CSE 232

Programming with C++

Lecture 5

Control Structures

Prepared by

Md. Mijanur Rahman, Prof. Dr.
Dept. of Computer Science and Engineering
Jatiya Kabi Kazi Nazrul Islam University, Bangladesh
Email: mijan@jkkniu.edu.bd
Web: www.mijanrahman.com

Contents

Control Structures

Conditional Control Structures

- **Control Structures** are statements that change the flow of a program to a different code segment based on certain conditions.
- The control structures are categorized into three major Conditional types; they are:
  1. Decision making and branching statements
     a) Selection statements
     b) Jump statements
  2. Decision making and looping (Iteration)

Conditional Control Structures

- **Conditional Control Structures statements in C:**

1. Selection Statements:

- If statement
- If Else Statement
- Else If statement
- Nested If statement
- Switch statement

2. Iteration Statements:

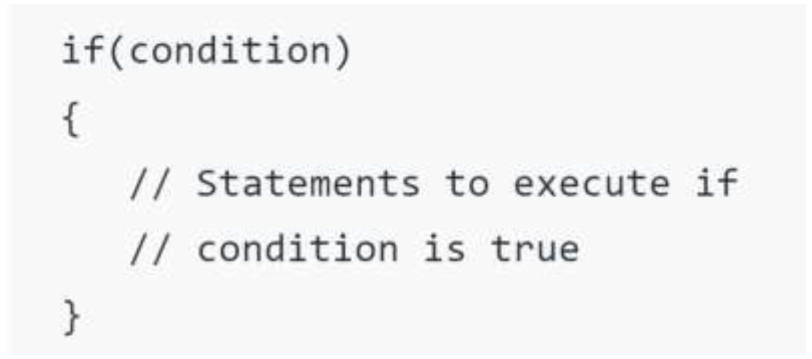
- For loop
- While loop
- do while loop

3. Jump Statements:

- return
- goto
- exit()
- break
- continue

If statement in C++

- if statement is the simplest decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not; i.e., if a certain condition is true then a block of statement is executed otherwise not.



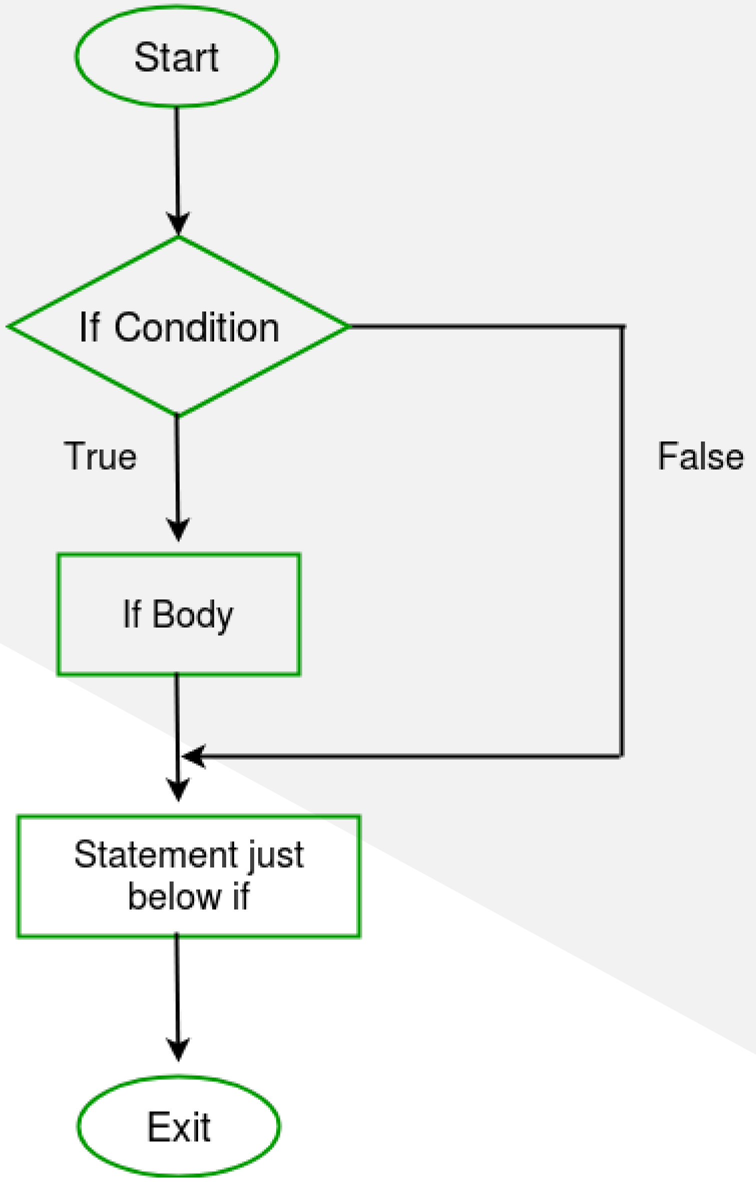
```
if(condition)
{
   // Statements to execute if
   // condition is true
}
```

- Here, the **condition** after evaluation will be either true or false. C++ if statement accepts Boolean values – if the value is true then it will execute the block of statements below it otherwise not. If we do not provide the curly braces ‘{‘ and ‘}’ after if(condition) then by default if statement will consider the first immediately below statement to be inside its block.

If Statement in C++

- Example:

```
if(condition)
   statement1;
   statement2;

// Here if the condition is true, if block
// will consider only statement1 to be inside
// its block.
```

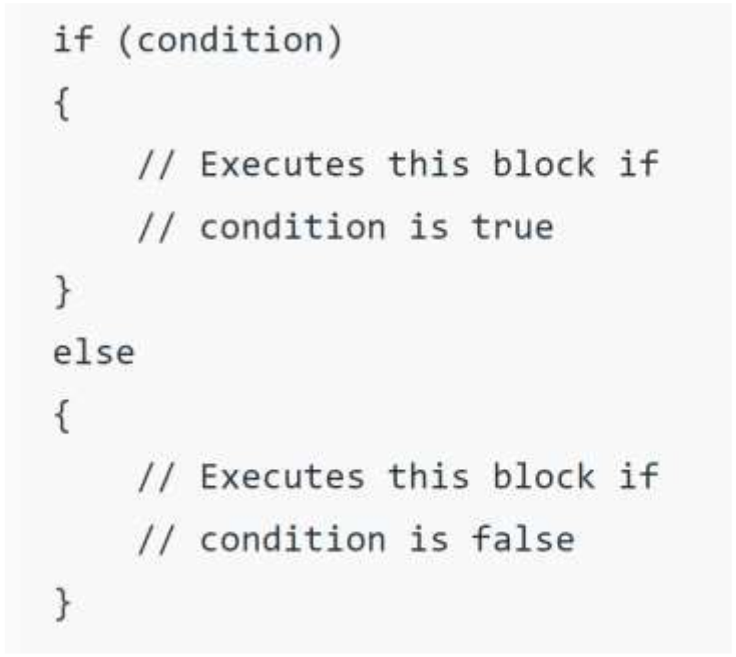


Flowchart of IF statement

If-else Statement in C++

- The *if* statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else if the condition is false. Here comes the C *else* statement.
- We can use the *else* statement with *if* statement to execute a block of code when the condition is false.

