Chapter 7 Jump Statements

Program control is how a program makes decisions or organizes its activities**. Jump Statements** are used to transfer the program control from one part of the code to another.

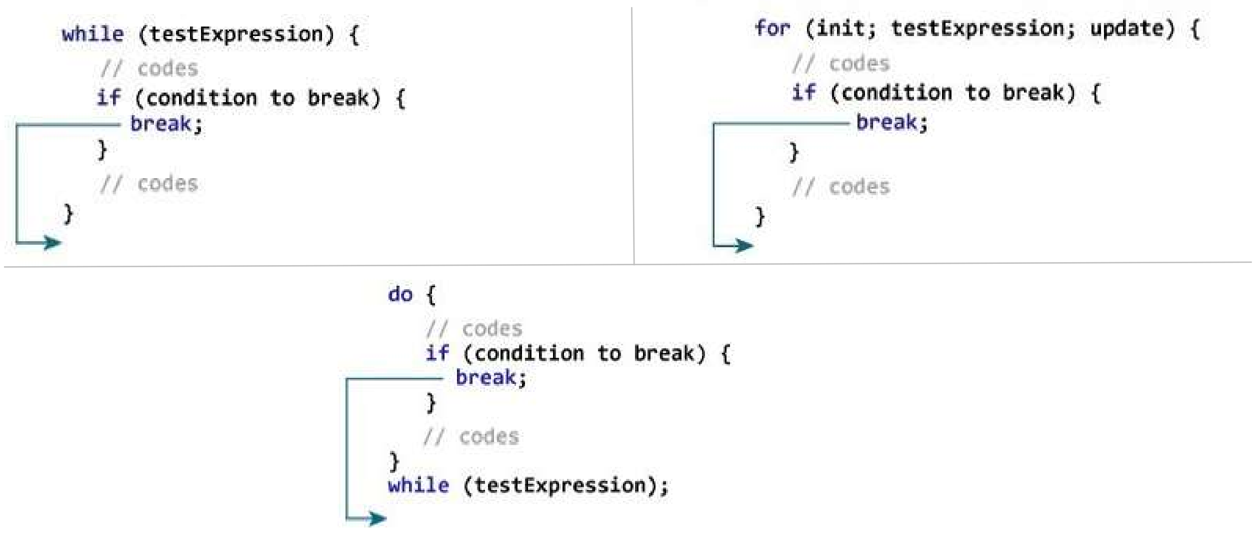
There are 4 types of Jump statements in C++.

- `break` Statement
- `continue` Statement
- `goto` Statement
- `return` Statement.

1. `break` Statement

The `break` statement is used to exit (end, stop or terminate) a loop (`for`, `while`, and `do`-`while`) or `switch` statement.

As shown in the codes below, the `break` statement exits the loop immediately when it is encountered.



```
while (testExpression) {
   // codes
   if (condition to break) {
     break;
   }
   // codes
}
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
     break;
   }
   // codes
}
```

```
do {
   // codes
   if (condition to break) {
     break;
   }
   // codes
}
while (testExpression);
```

Example 01: Use `break` to exit a while loop.

```
#include <iostream>
using namespace std;
int main(){
    int i = 0;

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

Output

```
0 1
```

If we write `while true` then the loop will run forever. Which is why it is often used in conjunction with a `break` statement, which allows the loop to be exited under a certain condition.

Example 02: Using `while true`.

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

int main() {
    int count = 0;

    while(true){
        cout << "Hello ";
        count += 1;

        if(count == 5)
            break;
    }

    return 0;
}
```

Output:

```
Hello Hello Hello Hello Hello
```

`while True` should be used when you need to place the condition somewhere else besides the beginning of the loop in order for the loop body to execute first before checking the condition.

Example 03: Use **`break`** to exit a for loop.

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

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

Output

```
0 1
```

Example 04: Use **break** to exit a do-while loop.

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

int main(){
    int i = 0;

    do {
        if(i == 2)
            break;
        cout << i << " ";
        i++;

    } while(i < 5);

    return 0;
}
```

Output

```
0 1
```

In a nested loop, a break statement only exits the loop it is placed in. Therefore, if a break is placed in the inner loop, it breaks the inner loop; the outer loop still continues.

Example 05: Use **break** in a nested loop.

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

int main(){
    // Outer loop
    for (int i = 1; i <= 2; i++) {
        cout << "Outer: " << i << endl; // Executes 2 times

        // Inner loop
        for (int j = 1; j <= 3; j++) {
            if(j == 2)
                break;
            cout << "\tInner: " << j << endl;
        }
    }
    return 0;
}
```

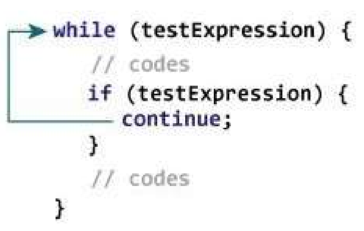
Output

```
Outer: 1
        Inner: 1
Outer: 2
        Inner: 1
```

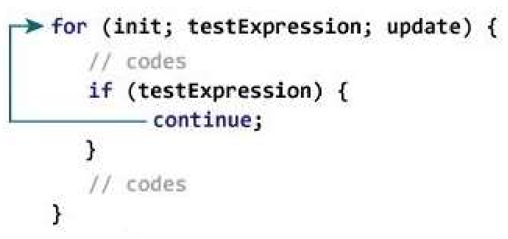
2. continue Statement

The continue statement is used to end the current iteration of a loop, and continues to the next iteration. It does not exit the loop. The continue statement can be placed wherever we want in the loop body.

