1. Introduction

In C++, `break` and `continue` are control statements used inside loops to change the normal execution of a loop.

Normally, a loop executes all its statements repeatedly until its condition becomes false. However, sometimes we need to:

- Stop a loop immediately, even if its condition is still true.
- Skip the remaining statements of the current iteration and move to the next iteration.

C++ provides two statements for these purposes:

1. `break` – Terminates the loop immediately.
2. `continue` – Skips the remaining statements of the current iteration.

2. The break Statement

The `break` statement is used to terminate a loop immediately.

When the `break` statement executes:

- The loop stops immediately.
- The remaining iterations are not executed.
- Program execution continues with the first statement after the loop.

Syntax

```
break;
```

Example 1: Using break in a for loop

Task: Write a C++ program to print numbers from 1 to 10. Stop the loop when the number becomes 5.

Program

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

int main()
{
    for (int i = 1; i <= 10; i++)
    {
        if (i == 5)
        {
            break;
        }

        cout << i << endl;
    }
```

```
    return 0;
}
```

Output

```
1
2
3
4
```

Explanation

1. The loop starts with `i = 1`.
2. The `if` statement checks whether `i` is equal to 5.
3. When `i` becomes 5, the `break` statement executes.
4. The loop terminates immediately.
5. Numbers 5 to 10 are not printed.

Dry Run

Iteration	Value of i	Condition i == 5	Action
1	1	False	Print 1
2	2	False	Print 2
3	3	False	Print 3
4	4	False	Print 4
5	5	True	Exit loop

Example 2: Using break in a while loop

Task: Write a program to print numbers from 1 to 10 using a while loop. Stop when the number becomes 6.

Program

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

int main()
{
    int i = 1;

    while (i <= 10)
    {
        if (i == 6)
        {
            break;
```

```
        }

        cout << i << endl;
        i++;
    }

    return 0;
}
```

Output

```
1
2
3
4
5
```

Explanation

The loop continues while `i` is less than or equal to 10. When `i` becomes 6, the `break` statement terminates the loop.

3. The continue Statement

The `continue` statement is used to skip the remaining statements of the current iteration and move to the next iteration.

When the `continue` statement executes:

- The remaining statements in the current iteration are skipped.
- The loop moves to its next iteration.
- The loop does not terminate.

Syntax

```
continue;
```

Example 1: Using continue in a for loop

Task: Write a C++ program to print numbers from 1 to 10, but skip number 5.

Program

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

int main()
{
    for (int i = 1; i <= 10; i++)
```

```
    {
        if (i == 5)
        {
            continue;
        }

        cout << i << endl;
    }

    return 0;
}
```

Output

```
1
2
3
4
6
7
8
9
10
```

Explanation

1. The loop starts with `i = 1`.
2. The `if` statement checks whether `i` is equal to 5.
3. When `i` becomes 5, the `continue` statement executes.
4. The `cout` statement is skipped for that iteration.
5. The loop continues with the next value, which is 6.

Dry Run

Iteration	Value of i	Condition i == 5	Action
1	1	False	Print 1
2	2	False	Print 2
3	3	False	Print 3
4	4	False	Print 4
5	5	True	Skip printing
6	6	False	Print 6

Example 2: Using continue in a while loop

Task: Write a program to print odd numbers from 1 to 10 using a while loop.

Program

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

int main()
{
    int i = 0;

    while (i < 10)
    {
        i++;

        if (i % 2 == 0)
        {
            continue;
        }

        cout << i << endl;
    }

    return 0;
}
```

Output

```
1
3
5
7
9
```

Explanation

1. The variable `i` starts from 0.
2. The value of `i` is increased by 1.
3. The condition `i % 2 == 0` checks whether the number is even.
4. If the number is even, `continue` skips the printing statement.
5. If the number is odd, it is printed.
6. The loop continues until `i` becomes 10.

Important: In a `while` loop, make sure the loop-control variable is updated before executing `continue` when necessary. Otherwise, the loop may become an infinite loop.

4. Important Concept: `break` in Nested Loops

A `break` inside an inner loop terminates only the inner loop. The outer loop continues normally.

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

int main()
{
    for (int i = 1; i <= 3; i++)
    {
        for (int j = 1; j <= 4; j++)
        {
            if (j == 3)
            {
                break;
            }

            cout << j << " ";
        }

        cout << endl;
    }

    return 0;
}
```

Output:

```
1 2
1 2
1 2
```

The inner loop stops whenever `j` reaches 3. The outer loop still creates all three rows.

5. Difference Between break and continue

break	**continue**
Terminates the loop immediately.	Skips the current iteration.
No further iterations are executed.	Remaining iterations continue normally.
Control moves outside the loop.	Control moves to the next iteration.
Used to stop a loop.	Used to skip selected iterations.

5. Important Points to Remember

1. Both `break` and `continue` are used inside loops.
2. `break` terminates the nearest enclosing loop.
3. `continue` skips the remaining statements of the current iteration.
4. Both statements can be used with `for`, `while`, and `do-while` loops.
5. In nested loops, `break` and `continue` affect only the innermost loop in which they are used.

6. Practice Tasks

Tasks 1: Understand the Code

Predict the output of the following program.

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

int main()
{
    for (int i = 1; i <= 7; i++)
    {
        if (i == 4)
        {
            break;
        }

        cout << i << " ";
    }

    return 0;
```

Task 2: Modify the Code

Modify the following program to skip all even numbers.

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

int main()
{
    for (int i = 1; i <= 10; i++)
    {
        cout << i << " ";
    }

    return 0;
}
```

Task 3: Write a Program

Write a C++ program that:

1. Prints numbers from 1 to 20.
2. Skips multiples of 3 using `continue`.
3. Stops the loop when the number becomes 15 using `break`.

Task 4: Find and Fix the Error

Identify and correct the logical error in the following program.

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

int main()
{
    int i = 1;

    while (i <= 10)
    {
        if (i == 5)
        {
            continue;
        }

        cout << i << " ";
        i++;
    }

    return 0;
}
```

5. Practice Coding Exercises

Solve Coding Exercises 32–33 in the `Break and Continue` section. Write, run, and test each C++ program to verify the output.