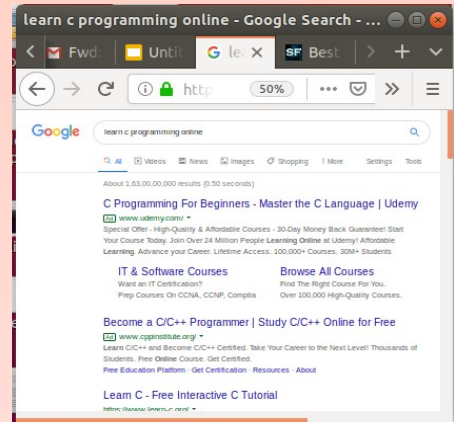
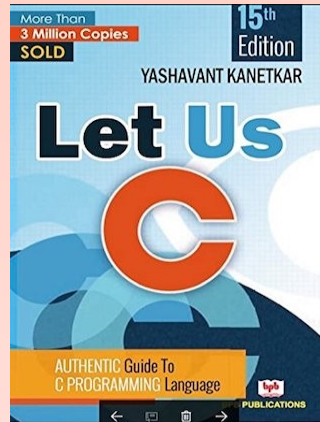
Teaching Scheme:Theory: 3 Hours/Week

Lab:2 Hours/Week

Tutorial:1 Hour / Week

For the first time learn C Programming from,

1. "Let Us C" by Yashavant Kanetkar
2. "The C Programming Language" by Brian W Kernighan / Dennis Ritchie
3. Over the web



1. if statement
2. if-else statement
3. if-else-if statement
4. compound if statement
5. comparison operators in expression
6. goto statement
7. conditional operator rules
8. switch case statement
9. switch case rules
- 10.If Statement Expression

Solving

1. switch case invalid ways
2. C mistakes – If-else if-elseif.
3. Loop Statement:
 - a. for loop
 - b. for loop – types
 - c. while loop
 - d. do-while loop
4. break statement
5. continue statement.

C Control Statements

Flow control in C

- C provides two styles of flow control:
 - Branching
 - Looping
- Branching is deciding what actions to take
- looping is deciding how many times to take a certain action.

Branching: if statement

- It takes an **expression in parenthesis** and an **statement or block of statements**.
- if the **expression is true** then the **statement or block of statements gets executed** **otherwise** these statements are **skipped**.

Form of if statement

```
if (expression)
    statement;

or

if (expression)
{
    Block of statements;
}

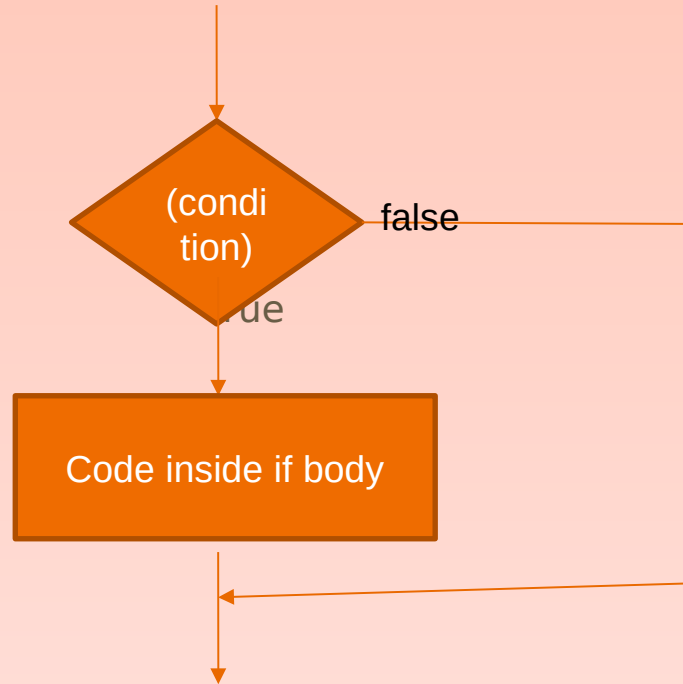
or

if (expression)
{
    Block of statements;
}
else
{
    Block of statements;
}

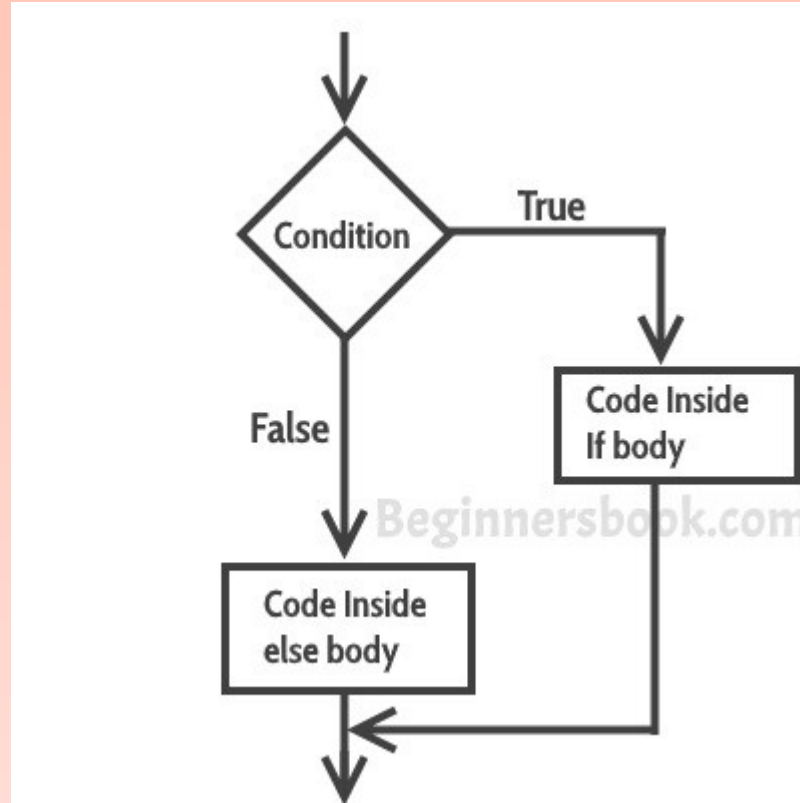
or

if (expression)
{
    Block of statements;
}
else if (expression)
{
    Block of statements;
}
else
{
    Block of statements;
}
```

