Chapter 5 Loops and Files

5.1 The Increment and Decrement Operators

Concept:

`++` and `--` are operators that add and subtract 1 from their operands.

- *Incrementing* a value means increasing it by one.
- *Decrementing* a value means decreasing it by one.
- C++ provides the unary increment operator (`++`) and decrement operator (`--`) for these tasks.
- For example, `num++;` increments `num`, and `num--;` decrements `num`.

Note:

The expression `num++` is pronounced "num plus plus," and `num--` is pronounced "num minus minus."

- **Postfix mode**: The operator is placed after the variable (e.g., `num++`).
- **Prefix mode**: The operator is placed before the variable (e.g., `++num`).
- In simple statements, both modes achieve the same result of modifying the variable's value by one.

Program 5-1

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int num = 4;
7
8       cout << "The variable num is " << num << endl;
9       cout << "I will now increment num.\n\n";
10
11      num++;
12      cout << "Now the variable num is " << num << endl;
13      cout << "I will increment num again.\n\n";
```

```
14
15        ++num;
16        cout << "Now the variable num is " << num << endl;
17        cout << "I will now decrement num.\n\n";
18
19        num--;
20        cout << "Now the variable num is " << num << endl;
21        cout << "I will decrement num again.\n\n";
22
23        --num;
24        cout << "Now the variable num is " << num << endl;
25        return 0;
26    }
```

Program Output

The Difference between Postfix and Prefix Modes

- The difference between postfix and prefix modes becomes important when the operators are used in more complex statements.
- **Postfix mode**: The variable's value is used in the expression *first*, and then it is incremented or decremented.
  - Example: In `cout << num++;` (where `num` is 4), the value 4 is displayed, and then `num` is incremented to 5.
- **Prefix mode**: The variable is incremented or decremented *first*, and then its new value is used in the expression.
  - Example: In `cout << ++num;` (where `num` is 4), `num` is first incremented to 5, and then the value 5 is displayed.

Program 5-2

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

int main()
{
    int num = 4;

    cout << num << endl;
    cout << num++ << endl;
    cout << num << endl;
    cout << ++num << endl;
    cout << endl;

    cout << num << endl;
    cout << num-- << endl;
    cout << num << endl;
    cout << --num << endl;

    return 0;
}
```

💻 **Program Output**

- Consider the following code:

```cpp
int x = 1;
int y
y = x++;
```

- In postfix mode, the value of `x` (which is 1) is assigned to `y` *before* `x` is incremented.
- After execution, `y` will be 1, and `x` will be 2.
- Now consider prefix mode:

```
int x = 1;
int y;
y = ++x;
```

- In prefix mode, `x` is incremented to 2 *before* its value is assigned to `y`.
- After execution, both `y` and `x` will be 2.

Using `++` and `--` in Mathematical Expressions

- These operators can be used within mathematical expressions, where the order of operations matters.
- Consider this example:

```
a = 2;
b = 5;
c = a * b++;
cout << a << " " << b << " " << c;
```

- `c` is assigned `a * b` (which is 10), and *then* `b` is incremented. The output will be `2 6 10`.
- If the expression were `c = a * ++b;`, `b` would be incremented to 6 *before* the multiplication.
- `c` would be assigned `2 * 6` (which is 12). The output would be `2 6 12`.
- The operand of `++` and `--` must be an lvalue (something that identifies a memory location, like a variable). You cannot use it on an expression like `(a * b)`.

Using `++` and `--` in Relational Expressions

- The operators can also be used in relational expressions, where the mode affects the outcome of the comparison.
- Consider this code:

```
x = 10;
if (x++ > 10)
   cout << "x is greater than 10.\n";
```

- In postfix mode, the comparison `10 > 10` happens first (which is false), and then `x` is incremented. The `cout` statement will not execute.

- If prefix mode is used:

```
x = 10;
if (++x > 10)
   cout << "x is greater than 10.\n";
```

- `x` is first incremented to 11, and then the comparison `11 > 10` is performed (which is true). The `cout` statement will execute.

