Unit 3

Control Statements and Loop In C

CONTROL STATEMENTS

Control statement determines the flow of control. They define how the control is transferred to other parts of the program. These statements enable us to specify the order in which the various instructions in the program are to be executed. Control statement defines the flow in which the execution of statements should take place to achieve the required result.

Types of control statements are:

- Decision-Making Statements
- Looping (Iteration) Statements
- Jump Statements

if STATEMENT

The if statement is a conditional control structure used to test a specific condition. If the condition evaluates to true, a block of code is executed; otherwise, it is skipped. It is the simplest form of decision-making in C and helps to perform selective execution based on logic.

```
if(condition)
{
  // Code to execute if condition is true
}
```

if-else STATEMENT

The if-else statement is an extension of the if statement. It allows the program to choose between two sets of code blocks: one executed if the condition is true, and another if the condition is false. It ensures that one of the two alternative actions is always performed based on the outcome of the condition.

```
if(condition){
  // Code if true
} else {
  // Code if false
}
```

Nested if STATEMENT

A nested if statement is a type of control structure where an if statement is placed inside another if or else block. This is used when multiple levels of decision-making are required. Each condition is checked in a hierarchical manner, allowing complex logical structures to be handled effectively.

```
if (condition1) {
  if (condition2) {
    // Code if both conditions are true
  }
}
```

else-if LADDER

The else-if ladder is a chain of conditions that are checked sequentially. It provides a way to test multiple expressions, where each else-if represents a new condition. The program evaluates each condition in order and executes the block of code corresponding to the first true condition, skipping the rest.

```
if(condition1)
{
  // Code block 1
} else if (condition2) {
  // Code block 2
} else {
  // Default block
}
```

switch STATEMENT

The switch statement is a multi-way branch control structure used to compare the value of a variable against multiple constant expressions. Each case represents a potential match. When a match is found, the corresponding block is executed. It is commonly used as an alternative to long if-else-if ladders when dealing with discrete values.

```
switch(expression)
{
  case value1:
    // Code block 1
    break;
  case value2:
    // Code block 2
    break;
  default:
    // Default code
}
```

CONDITIONAL OPERATOR (? :)

The conditional operator, also known as the ternary operator, provides a compact syntax for performing conditional assignments or evaluations. It takes three operands: a condition, an expression to execute if the condition is true, and another if it is false. It serves as a shorthand for simple if-else statements.

```
(condition) ? expression1 : expression2;
```

goto STATEMENT

The goto statement is a jump control structure that causes an unconditional jump to another part of the program marked by a label. Though rarely used, it can be helpful in breaking out of deeply nested loops or for certain error-handling scenarios. However, its use is generally discouraged as it can lead to unstructured and hard-to-maintain code.

```
goto label;
// some code
label:
// target code
```

LOOPING AND ITERATION

Looping and iteration are programming constructs that allow a set of statements to be executed repeatedly until a specified condition is met or fails. These constructs reduce code duplication, improve code efficiency, and handle repetitive tasks logically and effectively.

for LOOP

The for loop is a control structure that allows a block of code to be executed repeatedly based on a defined iteration count. It includes initialization, condition checking, and increment/decrement in a single line, making it ideal for situations where the number of iterations is known in advance.

```
for (initialization; condition; increment/decrement) {
  // Loop body
}
```

while LOOP

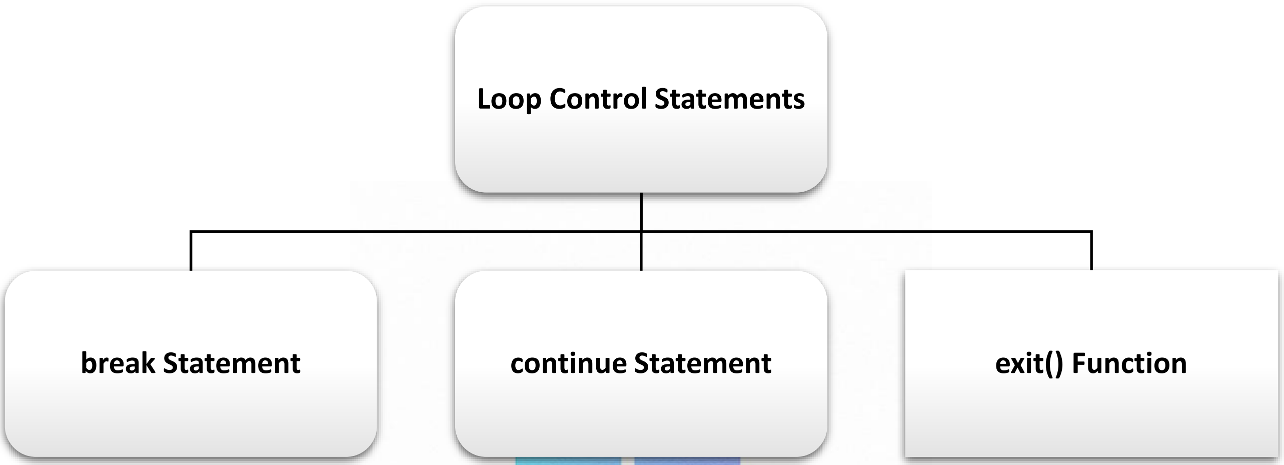
The while loop is a pre-test loop that continues to execute a block of code as long as the specified condition remains true. It is used when the number of iterations is not known in advance and depends on dynamic conditions evaluated before each iteration.

```
while(condition)
{
  // Loop body
}
```

do-while LOOP

The do-while loop is a post-test loop that ensures the block of code is executed at least once, regardless of the condition. The condition is checked after the first execution. This loop is useful when the task must be performed at least once before evaluating any condition.

```
do {
  // Loop body
} while (condition);
```



break STATEMENT

The break statement is used to exit from loops or switch statements prematurely, skipping any remaining iterations or cases. It immediately transfers the control to the statement following the loop or switch, providing a way to interrupt normal loop execution based on a condition.

Syntax: *break*;

continue STATEMENT

The continue statement is used within loops to skip the current iteration and jump directly to the next iteration. It does not terminate the loop but avoids the remaining code in the loop body for the current pass, proceeding with the next evaluation of the loop condition.

Syntax: *continue*;

exit() FUNCTION

The exit() function is used to terminate the program immediately. It is often used in cases of error handling or when a specific condition requires the program to stop running. It is a library function defined in <stdlib.h> and can return an exit status to the operating system.

Syntax: *exit*(0); // *From* < *stdlib.h* >