

# ELEG 1043

## Computer Applications in Engineering





# Chapter 5: Repetition Statements

**C++** FOR ENGINEERS  
AND SCIENTISTS

# Acknowledgement

- Some of the slides or images are from various sources. The copyright of those materials belongs to their original owners.

# Objectives

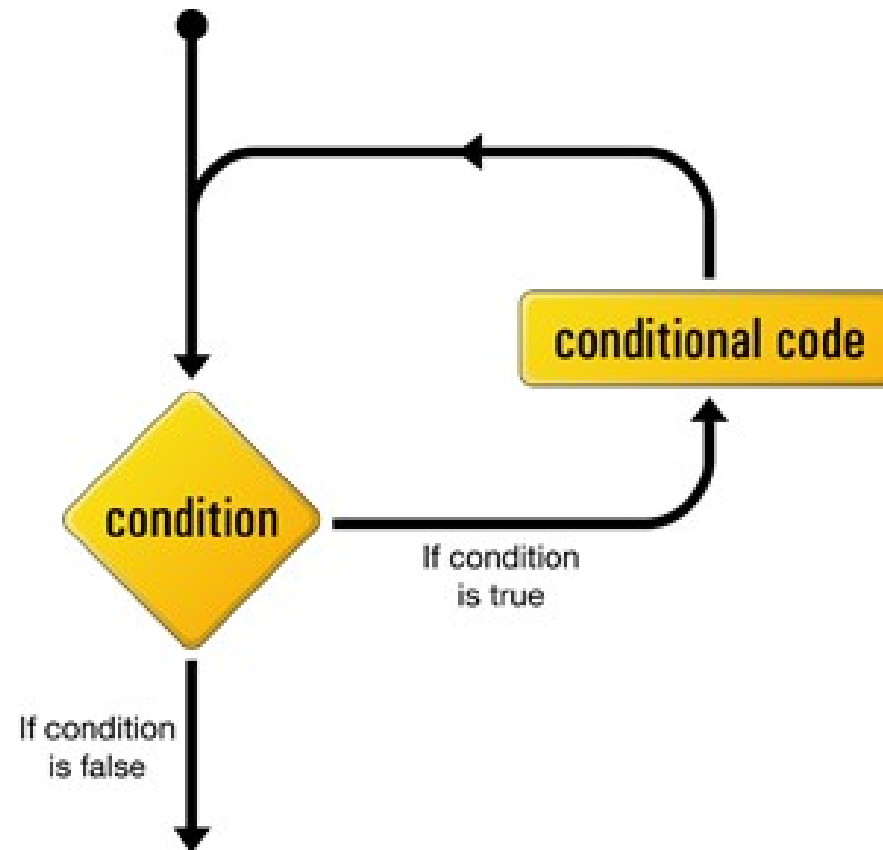
In this chapter, you will learn about:

- Basic loop structures
- **while** loops
- Interactive **while** loops
- **for** loops
- Loop programming techniques

# Objectives (continued)

- Nested loops
- **do while** loops
- Common programming errors

# LOOP



# Basic Loop Structures

- Repetition structure has **four** required elements:
  - Repetition statement
  - Condition to **be evaluated**
  - **Initial** value for the condition
  - Loop **termination**
- Repetition statements include:
  - **while**
  - **for**
  - **do while**

# Basic Loop Structures (continued)

- The **condition** can be tested
  - At the beginning: **Pretest** or **entrance-controlled** loop
  - At the end: **Posttest** or **exit-controlled** loop
- Something in the loop body must cause the condition to **change**, to avoid an **infinite loop**, which **never terminates**



# Pretest and Posttest Loops

- Pretest loop: Condition is tested **first**; if false, statements in the loop body are **never executed**
  - **while** and **for** loops are pretest loops
- do while?**

