Unit-4
Control Structures/Statements
(Marks -16)

CTEVT Diploma in Computer Engineering
Subject : C Programming
Shree Sarwajanik Secondary Technical School
Prepared By: Er. Abinash Adhikari

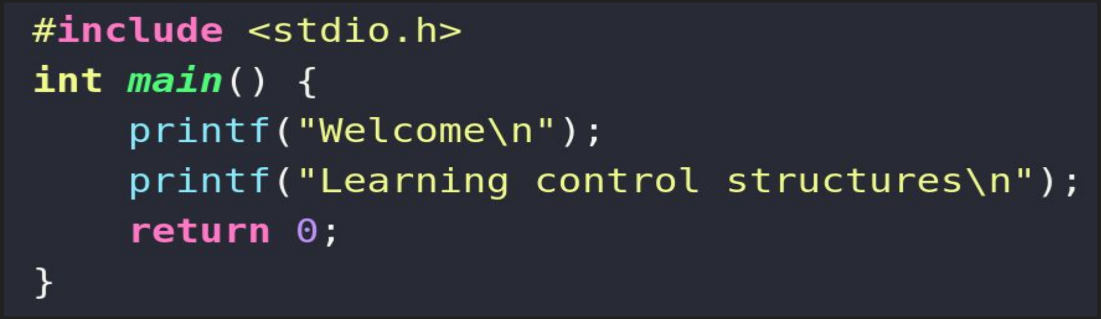
Control Structures Overview

- **Definition:**
  - Control structures are blocks of programming that determine the flow of control in a program.
  - They allow the execution of specific sections of code based on conditions or repetitions, enabling dynamic and flexible program behavior.
- **Categories:**
  - Sequential Statements
  - Decision/Selection/Conditional Statements
  - Loops
  - Jump Statements

Sequential Statement

- **Definition**:
  - A sequential statement is the simplest form of control structure where statements are executed one after the other in the order in which they appear in the program. It represents the default flow of execution.
- **Key Characteristics:**
  - No branching or repetition.
  - Executes step-by-step in a linear manner.
- **Example:**

```
int a = 5;
int b = 10;
int sum = a + b;
printf("Sum: %d", sum);
```



```
#include <stdio.h>
int main() {
    printf("Welcome\n");
    printf("Learning control structures\n");
    return 0;
}
```

Decision/Selection/Conditional Statements

- **Definition**:
  - These statements allow the program to make decisions based on conditions. They evaluate a condition and execute a block of code only if the condition is true.
- **Types**:
  - If Statement
  - If...Else Statement
  - If...Else If...Else Statement
  - Nested If...Else Statement
  - Switch Statement

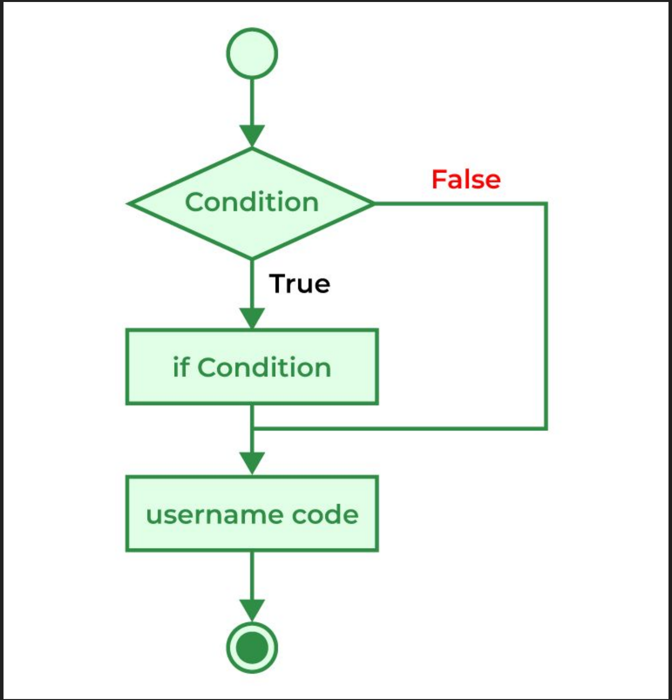
If Statement

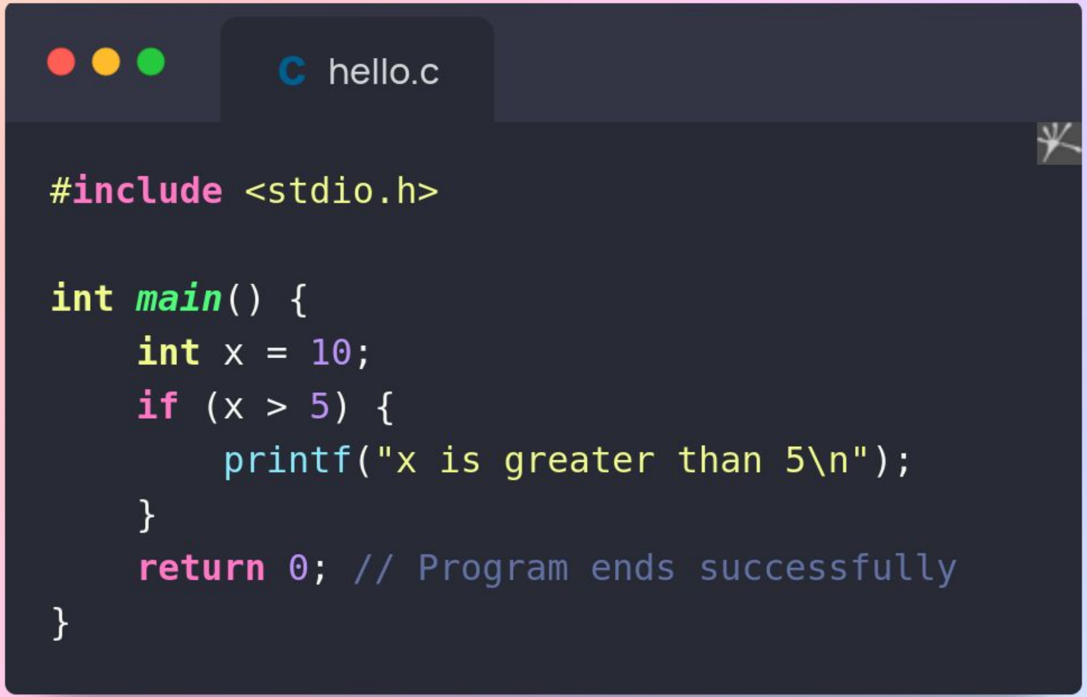
- **Definition:**
  - The simplest form of decision-making statement. It executes a block of code only if the specified condition evaluates to true.
- **Syntax:**

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

- **Example:**

```
int x = 10;
if (x > 5) {
   printf("x is greater than 5");
}
```





```c
#include <stdio.h>

int main() {
    int x = 10;
    if (x > 5) {
        printf("x is greater than 5\n");
    }
    return 0; // Program ends successfully
}
```

