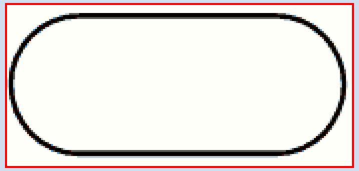
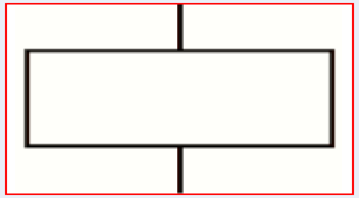
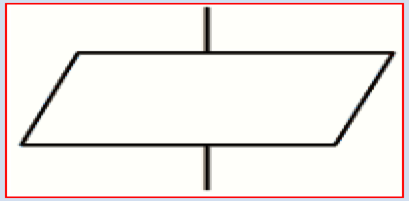
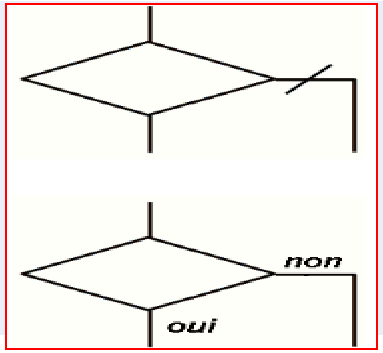
Conditional statement

The Role of Flowcharts in Algorithm Design

- Flowcharts are graphical representations of algorithms that depict the logical steps involved in solving a problem or performing a task. They serve as a visual guide for understanding, planning, and implementing algorithms.

Key Components of a Flowchart for Algorithm Design

- **Start and End Points**:
Every flowchart begins with a start point and concludes with an end point, denoting the algorithm's initiation and conclusion.
- **Processes**:
Rectangles represent processes or operations. In algorithm design, these correspond to specific actions or computations.
- **Decisions**:
Diamonds symbolize decision points where conditions are evaluated. Based on the outcome, the algorithm proceeds along different paths.
- **Input/Output**:
Parallelograms denote input or output operations. In algorithm design, this includes operations such as reading data or displaying results.

Symbol	Description
	Start and End Points
	Processes or operations
	Parallelograms denote input or output operations including reading data or displaying results.
	Diamonds symbolize decision points

Introduction

- Conditional statements are fundamental programming constructs that allow the execution of different code blocks based on specified conditions.

  - **If Statement**
  - **If-Else Statement**
  - **Nested If-Else Statement**
  - **Switch Statement**

if statement

```
if(condition)
{
      //if true
}
```

if-else

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

if-else-if

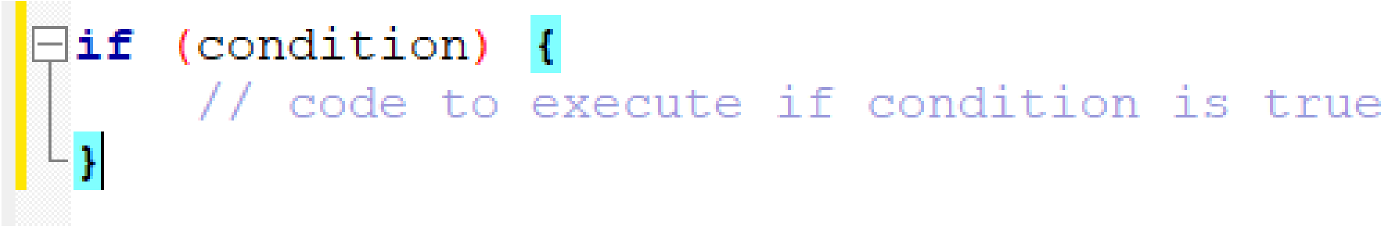
```
if(condition)
{
     // true
}
else if(condition 2)
{
    // cond 2 true
}
else if(condition 3)
{
    // cond 3 true
}
else
{
    //false
}
```

Nested if-else

```
if(condition)
{
   if(condition 2)
   {
      if(condition 3)
         {

          }
   }
}
else
{
}
```