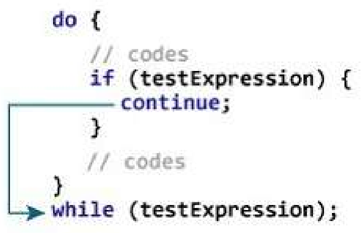
As shown in the codes below, the continue statement skips the current iteration of the loop and continues with the next iteration.



```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```



```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```



```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

Example 06: Use continue to skip a current iteration of a while loop.

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

int main(){
    int i = 0;

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

Output

```
0 1 3 4
```

Example 07: Use **continue** to skip a current iteration of a for loop.

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

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

Output

```
0 1 3 4
```

Example 08: Use **continue** to skip a current iteration of a do-while loop.

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

int main(){
    int i = 0;

    do{
        if(i == 2){
            i++;
            continue;
        }
        cout << i << " ";
        i++;

    } while(i < 5);

    return 0;
}
```

Output

```
0 1 3 4
```

In a nested loop, the `continue` statement skips the current iteration of the loop it is placed in. Therefore, if a `continue` is placed in the inner loop, it skips the current iteration of the inner loop.

Example 09: Use `continue` in a nested loop.

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

int main(){
    // Outer loop
    for (int i = 1; i <= 2; i++) {
        cout << "Outer: " << i << endl; // Executes 2 times

        // Inner loop
        for (int j = 1; j <= 3; j++) {
            if(j == 2)
                continue;
            cout << "\tInner: " << j << endl;
        }
    }
    return 0;
}
```

Output

```
Outer: 1
        Inner: 1
        Inner: 3
Outer: 2
        Inner: 1
        Inner: 3
```

3. goto Statement

goto statements are used to specify where the program control should go. Unlike continue statement, the goto statements can be placed outside the loops.

Syntax:

```
goto label;
// ...
label:
// statements to execute
```

Example 10: Use goto statement.

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

int main(){
    cout << "Hello";

    goto end;  // jumps to a label named "end"

    cout << "C Programing";

    end:  // This is where the "end" label is defined
    cout << "Done";

    return 0;
}
```

Output

```
Hello
Done
```

Example 11: Another example of using **goto** statement.

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

int main(){
    int x = -10;

    if(x < 0)
        goto first;
    else
        goto second;

    first:
    cout << "x is negative.";
    goto third;

    second:
    cout << "x is positive.";

    third:

    return 0;
}
```

Output

```
x is negative
```

In a nested loop, a **break** statement only exits the loop it is placed in. To create a multi-level **break** statements (that exits both inner loop and outer loop), you can use the **goto** statement.

Example 12: Use **goto** in a nested loop to exit both inner loop and outer loop.

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

int main(){
    // Outer loop
    for (int i = 1; i <= 2; i++) {
        cout << "Outer: " << i << endl;

        // Inner loop
        for (int j = 1; j <= 3; j++) {
            if(j == 2)
                goto end;
            cout << "\tInner: " << j << endl;
        }
    }
    end:
    cout << "Done";

    return 0;
}
```

Output

```
Outer: 1
    Inner: 1
Done
```

Note: the use of the **goto** statement is not a good practice. Using **goto** excessively can make the code difficult to understand and maintain. The **goto** statements are usually to be avoided, but are acceptable in a few well-constrained situations, if necessary; such as multi-level **break** statements.

4. `return` Statement

`return` statement is used to stop the execution of a function immediately. It can be anywhere in the function.

Syntax:

```
return;
```

or

```
return expression;
```

Example 13: Use `return` statement.

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

int main(){
    cout << "Hello";

    return 0;

    cout << "Done";
}
```

Output

```
Hello
```

Example 14: Another example of using `return` statement.

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

int main(){
    // Outer loop
    for (int i = 1; i <= 2; i++) {
        cout << "Outer: " << i << endl;

        // Inner loop
        for (int j = 1; j <= 3; j++) {
            return 0;
            cout << "\tInner: " << j << endl;
        }
    }
    cout << "Done";

    return 0;
}
```

Output

```
Outer: 1
```

Exercises

Write the following programs:

1. (Palindrome Array Check) Write a program that determines whether an integer array is a palindrome (i.e., it reads the same forward and backward). The program should compare elements from the start and end of the array, moving toward the center. If a mismatch is found, exit the loop immediately using `break`.
2. Display all numbers from an array but skips numbers that are multiples of `5` using `continue`.
3. Count how many numbers are positive and how many are negative in an array. Use `continue` to skip zeros.
4. Find the first duplicate number in a 2D array and stops checking using `goto`. If no duplicate number is found, display "No duplicate numbers".
5. Ask the user to enter a number. If the number exists in an array, print "Number found!" and exit the program immediately using `return`. If not found, display "Number not found." after checking all elements.

Reference

[1] Y. Daniel Liang. 'Introduction to Programming with C++', 3e – 2014

[2] Dey, P., & Ghosh, M. 'Computer Fundamentals and Programming in C', 2e – 2013

[3] https://cplusplus.com/doc/tutorial/

[4] https://www.programiz.com/cpp-programming

[5] https://www.studytonight.com/cpp/