Checkpoint

5.1 What will the following program segments display?

```
x = 2; y = x++; cout << x << y;
```

```
x = 2; y = ++x; cout << x << y;
```

```
x = 2; y = 4; cout << x++ << --y;
```

```
x = 2; y = 2 * x++; cout << x << y;
```

```
x = 99; if (x++ < 100) cout "It is true!\n"; else cout << "It is false!\n";
```

```
x = 0; if (++x) cout << "It is true!\n"; else cout << "It is false!\n";
```

5.2 Introduction to Loops: The `while` Loop

Concept:

A loop is part of a program that repeats.

The while Loop

- A **loop** is a control structure that causes a statement or group of statements to repeat.
- C++ offers three looping structures: `while`, `do-while`, and `for`. They differ in how they control repetition.

The `while` Loop

- A `while` loop has two main parts:

  1. An expression that is tested for a true or false value.
  2. A statement or block that is repeated as long as the expression is true.

- Here is the general format:

```
while (expression)
   statement;
```

- To repeat a block of statements, use braces:

```
while (expression)
{
   statement;
   statement;
}
```

- **How it works**:
  - The `expression` is tested.
  - If the expression is true, the loop *body* (the statement or block) is executed.
  - This cycle repeats until the `expression` becomes false.

Program 5-3

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int number = 0;
7
8       while (number < 5)
9       {
10          cout << "Hello\n";
11          number++;
12      }
13      cout << "That's all!\n";
14      return 0;
15  }
```

Program Output

- Each repetition of a loop is called an **iteration**.
- In Program 5-3, the loop performs five iterations.
- The variable `number` is the **loop control variable** because it controls the number of times the loop iterates.

The `while` Loop Is a Pretest Loop

- The `while` loop is a **pretest** loop, meaning it tests its expression *before* each iteration.
- If the test expression is false initially, the loop will never execute.
- To ensure a `while` loop runs at least once, you must initialize the relevant data so the test expression is true from the start.

Infinite Loops

- Loops must contain a mechanism to terminate. Something inside the loop must eventually make the test expression false.
- An **infinite loop** continues to repeat until the program is interrupted.
- This happens if the loop lacks a way to alter the loop control variable.

```
int number = 0;
while (number < 5)
{
   cout << "Hello\n";
}
```

- Accidentally placing a semicolon after the `while` header also creates an infinite loop with a null statement as its body.

```
while (number < 5);
```

Don't Forget the Braces with a Block of Statements

- If you intend for a loop to repeat a block of statements, you must enclose them in braces `{}`.

- Without braces, the `while` statement only controls the single statement immediately following it, which can lead to infinite loops.

```
while (number < 5)
   cout << "Hello\n";
   number++;
```

Programming Style and the `while` Loop

- For good programming style, the body of the loop should be indented.
- If the body is a single statement, place it on the line after the `while` header.
- If the body is a block, indent each statement within the braces. This makes the code more readable.

In the Spotlight:

Designing a Program with a `while` Loop

A project currently underway at Chemical Labs, Inc. requires that a substance be continually heated in a vat. A technician must check the substance's temperature every 15 minutes. If the substance's temperature does not exceed 102.5 degrees Celsius, then the technician does nothing. However, if the temperature is greater than 102.5 degrees Celsius, the technician must turn down the vat's thermostat, wait 5 minutes, and check the temperature again. The technician repeats these steps until the temperature does not exceed 102.5 degrees Celsius. The director of engineering has asked you to write a program that guides the technician through this process.

Here is the algorithm:

1. Prompt the user to enter the substance's temperature.
2. Repeat the following steps as long as the temperature is greater than 102.5 degrees Celsius:
    1. Tell the technician to turn down the thermostat, wait 5 minutes, and check the temperature again.
    2. Prompt the user to enter the substance's temperature.
3. After the loop finishes, tell the technician that the temperature is acceptable and to check it again in 15 minutes.

After reviewing this algorithm, you realize that steps 2a and 2b should not be performed if the test condition (temperature is greater than 102.5) is false to begin with. The `while` loop will work well in this situation, because it will not execute even once if its condition is false. Program 5-4 shows the code for the program.

Program 5-4

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

int main()
{
    const double MAX_TEMP = 102.5;
    double temperature;

    cout << "Enter the substance's Celsius temperature: ";
    cin >> temperature;

    while (temperature > MAX_TEMP)
    {
        cout << "The temperature is too high. Turn the\n";
        cout << "thermostat down and wait 5 minutes.\n";
        cout << "Then take the Celsius temperature again\n";
        cout << "and enter it here: ";
        cin >> temperature;
    }

    cout << "The temperature is acceptable.\n";
    cout << "Check it again in 15 minutes.\n";

    return 0;
}
```

