Conditional Statements

Rakib Mahmud

Operators

- Conditional statements mainly work with two operators.

  They are:

  - Relational Operators
  - Logical Operators

Relational Operators

- They are used to compare data
- They return boolean results

Operator Symbol	Operator Name
<	Less Than
<=	Less Than Equal To
>	Greater Than
>=	Greater Than Equal To
==	Equal To
!=	Not Equal To

Relational Operators

- Int a=5, b=10, c=15

Operations	Return Type
a<b	True
b>c	False
a==c	False
a!=c	True
Operations	Return Type

Logical Operators

- If there is multiple conditions, then to combine them, we use logical operations.
- In order to use logical operators in both sides there need to be boolean value.
- The entire statement result will be also boolean

Operator Symbol	Operator Name
&&	Logical And
||	Logical Or
!	Logical Not

Logical Operators

A	B	A && B	A || B	! A
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Conditional Statements

- They are also known as Decision-Making Statements.
- Following are the decision-making statements available in C:
  - if Statement
  - if-else Statement
  - Nested if else Statement
  - if-else-if Ladder
  - switch Statement
  - Conditional Operator
  - Jump Statements: break, continue, jump, goto

C “if” statement

- The if statement is the most simple decision-making statement.
- It is used to decide whether a certain statement or block of statements will be executed or not.
- If the condition evaluates to true, the code block enclosed within the curly braces {} is executed.
- If the condition evaluates to false, the code block is skipped, and the program continues to the next statement after the if block.

C “if” statement

Syntax:

```
if (condition)
{
//Statements to be executed
}
```

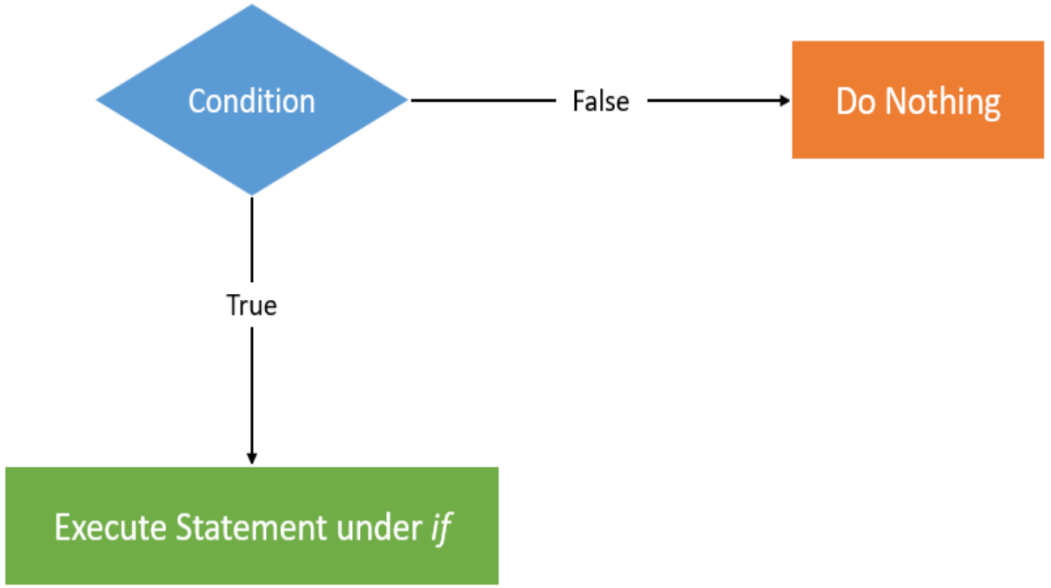

Figure 1: Block diagram of c if statement

C “if” statement

main.c

```c
#include<stdio.h>
int main()
{
    int i = 10;

    if (i > 15) {
        printf("10 is greater than 15");
    }

    printf("I am Not in if");
}
```

Output

```
/tmp/UMGPbShrDn.o
I am Not in if

=== Code Execution Successful ===
```

C “if else” statement

- The if-else statement is a decision-making statement that is used to decide whether the part of the code will be executed or not based on the specified condition.
- If the given condition is true, then the code inside the if block is executed, otherwise the code inside the else block is executed.