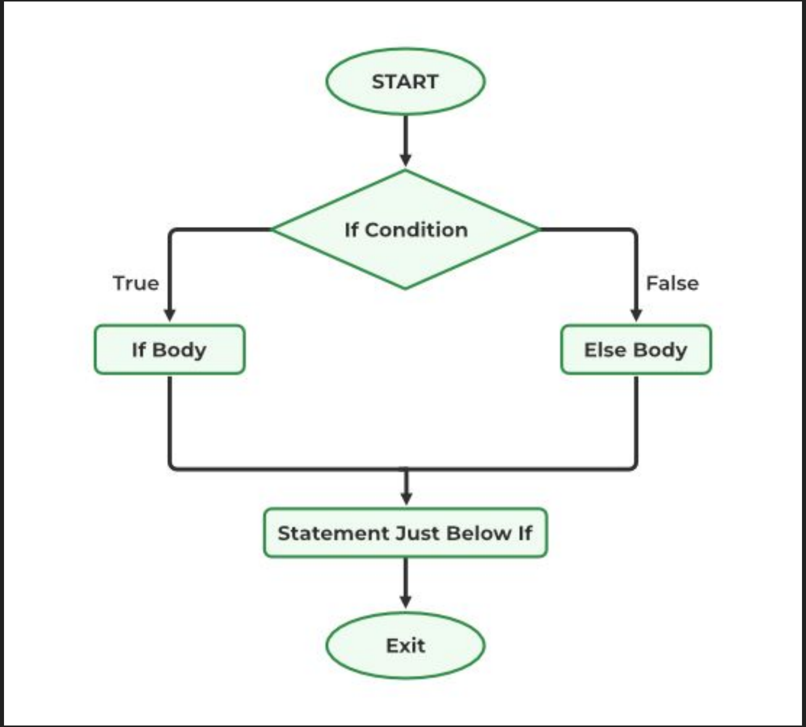
If...Else Statement

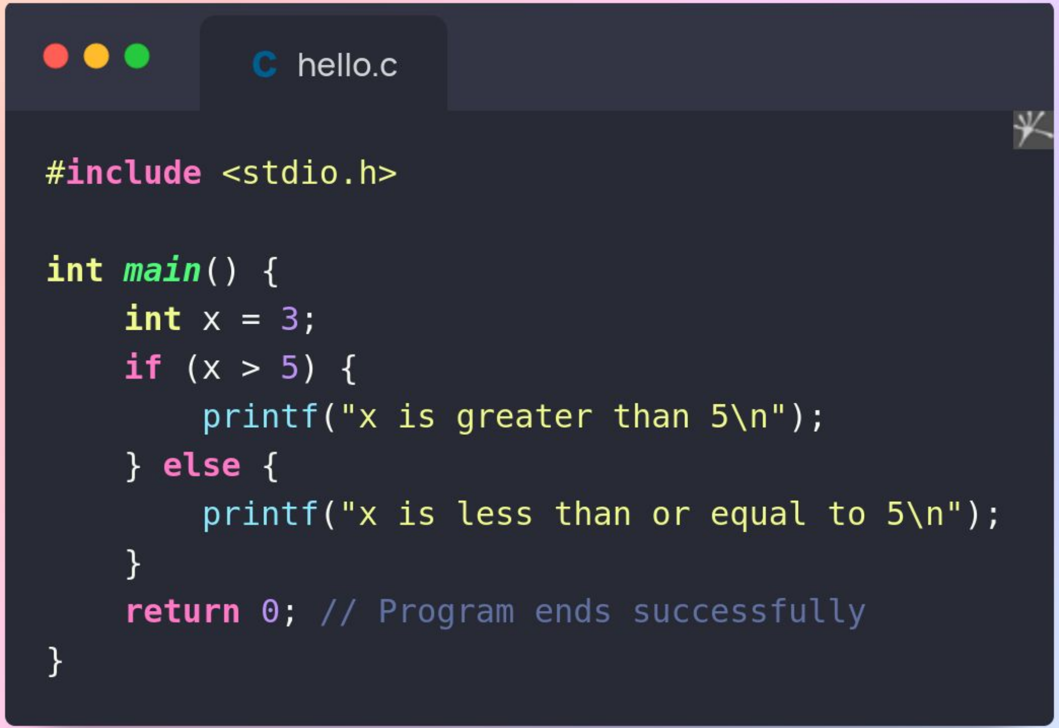
- **Definition:**
  - An extension of the if statement that specifies an alternate block of code to execute if the condition is false.
- **Syntax:**

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

- **Example:**

```
int x = 3;
if (x > 5) {
    printf("x is greater than 5");
} else {
    printf("x is less than or equal to 5");
}
```





```c
#include <stdio.h>

int main() {
    int x = 3;
    if (x > 5) {
        printf("x is greater than 5\n");
    } else {
        printf("x is less than or equal to 5\n");
    }
    return 0; // Program ends successfully
}
```

If...Else If...Else Statement

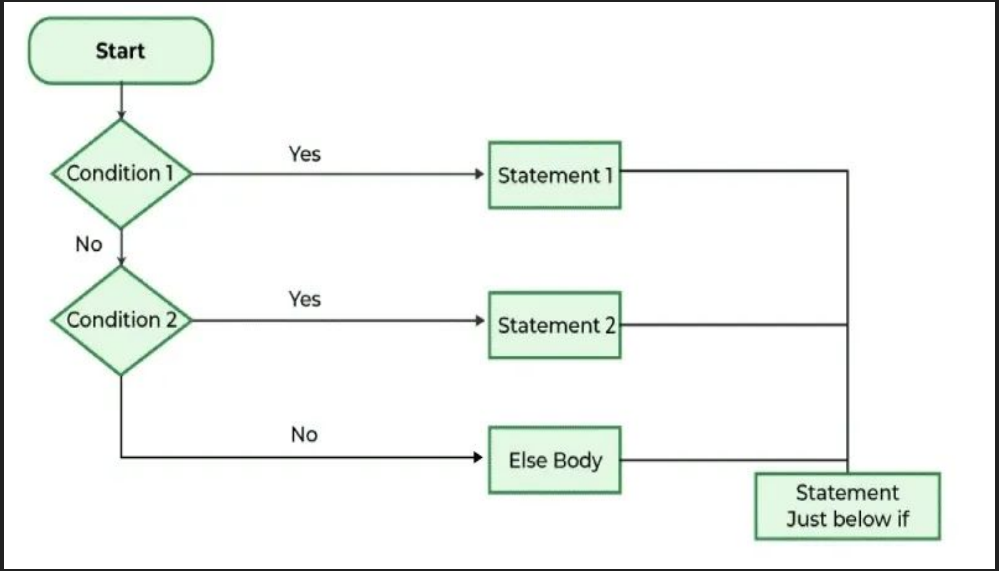
- **Definition:**
  - A decision-making structure that allows multiple conditions to be checked sequentially. If one condition is true, its corresponding block of code is executed, and the rest are ignored.

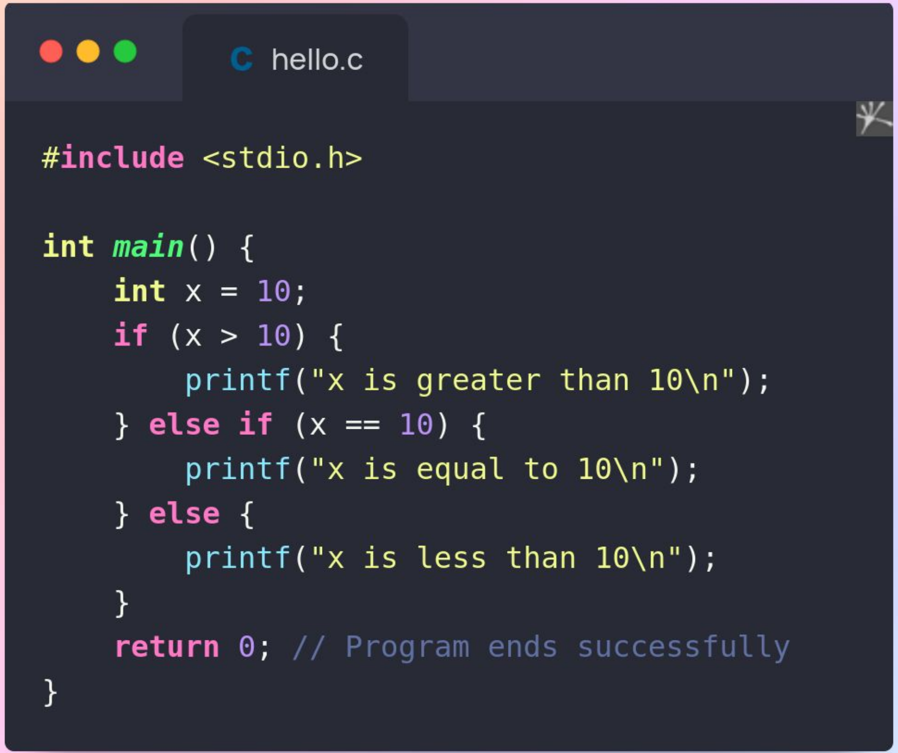
- **Syntax:**

```
if (condition1) {
   // Code for condition1
} else if (condition2) {
   // Code for condition2
} else {
   // Code if none of the conditions are true
}
```

- **Example:**

```
int x = 10;
if (x > 10) {
   printf("x is greater than 10");
} else if (x == 10) {
   printf("x is equal to 10");
} else {
   printf("x is less than 10");
}
```





```c
#include <stdio.h>

int main() {
    int x = 10;
    if (x > 10) {
        printf("x is greater than 10\n");
    } else if (x == 10) {
        printf("x is equal to 10\n");
    } else {
        printf("x is less than 10\n");
    }
    return 0; // Program ends successfully
}
```