Program Output

5.3 Using the `while` Loop for Input Validation

Concept:

The `while` loop can be used to create input routines that repeat until acceptable data is entered.

- **Input validation** is the process of inspecting user-provided data to determine if it's valid.
- The phrase "garbage in, garbage out" highlights the importance of ensuring a program receives good input to produce good output.
- A `while` loop is excellent for input validation. If the input is invalid, the loop can prompt the user to re-enter the data until a valid value is provided.
- Example validation loop for a number between 1 and 100:

```
cout << "Enter a number in the range 1-100: ";
cin >> number;
while (number < 1 || number > 100)
{
   cout << "ERROR: Enter a value in the range 1-100: ";
   cin >> number;
}
```

- The initial read operation before the loop is known as a **priming read**. It provides the first value for the loop to test.

Program 5-5

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       const int MIN_PLAYERS = 9,
```

```
 7                MAX_PLAYERS = 15;
 8
 9       int players,
10           teamPlayers,
11           numTeams,
12           leftOver;
13
14       cout << "How many players do you wish per team? ";
15       cin >> teamPlayers;
16
17       while (teamPlayers < MIN_PLAYERS || teamPlayers > MAX_PLAYERS)
18       {
19           cout << "You should have at least " << MIN_PLAYERS
20                << " but no more than " << MAX_PLAYERS << " per team.\n";
21
22           cout << "How many players do you wish per team? ";
23           cin >> teamPlayers;
24       }
25
26       cout << "How many players are available? ";
27       cin >> players;
28
29       while (players <= 0)
30       {
31           cout << "Please enter 0 or greater: ";
32           cin >> players;
33       }
34
35       numTeams = players / teamPlayers;
36
37       leftOver = players % teamPlayers;
38
39       cout << "There will be " << numTeams << " teams with "
40            << leftOver << " players left over.\n";
41       return 0;
42   }
```

Program Output

Checkpoint

5.2 Write an input validation loop that asks the user to enter a number in the range of 10 through 25.

5.3 Write an input validation loop that asks the user to enter 'Y', 'y', 'N', or 'n'.

5.4 Write an input validation loop that asks the user to enter "Yes" or "No".

5.4 Counters

Concept:

A counter is a variable that is regularly incremented or decremented each time a loop iterates.

- A **counter** is a variable used to keep track of the number of loop iterations.
- It is typically initialized before the loop and incremented or decremented within the loop body.
- This allows a program to control a loop to execute a specific number of times.

Program 5-6

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       const int MIN_NUMBER = 1,
7                 MAX_NUMBER = 10;
8
9       int num = MIN_NUMBER;
10
11      cout << "Number Number Squared\n";
12      cout << "-------------------------\n";
13      while (num <= MAX_NUMBER)
14      {
15          cout << num << "\t\t" << (num * num) << endl;
```

```
16            num++;
17        }
18        return 0;
19    }
```

Program Output

- In Program 5-6, the `num` variable serves as a counter.
- It starts at 1 and is incremented in each iteration.
- The loop terminates when `num` reaches 11.
- Proper initialization of a counter variable is crucial.

Note:

It's important that `num` be properly initialized. Remember, variables defined inside a function have no guaranteed starting value.

5.5 The `do-while` Loop

Concept:

The `do-while` loop is a posttest loop, which means its expression is tested after each iteration.