Flowchart for simple if



Flowchart for if -else



How to Make use of Comparison Statement inside If Statement

1. While Solving such a Long expression , First Scan Expression from Left to Right .
2. Solve Expression bounded in Pair of (

Step 1 :

1. Always Solve **Bracket First**

```
c = !(a>=0) >= 0
```

Step 2 :

1. $a \geq 0$ is True
2. So Replace **"a>=0" by 1**

```
c = !1 >= 0
```

Step 3 :

1. **NOT Operator** has **Higher Priority** Than $\geq$ Operator .
2. $!1 = 0$
3. Replace **"!1" by "0"**

```
c = 0 >= 0
```

Step 4 :

1. Obviously $0 \geq 0$ is **True** Condition
2. **Replace Whole Expression by 1**

```
c = 1
```

Thus C = 1

C Programming if-else mistake

1. Assignment operator (=) and comparison operators (==)

```
if(var = 10)
{
    //Statement 1
    //Statement 2
    //Statement 3
    //Statement n
}
```

Any non-zero number inside if condition is considered as true condition so it will always executes to true

```
if(var = 0)
{
    //Statement 1
    //Statement 2
    //Statement 3
    //Statement n
}
```

statement executes to be false as we are assigning 0 to variable.

Best practice of using if statement is -

```
if(var == 10)
{
    //Statement 1
    //Statement 2
    //Statement 3
    //Statement n
}
```

C Programming if-else mistake

2.Use of Semicolon

```
if(var == 10);  
{  
    //Statement 1  
    //Statement 2  
    //Statement 3  
    //Statement n  
}
```

