C++ LESSON MODULE: LOOP CONTROL STATEMENTS – BREAK AND CONTINUE

INTRODUCTION

In C++, loop control statements allow changes to the normal flow of loop execution. During looping, you sometimes need to exit a loop prematurely or skip selected iterations based on specific conditions. The two primary control statements used for this are the break and continue statements.

BREAK STATEMENT

The break statement immediately terminates the smallest enclosing loop (for, while, or do-while) regardless of the test condition.
Once a break executes, the program continues from the first statement after the loop.

EXAMPLE 1: BREAK IN A FOR LOOP

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

int main() {
    for (int i = 0; i < 5; i++) {
        if (i == 3)
            break;
        cout << "i = " << i << endl;
    }
    cout << "Loop exited at i == 3";
    return 0;
}
```

OUTPUT: i = 0

i = 1

i = 2

Loop exited at i == 3

Explanation: When i equals 3, the break statement stops the loop execution entirely and control moves outside the loop.

EXAMPLE 2: BREAK IN A WHILE LOOP

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

int main() {
    int i = 1;
    while (i <= 5) {
        if (i == 4)
            break;
        cout << i << " ";
        i++;
    }
    return 0;
}
```

OUTPUT: 1 2 3

Explanation: When the counter i reaches 4, the break statement terminates the loop, even though the condition (i <= 5) is still true.

CONTINUE STATEMENT

The continue statement skips the remaining code in the current iteration and moves directly to the next iteration of the loop.

Unlike break, it does not terminate the loop entirely – it only ignores the remaining statements for that cycle.

EXAMPLE 3: CONTINUE IN A FOR LOOP

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

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3)
            continue;
        cout << i << " ";
    }
    return 0;
}
```

OUTPUT: 1 2 4 5

Explanation: When i equals 3, the continue statement skips that iteration – the print statement is not executed for 3, but the loop continues for 4 and 5.

EXAMPLE 4: CONTINUE IN A WHILE LOOP

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

int main() {
   int i = 0;
   while (i < 5) {
      i++;
      if (i == 2)
         continue;
      cout << i << " ";
   }
   return 0;
}
```

OUTPUT: 1 3 4 5

Explanation: For i == 2, the loop skips the cout statement and immediately continues with the next increment.

VISUALIZATION OF LOOP CONTROL FLOW

To visualize the behavior of break and continue:

1. Loop starts.
2. Condition is evaluated.
3. Loop body executes.
4. If break encountered → control exits loop.
5. If continue encountered → skip remaining body and go to next iteration.
6. Repeat steps 2-5.

EXAMPLE 5: DIFFERENCE BETWEEN BREAK AND CONTINUE

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

int main() {
   cout << "Break example:" <<
endl;
   for (int i = 1; i <= 5; i++) {
      if (i == 3)
         break;
      cout << i << " ";
   }

   cout << "\nContinue
example:" << endl;
   for (int i = 1; i <= 5; i++) {
      if (i == 3)
         continue;
      cout << i << " ";
   }
   return 0;
}
```

OUTPUT:

Break example:
1 2
Continue example:
1 2 4 5

Explanation:

- break exits the loop immediately when i == 3.
- continue skips printing 3 but continues executing the loop for 4 and 5.

TABLE: COMPARISON OF BREAK AND CONTINUE

Feature	break	continue
Function	Terminates the loop immediately	Skips to next iteration
Execution Control	Exits the loop body	Remains inside the loop
Common Use	To stop looping when a condition is met	To skip unwanted iterations
Usable In	for, while, do-while, switch	for, while, do-while
Example Behavior	Exit on condition met	Skip only a single iteration

REFERENCES:

- GEEKSFORGEEKS: HTTPS://WWW.GEEKSFORGEEKS.ORG/CPP/CPP-LOOPS/
- GEEKSFORGEEKS: HTTPS://WWW.GEEKSFORGEEKS.ORG/CPP/CONTINUE-STATEMENT-CPP/
- GEEKSFORGEEKS: HTTPS://WWW.GEEKSFORGEEKS.ORG/DIFFERENCE-BETWEEN-CONTINUE-AND-BREAK-STATEMENTS-IN-C/
- W3SCHOOLS: HTTPS://WWW.W3SCHOOLS.COM/CPP/CPP_BREAK.ASP

THANK YOU