- The `do-while` loop is like an inverted `while` loop.
- It is a **posttest** loop, meaning the expression is tested *after* the loop body has executed.

- This guarantees that the `do-while` loop will always perform at least one iteration, even if the expression is initially false.
- Format for a single statement:

```
do
  statement;
while (expression);
```

- Format for a block of statements:

```
do
{
  statement;
  statement;
} while (expression);
```

Note:

The `do-while` loop must be terminated with a semicolon.

- Use a `do-while` loop when you want to ensure the loop executes at least once.
- Program 5-7 demonstrates a **user-controlled loop**, where the user decides whether to repeat the process.

Program 5-7

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int score1, score2, score3;
7       double average;
8       char again;
9
10      do
11      {
12          cout << "Enter 3 scores and I will average them: ";
13          cin >> score1 >> score2 >> score3;
14
15          average = (score1 + score2 + score3) / 3.0;
```

```
16          cout << "The average is " << average << ".\n";
17
18          cout << "Do you want to average another set? (Y/N) ";
19          cin >> again;
20      } while (again == 'Y' || again == 'y');
21      return 0;
22  }
```

Program Output

Using `do-while` with Menus

- The `do-while` loop is an excellent choice for displaying a menu and repeating it until the user chooses to quit.
- The menu is guaranteed to be displayed at least once.

Program 5-8

```
1   #include <iostream>
2   #include <iomanip>
3   using namespace std;
4
5   int main()
6   {
7       const int ADULT_CHOICE = 1,
8                 CHILD_CHOICE = 2,
9                 SENIOR_CHOICE = 3,
10                QUIT_CHOICE = 4;
11
12      const double ADULT = 40.0,
13                   CHILD = 20.0,
14                   SENIOR = 30.0;
15
16      int choice;
17      int months;
18      double charges;
```

```
19
20      cout << fixed << showpoint << setprecision(2);
21
22      do
23      {
24          cout << "\n\t\tHealth Club Membership Menu\n\n"
25              << "1. Standard Adult Membership\n"
26              << "2. Child Membership\n"
27              << "3. Senior Citizen Membership\n"
28              << "4. Quit the Program\n\n"
29              << "Enter your choice: ";
30          cin >> choice;
31
32          while (choice < ADULT_CHOICE || choice > QUIT_CHOICE)
33          {
34              cout << "Please enter a valid menu choice: ";
35              cin >> choice;
36          }
37
38          if (choice != QUIT_CHOICE)
39          {
40               cout << "For how many months? ";
41               cin >> months;
42
43               switch (choice)
44               {
45                   case ADULT_CHOICE:
46                        charges = months * ADULT;
47                        break;
48                   case CHILD_CHOICE:
49                        charges = months * CHILD;
50                        break;
51                   case SENIOR_CHOICE:
52                        charges = months * SENIOR;
53               }
54
55               cout << "The total charges are $"
56                    << charges << endl;
57           }
58      } while (choice != QUIT_CHOICE);
59      return 0;
60  }
```

Program Output

Checkpoint

5.5 What will the following program segments display?

```
int count = 10; do { cout << "Hello World\n"; count++; } while (count < 1);
```

```
int v = 10; do cout << v << end1; while (v < 5);
```

```
int count = 0, number = 0, limit = 4; do { number += 2; count++; } while (count < limit); cout << number << " " << count << endl;
```

5.6 The `for` Loop

Concept:

The for loop is ideal for performing a known number of iterations.

- Loops can be categorized as conditional or count-controlled.
  - A **conditional loop** executes as long as a condition is true (e.g., `while`).
  - A **count-controlled** loop repeats a specific number of times.

The for Loop

- A count-controlled loop requires three elements:

  1. **Initialization** of a counter variable.
  2. A **test** of the counter against a maximum value.

3. An **update** (usually incrementing) of the counter in each iteration.

- The `for` loop is specifically designed for count-controlled situations.
- Format for a single statement:

```
for (initialization; test; update)
  statement;
```

- Format for a block:

```
for (initialization; test; update)
{
  statement;
}
```

- The loop header contains three expressions separated by semicolons:

1. The **initialization expression** runs once at the beginning.
2. The **test expression** is evaluated before each iteration. The loop continues as long as it's true.
3. The **update expression** executes at the end of each iteration.

Program 5-9

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       const int MIN_NUMBER = 1,
7                 MAX_NUMBER = 10;
8       int num;
9
10      cout << "Number Number Squared\n";
11      cout << "-------------------------\n";
12
13      for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
14          cout << num << "\t\t" << (num * num) << endl;
15
16      return 0;
17  }
```

💻 **Program Output**

Using the `for` Loop instead of `while` or `do-while`

- The `for` loop is the best choice when a loop requires an initialization, a test condition to stop, and an update at the end of each iteration.
- Many `while` loops that use a counter can be easily converted into `for` loops, making the code more compact and readable.

The `for` Loop Is a Pretest Loop

- Like the `while` loop, the `for` loop is a pretest loop.
- It evaluates the test expression *before* each iteration.
- If the test expression is initially false, the loop will not execute at all.

Avoid Modifying the Counter Variable in the Body of the `for` Loop

- All updates to the loop's counter variable should be done in the update expression of the loop header.
- Modifying the counter variable inside the loop's body can lead to unexpected behavior and logical errors.

Other Forms of the Update Expression

- The update expression is not limited to simple increments (`++`).
- You can use other expressions, such as `num += 2` to count by twos, or `num--` to count backward.

Defining a Variable in the `for` Loop's Initialization Expression

- You can define the counter variable directly within the `for` loop's initialization expression.

```
for (int num = 1; num <= 10; num++)
   cout << num << endl;
```

- When a variable is defined this way, its **scope** is limited to the loop. It cannot be accessed outside the loop.

Creating a User-Controlled `for` Loop

- The starting and ending values for the counter variable can be provided by the user, allowing them to control the number of iterations.

Program 5-10

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int minNumber,
7           maxNumber;
8
9       cout << "I will display a table of numbers and "
10           << "their squares.\n"
11           << "Enter the starting number: ";
12      cin >> minNumber;
13      cout << "Enter the ending number: ";
14      cin >> maxNumber;
15
16      cout << "Number Number Squared\n"
17           << "-------------------------\n";
18
19      for (int num = minNumber; num <= maxNumber; num++)
20          cout << num << "\t\t" << (num * num) << endl;
21
22      return 0;
23  }
```

Program Output

Using Multiple Statements in the Initialization and Update Expressions

- You can execute multiple statements in the initialization and update expressions by separating them with commas.
- Example: `for (x = 1, y = 1; x <= 5; x++, y++)`
- This technique does not apply to the test expression; for multiple conditions there, use logical operators like `&&` or `||`.

Omitting the `for` Loop's Expressions

- Any of the three expressions in the `for` loop header can be omitted.
- If the initialization is done before the loop, it can be left blank.
- If the update is handled inside the loop body, it can be left blank.
- Omitting the test expression creates an infinite loop by default.

```
for ( ; ; )
   cout << "Hello World\n";
```

In the Spotlight:

Designing a Count-Controlled Loop with the `for` Statement

Your friend Amanda just inherited a European sports car from her uncle. Amanda lives in the United States, and she is afraid she will get a speeding ticket because the car's speedometer indicates kilometers per hour. She has asked you to write a program that displays a table of

speeds in kilometers per hour with their values converted to miles per hour. The formula for converting kilometers per hour to miles per hour is:

$$MPH = KPH * 0.6214$$

In the formula, *MPH* is the speed in miles per hour and *KPH* is the speed in kilometers per hour.

The table your program displays should show speeds from 60 kilometers per hour through 130 kilometers per hour, in increments of 10, along with their values converted to miles per hour. The table should look something like this:

KPH	MPH
60	37.3
70	43.5
80	49.7
⋮	
130	80.8

After thinking about this table of values, you decide that you will write a `for` loop that uses a counter variable to hold the kilometer-per-hour speeds. The counter's starting value will be 60, its ending value will be 130, and you will add 10 to the counter variable after each iteration. Inside the loop, you will use the counter variable to calculate a speed in miles per hour. Program 5-11 shows the code.

Program 5-11