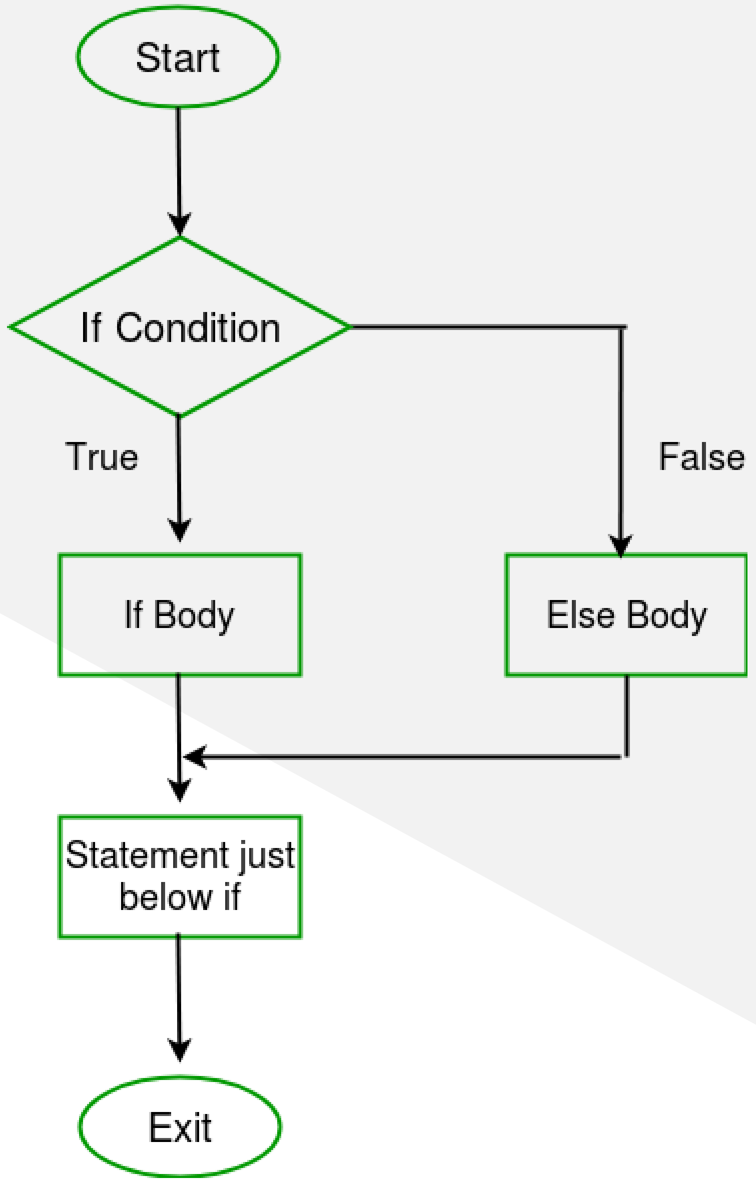
Syntax:



```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

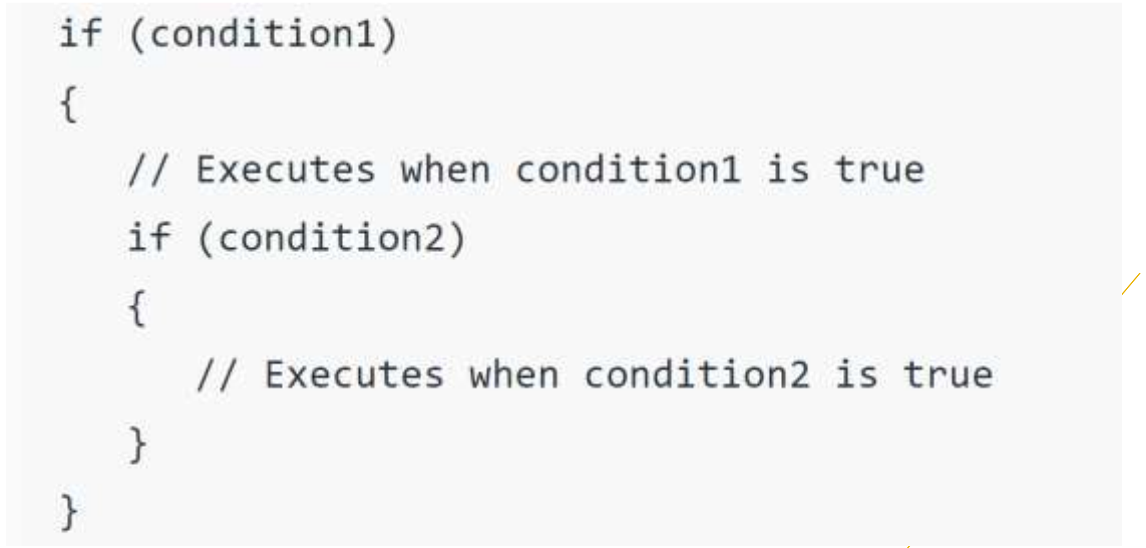
If-else Statement in C++

- **Flowchart of IF-ELSE statement:**



Nested If-else statement in C++

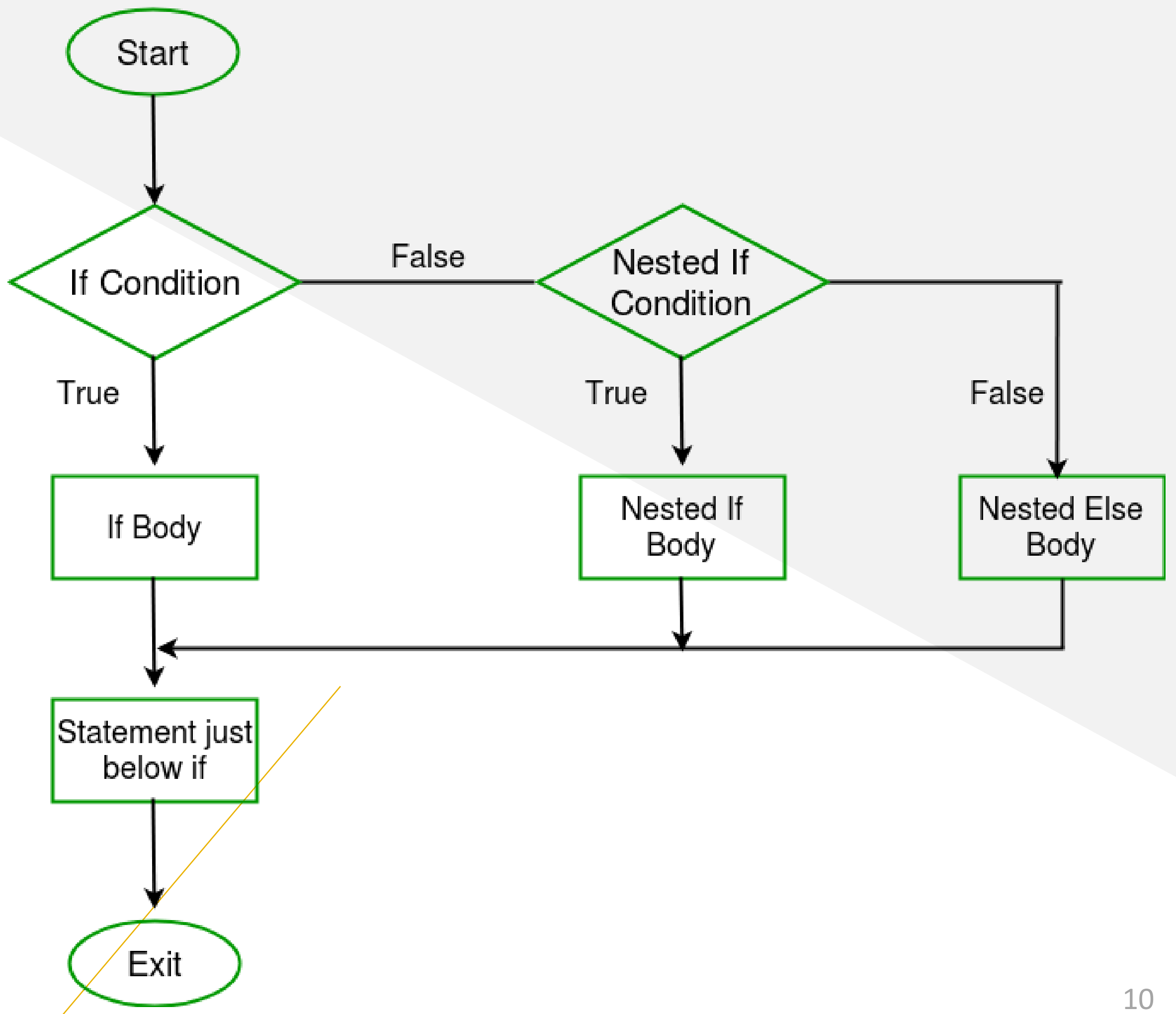
- A nested if in C++ is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, both C and C++ allow us to nested if statements within if statements, i.e., we can place an if statement inside another if statement.
- **Syntax:**