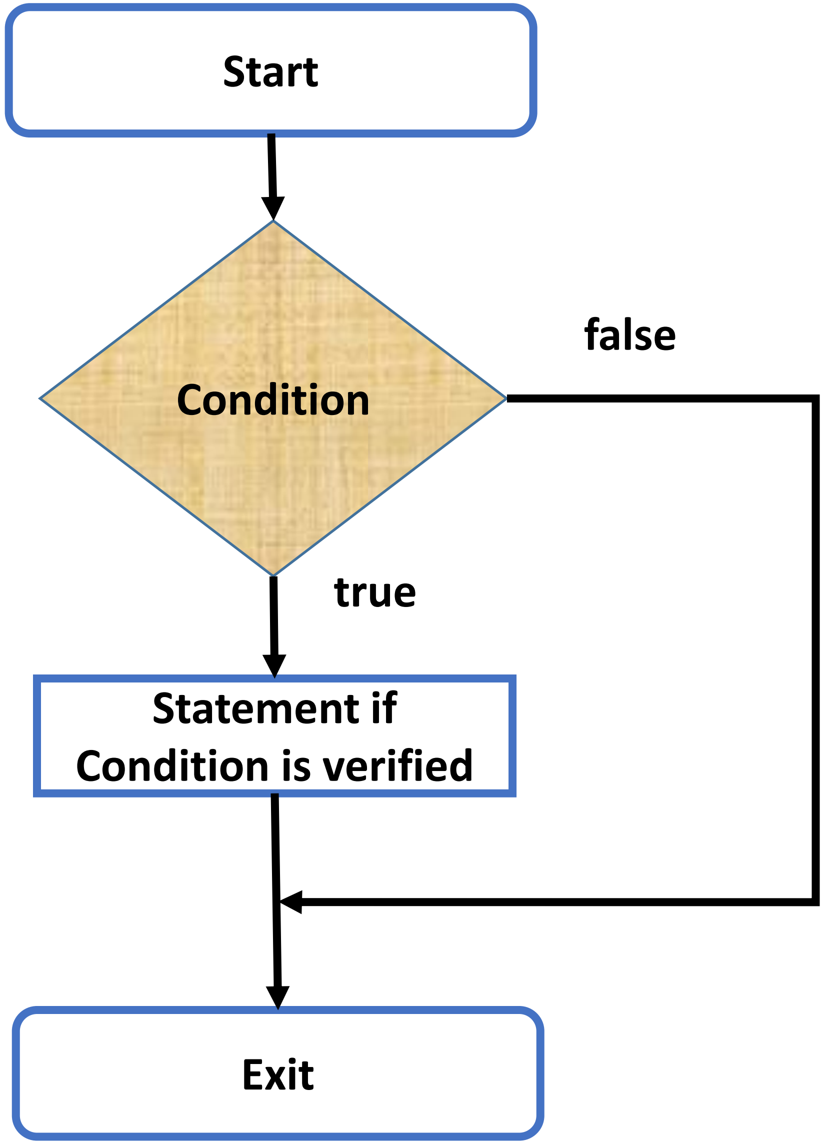
if statement

- The if statement allows you to execute a block of code if a specified condition is true.
- Syntax:



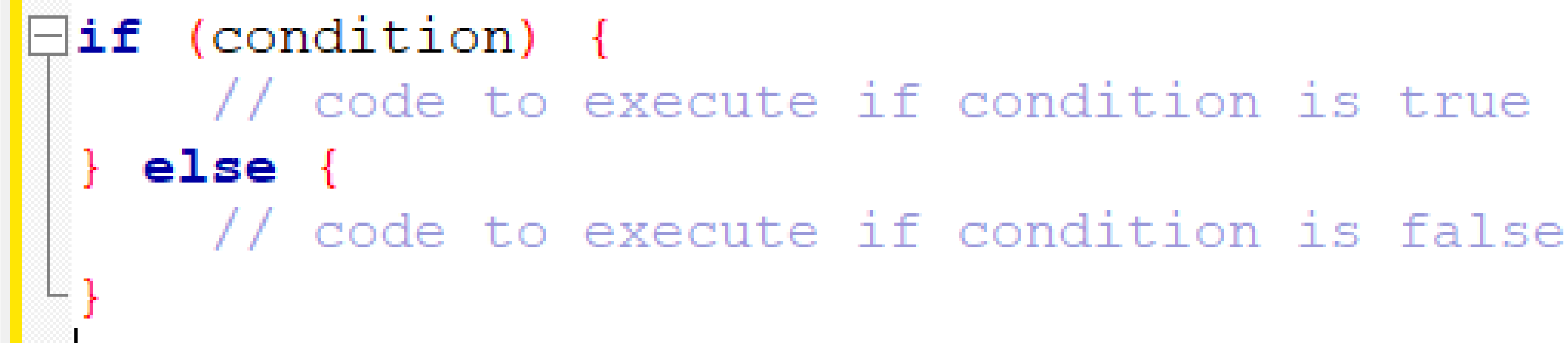
```
if (condition) {
    // code to execute if condition is true
}
```