Nested If...Else Statement

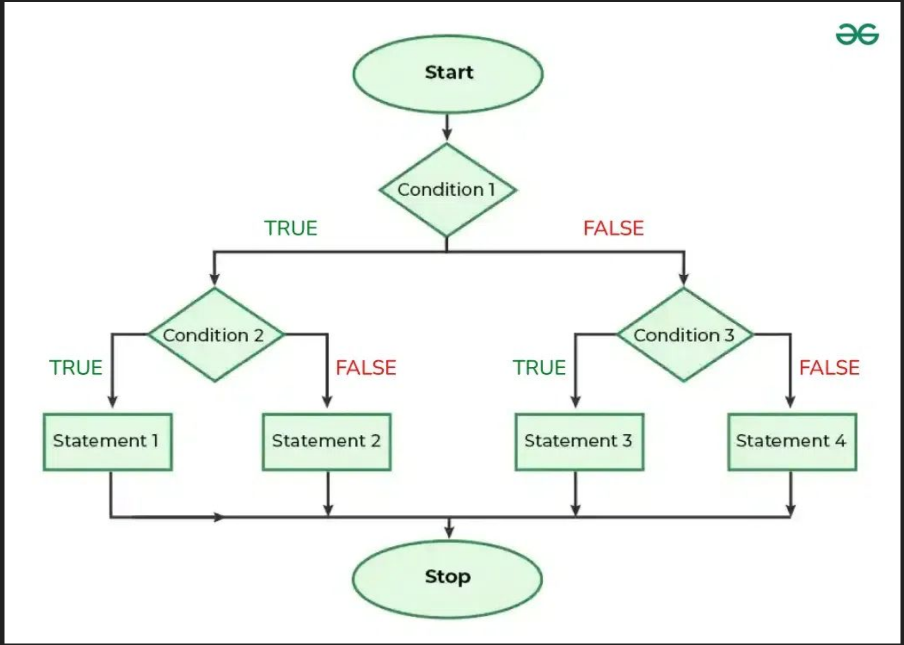
- **Definition:**
  - An if or if-else statement placed inside another if or if-else statement. This structure allows testing of multiple conditions.

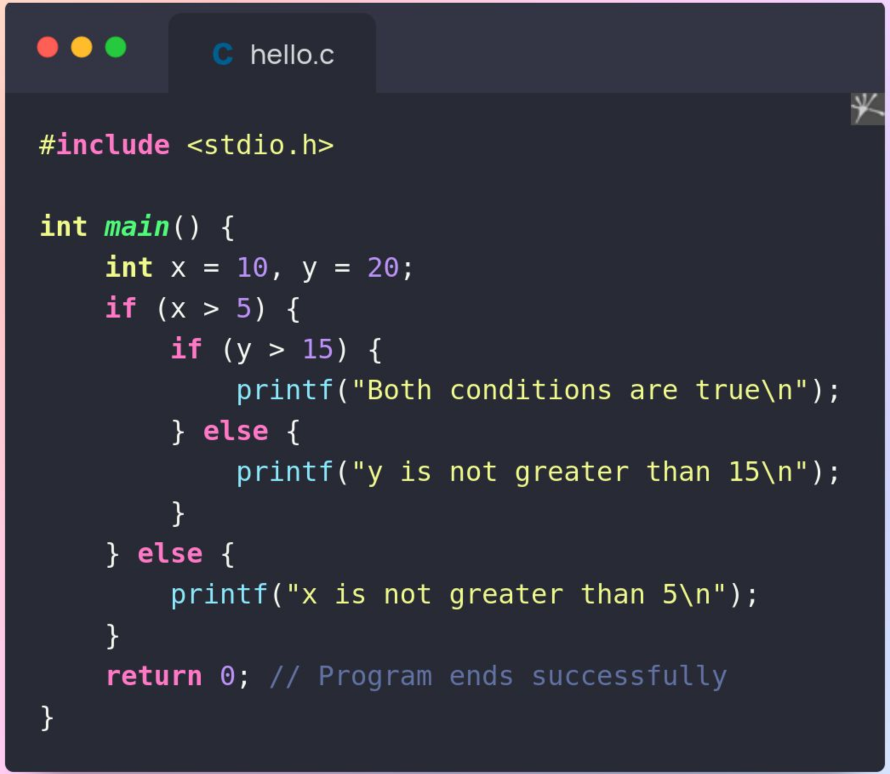
- **Syntax:**

```
if (condition1) {
    if (condition2) {
        // Code for condition2
    } else {
        // Code if condition2 is false
    }
} else {
    // Code if condition1 is false
}
```

- **Example:**

```
int x = 10, y = 20;
if (x > 5) {
    if (y > 15) {
        printf("Both conditions are true");
    }
}
```





```c
#include <stdio.h>

int main() {
    int x = 10, y = 20;
    if (x > 5) {
        if (y > 15) {
            printf("Both conditions are true\n");
        } else {
            printf("y is not greater than 15\n");
        }
    } else {
        printf("x is not greater than 5\n");
    }
    return 0; // Program ends successfully
}
```

