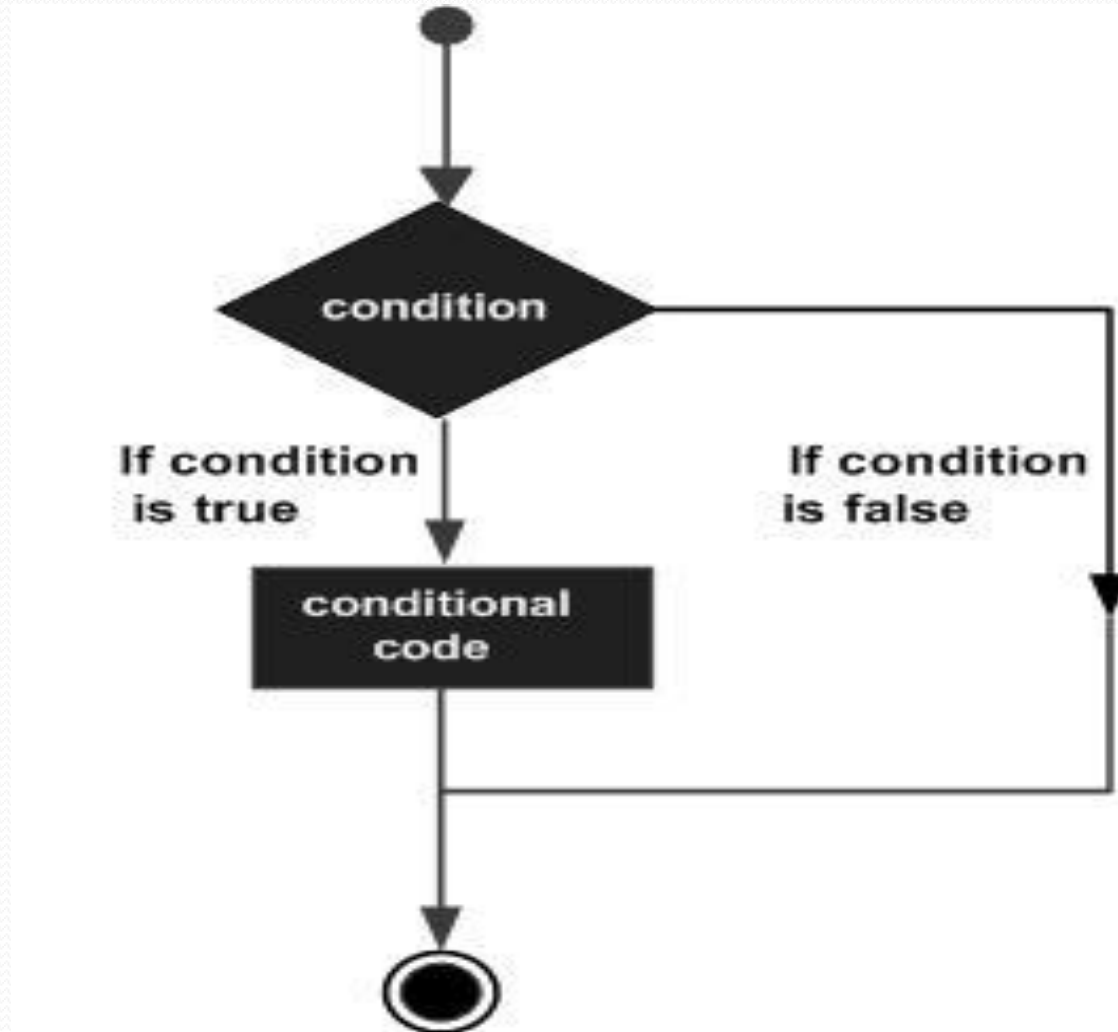


Unit : 2

Decision Making

C - Decision Making

- Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.



- C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

If Statement

- An **if** statement consists of a Boolean expression followed by one or more statements.


```
if(boolean_expression)
{
    /* statement(s) will execute if the boolean expression
    is true */
}
```

- If the Boolean expression evaluates to **true**, then the block of code inside the 'if' statement will be executed. If the Boolean expression evaluates to **false**, then the first set of code after the end of the 'if' statement (after the closing curly brace) will be executed.

If..else statement

- An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.

```
if(boolean_expression)
{
    /* statement(s) will execute if the boolean
    expression is true */
}
else
{
    /* statement(s) will execute if the boolean
    expression is false */
}
```


- 
- If the Boolean expression evaluates to **true**, then the **if block** will be executed, otherwise, the **else block** will be executed.