Flowchart of if Statement



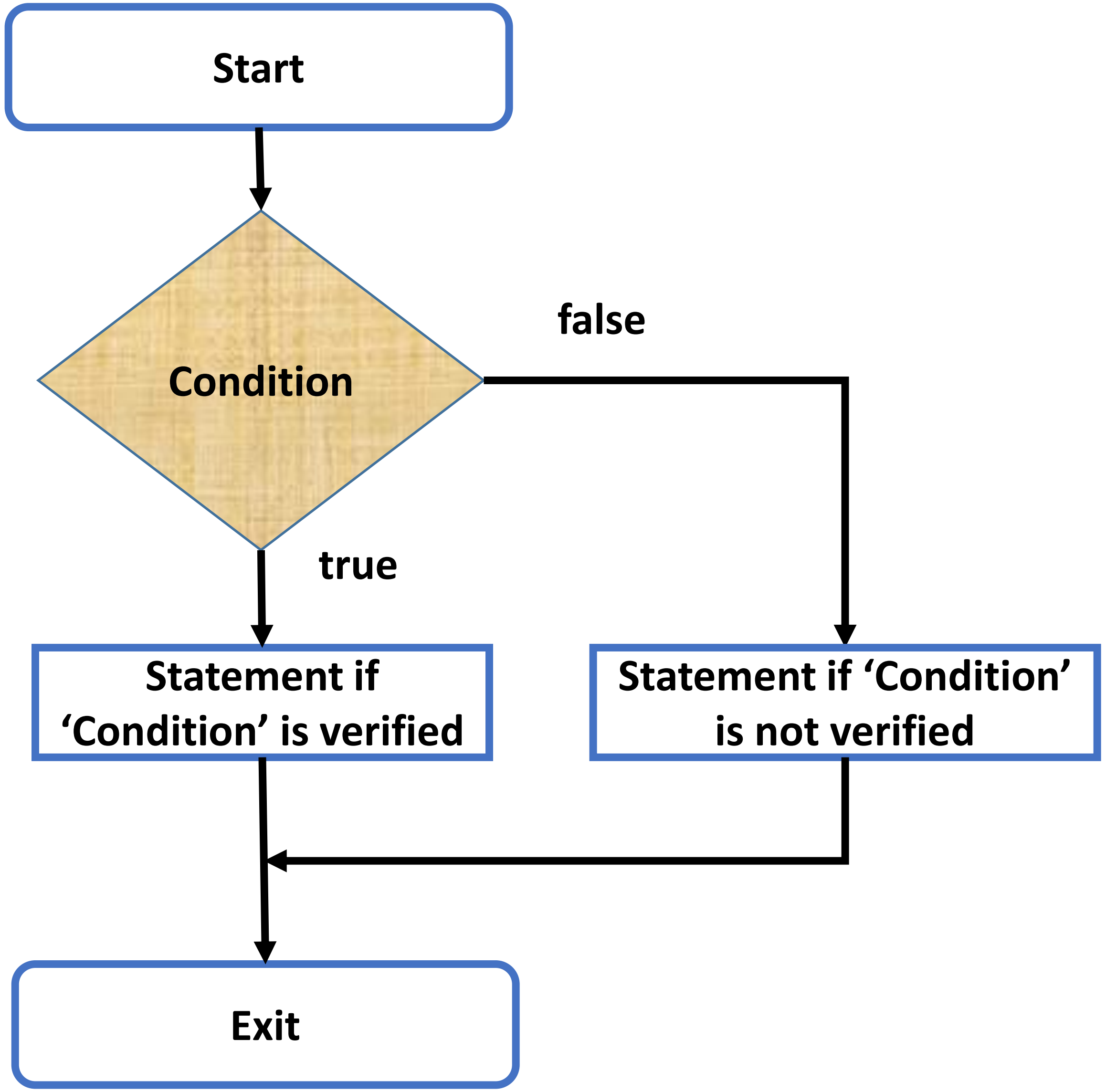
If-Else Statement

- The if-else statement provides an alternative block of code to execute if the condition is false.
- Syntax:



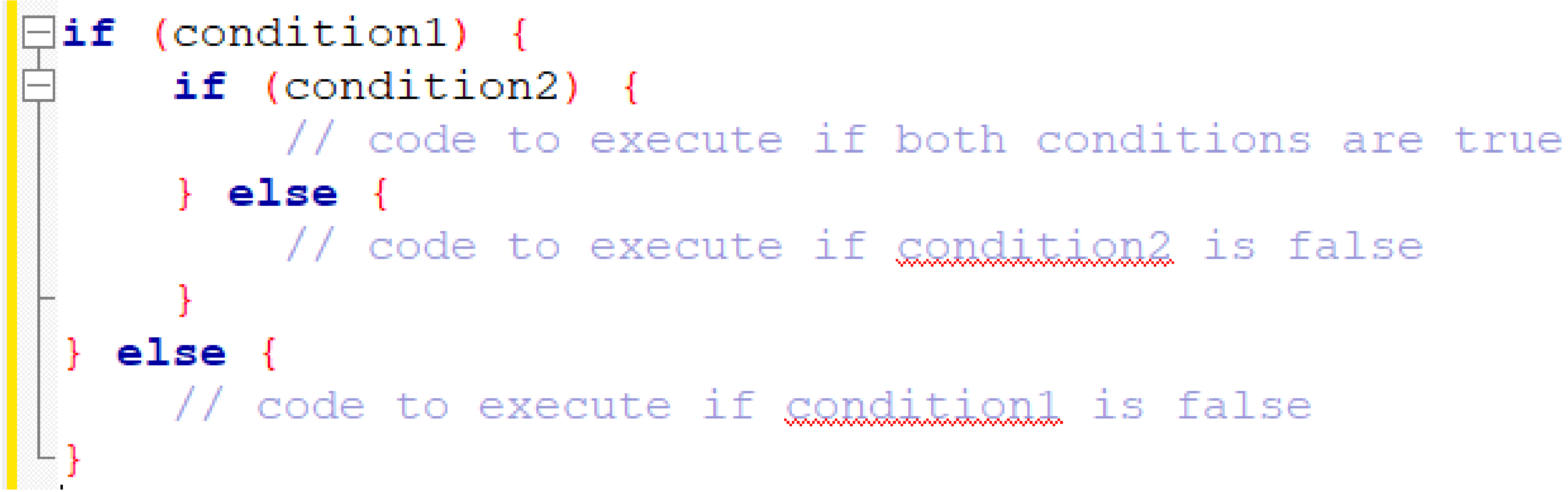
```
if (condition) {
    // code to execute if condition is true
} else {
    // code to execute if condition is false
}
```