C “if else” statement

Syntax:

```
if (condition)
{
//Statements to be executed if condition satisfies
}
else
{
//Statements to be executed if the condition is not satisfied
}
```

C “if else” statement

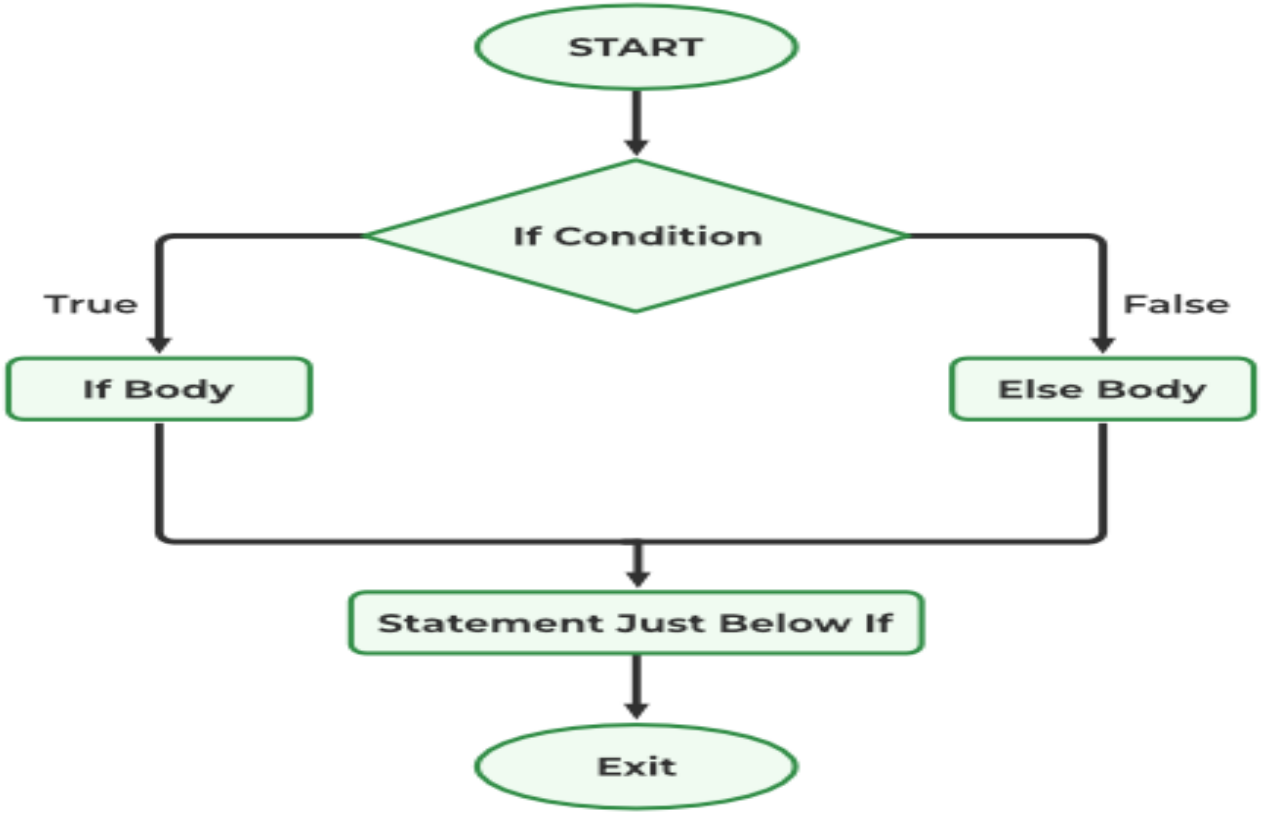

Figure 2: Block diagram of c if-else statement

C “if-else” statement

main.c

```c
#include <stdio.h>
int main()
{
    int i = 20;
    if (i < 15) {
        printf("i is smaller than 15\n");
    }
    else {
        printf("i is greater than 15\n");
    }
    printf("Hello CSE98 batch");
    return 0;
}
```

Output

```
/tmp/VTjPgARPQd.o
i is greater than 15
Hello CSE98 batch

=== Code Execution Successful ===
```

Two key points

- For every else block there have to be a corresponding if block. That is there is no else block without a respective if block.
- For every if block there may or may not be an else block.

C “Nested if else” statement

- Nested if statement C represents the if-else statement within another if-else statement.
- The inner if block executes only when the outer if condition is true.