Nested if Statement

- It is always legal in C programming to **nest** if-else statements, which means you can use one if or else if statement inside another if or else if statement(s)

```
if( boolean_expression 1)
{
    /* Executes when the boolean expression 1 is true */
    if(boolean_expression 2)
    {
        /* Executes when the boolean expression 2
        is true */
    }
}
```

Ladder if Statement

- It is use to check set of conditions in sequence. If the first condition is false then it check the second one.

```
if( boolean_expression 1)
{
    /* Executes when the boolean expression 1 is true */
}
else if(boolean_expression 2)
{
    /* Executes when the boolean expression 2 is true */
}
else
{
    /* Executes when both are false*/
}
```

Switch Case

- A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.



```
Switch(expression)
```

```
{
```

```
    case constant-expression :
```

```
        statement(s);
```

```
        break; /* optional */
```

```
    case constant-expression :
```

```
        statement(s);
```

```
        break; /* optional */
```

```
    /* you can have any number of case statements */
```

```
    default : /* Optional */
```

```
        statement(s);
```

```
}
```


Rules for Switch case

- The **expression** used in a **switch** statement must have an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.

- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

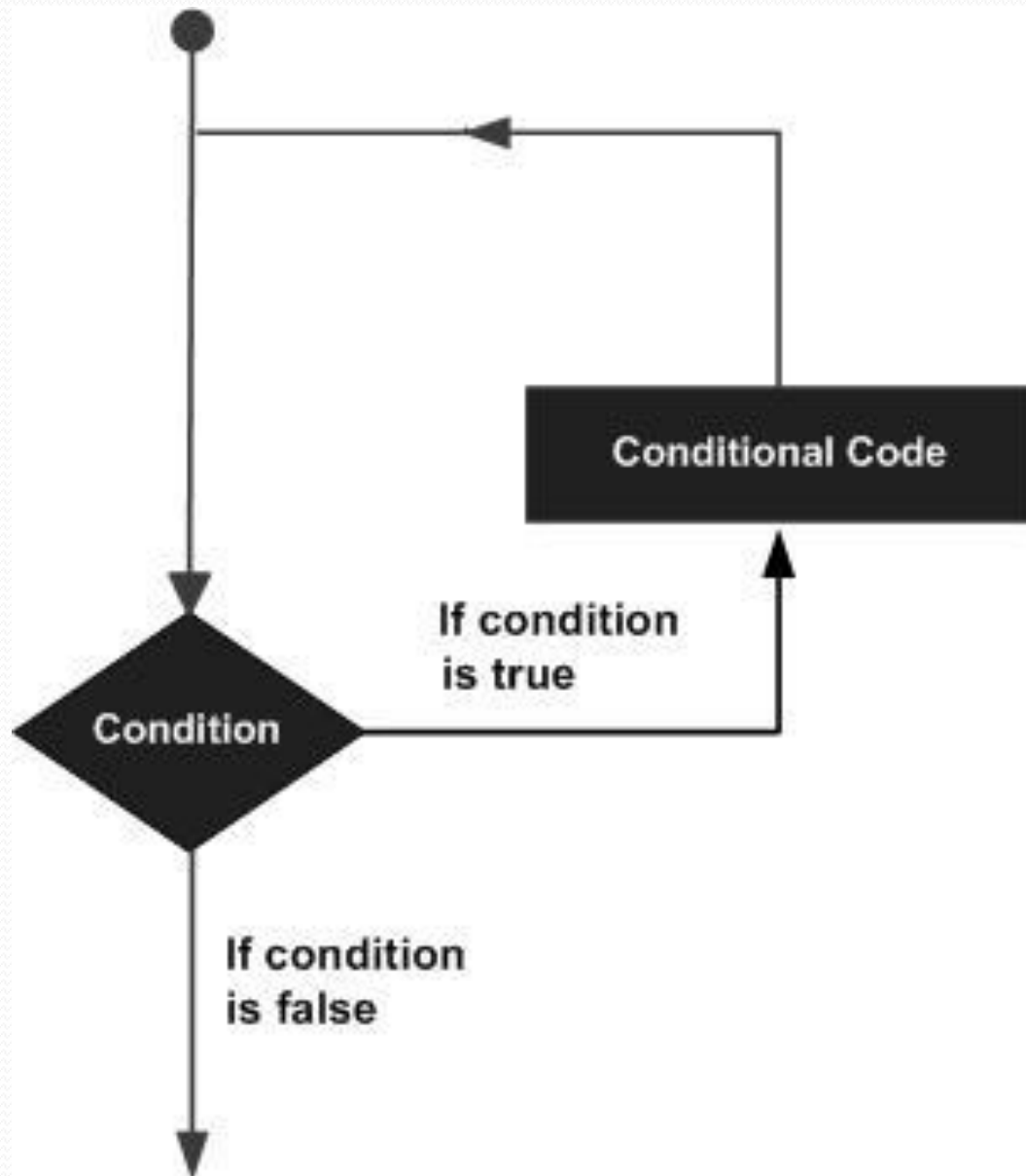
Loops

- You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.

- 
- A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages.

Types of Looping

- Entry Control loop
 - Check condition when control start of loop.
- Exit Control loop
 - Check condition when control end of loop.