```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
      // Executes when condition2 is true
   }
}
```

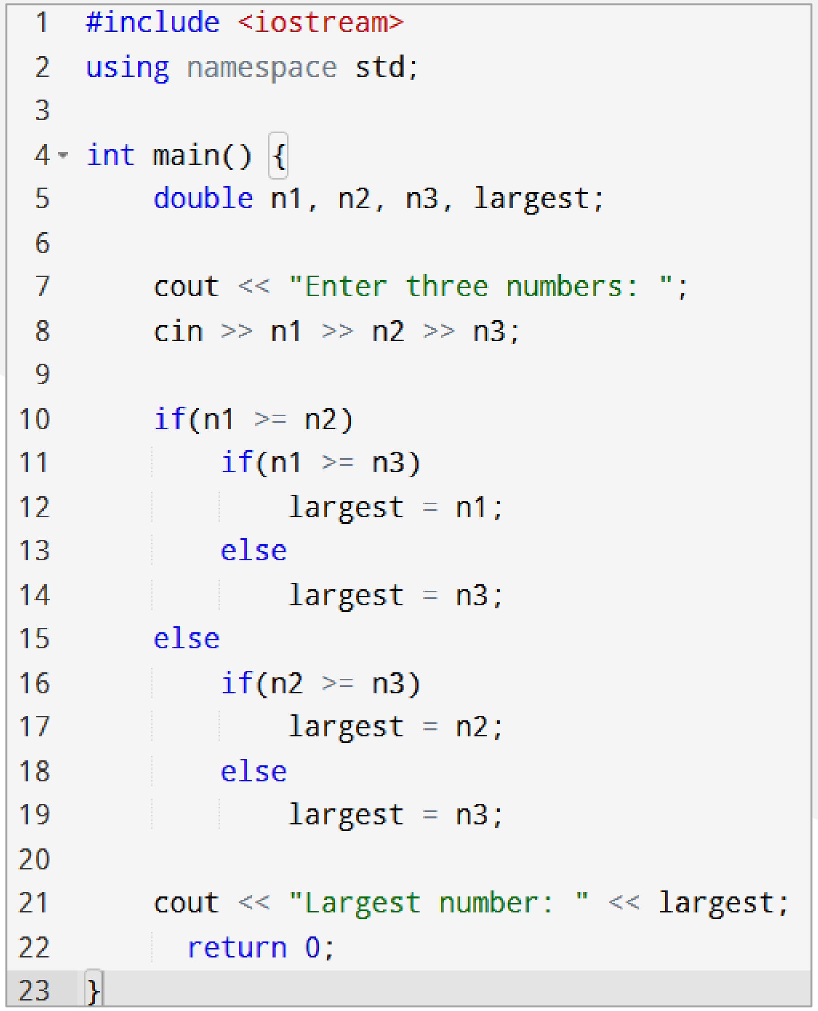
Nested If-else statement in C++

- **Flowchart of Nested IF-ELSE statement:**

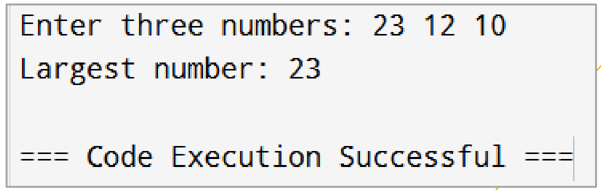


Nested If-else statement in C++

- **Example: Find the Largest Number Among Three Numbers**



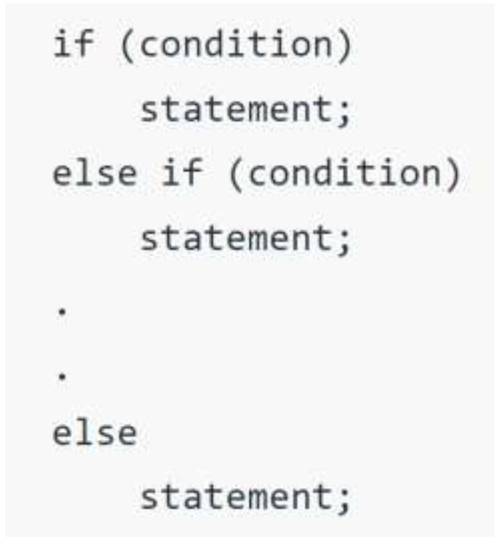
```cpp
#include <iostream>
using namespace std;

int main() {
    double n1, n2, n3, largest;

    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;

    if(n1 >= n2)
        if(n1 >= n3)
            largest = n1;
        else
            largest = n3;
    else
        if(n2 >= n3)
            largest = n2;
        else
            largest = n3;