C “Nested if else” statement

Syntax:

```
if (condition1)
{
    // Executes when condition1 is true
    if (condition_2)
    {
        // statement 1
    }
    else
    {
            // Statement 2
      }
}
else {
       if (condition_3)
    {
        // statement 3
    }
    else
    {
            // Statement 4
      }
}
```

C “Nested if else” statement

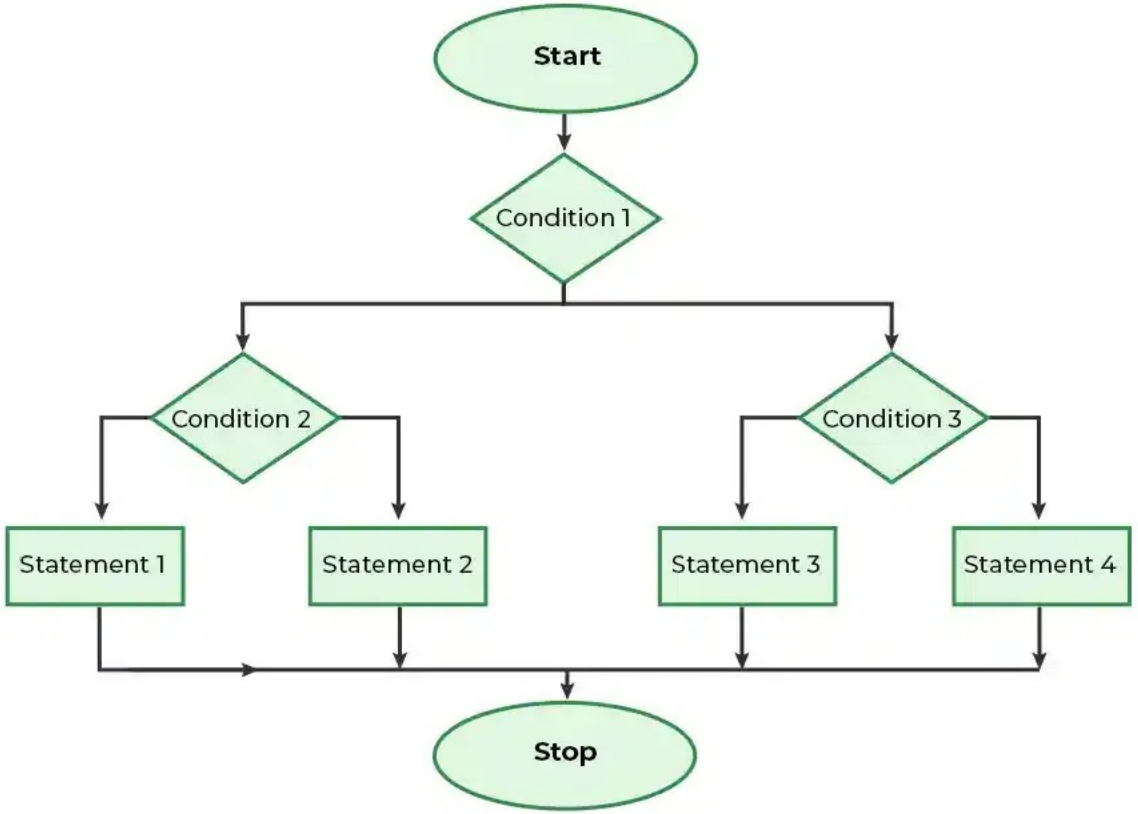

Figure 3: Block diagram of c Nested if-else statement

C “Nested if else” statement

```c
#include <stdio.h>
int main()
{
    int i = 10;
    if (i == 10) {
        if (i < 15)
            printf("i is smaller than 15\n");
        if (i < 12)
            printf("i is smaller than 12 too\n");
        else
            printf("i is greater than 15");
    }
    else {
        if (i == 20) {
            if (i < 22)
                printf("i is smaller than 22 too\n");
            else
                printf("i is greater than 25");
        }
    }
    return 0;
}
```

```
/tmp/kACt5vUTJt.o
i is smaller than 15
i is smaller than 12 too


=== Code Execution Successful ===
```

C “if…else if” statement

- The if else if statements are used when the user has to decide among multiple options.
- The C if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed.
- If none of the conditions is true, then the final else statement will be executed.

C “if…else if” statement

Syntax:

```
if (condition)
{
//Statements set 1
}
else if (condition 2)
{
//Statements set 2
}

. . .

else
{
//Statements to be executed if no condition is satisfied.
}
```

