

1. Introduction to the `goto` Statement

The `goto` statement in C++ is a jump statement. It transfers the program's control from one part of the code to another part identified by a label.

Normally, a program executes statements from top to bottom. The `goto` statement allows us to jump to a different location in the same function.

Real-life example: Imagine reading a book from page 1 to page 10. Your teacher asks you to skip directly to page 7. You jump to page 7 instead of reading every page in between. The `goto` statement works in a similar way.

2. Why Do We Use the `goto` Statement?

The `goto` statement can be used to:

- Jump to a specific part of a program.
- Repeat a block of code by jumping backward.
- Skip some statements by jumping forward.
- Handle certain situations where a direct jump is useful.

However, excessive use of `goto` can make a program difficult to understand and maintain. In most cases, `for`, `while`, and `do-while` loops are better choices for repetition.

3. Syntax of the `goto` Statement

```
goto label;  
  
// Other statements  
  
label:  
    // Statements to execute
```

Understanding the syntax:

- `goto` is a C++ keyword used to jump to a label.
- `label` is a name chosen by the programmer.
- `:` (colon) marks the label's location.
- The statements after the label execute when control reaches that location.

The label name used with `goto` must match the label defined in the program.

4. Example 1: Simple `goto` Statement

Program

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

```
int main()
{
    cout << "Statement 1" << endl;

    goto second;

    cout << "Statement 2" << endl;

    second:
        cout << "Statement 3" << endl;

    return 0;
}
```

Understand the Code

The program jumps directly to the label `second`. Therefore, `Statement 2` is skipped.

Output

```
Statement 1
Statement 3
```

How the program executes

1. Print Statement 1
2. Execute `goto second`
3. Skip Statement 2
4. Jump to `second` and print Statement 3

Modify the Code

Make the following changes yourself:

1. Change `Statement 1` to `Start of Program`.
2. Change `Statement 2` to `This line is skipped`.
3. Change `Statement 3` to `Program continues here`.
4. Add another `cout` statement before `goto second`.

Question: Which statements will appear in the output, and which statement will be skipped?

5. Example 2: Jumping Forward

A forward jump moves control to a label that appears later in the program.

Program

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

int main()
{
    cout << "Welcome to C++" << endl;

    goto finish;

    cout << "Learning variables" << endl;
    cout << "Learning conditions" << endl;

    finish:
        cout << "End of program" << endl;

    return 0;
}
```

Understand the Code

1. The program prints `Welcome to C++`.
2. The `goto finish` statement transfers control to the `finish` label.
3. Both statements about variables and conditions are skipped.
4. The program prints `End of program`.

Output

```
Welcome to C++
End of program
```

Modify the Code

Modify the program so that:

- It prints `Student Registration`.
- It jumps to the label `result`.
- The messages `Enter Student Name` and `Enter Student Roll Number` are skipped.
- The label `result` prints `Registration Completed`.

Expected output

```
Student Registration
Registration Completed
```

6. Example 3: Using `goto` to Repeat Code

A `goto` statement can jump backward to repeat statements. This example demonstrates repetition without using a loop.

Program

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

int main()
{
    int count = 1;

    repeat:
        cout << count << endl;
        count++;

        if (count <= 5)
        {
            goto repeat;
        }

    return 0;
}
```

Understand the Code

- `int count = 1;` initializes the counter.
- `repeat:` defines the location to which the program can jump.
- `cout << count` displays the current number.
- `count++` increases the counter by 1.
- `if (count <= 5)` checks whether another repetition is needed.
- `goto repeat` sends control back to the label if the condition is true.

Output

```
1
2
3
4
5
```

Trace the Code

Iteration	Printed value	Counter after <code>count++</code>	Jump back?
1	1	2	Yes
2	2	3	Yes
3	3	4	Yes
4	4	5	Yes

Iteration	Printed value	Counter after <code>count++</code>	Jump back?
5	5	6	No

When `count` becomes 6, the condition `count <= 5` becomes false. The program exits the repetition and reaches `return 0;`.

Modify the Code

Change the program to print numbers from 1 to 10 instead of 1 to 5.

Hints:

- Keep `count = 1`.
- Update the condition to `count <= 10`.
- Keep the label name `repeat` unchanged.

Challenge: Modify the program to print even numbers from 2 to 10 using `goto`.

7. Example 4: Jumping Based on a Condition

In this example, the program checks a student's marks and uses `goto` to jump to a result message.

Program

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

int main()
{
    int marks;

    cout << "Enter your marks: ";
    cin >> marks;

    if (marks >= 50)
    {
        goto pass;
    }

    cout << "You need to improve." << endl;
    goto finish;

pass:
    cout << "Congratulations! You passed." << endl;

finish:
    cout << "Result checking completed." << endl;

    return 0;
}
```

Understand the Code

If the student enters 65:

- The condition `marks >= 50` is true.
- The program jumps to `pass`.
- The pass message is displayed.
- The program continues to `finish`.

Output:

```
Enter your marks: 65
Congratulations! You passed.
Result checking completed.
```

If the student enters 35, the condition is false. The program prints the improvement message and then jumps to `finish`.

Output:

```
Enter your marks: 35
You need to improve.
Result checking completed.
```

Modify the Code

Change the program to use these rules:

- If marks are 80 or above, jump to the label `excellent` and print `Excellent Performance!`.
- Otherwise, print `Keep Practicing!`.
- At the end, print `Assessment Completed`.

Test the program with marks of 85 and 60.

8. Important Rules of the `goto` Statement

1. A label must be followed by a colon (`:`).
2. The `goto` statement and its target label must be inside the same function.
3. The label name in the `goto` statement must match the defined label.
4. A forward jump can skip statements.
5. A backward jump can repeat statements.
6. Incorrect use of backward jumps may cause an infinite loop.
7. Avoid jumping into the scope of variables in ways that violate C++ initialization rules.
8. Prefer loops and structured control statements when they make the code easier to understand.

9. Practice Coding Exercises

Solve these exercises yourself before looking for help. Each exercise includes a program statement and a suggested task.

Exercise 1 — Skip a Message

Beginner

Program statement: Write a C++ program that prints **Start**, jumps over a message saying **This message is skipped**, and then prints **Finish**.

Requirements: Use the **goto** statement and a label named **end**.

Exercise 2 — Print Numbers

Beginner

Program statement: Write a program that prints numbers from 1 to 7 using a counter, an **if** statement, and a backward **goto**.

Requirements: Use the label **again**. Do not use a loop.

Exercise 3 — Positive or Negative

Beginner–Intermediate

Program statement: Ask the user to enter an integer. If the number is positive, jump to the label **positive** and print **Positive Number**. Otherwise, print **Zero or Negative Number**.

Requirements: Use **if**, **goto**, and a suitable label.

Exercise 4 — Pass or Fail

Beginner–Intermediate

Program statement: Input a student's marks. If marks are 50 or above, jump to **passed** and display **Pass**. Otherwise, display **Fail**. Both paths must eventually display **Result Completed**.

Requirements: Use an **if** statement, **goto**, and at least two labels.

Exercise 5 — Display a Menu Again

Intermediate

Program statement: Display a menu with two choices: **1. Say Hello** and **2. Exit**. If the user selects 1, print **Hello Student!** and use **goto** to display the menu again. If the user selects 2, end the program.

Requirements: Use a label named **menu**. Prevent the program from repeating forever after the user selects 2. For this exercise, you may assume the user enters only 1 or 2.