    cout << "Largest number: " << largest;
     return 0;
}
```



```
Enter three numbers: 23 12 10
Largest number: 23

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

Else-If Ladder Statement

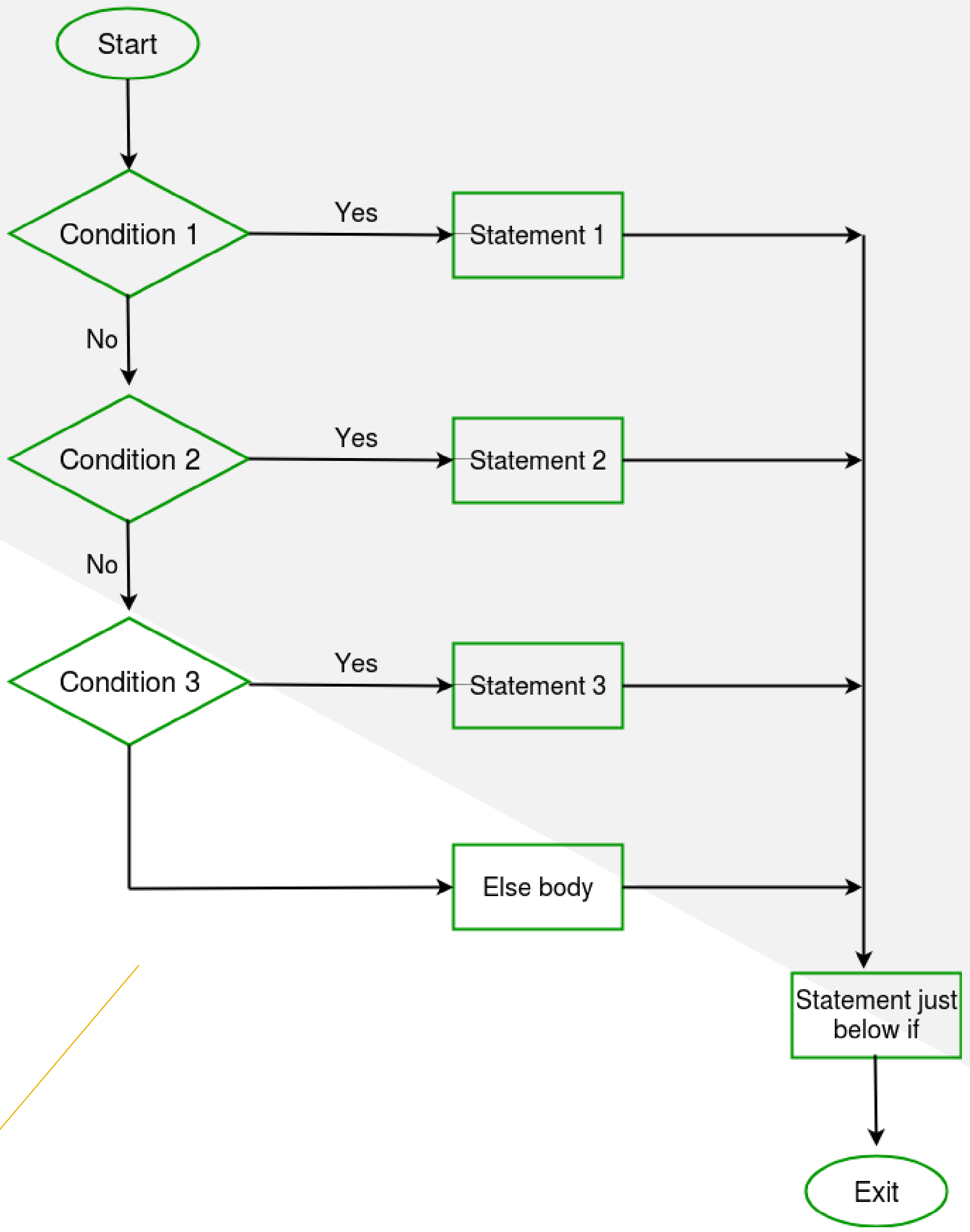
- The **else if statement** is an extension of the "if else" conditional branching statement. When the expression in the "if" condition is "false" another "if else" construct is used to execute a set statements based on an expression.
- This control structure statement also known as **else if ladder** statement.
- **Syntax:**



```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```

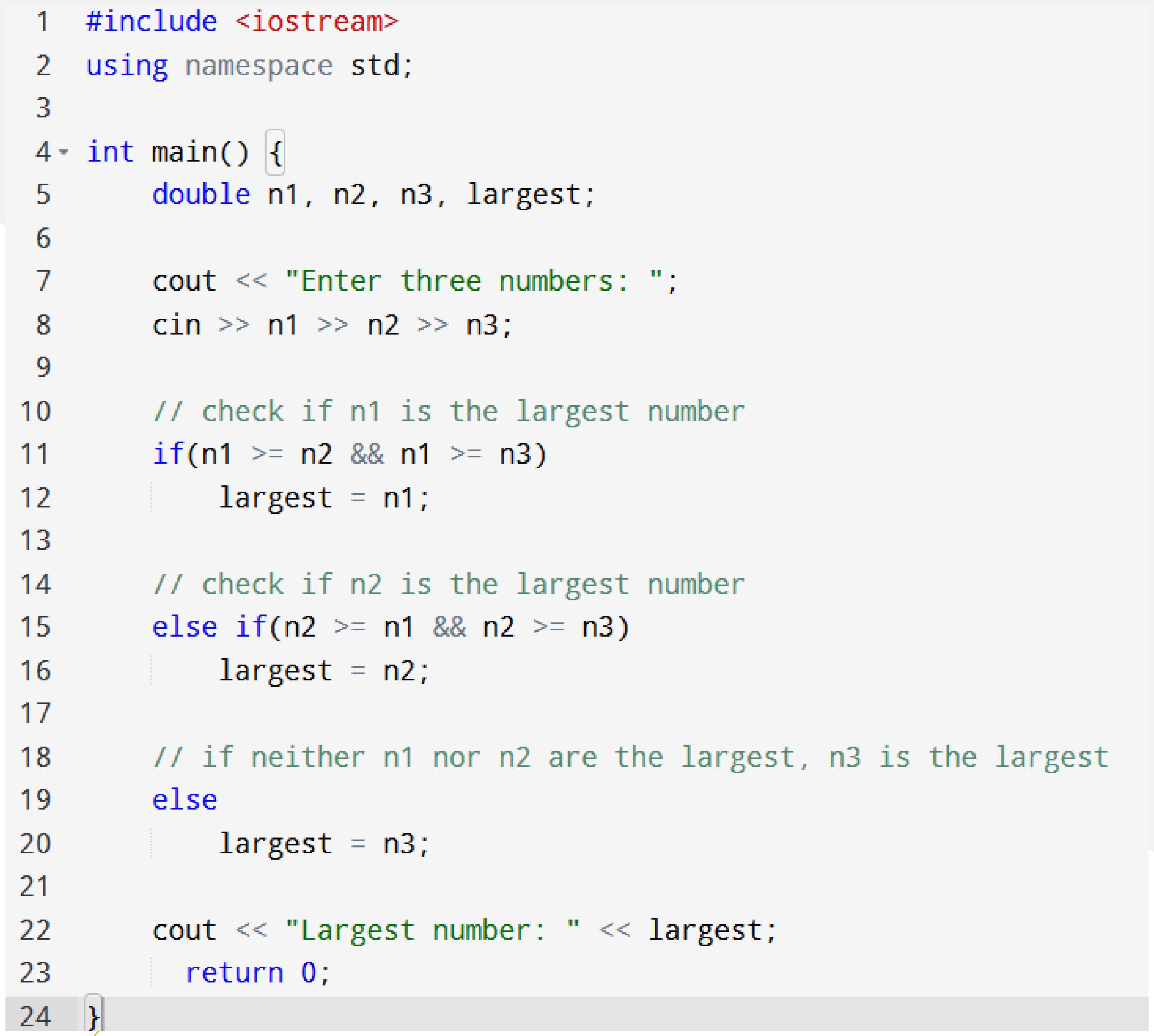
Else-If Ladder Statement

- **Flowchart:**



Else-If Ladder Statement

- **Example: Find the Largest Number Among Three Numbers**



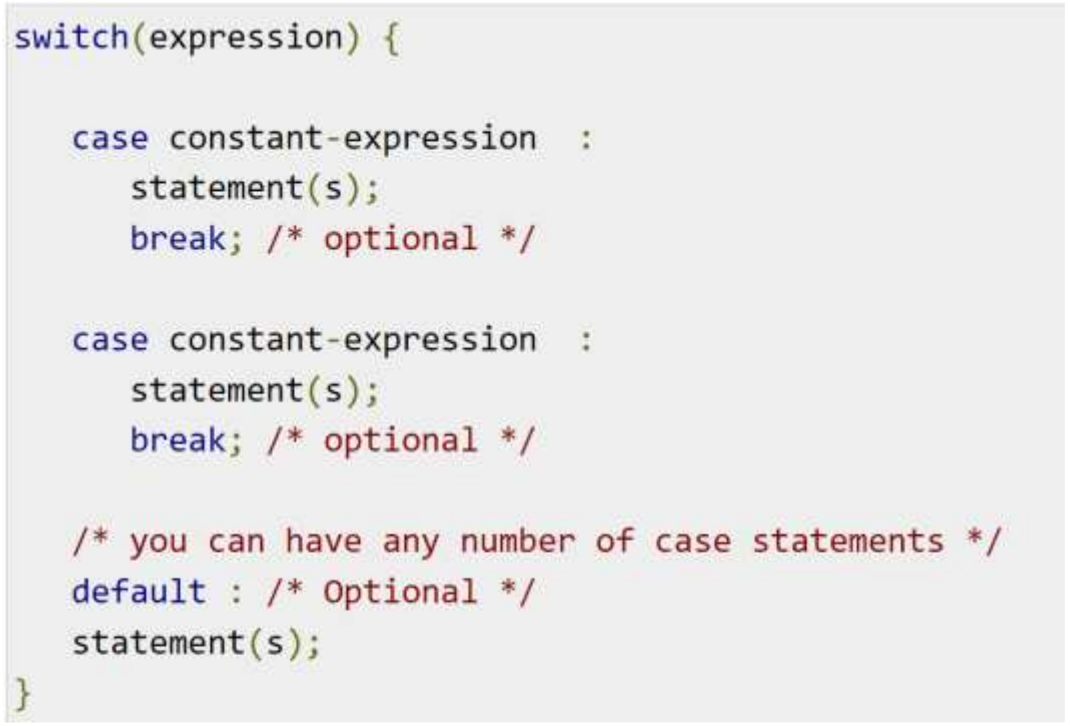
```cpp
#include <iostream>
using namespace std;