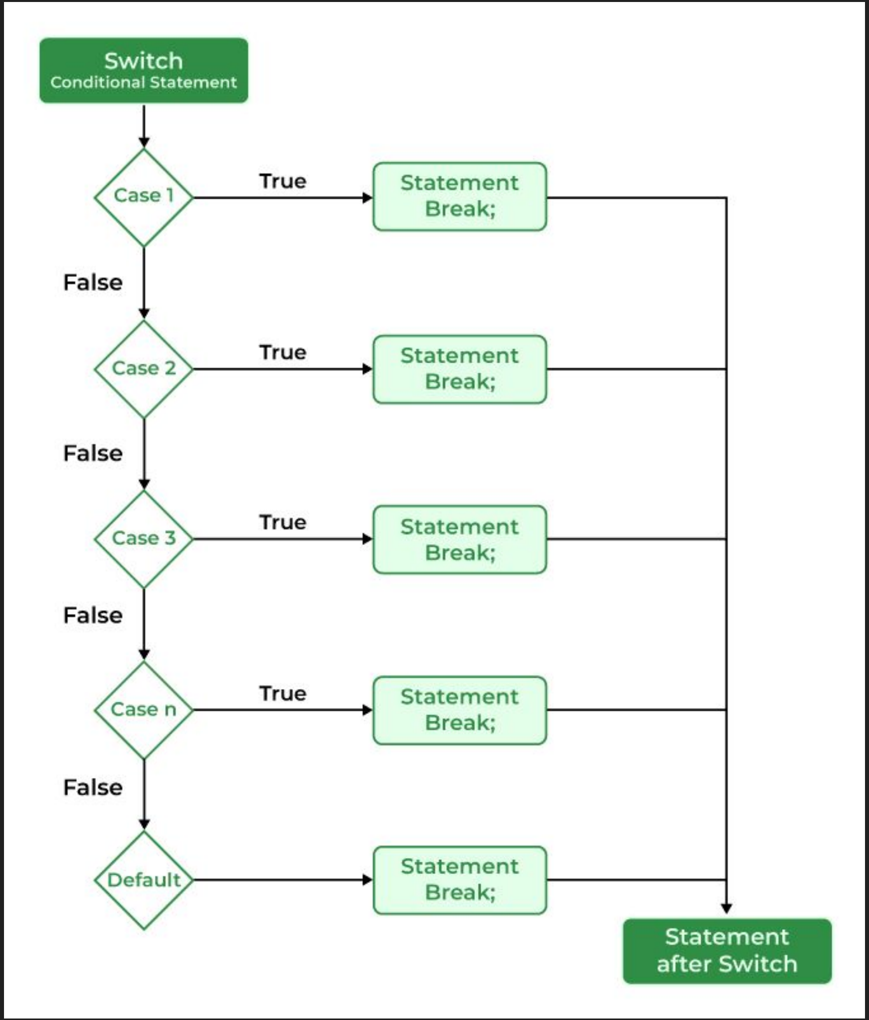
Switch Statement

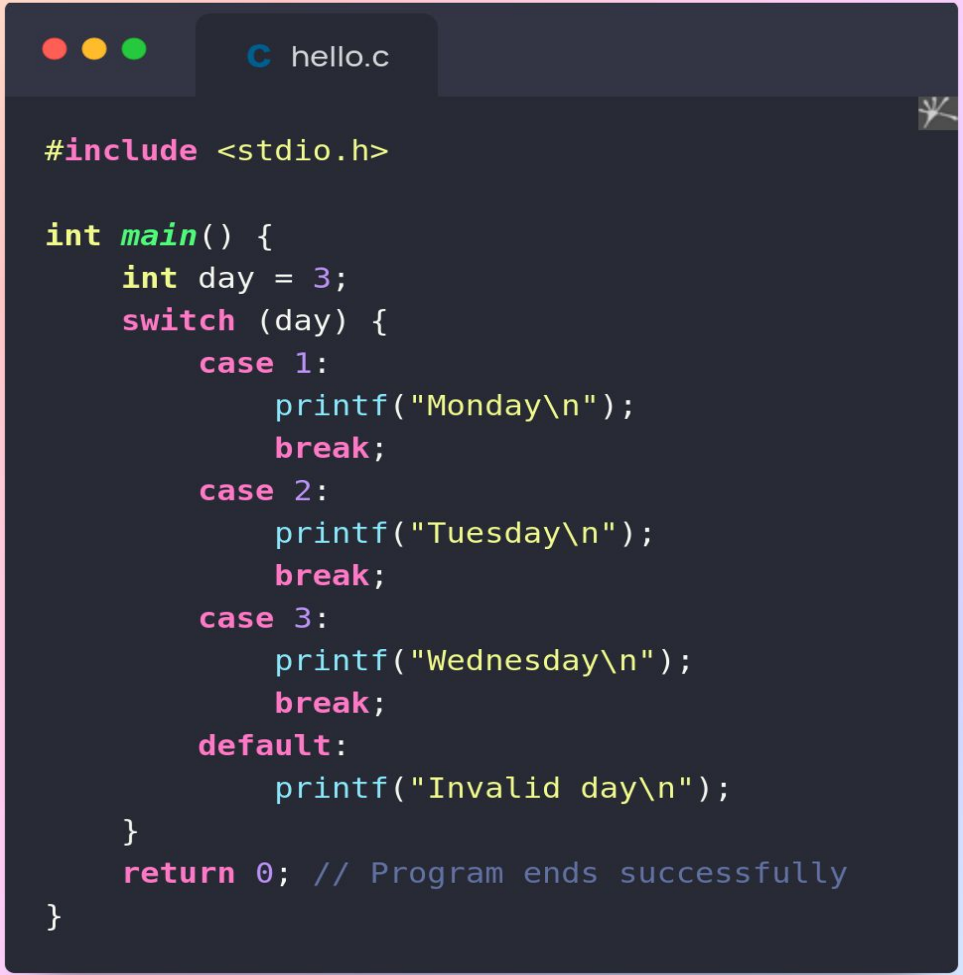
- **Definition:**
  - ➢ A control statement that allows a variable to be tested for equality against a list of values. Each value is called a case.
- **Syntax:**

```
switch (variable) {
    case value1:
        // Code for case value1
        break;
    case value2:
        // Code for case value2
        break;
    default:
        // Code if no cases match
}
```

❖ **Example:**

```
int day = 3;
switch (day) {
   case 1:
      printf("Monday");
      break;
   case 2:
      printf("Tuesday");
      break;
   case 3:
      printf("Wednesday");
      break;
   default:
      printf("Invalid day");
}
```



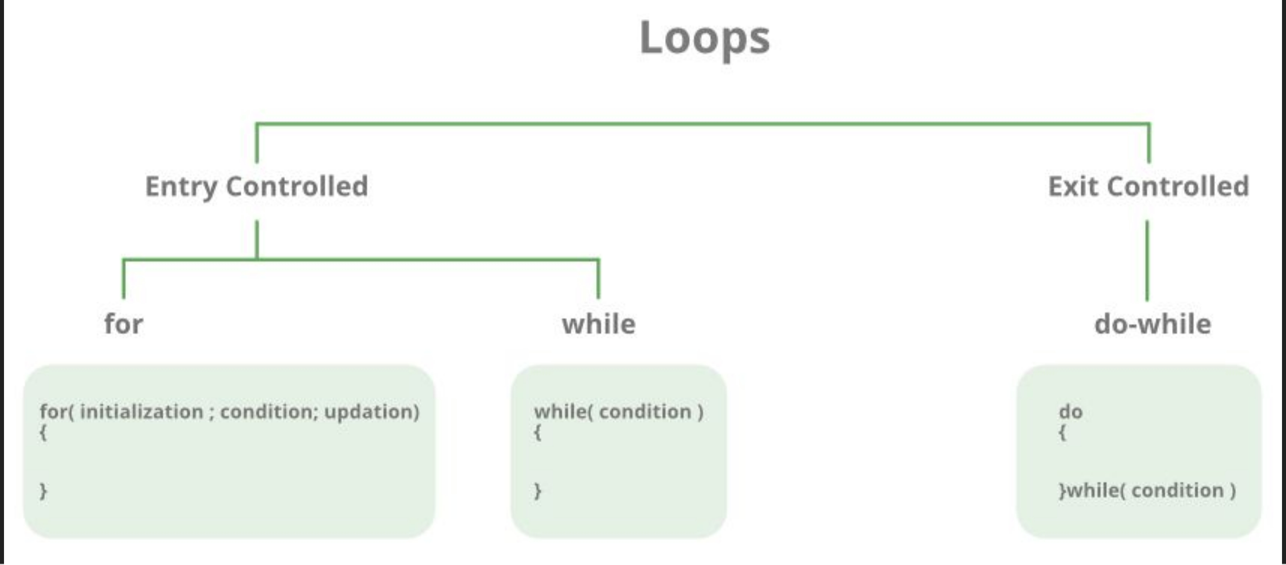

hello.c

```c
#include <stdio.h>

int main() {
    int day = 3;
    switch (day) {
        case 1:
            printf("Monday\n");
            break;
        case 2:
            printf("Tuesday\n");
            break;
        case 3:
            printf("Wednesday\n");
            break;
        default:
            printf("Invalid day\n");
    }
    return 0; // Program ends successfully
}
```

Loops

- Loops in programming are used to repeat a block of code until the specified condition is met. A loop statement allows programmers to execute a statement or group of statements multiple times without repetition of code.
- There are mainly two types of loops in C Programming:
  - **Entry Controlled loops:**
    - In Entry controlled loops the test condition is checked before entering the main body of the loop.
    - For Loop and While Loop is Entry-controlled loops.
  - **Exit Controlled loops:**
    - In Exit controlled loops the test condition is evaluated at the end of the loop body.
    - The loop body will execute at least once, irrespective of whether the condition is true or false. do-while Loop is Exit Controlled loop.



```
for( initialization ; condition; updation)
{

}
```

```
while( condition )
{

}
```

```
do
{

}while( condition )
```

Loop Type	Description
for loop	first Initializes, then condition check, then executes the body and at last, the update is done.
while loop	first Initializes, then condition checks, and then executes the body, and updating can be inside the body.
do-while loop	do-while first executes the body and then the condition check is done.