Flowchart of if-else Statement


Start
Condition
false
true
Statement if ‘Condition’ is verified
Statement if ‘Condition’ is not verified
Exit

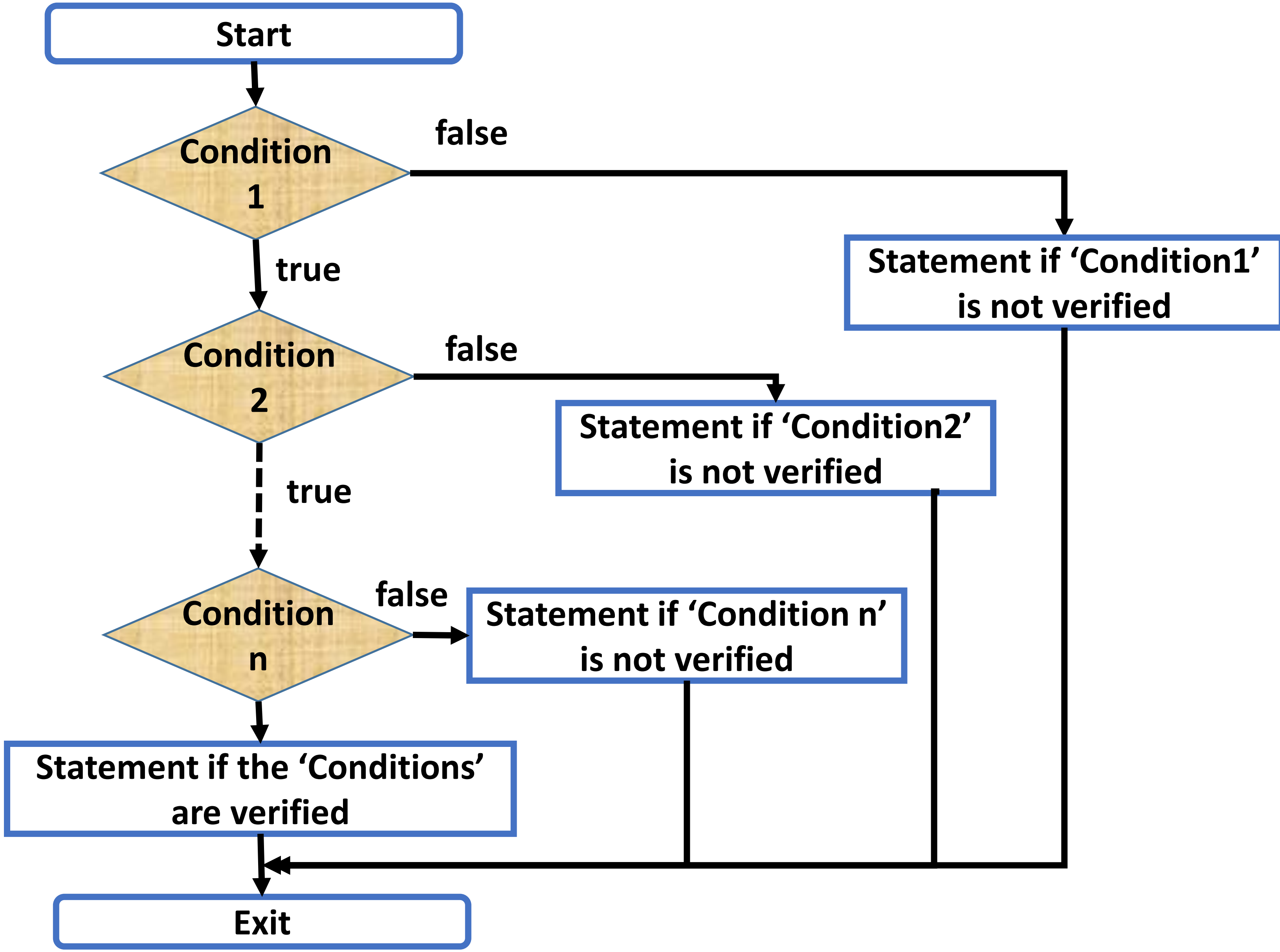
Nested If-Else Statement

- Nesting involves placing one conditional statement inside another.
- It allows for multiple levels of decision-making.
- Example:



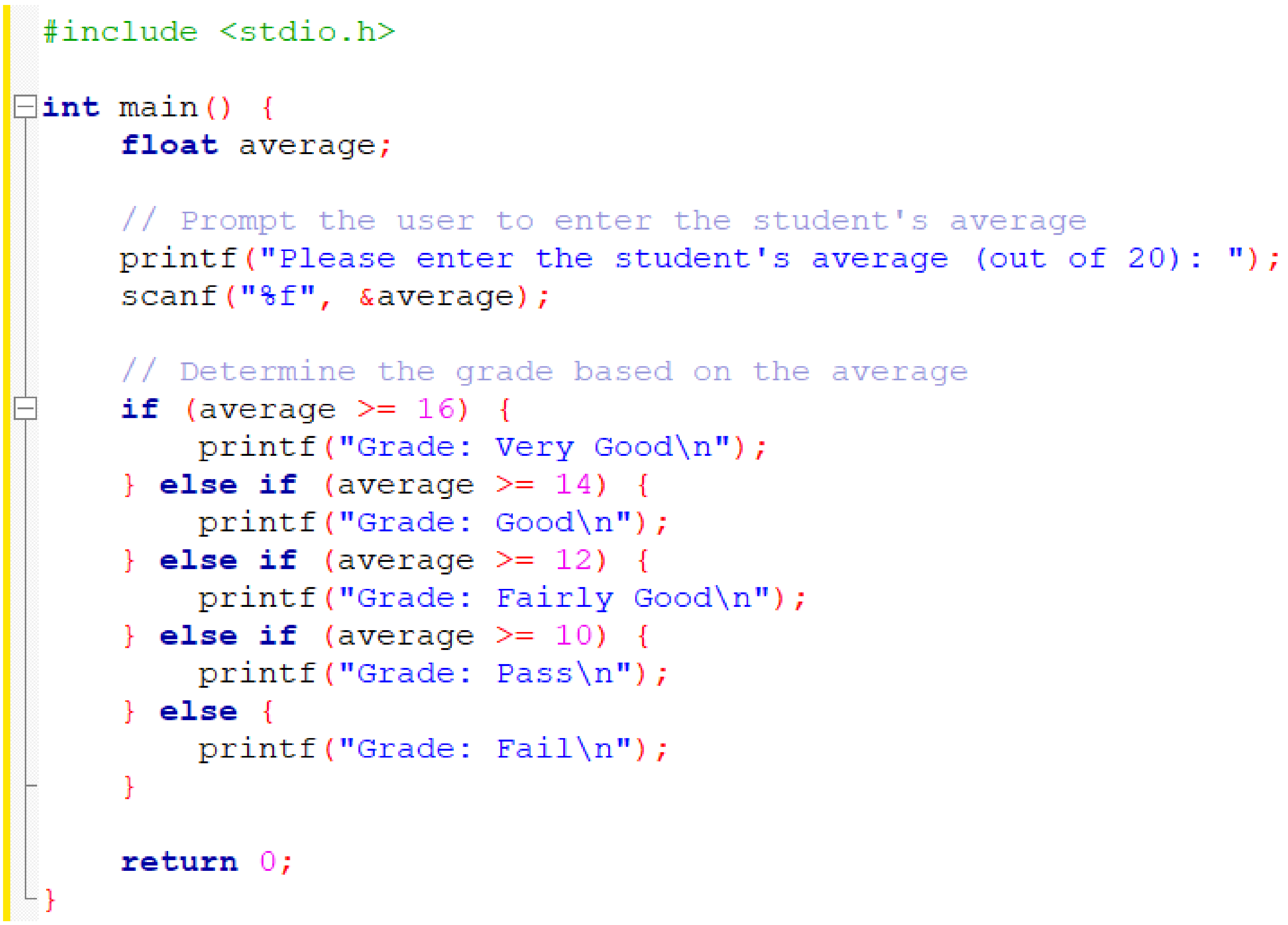
```
if (condition1) {
    if (condition2) {
        // code to execute if both conditions are true
    } else {
        // code to execute if condition2 is false
    }
} else {
    // code to execute if condition1 is false
}
```

Nested If-Else Statement


Start
Condition 1
false
true
Statement if ‘Condition1’ is not verified
Condition 2
false
Statement if ‘Condition2’ is not verified
true
Condition n
false
Statement if ‘Condition n’ is not verified
Statement if the ‘Conditions’ are verified
Exit