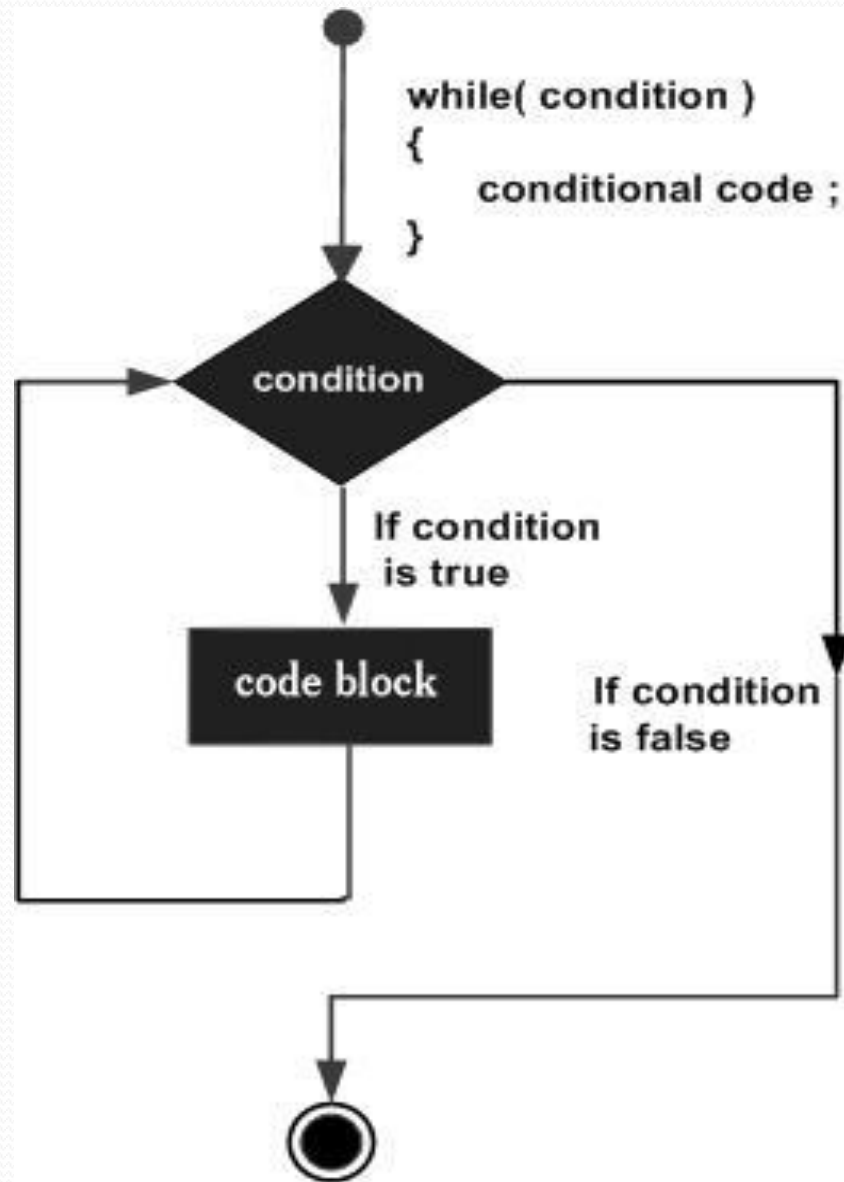


while loop

- A **while** loop in C programming repeatedly executes a target statement as long as a given condition is true.

```
while(condition)
{
    statement(s);
}
```

- **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true.



for loop

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

```
for ( init; condition; increment )  
{  
    statement(s);  
}
```

- Here is the flow of control in a 'for' loop


- 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.

do while loop

- Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming checks its condition at the bottom of the loop.
- A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

```
do
{
    statement(s);
} while( condition );
```

- 
- Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.
 - If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

Loop Control Statements

- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

break statement in C

- When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- It can be used to terminate a case in the **switch** statement.

continue statement in C

- The **continue** statement in C programming works somewhat like the **break** statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.
- For the **for** loop, **continue** statement causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops, **continue** statement causes the program control to pass to the conditional tests.

- A **goto** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

The Infinite Loop

- A loop becomes an infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.
- You can terminate an infinite loop by pressing Ctrl + C keys

Practice From this Unit

- Check entered value is five or not.
- Check entered number is positive or not
- Check entered number is odd or even
- Check entered year is leap year or not.
- Check entered number is between 0 to 100 or not.
- Simple mark sheet with pass or fail result.
- Light Bill
- Check Entered character is vowels or not.
- Income tax Calculator.

- Even and odd number between 1 to 100
- Sum of even and odd numbers between 1 to 100
- Reverse the number
- enter the no is or not Palindrome no
- Find the power of number
- Factorial number
- Armstrong number
- Fibonacci number