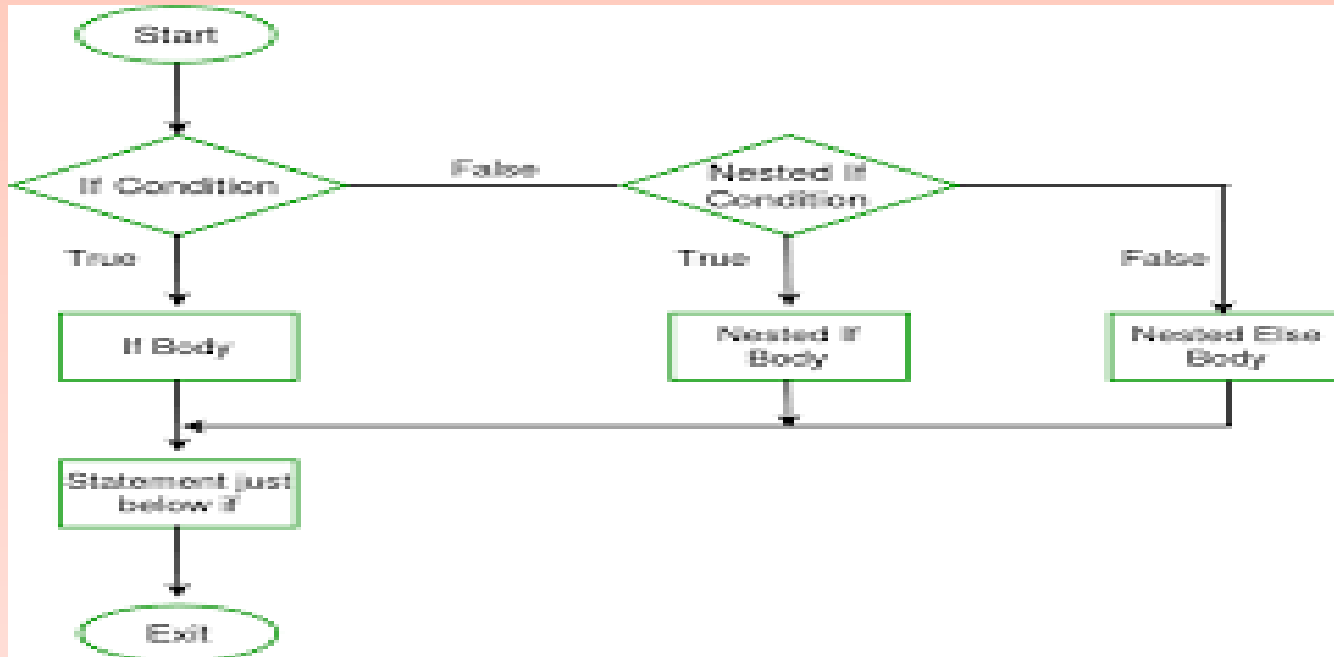
3.Else without using if

```
if  
Statement1 ;  
Statement2 ;  
..  
    else  
    {  
        Statement 3;  
        Statement 4;  
        .  
        .  
    }  
}
```

4.Opening and Closing Braces

```
Statement1 ;  
    if(condition)  
Statement1 ;  
Statement2 ;  
  
else  
{  
Statement 3;  
Statement 4;  
}
```

Nested if -else flowchart



Example 1: if...

```
// Program to display a number if it is negative
#include <stdio.h>
int main() {
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);
    // true if number is less than 0
    if (number < 0) {
        printf("You entered %d.\n", number);
    }
    printf("The if statement is easy.");
    return 0;
}
```

Output 1

```
Enter an integer: -2
You entered -2.
The if statement is easy.
```

Output 2

```
Enter an integer: 5
The if statement is easy.
```

Example 2: if...else statement

// Check whether an integer is odd or even

```
#include <stdio.h>
```

```
int main() {
```

```
    int number;
```

```
    printf("Enter an integer: ");
```

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

```
    // True if the remainder is 0
```

```
    if (number%2 == 0) {
```

```
        printf("%d is an even integer.",number);
```

```
    }
```

```
    else {
```

```
        printf("%d is an odd integer.",number);
```

```
    }
```

```
    return 0;
```

```
}
```

Output

```
Enter an integer: 7
7 is an odd integer.
```

Example 3: if...else if Ladder

```
// Program to relate two integers using =, > or < symbol
#include <stdio.h>
int main() {
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);
    //checks if the two integers are equal.
    if(number1 == number2) {
        printf("Result: %d = %d",number1,number2);
    }
    //checks if number1 is greater than number2.
    else if (number1 > number2) {
        printf("Result: %d > %d", number1, number2);
    }
    //checks if both test expressions are false
    else {
        printf("Result: %d < %d",number1, number2);    }
    return 0;}
```

Output

```
Enter two integers: 12
23
Result: 12 < 23
```

Example 4: Nested if...else

```
#include <stdio.h>
int main() {
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    if (number1 >= number2) {
        if (number1 == number2) {
            printf("Result: %d = %d", number1, number2);
        }
        else {
            printf("Result: %d > %d", number1, number2);
        }
    }
    else {
        printf("Result: %d < %d", number1, number2);
    }

    return 0;
}
```