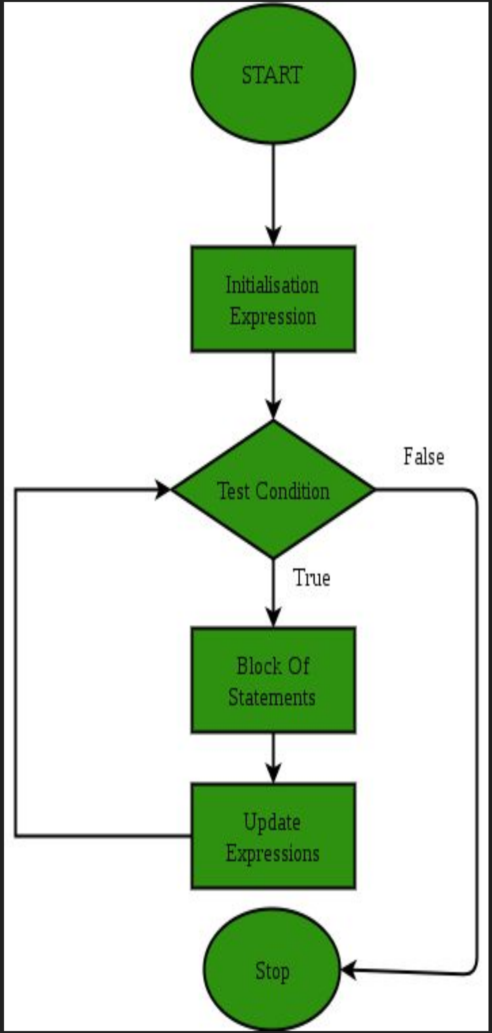
For Loop

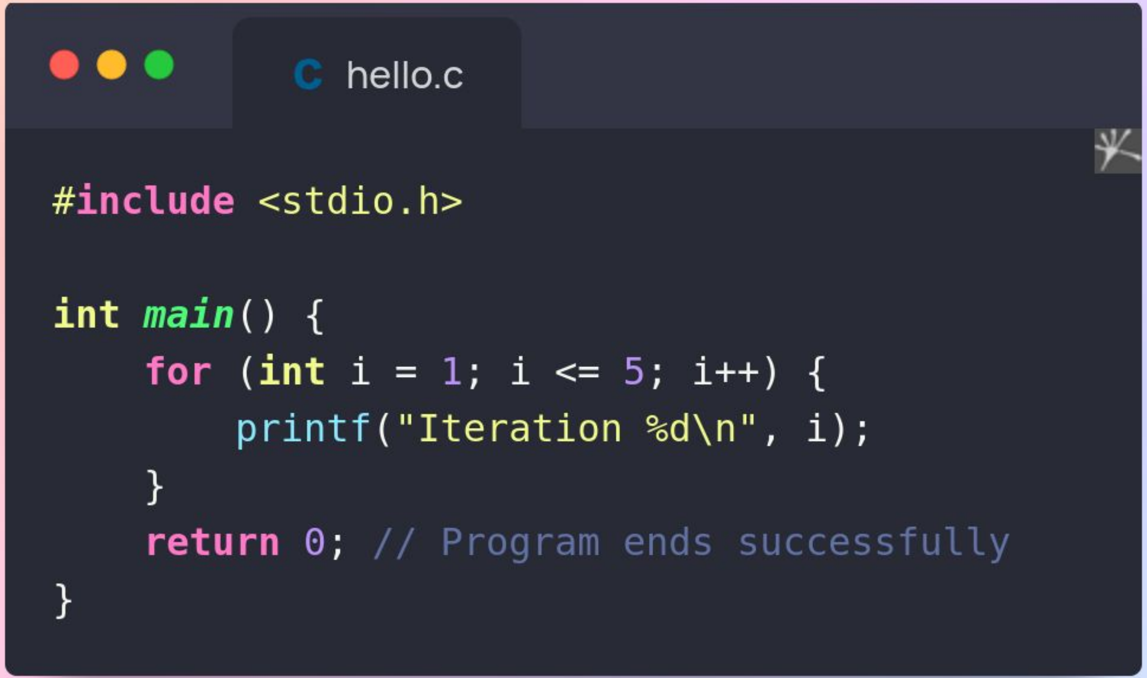
- for loop in C programming is a repetition control structure that allows programmers to write a loop that will be executed a specific number of times. for loop enables programmers to perform n number of steps together in a single line.
- **Syntax:**

```
for (initialization; condition; update) {
    // Code to execute
}
```

- **Example:**

```
for (int i = 1; i <= 5; i++) {
    printf("Iteration %d\n", i);
}
```





```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        printf("Iteration %d\n", i);
    }
    return 0; // Program ends successfully
}
```

While Loop

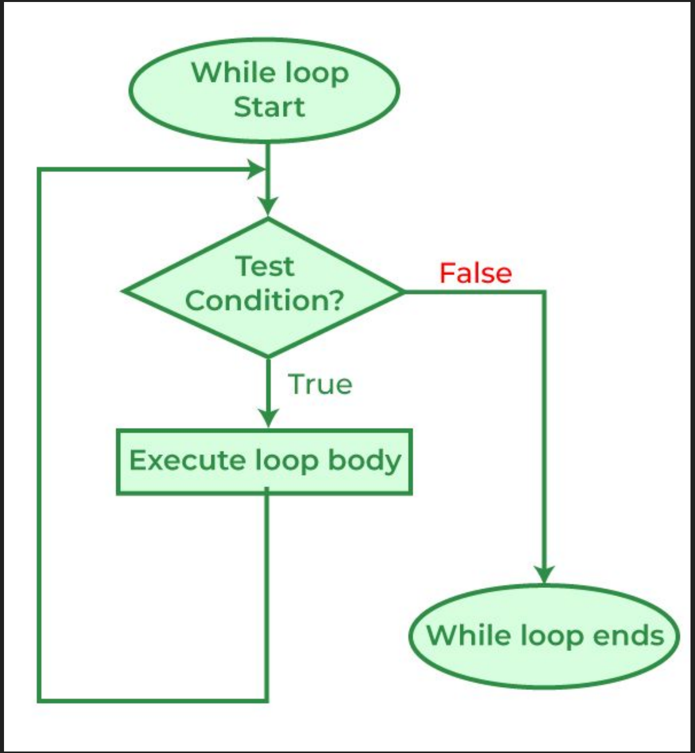
While loop does not depend upon the number of iterations. In for loop the number of iterations was previously known to us but in the While loop, the execution is terminated on the basis of the test condition. If the test condition will become false then it will break from the while loop else body will be executed.

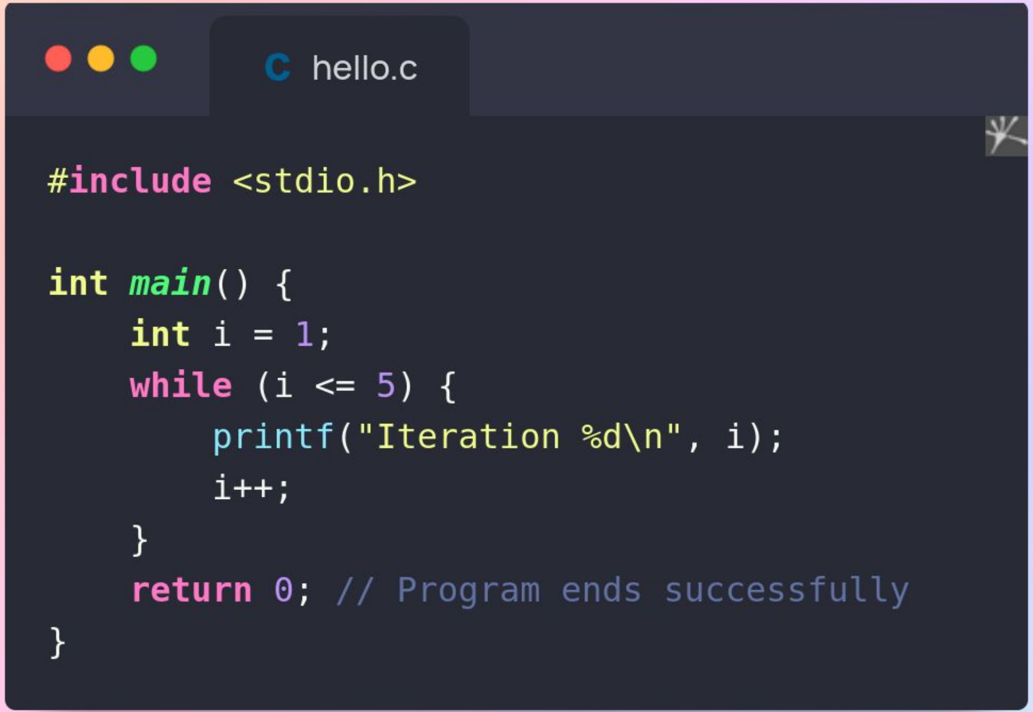
Syntax

```
while (condition) {
    // Code to execute
}
```

Example

```
int i = 1;
while (i <= 5) {
    printf("Iteration %d\n", i);
    i++;
}
```





```c
#include <stdio.h>

int main() {
    int i = 1;
    while (i <= 5) {
        printf("Iteration %d\n", i);
        i++;
    }
    return 0; // Program ends successfully
}
```

Do-While Loop

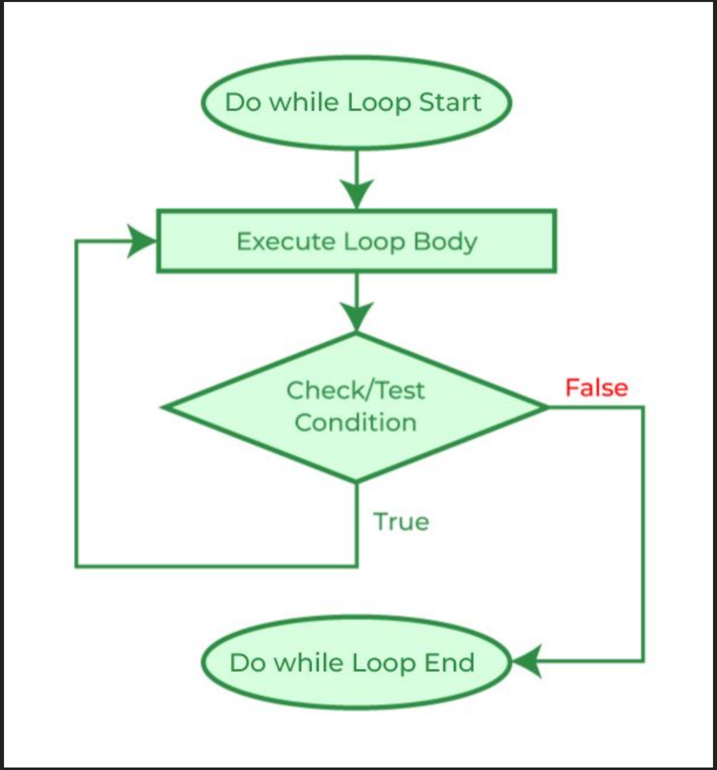
The do-while loop is similar to a while loop but the only difference lies in the do-while loop test condition which is tested at the end of the body. In the do-while loop, the loop body will execute at least once irrespective of the test condition.

