Lecture 07

M M Imran

Ternary Operator

- A ternary operator is an expression with one of two possible values.
- A ternary operator has a condition that is tested:
  - If the condition is true, the operator returns a true-expression.
  - If the condition is false, the operator returns a false-expression.
- The ternary operator has syntax:

  <condition> ? <true-expression> : <false-expression>
- Although not required, parentheses should be placed around the ternary operator to separate its logic from surrounding code.
- This is called William's compound ternary operator.

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

int main() {
  double marks;

  // take input from users
  cout << "Enter your marks: ";
  cin >> marks;

  // ternary operator checks if
  // marks is greater than 40
  string result = (marks >= 40) ? "passed" : "failed";

  cout << "You " << result << " the exam.";

  return 0;
}
```

Output 1

```
Enter your marks: 80
You passed the exam.
```

Output 2

```
Enter your marks: 39.5
You failed the exam.
```

When to use a Ternary Operator?

- We should only use the ternary operator if the resulting statement is short.
- When the use of the ternary operator makes our code more readable and clean.

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

int main() {
  // Create a variable
  int number = -4;

  if (number > 0)
    cout << "Positive Number";
  else
    cout << "Negative Number!";

  return 0;
}
```

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

int main() {
  int number = -4;
  string result;

  // Using ternary operator
  result = (number > 0) ? "Positive Number!" : "Negative Number!";

  cout << result << endl;

  return 0;
}
```

Output

```
Negative Number!
```

Nested Ternary Operators

- It is also possible to use one ternary operator inside another ternary operator. It is called the nested ternary operator in C++.
- It is not recommended to use nested ternary operators. This is because it makes our code more complex.

```
(number == 0) ? "Zero" : ((number > 0) ? "Positive" : "Negative");
```

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

int main() {
  int number = 0;
  string result;

  // nested ternary operator to find whether
  // number is positive, negative, or zero
  result = (number == 0) ? "Zero" : ((number > 0) ? "Positive" : "Negative");

  cout << "Number is " << result;

  return 0;
}
```

Output

```
Number is Zero
```

Switch Statement

- The switch statement is a limited alternative to the if statement.
- A switch statement has an expression that is tested against one or more cases.
- The data type of the expression may be:
  - char
  - short
  - int
  - long
  - long long
  - Enumerated type (to be discussed later)

Switch Statement

- The data type of the expression cannot be:
  - float
  - double
  - long double
  - string
- The data type of the expression should not be:
  - bool
- A switch statement may be easier to read when there are more than three conditions to test.
- A switch statement may have a default case that will run if all other case tests are false.