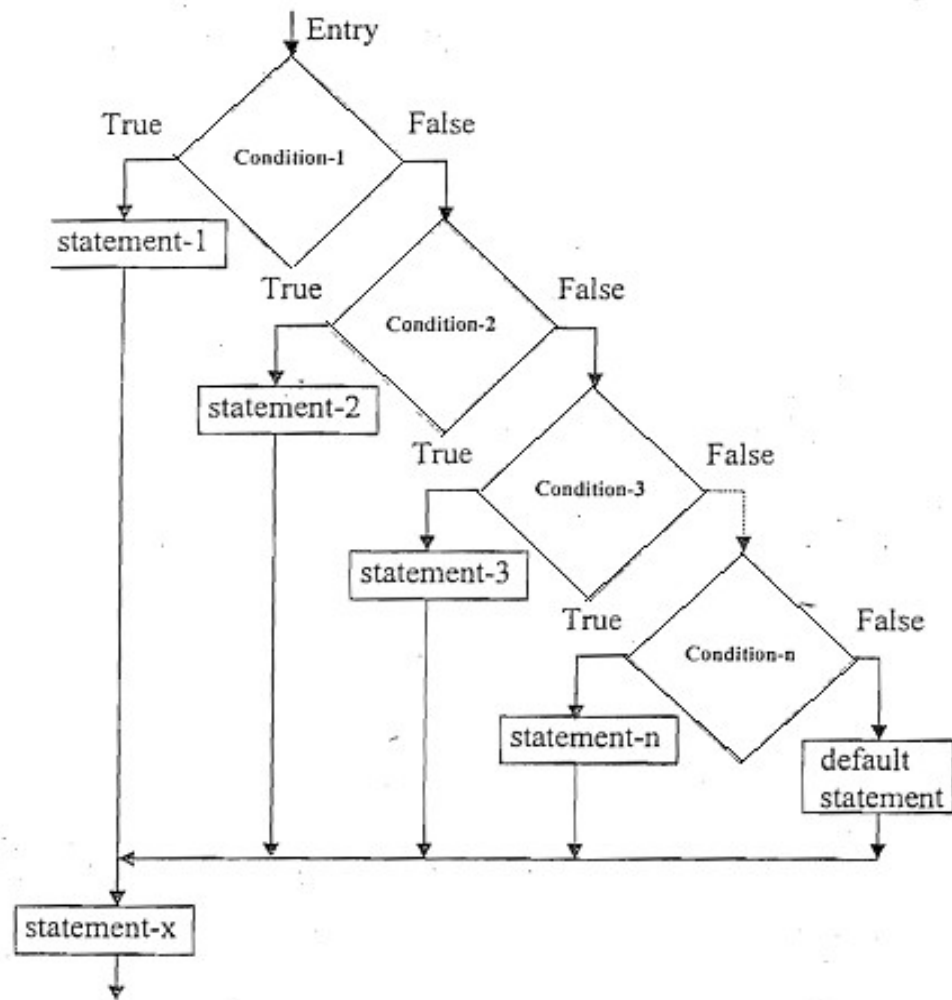
If-else-if ladder

- When we have multiple options available or we need to take multiple decisions based on available condition, we can use another form of if statement called ***else...if ladder***.
- In ***else...if ladder*** each else is associated with another if statement.
- Evaluation of condition starts from top to down.
- If condition becomes true then the associated block with if statement is executed and rest of conditions are skipped.
- If the condition becomes false then it will check for next condition in a sequential manner.
- It repeats until all conditions are checked or a true condition is found.
- If all condition available in ***else...if ladder*** evaluated to false then default else block will be executed

If-else-if ladder

```
If (condition1)
{
    block1
}
else if ( condition 2)
{
    block2
}
else if (condition 3)
{
    block 3
}
else
{
    default block;
}
```

If-else-if ladder



? : Operator

- The ? : operator is just **like an if ... else statement** except that because it is an operator you **can use it within expressions**.
- ? : is a **ternary operator** in that it takes **three values**, this is the only ternary operator C has.

? : Operator form

```
if condition is true ? then X return value : otherwise Y value;
```

Like,

```
if(expression1)
{
    expression2;
}
else
{
    expression3;
}
```

? : Operator form Example

```
#include<stdio.h>
int main()
{
    float num1, num2, max;
    printf("Enter two numbers: ");
    scanf("%f %f", &num1, &num2);
    max = (num1 > num2) ? num1 : num2;
    printf("Maximum of %.2f and %.2f = %.2f", num1, num2, max);
    return 0;
}
```

Output:- Enter two numbers: 12.5 10.5
Maximum of 12.50 and 10.50 = 12.50

Problem with if-else if ladder

- There is one potential problem with the if-else statement which is the complexity of the program increases whenever the number of alternative path increases. If you use multiple if-else constructs in the program, a program might become difficult to read and comprehend. Sometimes it may even confuse the developer who himself wrote the program.
- The solution to this problem is the switch statement

switch statement:

- The switch statement is much like a else if ladder statement
- switch statement can be slightly more efficient and easier to read.
- Switch case statements follow a selection-control mechanism and allow a value to change control of execution.
- They are a substitute for long if statements that compare a variable to several integral values.
- The switch statement is a multiway branch statement. It provides an easy way to dispatch

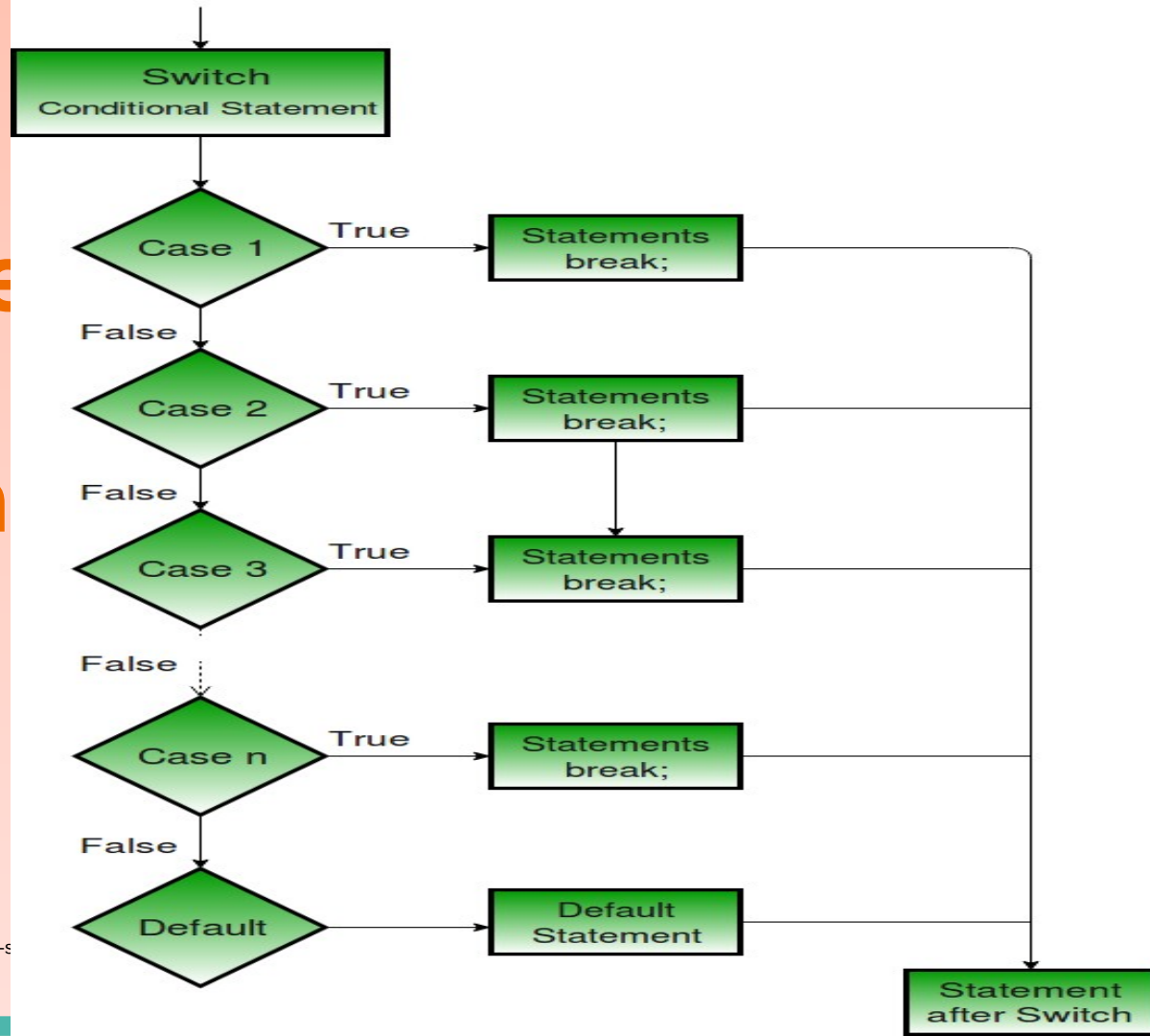
switch statement form

```
switch( expression )  
{  
    case constant-expression1:    statements1;  
    [case constant-expression2:    statements2;]  
    [case constant-expression3:    statements3;]  
    [default : statements4;]  
}
```

Switch statement syntax

```
switch (n)
{
    case 1: // code to be executed if n = 1;
        break;
    case 2: // code to be executed if n = 2;
        break;
    default: // code to be executed if n doesn't match any cases
}
```

switch statement flowchart



Switch case :

- **Some important keywords:**
- **1) Break:** This keyword is used to stop the execution inside a switch block. It helps to terminate the switch block and break out of it.
- **2) Default:** This keyword is used to specify the set of statements to execute if there is no case match.