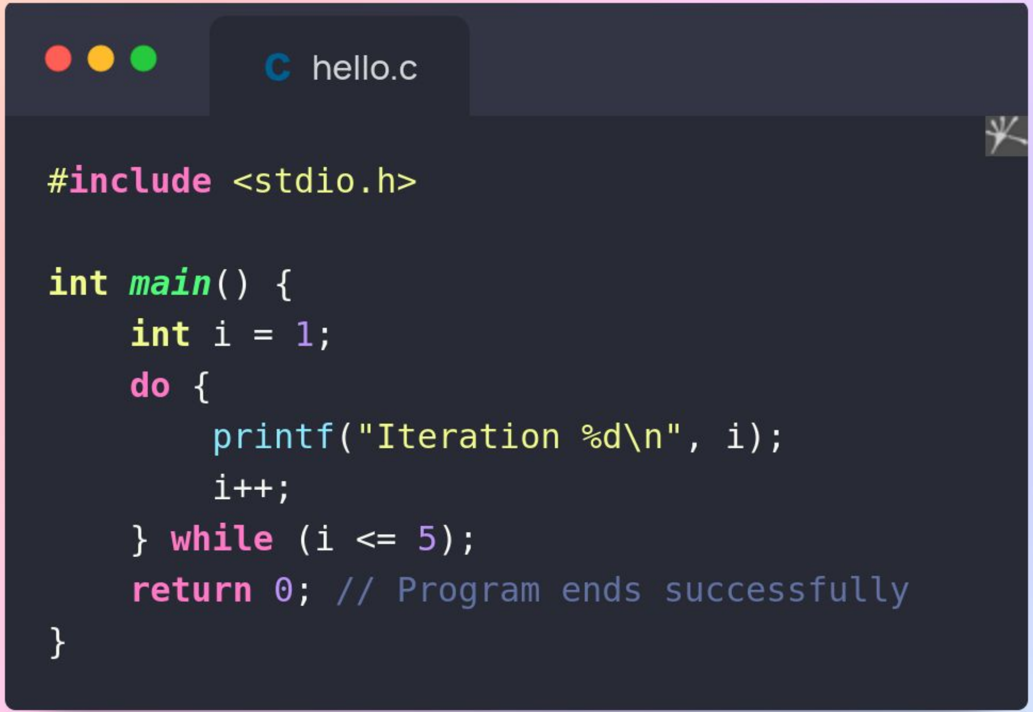
Syntax:

```
do {
    // Code to execute
} while (condition);
```

Example:

```
int i = 1;
do {
    printf("Iteration %d\n", i);
    i++;
} while (i <= 5);
```





```c
#include <stdio.h>

int main() {
    int i = 1;
    do {
        printf("Iteration %d\n", i);
        i++;
    } while (i <= 5);
    return 0; // Program ends successfully
}
```

Jump Statements

- **Definition:**
  - Jump statements transfer control to another part of the program.
  - These statements are used in C for the unconditional flow of control throughout the functions in a program. They support four types of jump statements:
- **Types:**
  - Break
  - Continue
  - Goto
  - return

break

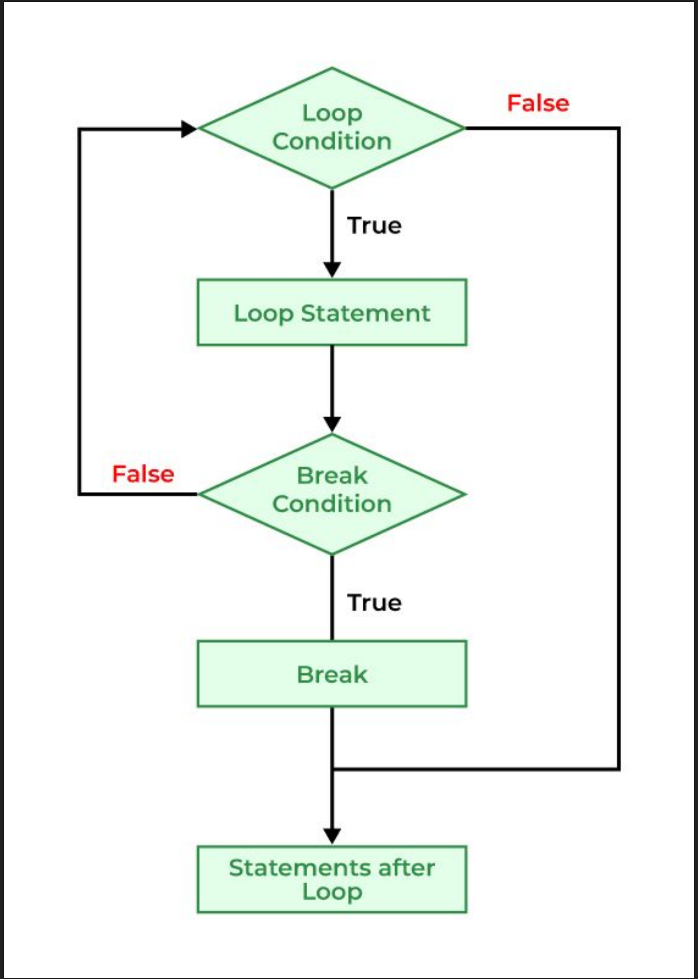
This loop control statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

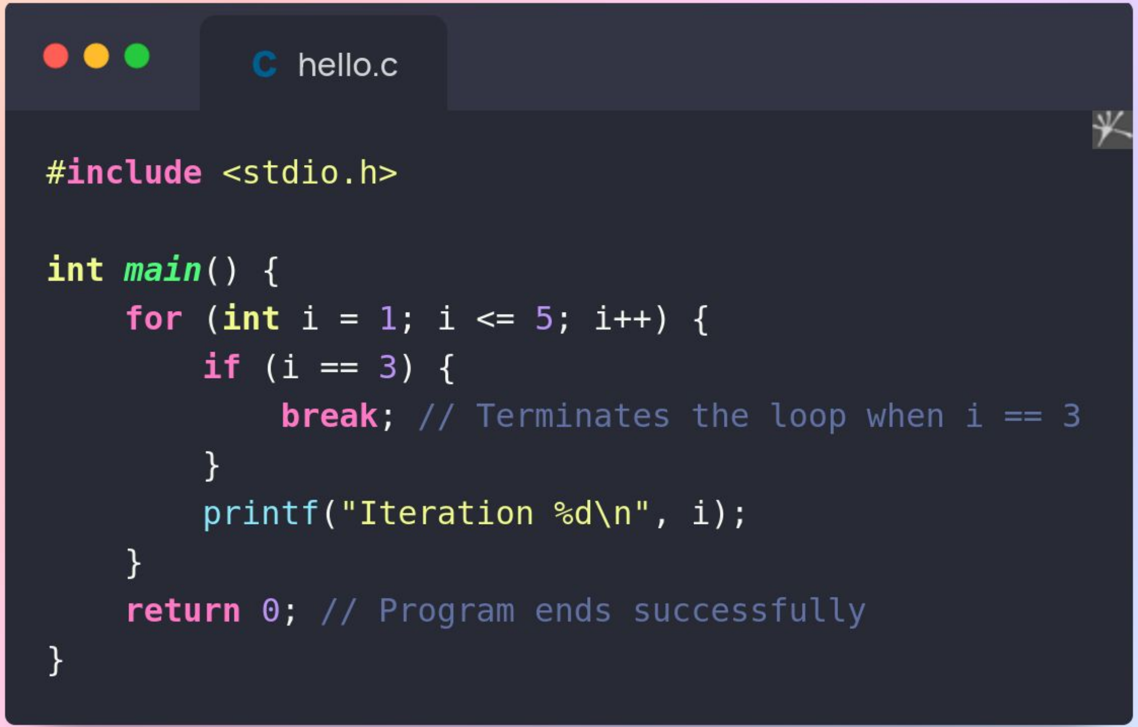
Syntax of break:

```
        break;
```