int main() {
    double n1, n2, n3, largest;

    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;

    // check if n1 is the largest number
    if(n1 >= n2 && n1 >= n3)
        largest = n1;

    // check if n2 is the largest number
    else if(n2 >= n1 && n2 >= n3)
        largest = n2;

    // if neither n1 nor n2 are the largest, n3 is the largest
    else
        largest = n3;

    cout << "Largest number: " << largest;
      return 0;
}
```

Switch Statement

- The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possible values of a single variable called switch variable.
- Here, We can define various statements in the multiple cases for the different values of a single variable.
- Thus, 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 Statement

- **Syntax:**
- The syntax for a **switch** statement in C programming language is as follows –



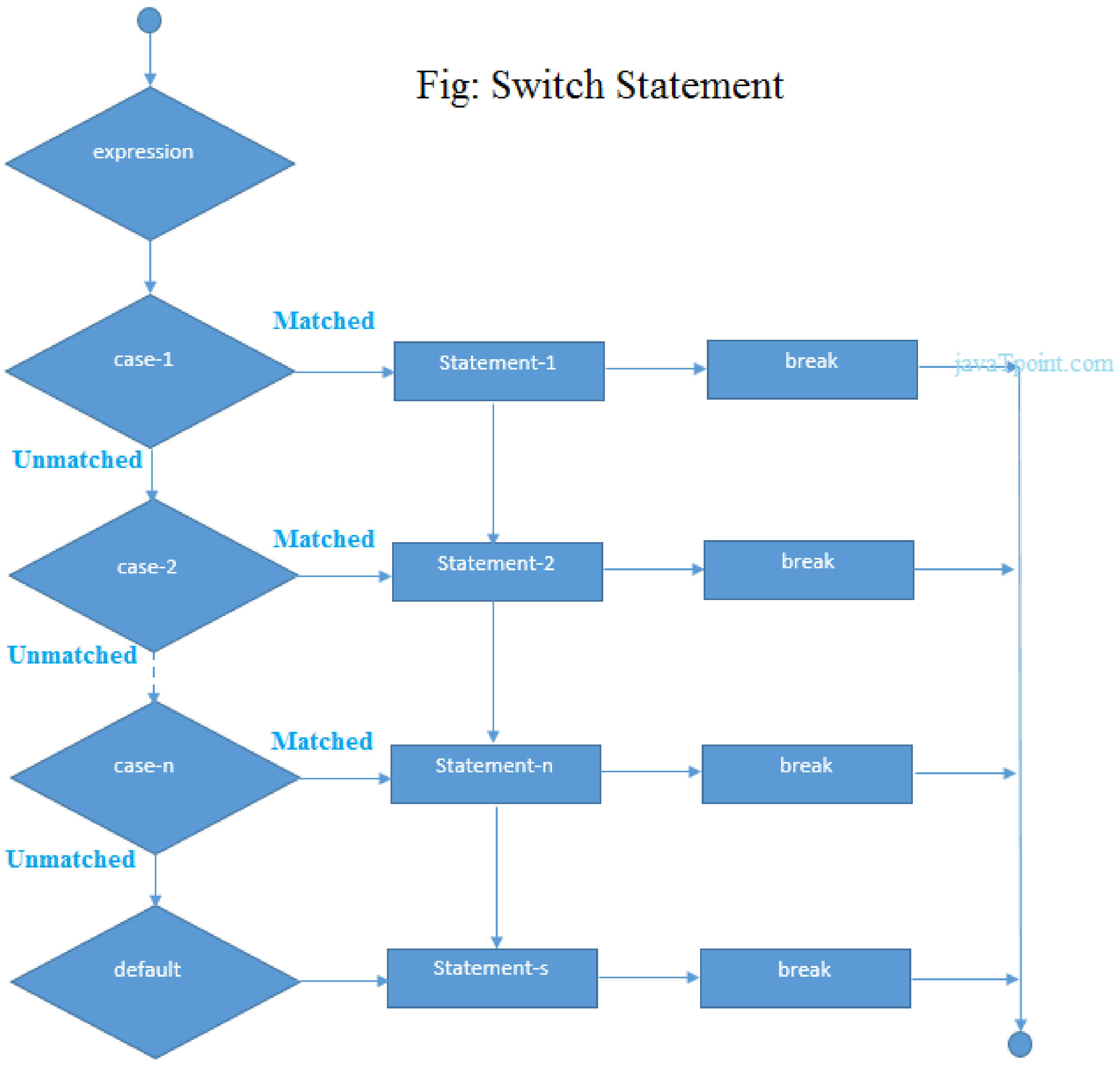
```
switch(expression) {

   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);
}
```

Switch Statement

- The following rules apply to a **switch** statement −
  - The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to 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.

Switch Statement

- **Flow Diagram:**



Switch Statement

- **Example:**

```
int main()
{
  int day;
  cout<<"Enter the day no (1-7):";
  cin>>day;
  switch(day)
  {
    case 1:
      cout<<"Saturday";
    break;
    case 2:
      cout<<"Sunday";
    break;
    case 3:
      cout<<"Monday";
    break;
    case 4:
      cout<<"Tuesday";
    break;
    case 5:
      cout<<"Wednesday";
    break;
    case 6:
      cout<<"Thursday";
    break;
    case 7:
      cout<<"Friday";
    break;
    default:
     cout<<"Invalid input!";
    break;
   }
return 0;
}
```

Output:

```
Enter the day no (1-7): 6
Thursday
```

Jump Statement

- These statements are used in C or C++ for the unconditional flow of control throughout the functions in a program.
- They support four types of jump statements:
  - **Break**
  - **Continue**
  - **Goto**
  - **Return**

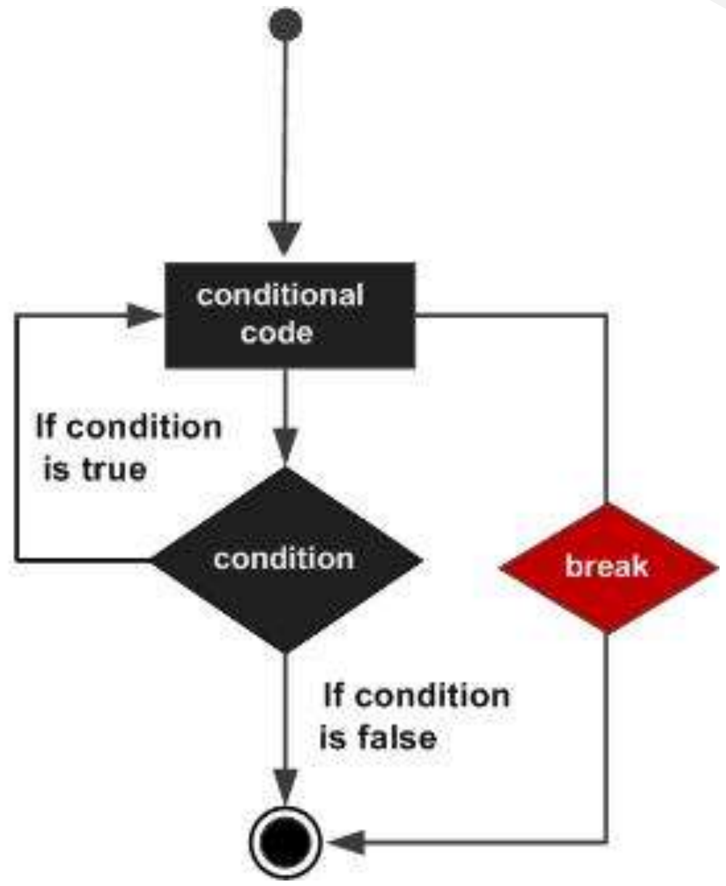
Break Statement

- The **break** statement in C programming has the following two usages –
  - This loop control statement is used to terminate the loop. 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.
- If you are using nested loops, the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.
- **Syntax:**
- The syntax for a **break** statement in C is as follows –

```
break;
```

Break Statement

- **Flow Diagram:**



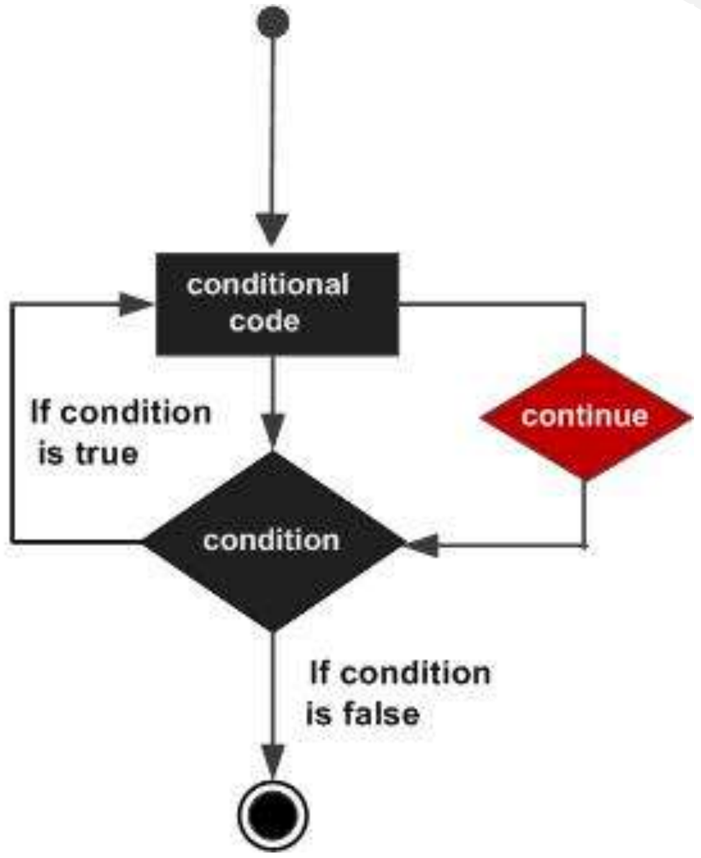
Continue Statement

- 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.
- **Syntax:**
- The syntax for a **continue** statement in C is as follows –

```
continue;
```

Continue Statement

- **Flow Diagram:**



Goto Statement

- The **goto** statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statement can be used to repeat some part of the code for a particular condition.
- It can also be used to break the multiple loops which can't be done by using a single break statement.
- However, using goto is avoided these days since it makes the program less readable and complicated.

Syntax:

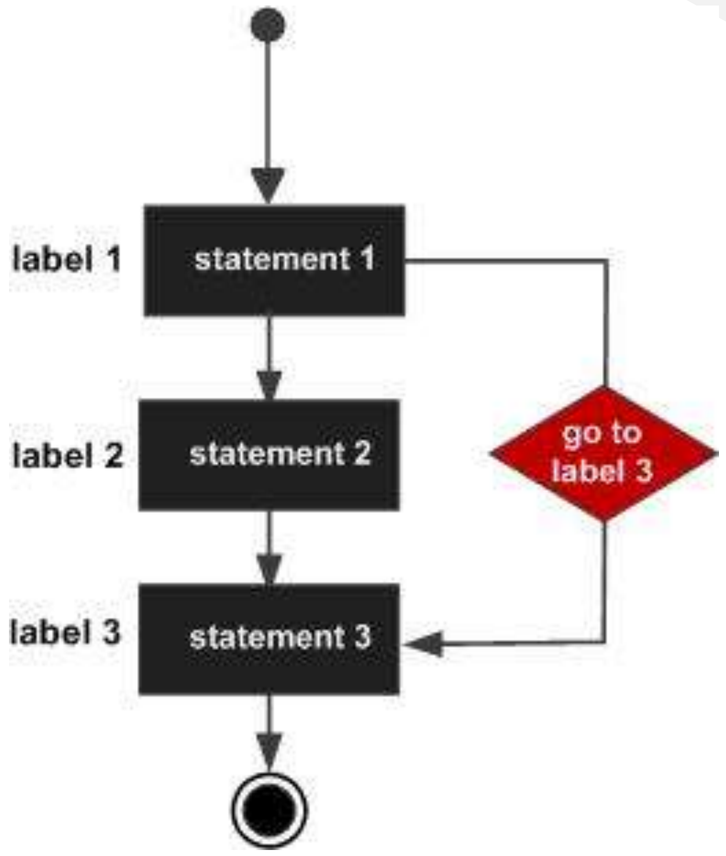
- The syntax for a **goto** statement in C is as follows –