Rules of Using Switch Case

1. Case Label **must be unique**
2. Case Labels must **ends with Colon**
3. Case labels must **have constants / constant expression**
4. Case label must be of **integral Type (Integer,Character)**
5. Case label should not be 'floating point number '
6. Switch case should have at most one default label
7. Default label is Optional
8. Default **can be placed anywhere in the switch**
9. Break Statement takes control out of the switch
10. **Two or more cases** may share **one break statement**
11. **Nesting (switch within switch)** is allowed.
12. Macro Identifier are allowed as Switch Case Label.

Switch case example

```
int id = 3 ;
switch(id)
{
    case 1:
        printf("C Programming Language");
        break;

    case 2:
        printf("C++ Programming Language");
        break;

    case 2:
        printf("Web Technology");
        break;

    default :
        printf("No student found");
        break;

}
```

Switch case example 2

```
// Program to create a simple calculator
#include <stdio.h>
int main() {
    char operator;    double n1, n2;
    printf("Enter an operator (+, -, *, /): ");    scanf("%c", &operator);
    printf("Enter two operands: ");    scanf("%lf %lf",&n1, &n2);
    switch(operator)
    { case '+':
        printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
        break;
      case '-':
        printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
        break;
      case '*':
        printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
        break;
      case '/':
        printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
        break;
      default:
        printf("Error! operator is not correct");
        // operator doesn't match any case constant +, -, *, /
    } return 0;}
```

Output

```
Enter an operator (+, -, *,,): -
Enter two operands: 32.5
12.4
32.5 - 12.4 = 20.1
```

Illegal way of using Switch Statement

1. **Floating point** values are **not allowed** as Case Label.
2. Case **Labels** Should be **Unique**.
3. **Variables** are **not allowed** in switch case.
4. **All statements** inside switch **must be wrapped** inside **any of** the case.

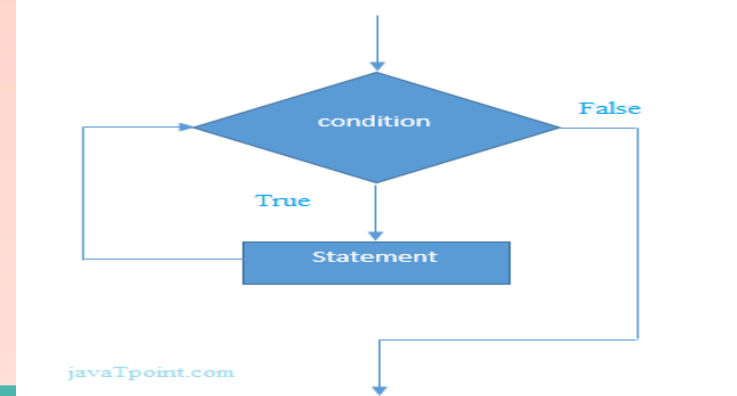
Looping

- Loops provide a **way to repeat commands and control** how many times user want they are repeated.
- A loop is used to repeat a block of code until the specified condition is met.
- C provides 3 looping statements
 - 1.while Loop
 - 2.do while Loop
 - 3.for Loop
- Types of Loops in C
- Depending upon the position of a control statement in a program, looping statement in C is classified into two types:
 1. Entry controlled loop
 2. Exit controlled loop
- In an entry control loop in C, a condition is checked before executing the body of a loop. It is also called as a pre-checking loop. Ex while loop
- In an exit controlled loop, a condition is checked after executing the body of a loop. It is also called as a post-checking loop. Ex. do while loop

while loop

- The most basic loop in C.
- It is like a repeating if statement.
- Like an If statement, if the test condition is true: the statements get executed.
- The difference is that after the statements have been executed, the test condition is checked again.
- If it is still true the statements get executed again.
- This cycle repeats until the test condition evaluates to false.

```
while ( expression )  
{  
    Single statement  
    or  
    Block of statements;  
}
```



Properties of while loop

- A conditional expression is used to check the condition. The statements defined inside the while loop will repeatedly execute until the given condition fails.
- In while loop, the condition expression is compulsory.
- Running a while loop without a body is possible.
- We can have more than one conditional expression in while loop.
- If the loop body contains only one statement, then the braces are optional.