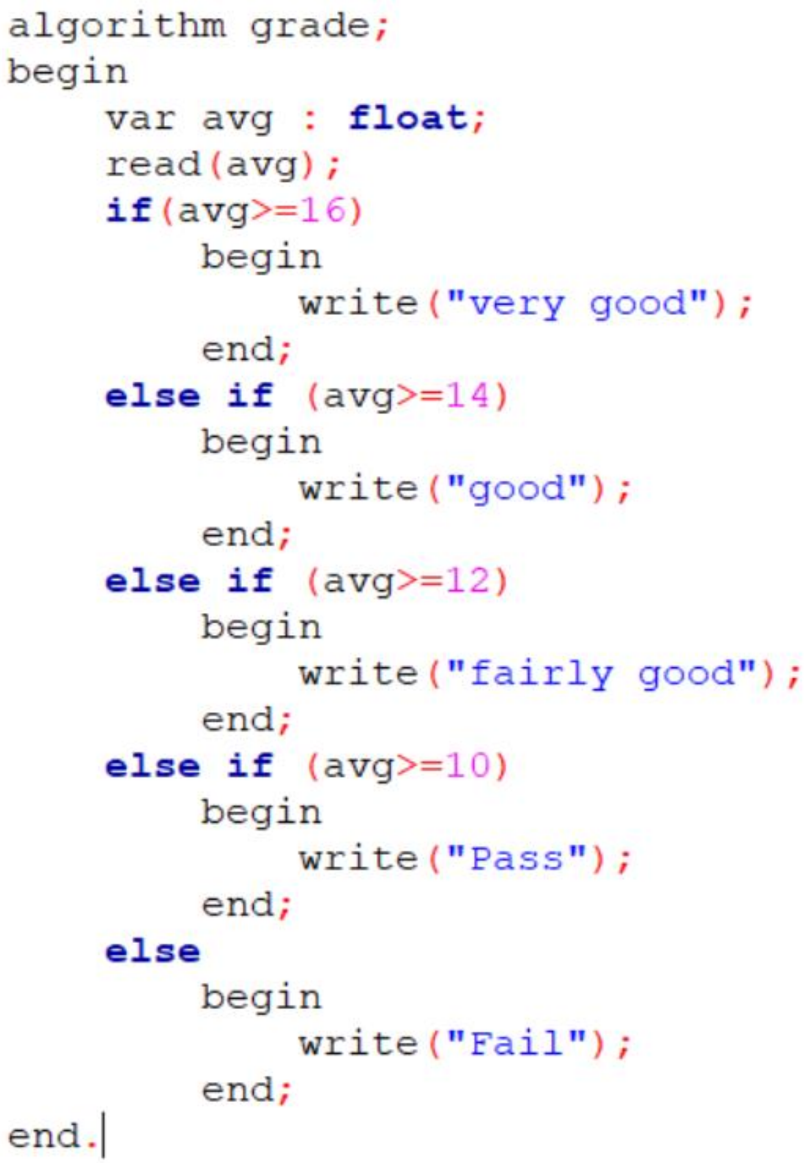
Example

- Write a C program that displays a student's grade based on their average



```c
#include <stdio.h>

int main() {
    float average;

    // Prompt the user to enter the student's average
    printf("Please enter the student's average (out of 20): ");
    scanf("%f", &average);

    // Determine the grade based on the average
    if (average >= 16) {
        printf("Grade: Very Good\n");
    } else if (average >= 14) {
        printf("Grade: Good\n");
    } else if (average >= 12) {
        printf("Grade: Fairly Good\n");
    } else if (average >= 10) {
        printf("Grade: Pass\n");
    } else {
        printf("Grade: Fail\n");
    }

    return 0;
}
```

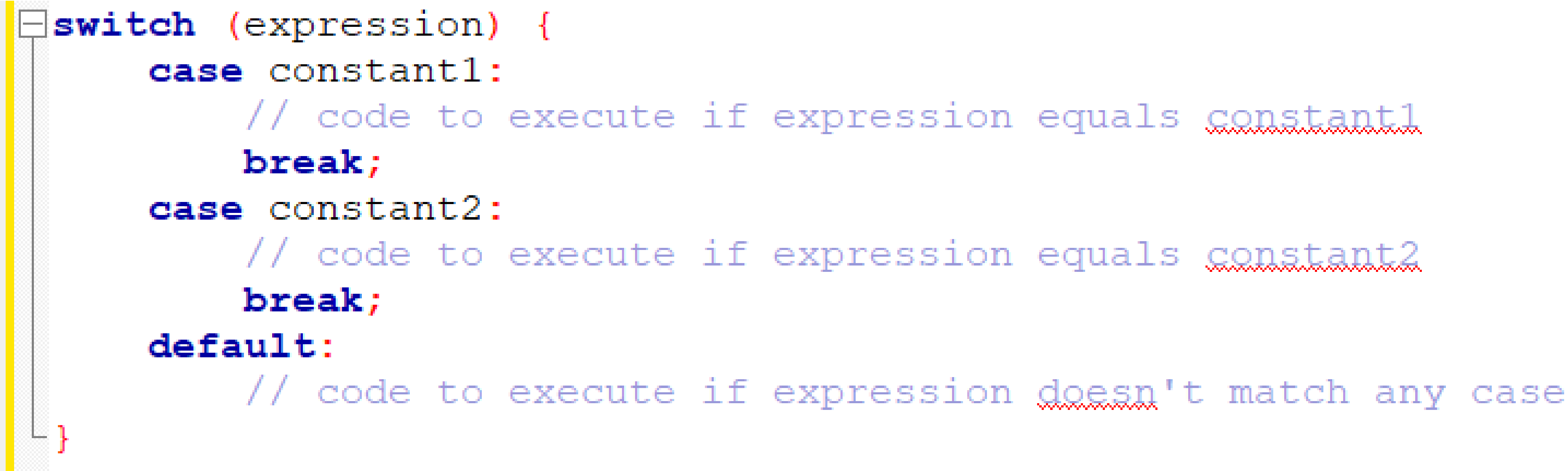
Algorithmic notation



```
algorithm grade;
begin
    var avg : float;
    read(avg);
    if(avg>=16)
        begin
            write("very good");
        end;
    else if (avg>=14)
        begin
            write("good");
        end;
    else if (avg>=12)
        begin
            write("fairly good");
        end;
    else if (avg>=10)
        begin
            write("Pass");
        end;
    else
        begin
            write("Fail");
        end;
end.
```