```
goto label;
..
.
label: statement;
```

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.

Goto Statement

- Flow Diagram:



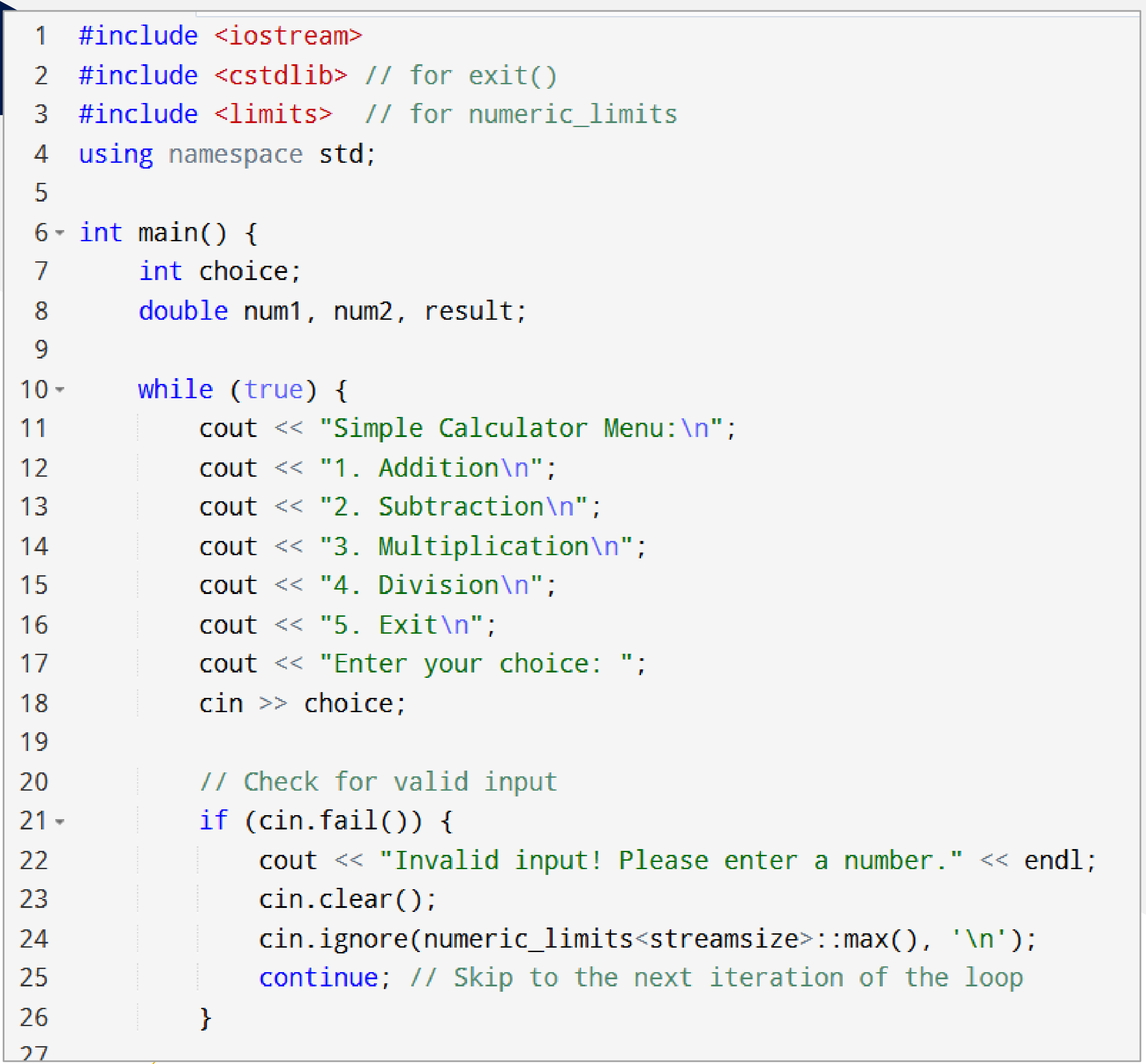
Return Statement

- The **return** in C or 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 return 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 is must be returned.
- **Syntax:**
- The syntax for a **return** statement in C is as follows –

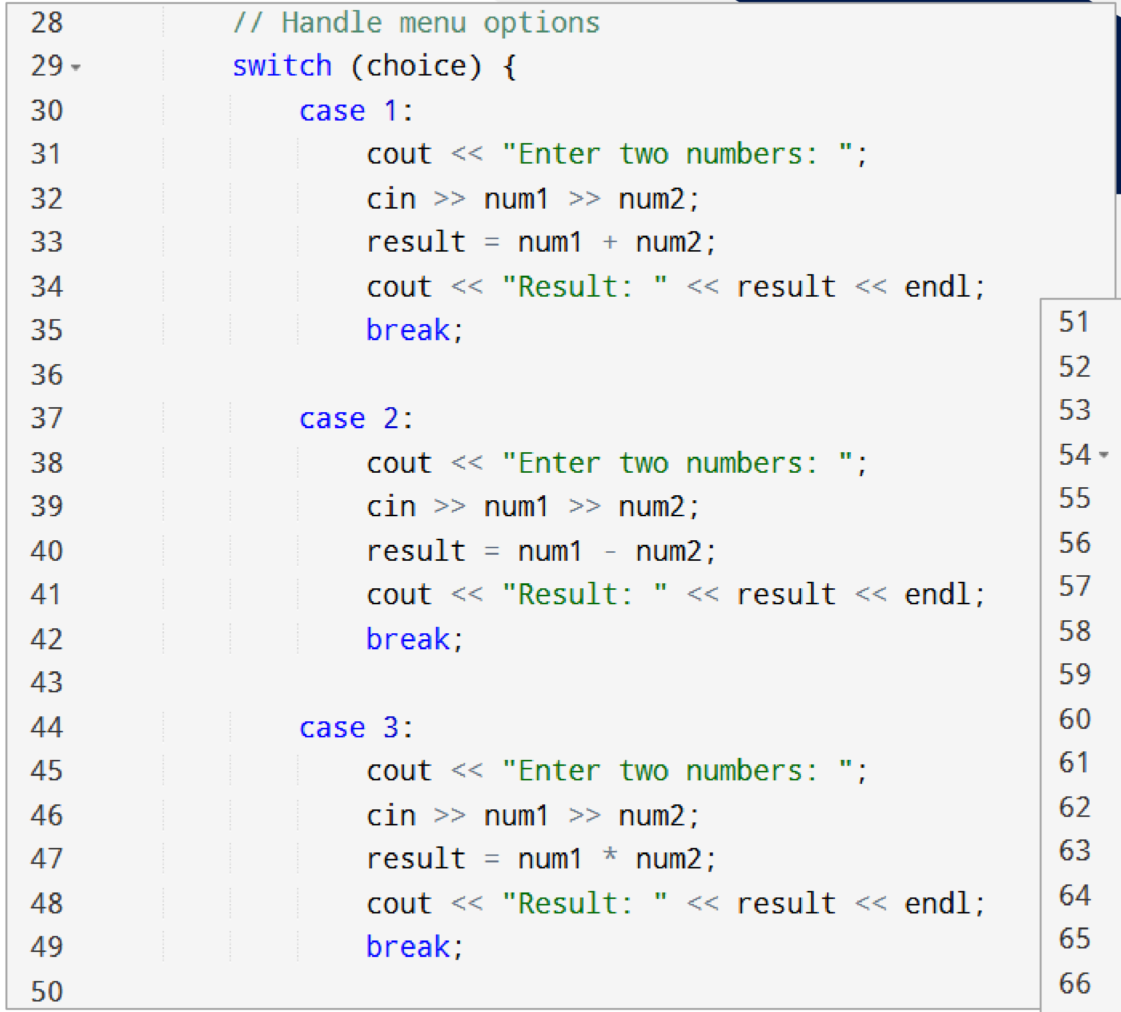
```
return [expression];
```

Example

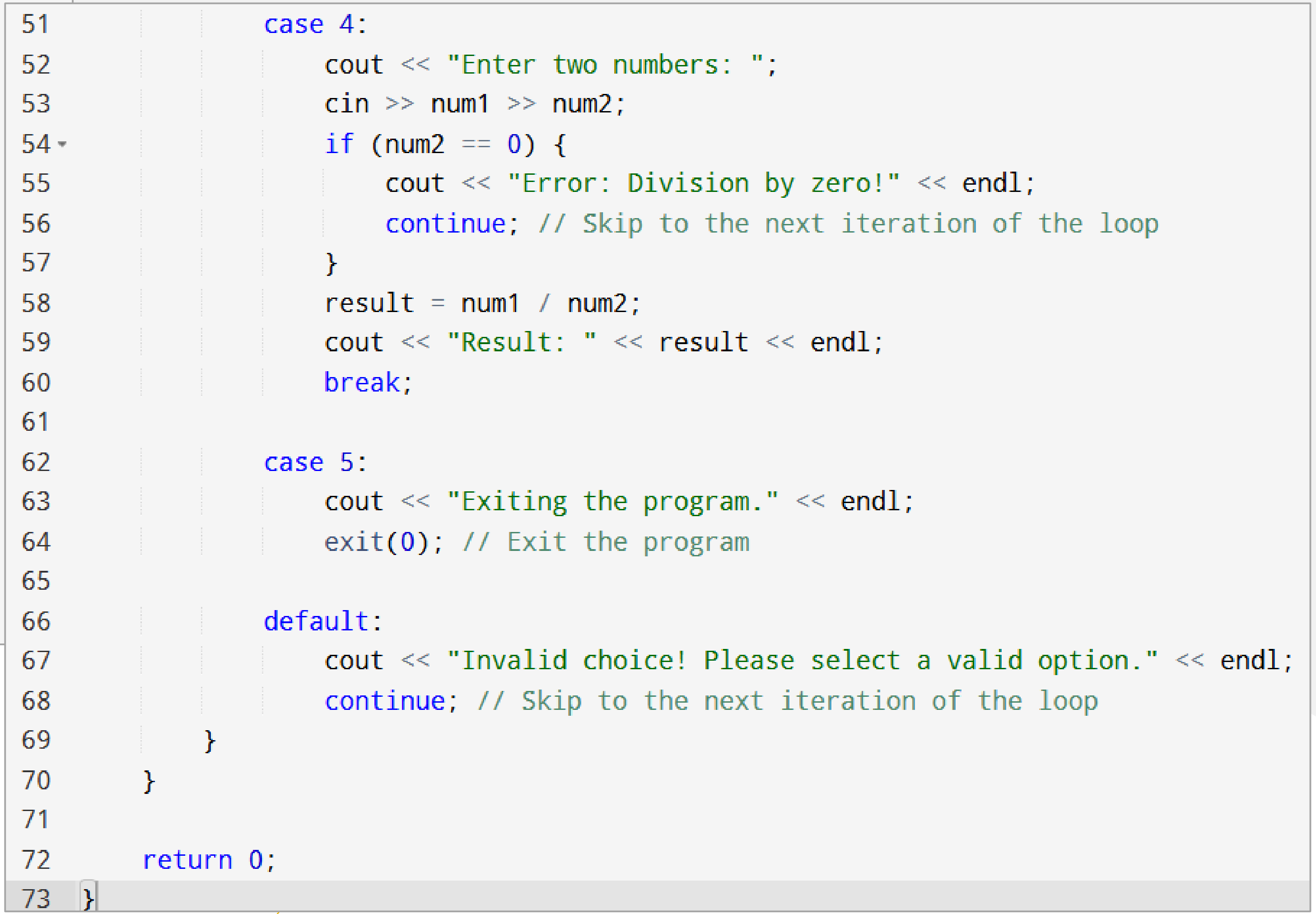
- C++ program that demonstrates the use of break, continue, goto, exit, and return statements.
- This program is a basic menu-driven calculator that performs addition, subtraction, multiplication, and division based on user input.



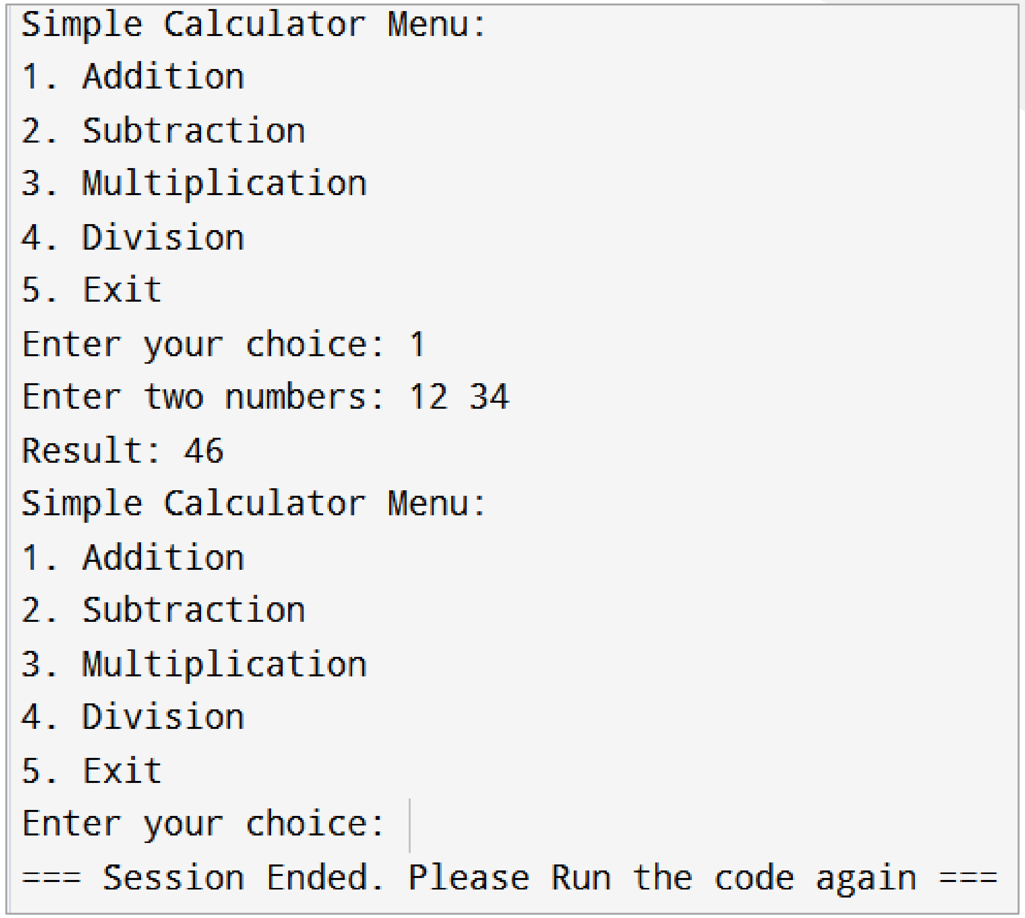
```cpp
#include <iostream>
#include <cstdlib> // for exit()
#include <limits>  // for numeric_limits
using namespace std;