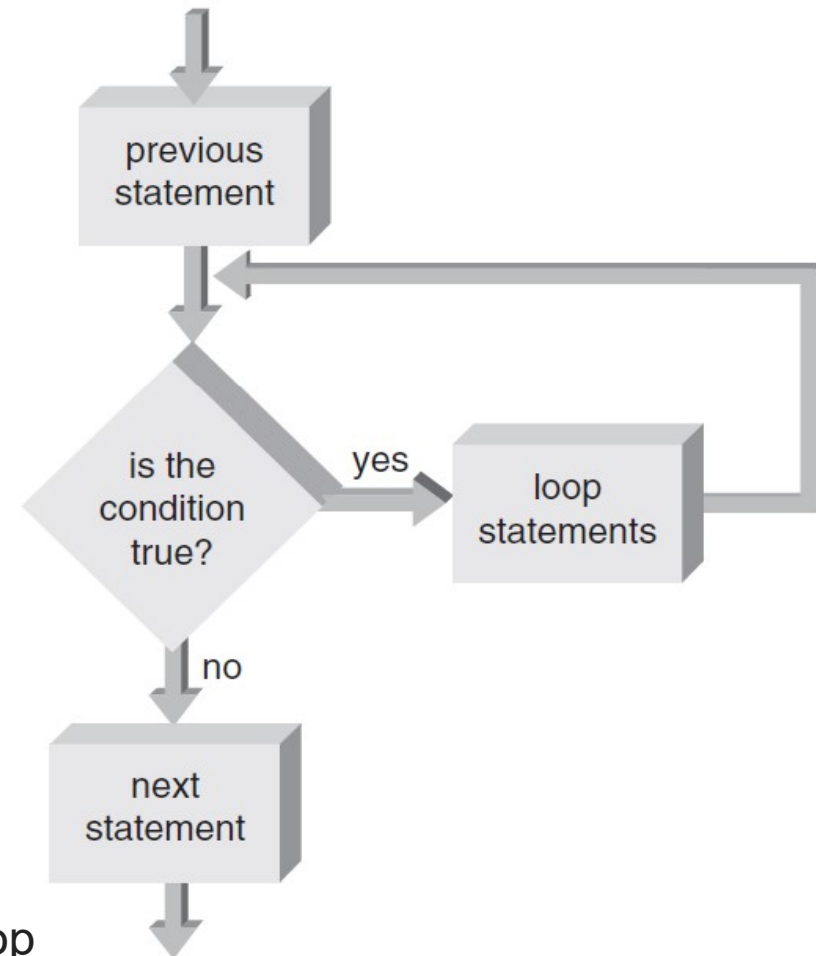


Figure 5.1 A pretest loop

# Pretest and Posttest Loops (continued)

- Posttest loop: Condition is tested **after** the loop body statements are executed; loop body **always executes** at least once
- **do while** is a **posttest loop**

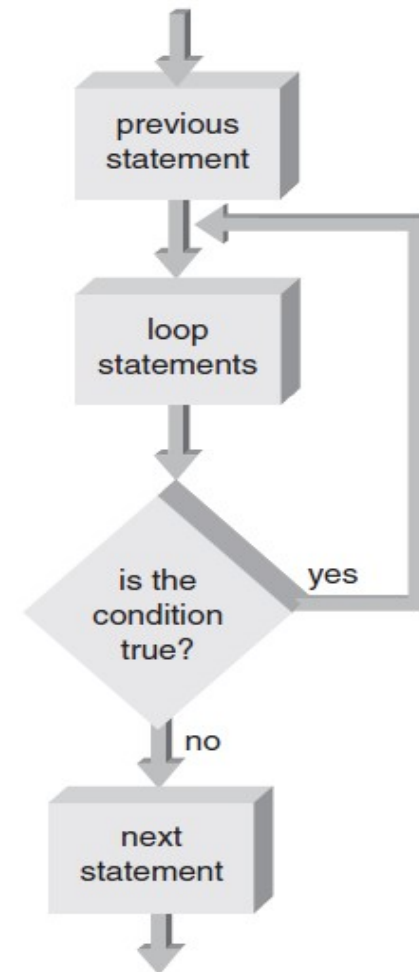


Figure 5.2 A posttest loop

# Fixed-Count **Versus** Variable-Condition Loops

- **Fixed-count loop:** Loop is processed for a fixed number of repetitions
- **Variable-condition loop:** Number of repetitions depends on the value of a variable

# while Loops



# while Loops

- **while statement** is used to create a `while` loop
  - Syntax:

***while (expression)  
statement;***

- Statements following the expressions are executed **as long as the expression condition remains true** (evaluates to a non-zero value)

# while Loops (continued)



## Program 5.1

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

int main()
{
    int count;

    count = 1;           // initialize count
    while (count <= 10)
    {
        cout << count << " ";
        count++;         // increment count
    }