C “if...else if” statement

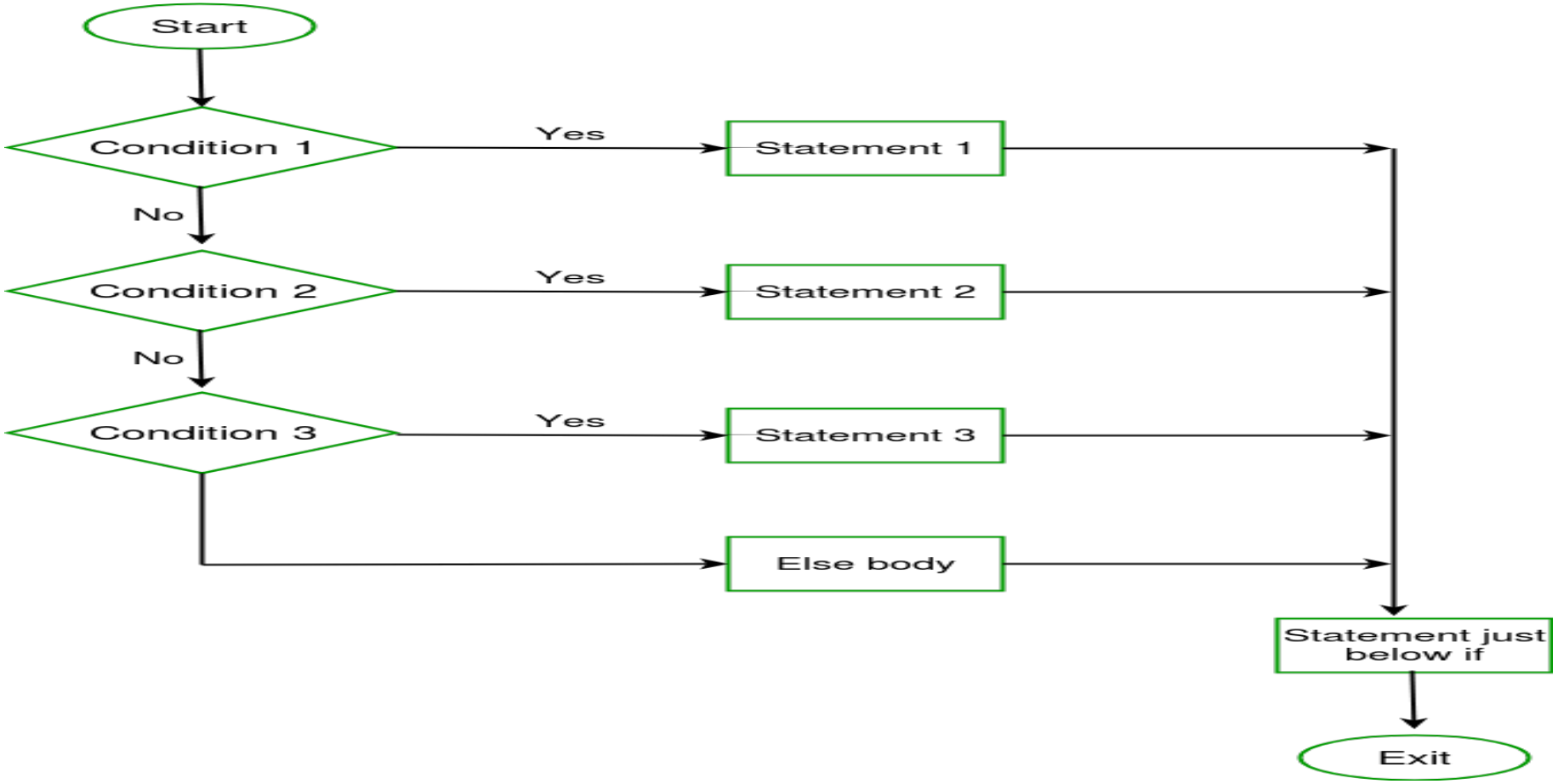

Figure 3: Block diagram of c if...else if statement

C “if...else if” statement

main.c

```c
#include <stdio.h>
int main()
{
    int i = 20;
    if (i == 10)
        printf("i is 10");
    else if (i == 15)
        printf("i is 15");
    else if (i == 20)
        printf("i is 20");
    else
        printf("i is not present");
}
```

Output

```
/tmp/leBhhN8A8K.o
i is 20

=== Code Execution Successful ===
```

C switch statement

- The switch case statement is an alternative to the if else if ladder.
- In the switch statement, the expression is evaluated, and its value is compared against each case value.
- If a case value matches the value of the expression, the corresponding code block is executed. The break statement is used to terminate the switch block and exit the statement.
- If a case doesn't have a break statement, the execution continues to the next case until a break is encountered or the switch block ends.

C switch statement

- The default case is optional and represents the code block to be executed when none of the case values match the expression.