while loop

```
// Print numbers from 1 to 5
#include <stdio.h>
int main()
{
    int i = 1;
    while (i <= 5)
    {
        printf("%d\n", i);
        ++i;
    }
    return 0;
}
```

Output

1
2
3
4
5

for loop

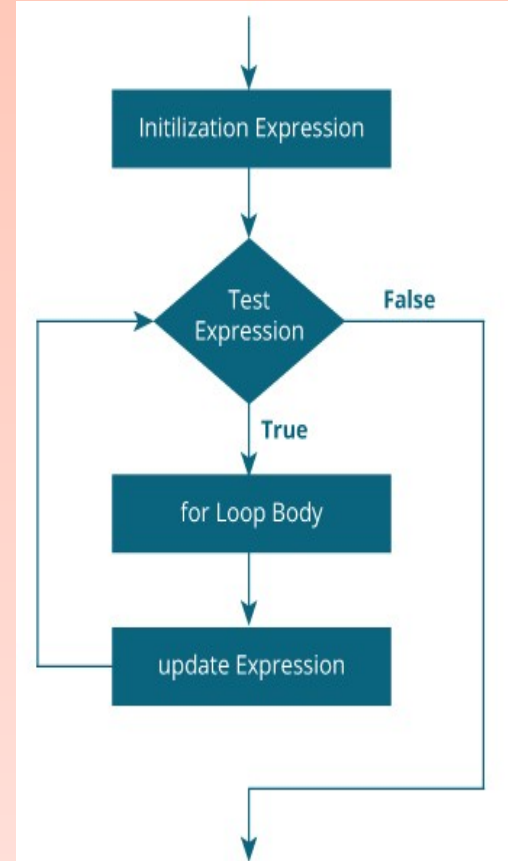
- **for** loop is similar to while, it's just written differently.
- When user want specific number of times to be action performed then for statement is frequently used.
- for statements are often used to process arrays and list .

for loop

- Syntax of for loop:

1. Expression 1 - Initialize variables.
2. Expression 2 - Conditional expression, as long as this condition is true, loop will keep executing.
3. Expression 3 - expression 3 is the modifier which may be simple increment of a variable.

```
for( expression1; expression2; expression3)
{
    Single statement
    or
    Block of statements;
}
```



How for loop executes

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

for loop

// Print numbers from 1 to 10

```
#include <stdio.h>
```

```
int main() {
```

```
    int i;
```

```
    for (i = 1; i < 11; ++i)
```

```
    {
```

```
        printf("%d ", i);
```

```
    }
```

```
    return 0;
```

```
}
```

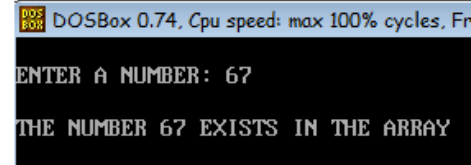
Output

1 2 3 4 5 6 7 8 9 10

for loop

```
1. #include <stdio.h>
2. #include <conio.h>
3. main()
4. {   int arr[]={12,23,78,98,67,56,45,19,65,9},key,i,flag=0;   clrscr();
5.     printf("\nENTER A NUMBER: ");
6.     scanf("%d",&key);
7.     for(i=0;i<10;i++)
8.     {
9.         if(key==arr[i])
10.            flag=1;
11.     }
12.     if(flag==1)
13.         printf("\nTHE NUMBER %d EXISTS IN THE ARRAY",key);
14.     else
15.         printf("\nTHE NUMBER %d DOES NOT EXIST IN THE ARRAY",key);
16.     getch();
17. }
```

Output



Nested for loop

- **Nested loop** means a loop statement inside another loop statement. That is why nested loops are also called as “**loop inside loop**”.
- **Syntax for Nested For loop:**

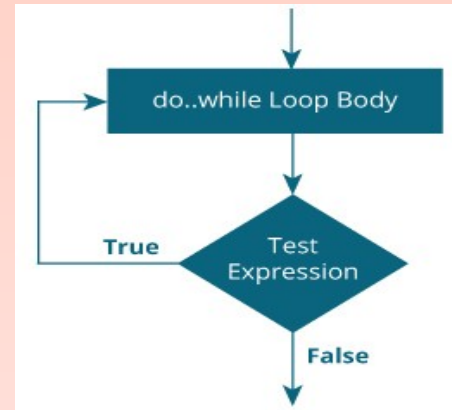
```
for ( initialization; condition; increment )  
{  
    for ( initialization; condition; increment )  
    {  
        // statement of inner loop  
    }  
    // statement of outer loop  
}
```

do...while loop

- just like a while loop except that the **test condition is checked at the end of the loop** rather than the start.
- This has the effect that the **content of the loop are always executed at least once.**