int main() {
    int choice;
    double num1, num2, result;

    while (true) {
        cout << "Simple Calculator Menu:\n";
        cout << "1. Addition\n";
        cout << "2. Subtraction\n";
        cout << "3. Multiplication\n";
        cout << "4. Division\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        // Check for valid input
        if (cin.fail()) {
            cout << "Invalid input! Please enter a number." << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue; // Skip to the next iteration of the loop
        }
```



```cpp
        // Handle menu options
        switch (choice) {
            case 1:
                cout << "Enter two numbers: ";
                cin >> num1 >> num2;
                result = num1 + num2;
                cout << "Result: " << result << endl;
                break;

            case 2:
                cout << "Enter two numbers: ";
                cin >> num1 >> num2;
                result = num1 - num2;
                cout << "Result: " << result << endl;
                break;

            case 3:
                cout << "Enter two numbers: ";
                cin >> num1 >> num2;
                result = num1 * num2;
                cout << "Result: " << result << endl;
                break;

```



```cpp
            case 4:
                cout << "Enter two numbers: ";
                cin >> num1 >> num2;
                if (num2 == 0) {
                    cout << "Error: Division by zero!" << endl;
                    continue; // Skip to the next iteration of the loop
                }
                result = num1 / num2;
                cout << "Result: " << result << endl;
                break;

            case 5:
                cout << "Exiting the program." << endl;
                exit(0); // Exit the program

            default:
                cout << "Invalid choice! Please select a valid option." << endl;
                continue; // Skip to the next iteration of the loop
        }
    }

    return 0;
}
```



```
Simple Calculator Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Enter your choice: 1
Enter two numbers: 12 34
Result: 46
Simple Calculator Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Enter your choice: 
=== Session Ended. Please Run the code again ===
```

Lecture 5

C++ Control Structures

?

THE END