Switch Statement Syntax

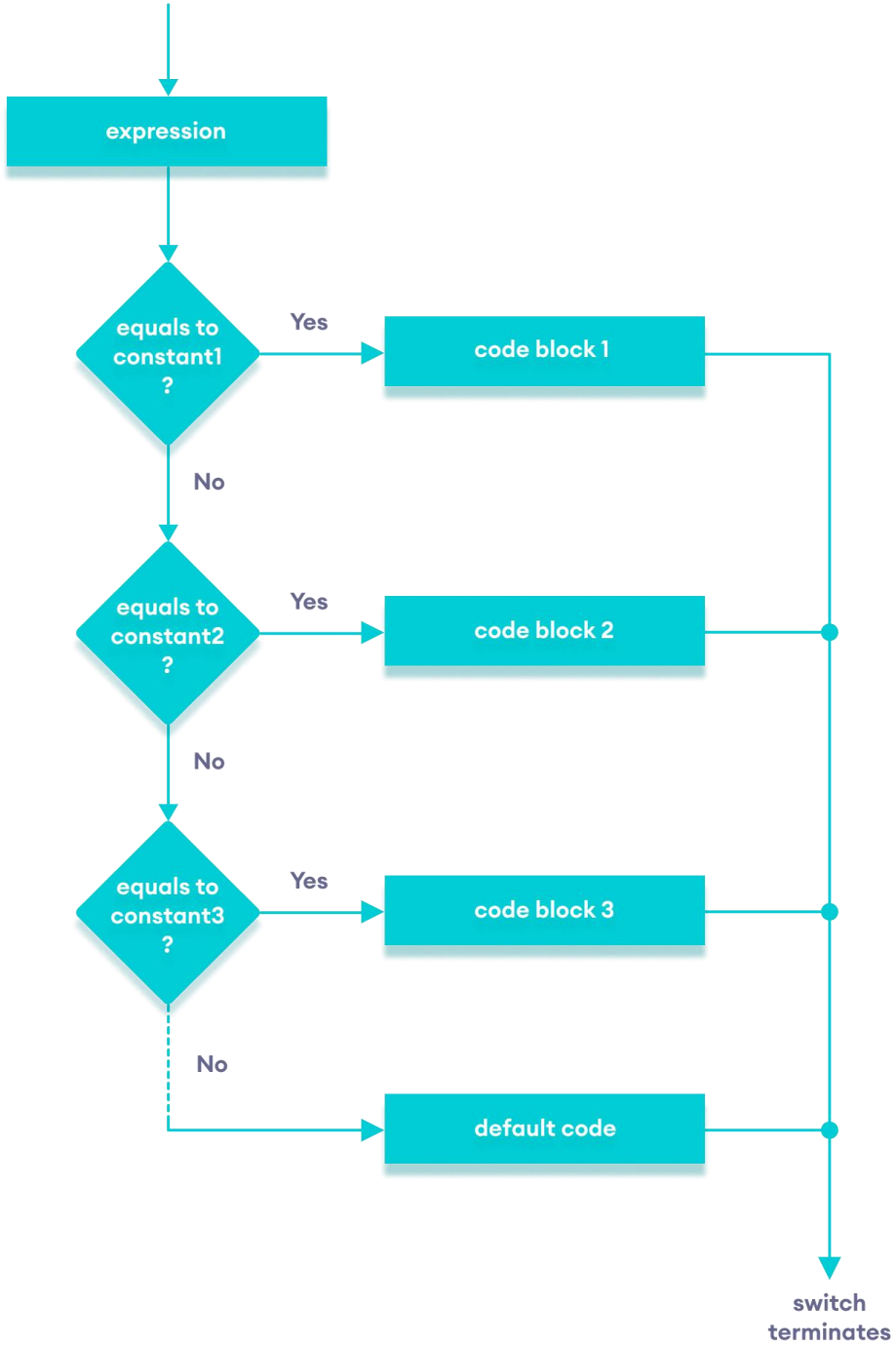
```
switch (expression)  {
    case constant1:
        // code to be executed if
        // expression is equal to constant1;
        break;

    case constant2:
        // code to be executed if
        // expression is equal to constant2;
        break;
        .
        .
        .
    default:
        // code to be executed if
        // expression doesn't match any constant
}
```

The **case** part can be repeated as many times as is needed.

Switch Statement

- Curly braces are not used to enclose each case.
- The break statement halts the switch statement and the program continues after the switch statement.
- In most cases, the break statement is required. If a break statement is not there and there is another case or default block, execution continues with that block.
- We can do the same thing with the if...else..if ladder. However, the syntax of the **switch** statement is cleaner and much easier to read and write.


expression
equals to constant1 ?
Yes
code block 1
No
equals to constant2 ?
Yes
code block 2
No
equals to constant3 ?
Yes
code block 3
No
default code
switch terminates

```
char oper;
float num1, num2;
cout << "Enter an operator (+, -, *, /): ";
cin >> oper;
cout << "Enter two numbers: " << endl;
cin >> num1 >> num2;

switch (oper) {
    case '+':
        cout << num1 << " + " << num2 << " = " << num1 + num2;
        break;
    case '-':
        cout << num1 << " - " << num2 << " = " << num1 - num2;
        break;
    case '*':
        cout << num1 << " * " << num2 << " = " << num1 * num2;
        break;
    case '/':
        cout << num1 << " / " << num2 << " = " << num1 / num2;
        break;
    default:
        // operator is doesn't match any case constant (+, -, *, /)
        cout << "Error! The operator is not correct";
        break;
}
```

Output 1

```
Enter an operator (+, -, *, /): +
Enter two numbers:
2.3
4.5
2.3 + 4.5 = 6.8
```

Output 2

```
Enter an operator (+, -, *, /): -
Enter two numbers:
2.3
4.5
2.3 - 4.5 = -2.2
```

Output 3

```
Enter an operator (+, -, *, /): *
Enter two numbers:
2.3
4.5
2.3 * 4.5 = 10.35
```

Switch without break

```cpp
#include <iostream>

using namespace std;

int main(){
    int i=2;

    switch(i) {
        case 1: cout<<"Case1 "<<endl;
        case 2: cout<<"Case2 "<<endl;
        case 3: cout<<"Case3 "<<endl;
        case 4: cout<<"Case4 "<<endl;
        default: cout<<"Default "<<endl;
    }

    return 0;
}
```

```
Case2
Case3
Case4
Default
```