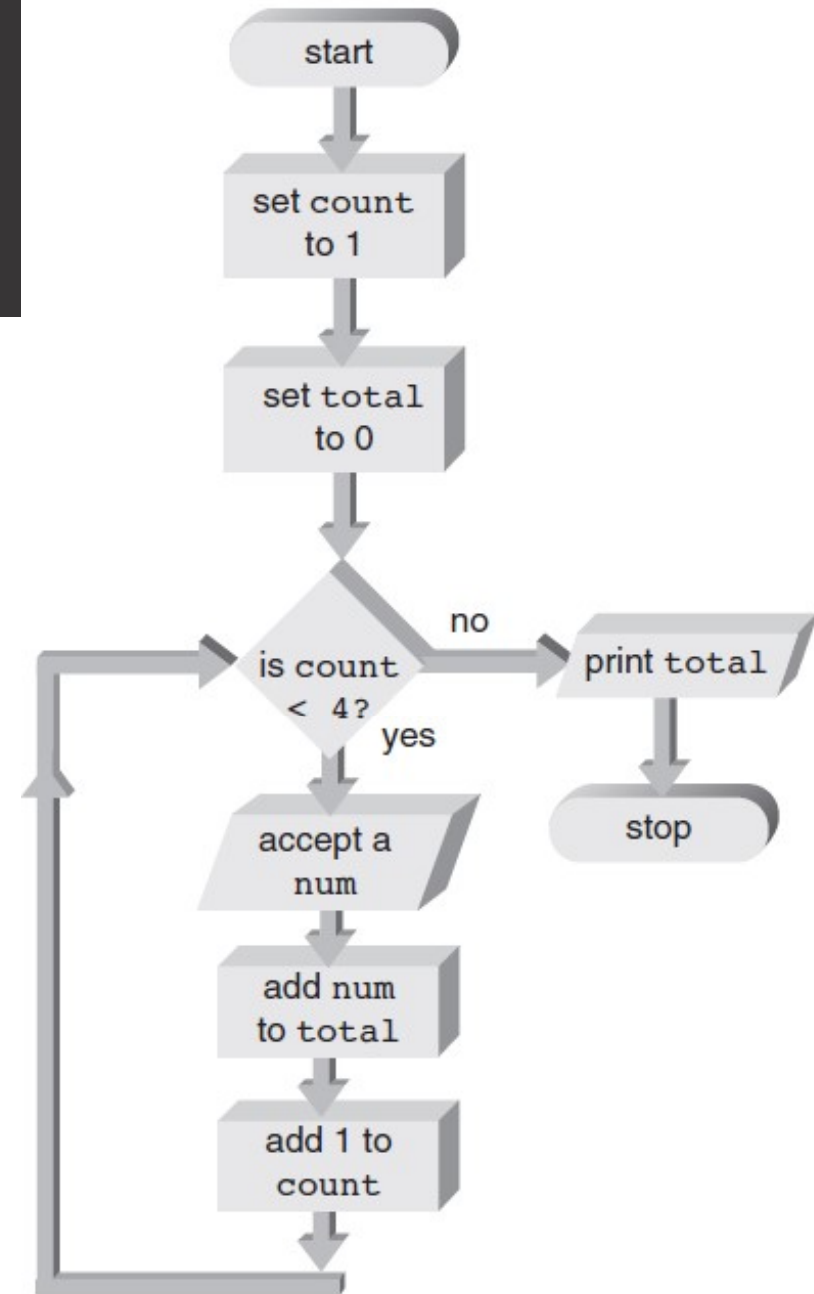
    return 0;
}
```

# Interactive `while` Loops

- Combining **interactive data entry** with the `while` statement provides for repetitive entry and accumulation of totals

# Interactive while Loops (cont'd)

**Figure 5.7** Accumulation  
flow of control





# Interactive `while` Loops (cont'd)

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

int main()
{
    int count = 0, total = 0;

    while(count < 4)
    {
        int num;
        cout<<"Enter a num \n";
        cin>>num;
        total = total + num;
    }

    cout<<"The total is "<<total;
    return 0;
}
```

# Interactive `while` Loops (cont'd)

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

int main()
{
    int count = 0, total = 0;

    while(count < 4)
    {
        int num;
        cout<<"Enter a num \n";
        cin>>num;
        total = total + num;
        count = count + 1;
    }

    cout<<"The total is "<<total;
    return 0;
}
```

# break and continue Statements

- **break** statement
  - Forces an immediate **break**, or exit, from **switch**, **while**, **for**, and **do-while** statements
  - Useful for breaking out of loops when an unusual condition is detected

# break and continue Statements (cont'd)

- Example of a break statement:

```
while (count <= 10)
{
    cout << "Enter a number: ";
    cin >> num;
    if (num > 76)
    {
        cout << "You lose!\n";
        break;           // break out of the loop
    }
    else
        cout << "Keep on trucking!\n";
    count++;
}
// break jumps to here
```

# break and continue Statements (cont'd)

- **continue** statement
  - Applies to **while**, **do-while**, and **for** statements; causes the next iteration of the loop to begin immediately
  - Useful for **skipping over data** that should not be processed in this iteration, while staying within the loop

# break and continue

## Statements (cont'd)

- A `continue` statement where invalid grades are **ignored**, and **only valid grades** are added to the total:

```
while (count < 30)
{
    cout << "Enter a grade: ";
    cin >> grade
    if(grade < 0 || grade > 100)
        continue;
    total = total + grade;
    count++;
}
```