Syntax:

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....

default:
  code to be executed if all cases are not matched;
}
```

Rules for switch statements in C

- The switch expression must be of an integer or character type.
- The case value must be an integer or character constant.
- The case value can be used only inside the switch statement.

C switch statement

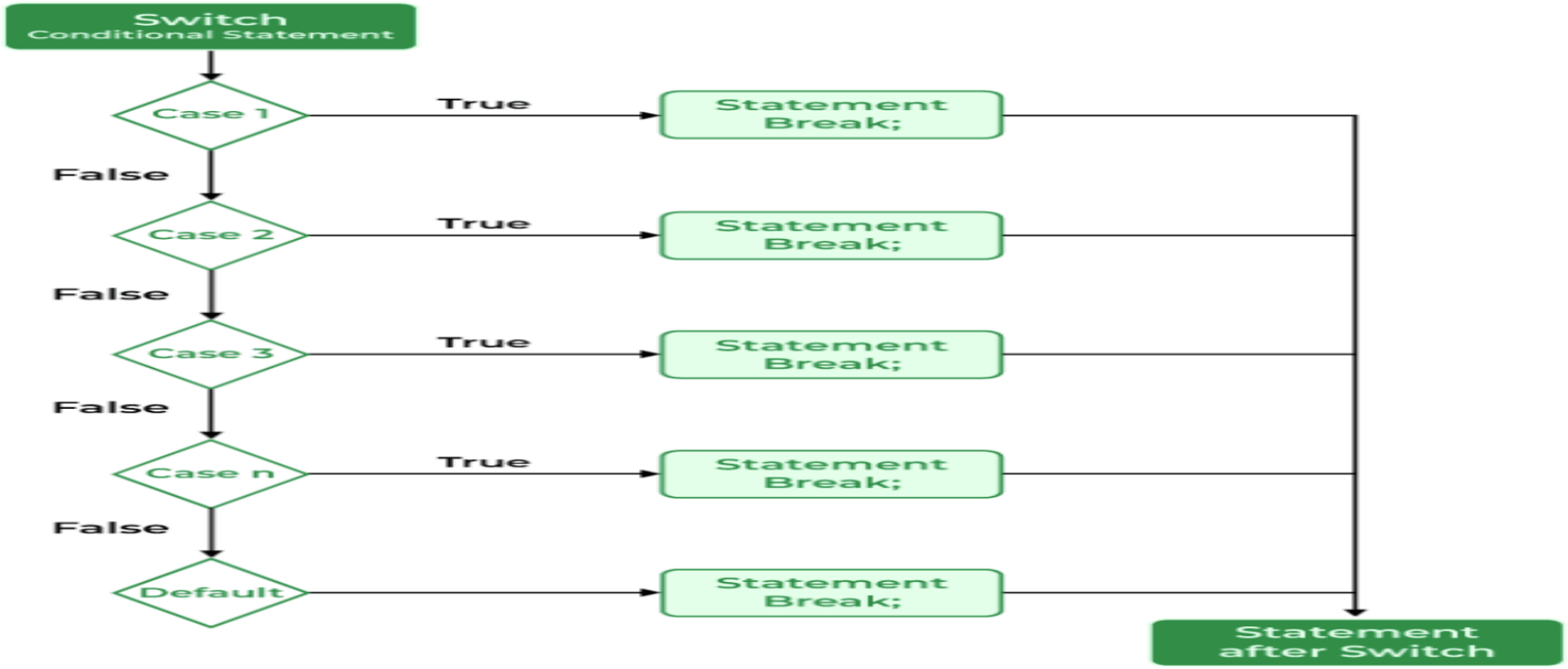

Figure 3: Block diagram of c switch statement

C switch statement

main.c

```c
#include <stdio.h>
int main()
{
    int var = 7;
    // declaring switch cases
    switch (var) {
    case 1:
        printf("Case 1 is executed\n");
        break;
    case 2:
        printf("Case 2 is executed\n");
        break;
    case 3:
        printf("Case 3 is executed\n");
        break;
    default:
        printf("Default Case is executed\n");
        break;
    }
    return 0;
}
```

Output

```
/tmp/RSNCU92lQO.o
Default Case is executed


=== Code Execution Successful ===
```

Differences

	If- Else	Switch
Flow of execution	Can have only two possible paths of execution either if or else	Can have multiple paths of execution in the forms of different cases
Type of outcome	Always evaluates an expression to a boolean value	Can evalute to an expression to a enum, short, char and int primitive type values
Readability	Less readble in case of multiple if else	More readable in case of multiple cases
Efficiency	It becomes less efficient in case of multiple if-else. Since it checks for every if/else and evaluates the corresponding statements.	More efficient working because once a valid match is found, the other cases are bypassed if handled carefully with break statement

Thank You