Example:

```
        for (int i = 1; i <= 5; i++) {
            if (i == 3) {
                break;
            }
            printf("Iteration %d\n", i);
        }
```




hello.c

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            break; // Terminates the loop when i == 3
        }
        printf("Iteration %d\n", i);
    }
    return 0; // Program ends successfully
}
```

continue

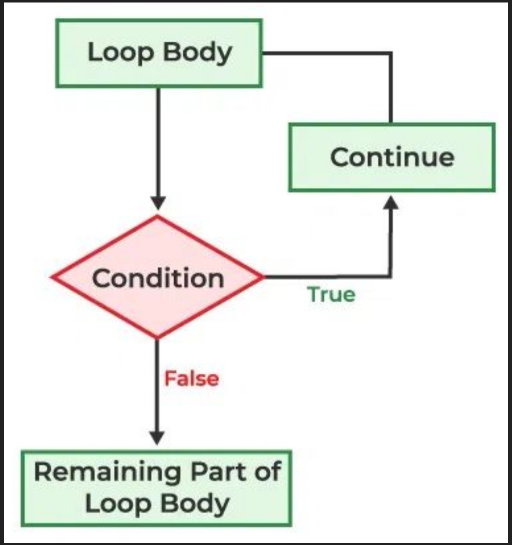
- This loop control statement is just like the break statement. The continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.
- As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

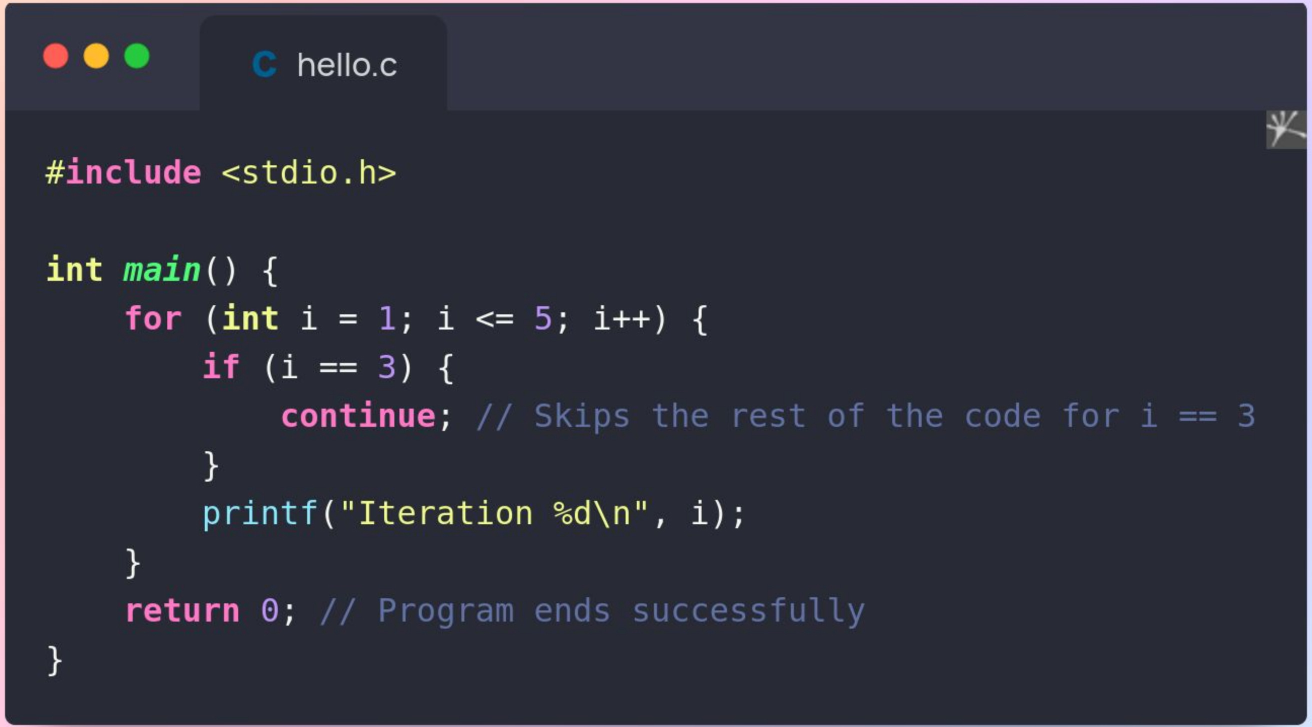
Syntax of continue

```
continue;
```

Example of continue

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        continue;
    }
    printf("Iteration %d\n", i);
}
```





```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            continue; // Skips the rest of the code for i == 3
        }
        printf("Iteration %d\n", i);
    }
    return 0; // Program ends successfully
}
```

goto

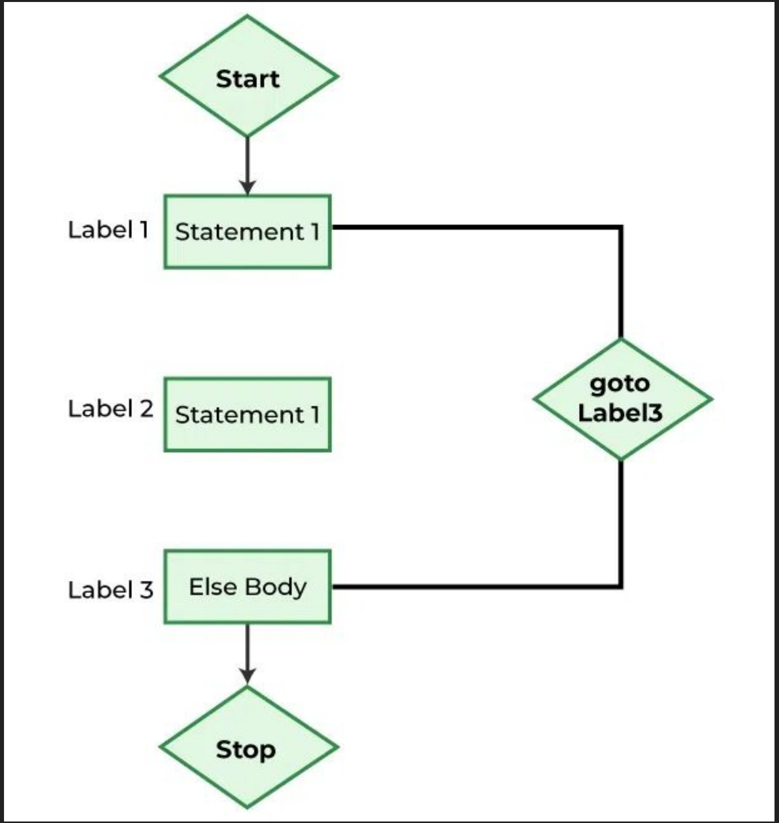
The goto statement in C also referred to as the unconditional jump statement can be used to jump from one point to another within a function.

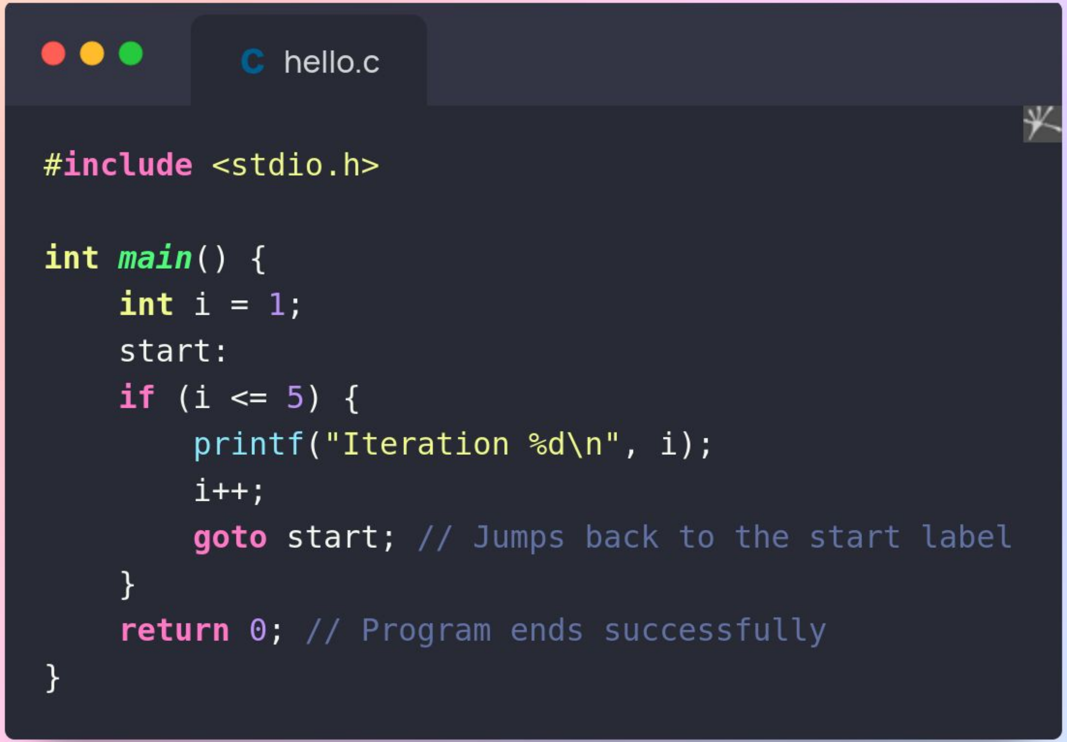
Syntax of goto

```
    goto label;
    label:
      // Code to execute
```