Basic syntax of do...while loop is as follows

```
do
{
    Single statement
    or
    Block of statements;
}while(expression);
```



do...while loop

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
int i=1;
```

```
do{
```

```
    printf("%d \n",i);
```

```
    i++;
```

```
}while(i<=10);
```

```
return 0;
```

```
}
```

Output

1

2

3

4

5

6

7

8

9

10

break and continue statements

- C provides two commands to control how we loop:
 - **break -- exit form loop or switch.**
 - **continue -- skip 1 iteration of loop.**
- The example of using break statement is seen in switch case.

Example of con

```
#include <stdio.h>

main()
{
    int i;
    int j = 10;

    for( i = 0; i <= j; i ++ )
    {
        if( i == 5 )
        {
            continue;
        }
        printf("Hello %d\n", i );
    }
}
```

This will produce following output:

```
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
Hello 6
Hello 7
Hello 8
Hello 9
Hello 10
```

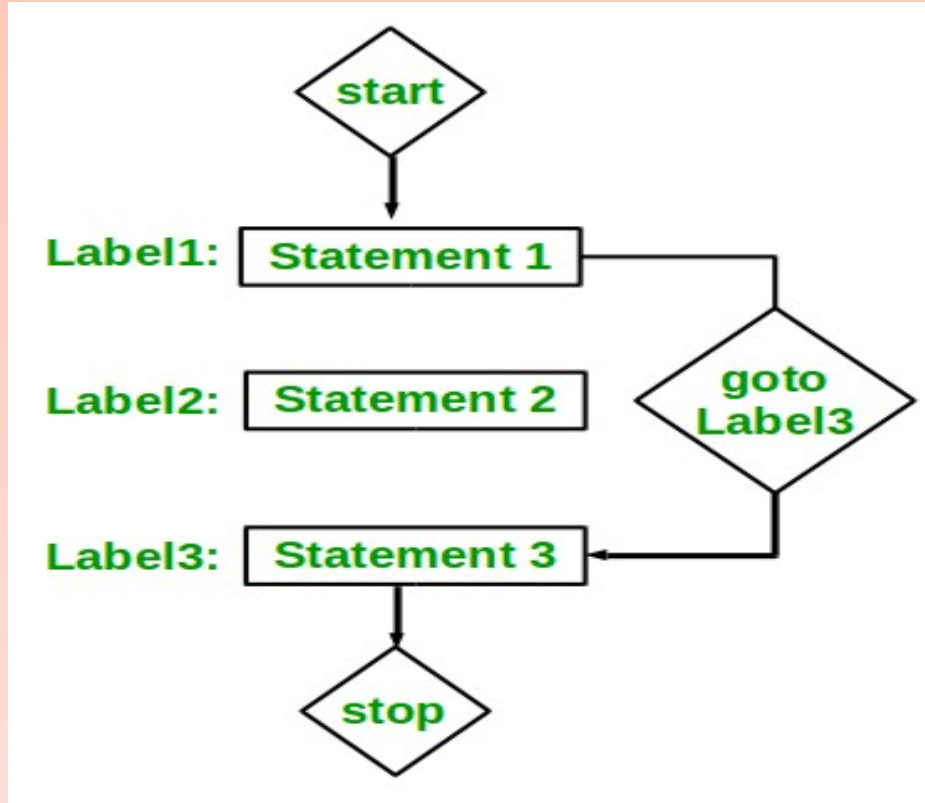
Goto statement

- a jump statement which is sometimes also referred to as unconditional jump statement.
- can be used to jump from anywhere to anywhere within a function

Syntax1		Syntax2

goto label;		label:
.		.
.		.
.		.
label:		goto label;

Goto statement



Goto statement example

```
// C program to check if a number is
// even or not using goto statement
#include <stdio.h>

// function to check even or not
void checkEvenOrNot(int num)
{
    if (num % 2 == 0)
        // jump to even
        goto even;
    else
        // jump to odd
        goto odd;

even:
    printf("%d is even", num);
    // return if even
    return;
odd:
    printf("%d is odd", num);
}

int main() {
    int num = 26;
    checkEvenOrNot(num);
    return 0;
}
```

Output:

26 is even

Thank You