Iteration / Loop

- In computer programming, loops are used to repeat a block of code.
- For example, let's say we want to show a message 100 times. Then instead of writing the print statement 100 times, we can use a loop.
- That was just a simple example; we can achieve much more efficiency and sophistication in our programs by making effective use of loops.
- There are 3 types of loops in C++.
  - for loop
  - while loop
  - do...while loop

for Loop

- A **for** statement is an iterative (looping) statement with a condition and a block of code.
- A **for** statement is a pre-test loop: the condition is tested before the block of code executes.
- A **for** statement does definite iteration: the number of times it loops is known ahead of time.
- A **for** statement continues to execute the block of code while the condition is true.

for Loop Syntax

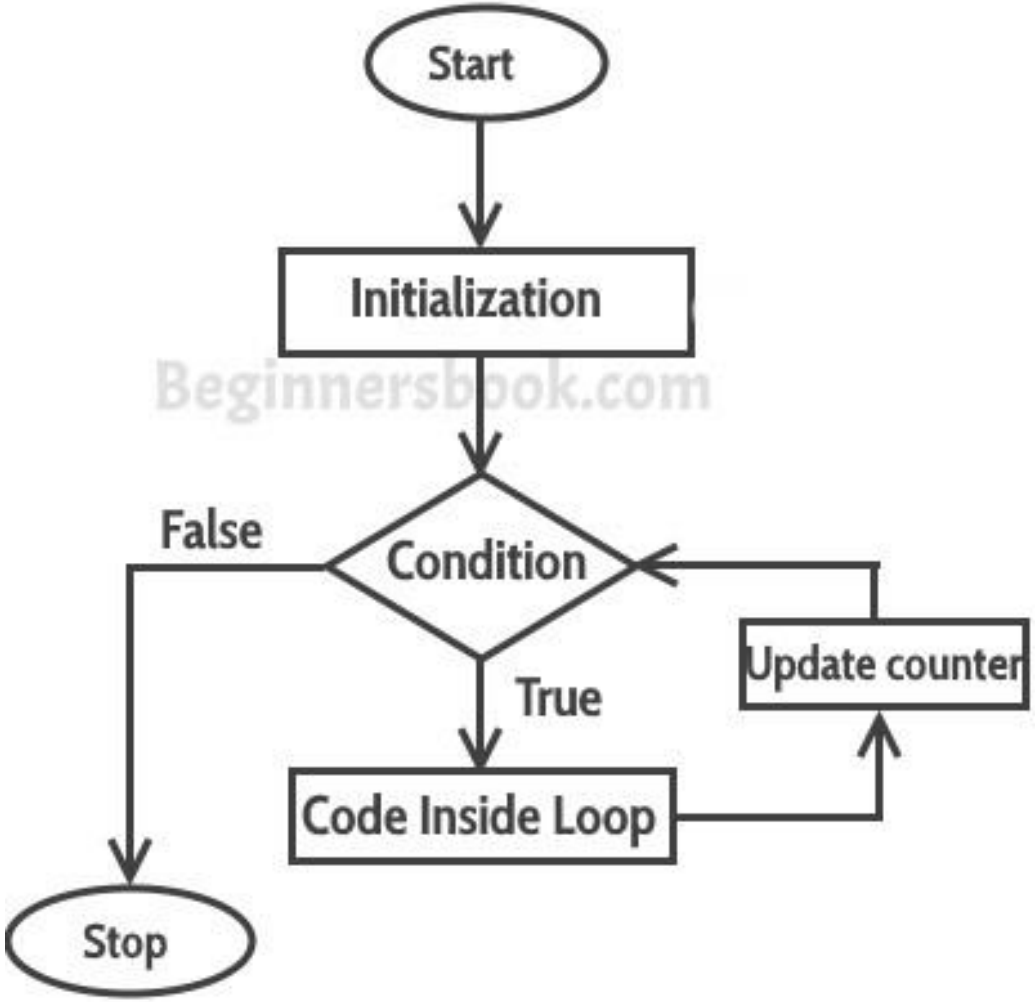
for (<initialization>; <condition>; <update>) {

    <block>

}

- **<initialization>** - initializes variables and is executed only once
- **<condition>** - if true, the body of for loop is executed, and if false, the for loop is terminated
- **<update>** - updates the value of initialized variables and again checks the condition

for Loop


Start
Initialization
Beginnersbook.com
False
Condition
True
Update counter
Code Inside Loop
Stop

for Loop

- A loop index is often used to control how many times a for statement loops. An integer variable is used for the loop index.
- Excellent programmer tip: only modify the loop index in the <update> part of the for statement; do not modify it in the <block>.