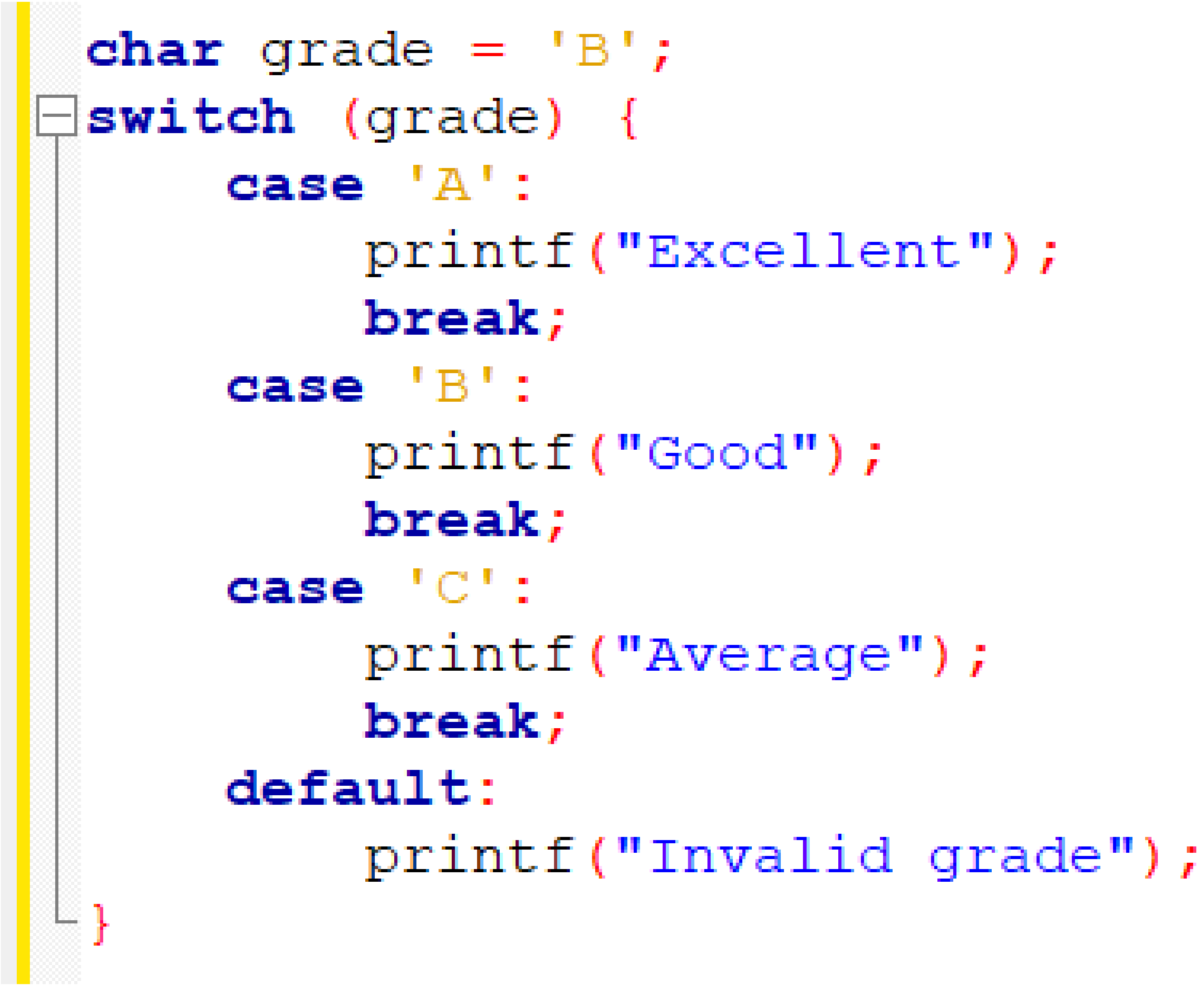
Switch Statement

- The switch statement allows to select one of many code blocks to be executed.
- Useful when there are multiple cases to consider.
- Syntax:



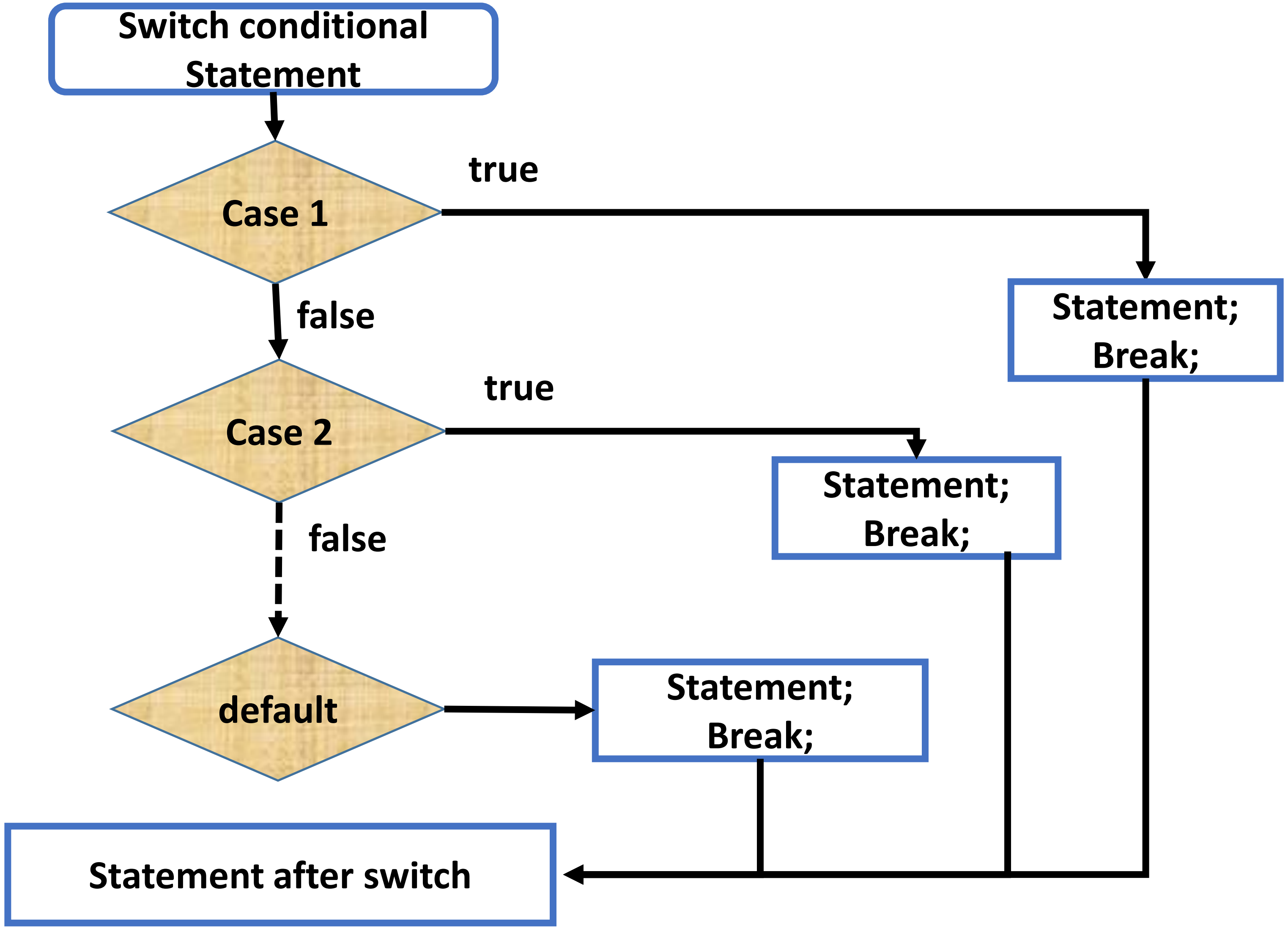
```
switch (expression) {
    case constant1:
        // code to execute if expression equals constant1
        break;
    case constant2:
        // code to execute if expression equals constant2
        break;
    default:
        // code to execute if expression doesn't match any case
}
```

Example of switch Statement



```c
char grade = 'B';
switch (grade) {
    case 'A':
        printf("Excellent");
        break;
    case 'B':
        printf("Good");
        break;
    case 'C':
        printf("Average");
        break;
    default:
        printf("Invalid grade");
}
```

Flowchart of Switch



Exercise

- Write a C program that implements a calculator with basic operations (+,-,*,/) using if-else, and then using switch-case.