Example:

```
    int i = 1;
    start:
    if (i <= 5) {
        printf("Iteration %d\n", i);
        i++;
        goto start;
    }
```





```c
#include <stdio.h>

int main() {
    int i = 1;
    start:
    if (i <= 5) {
        printf("Iteration %d\n", i);
        i++;
        goto start; // Jumps back to the start label
    }
    return 0; // Program ends successfully
}
```

return

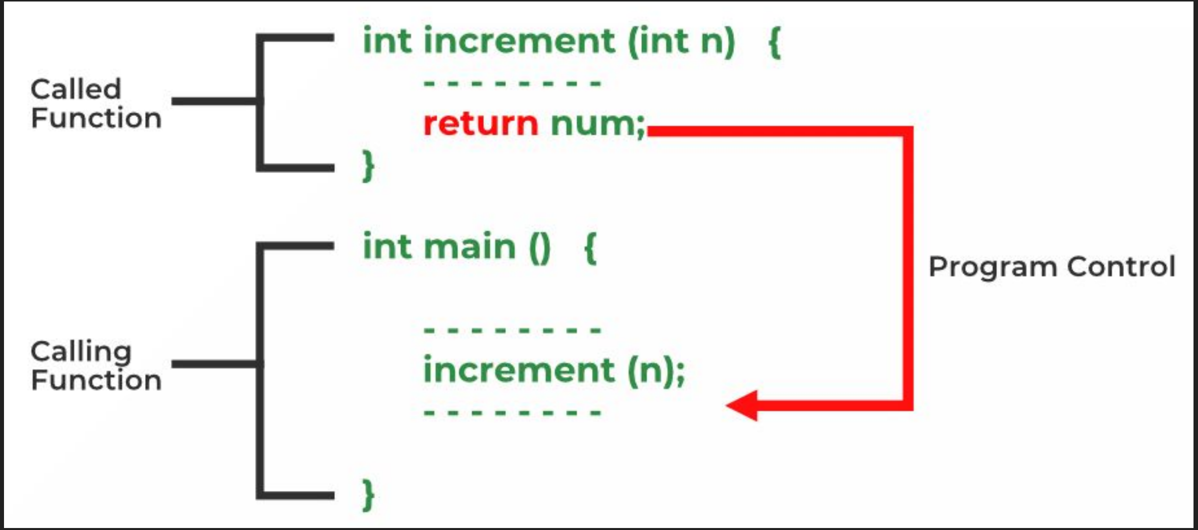
The return in C returns the flow of the execution to the function from where it is called. This statement does not mandatorily need any conditional statements. As soon as the statement is executed, the flow of the program stops immediately and returns the control from where it was called. The return statement may or may not return anything for a void function, but for a non-void function, a return value must be returned.

Syntax of return

```
        return [expression];
```

Example:

```
int SUM(int a, int b)
{
   return  a + b;
}
int main()
{
   int num1 = 10 , num2 = 10;
   int sum_of = SUM(num1, num2);
   printf("The sum is %d", sum_of);
   return 0;
}
```

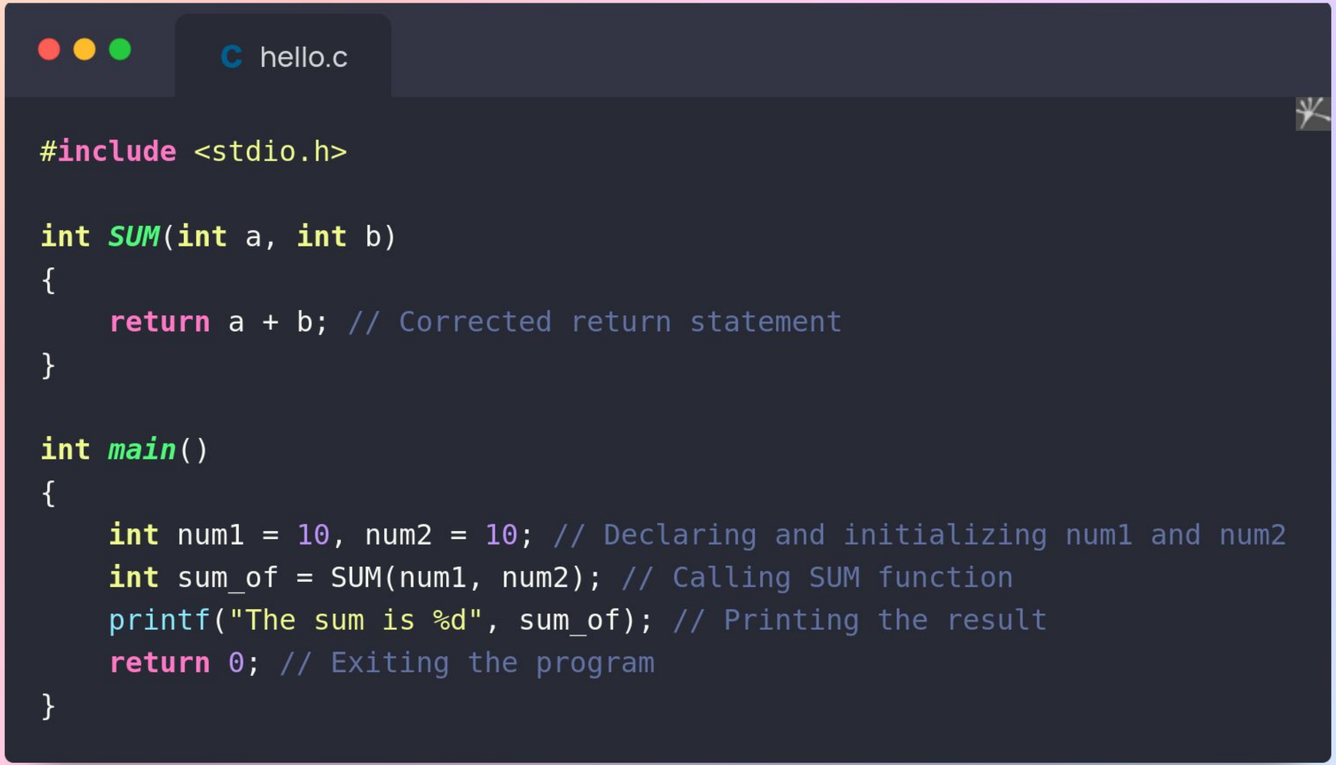


```
int increment (int n)   {
     - - - - - - - - -
     return num;
}

int main ()   {

     - - - - - - - - -
     increment (n);
     - - - - - - - - -

}
```


hello.c

```c
#include <stdio.h>

int SUM(int a, int b)
{
    return a + b; // Corrected return statement
}

int main()
{
    int num1 = 10, num2 = 10; // Declaring and initializing num1 and num2
    int sum_of = SUM(num1, num2); // Calling SUM function
    printf("The sum is %d", sum_of); // Printing the result
    return 0; // Exiting the program
}
```

Study Reference Links

https://www.geeksforgeeks.org/decision-making-c-cpp/

https://www.programiz.com/c-programming/c-if-else-statement

https://www.javatpoint.com/control-statements-in-c

https://www.simplilearn.com/tutorials/c-tutorial/conditional-control-statements-in-c

https://jsommers.github.io/cbook/control.html

https://www.naukri.com/code360/library/control-structures-in-c

THANK
YOU