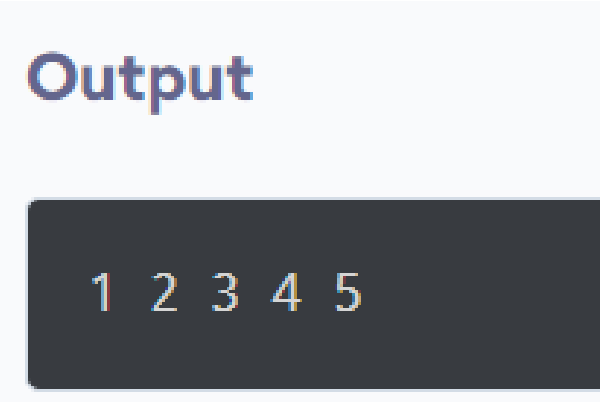
Printing Numbers From 1 to 5

```cpp
#include <iostream>

using namespace std;

int main() {
    for (int i = 1; i <= 5; ++i) {
        cout << i << " ";
    }

    return 0;
}
```



Explanation

Iteration	Variable	i <= 5	Action
1st	i = 1	true	1 is printed. i is increased to 2.
2nd	i = 2	true	2 is printed. i is increased to 3.
3rd	i = 3	true	3 is printed. i is increased to 4.
4th	i = 4	true	4 is printed. i is increased to 5.
5th	i = 5	true	5 is printed. i is increased to 6.
6th	i = 6	false	The loop is terminated

Find the sum of first n Natural Numbers

```cpp
#include <iostream>

using namespace std;

int main() {
    int num, sum;
    sum = 0;

    cout << "Enter a positive integer: ";
    cin >> num;

    for (int i = 1; i <= num; ++i) {
        sum += i;
    }

    cout << "Sum = " << sum << endl;

    return 0;
}
```

Output

```
Enter a positive integer: 10
Sum = 55
```

Infinite for Loop

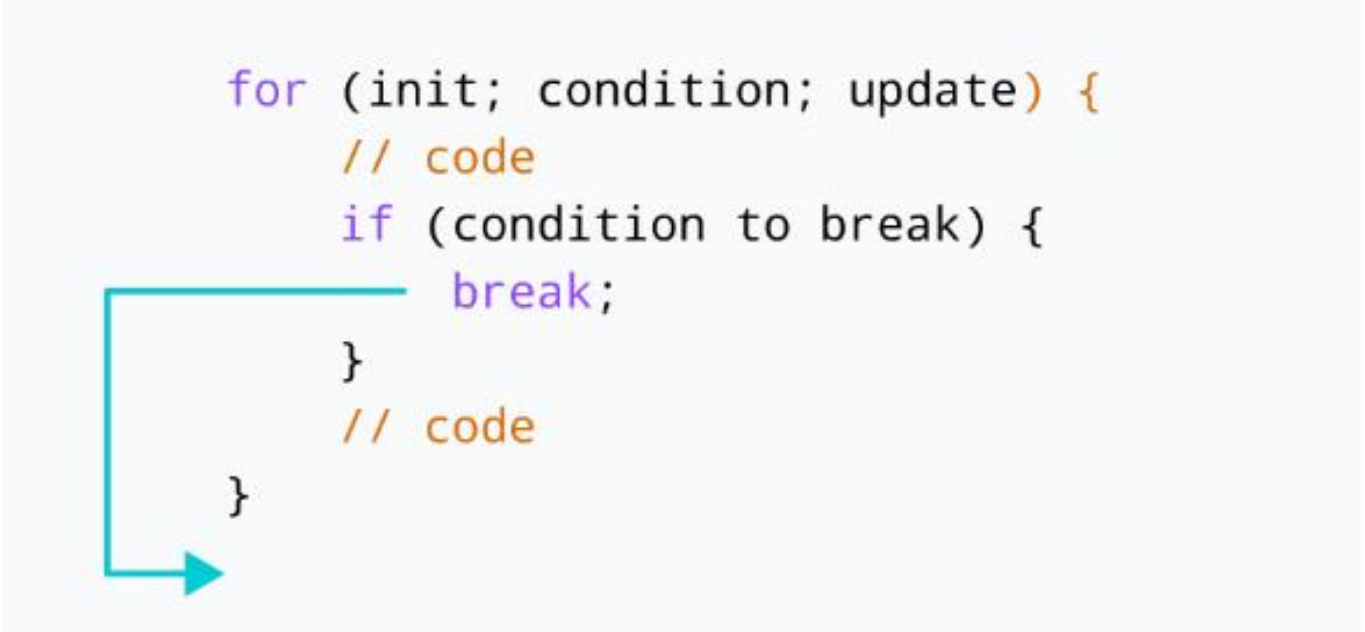
- A loop is said to be infinite when it executes repeatedly and never stops.
- This usually happens by mistake.
- When you set the condition in for loop in such a way that it never return false, it becomes infinite loop.

```
// infinite for loop
for(int i = 1; i > 0; i++) {
    // block of code
}
```

In the above program, the condition is always true which will then run the code for infinite times.

break Statement

- The break statement terminates the loop when it is encountered.
- It breaks the current flow of the program at the given condition.
- In case of inner loop, it breaks only inner loop.



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

int main() {
    for (int i = 1; i <= 5; i++) {
        // break condition
        if (i == 3) {
            break;
        }
        cout << i << endl;
    }

return 0;
}
```

Output

```
1
2
```

break With Nested Loop

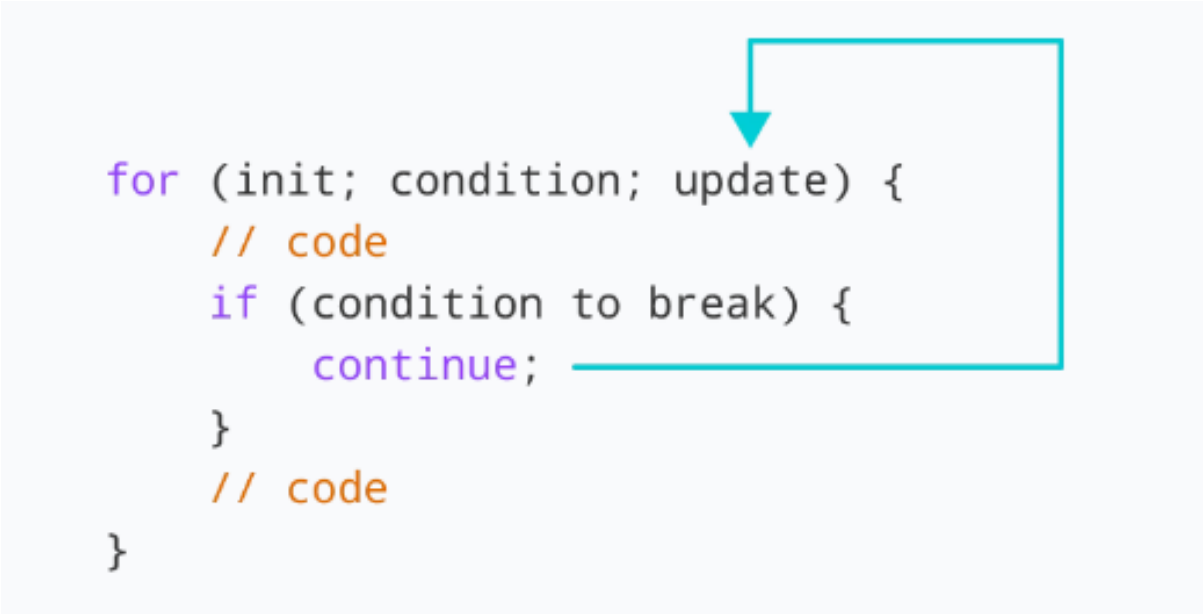
```
// first loop
for (int i = 1; i <= 3; i++) {
    // second loop
    for (int j = 1; j <= 3; j++) {
        if (i == 2) {
            break;
        }
        cout << "i = " << i << ", j = " << j << endl;
    }
}
```

Output

```
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 3, j = 1
i = 3, j = 2
i = 3, j = 3
```

continue Statement

- The **continue** statement is used to skip the current iteration of the loop and the control of the program goes to the next iteration.
- Whenever a continue statement is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements inside loop's body for the current iteration.



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

int main() {
    for (int i = 1; i <= 5; i++) {
        // condition to continue
        if (i == 3) {
            continue;
        }

        cout << i << endl;
    }

    return 0;
}
```

Output

```
1
2
4
5
```

continue With Nested Loop

```
// first loop
for (int i = 1; i <= 3; i++) {
    // second loop
    for (int j = 1; j <= 3; j++) {
        if (j == 2) {
            continue;
        }
        cout << "i = " << i << ", j = " << j << endl;
    }
}
```

Output

```
i = 1, j = 1
i = 1, j = 3
i = 2, j = 1
i = 2, j = 3
i = 3, j = 1
i = 3, j = 3
```

Let’s ***break*** the class