```
1   #include <iostream>
2   #include <iomanip>
3   using namespace std;
4
5   int main()
6   {
7       const int START_KPH = 60,
8                 END_KPH = 130,
9                 INCREMENT = 10;
10
11      const double CONVERSION_FACTOR = 0.6214;
12
```

```
13      int kph;
14      double mph;
15
16      cout << fixed << showpoint << setprecision(1);
17
18      cout << "KPH\tMPH\n";
19      cout << "---------------\n";
20
21      for (kph = START_KPH; kph <= END_KPH; kph += INCREMENT)
22      {
23          mph = kph * CONVERSION_FACTOR;
24
25          cout << kph << "\t" << mph << endl;
26
27      }
28      return 0;
29  }
```

Program Output

Checkpoint

5.6 Name the three expressions that appear inside the parentheses in the `for` loop's header.

5.7 You want to write a `for` loop that displays "I love to program" 50 times. Assume you will use a counter variable named `count`.

What initialization expression will you use?

What test expression will you use?

What update expression will you use?

Write the loop.

5.8 What will the following program segments display?

```
for (int count = 0; count < 6; count++) cout << (count + count);
```

```
for (int value = -5; value < 5; value++) cout << value;
```

```
int x; for (x = 5; x <= 14; x += 3) cout << x << endl; cout << x << endl;
```

5.9 Write a `for` loop that displays your name 10 times.

5.10 Write a `for` loop that displays all of the odd numbers, 1 through 49.

5.11 Write a `for` loop that displays every fifth number, 0 through 100.

5.7 Keeping a Running Total

Concept:

A *running total* is a sum of numbers that accumulates with each iteration of a loop. The variable used to keep the running total is called an *accumulator*.

- Many programming tasks involve calculating the sum of a series of numbers.
- This is typically achieved using two elements:
    1. A loop to read each number in the series.
    2. A variable, known as an **accumulator**, to hold the accumulating sum.
- The process of accumulating a sum within a loop is often called keeping a **running total**.
- The logic is as follows:
    1. Initialize the accumulator variable to 0. This is a critical step.
    2. Inside a loop, read a number.
    3. Add the number to the accumulator.
    4. Repeat steps 2 and 3 for all numbers.
- When the loop finishes, the accumulator will hold the total sum.

Program 5-12

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

int main()
{
    int days;
    double total = 0.0;

    cout << "For how many days do you have sales amounts? ";
    cin >> days;

    for (int count = 1; count <= days; count++)
    {
        double sales;
        cout << "Enter the sales for day " << count << ": ";
        cin >> sales;
        total += sales;
    }

    cout << fixed << showpoint << setprecision(2);
    cout << "The total sales are $" << total << endl;
    return 0;
}
```

Program Output

5.8 Sentinels

Concept:

A *sentinel* is a special value that marks the end of a list of values.

- Sometimes a user doesn't know the number of items they will enter in advance.
- In these cases, a **sentinel** value can be used to signal the end of the input.
- A sentinel is a special value that cannot be mistaken for a valid data item.
- When the program reads the sentinel value, the loop terminates.
- This technique avoids the need to count the items beforehand.

Program 5-13

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int game = 1,
7           points,
8           total = 0;
9
10      cout << "Enter the number of points your team has earned\n";
11      cout << "so far in the season, then enter -1 when finished.\n\n";
12      cout << "Enter the points for game " << game << ": ";
13      cin >> points;
14
15      while (points != -1)
16      {
17          total += points;
18          game++;
19          cout << "Enter the points for game " << game << ": ";
20          cin >> points;
21      }
22      cout << "\nThe total points are " << total << endl;
23      return 0;
24  }
```

Program Output

- In Program 5-13, -1 is chosen as the sentinel because a team cannot score negative points.
- A priming read is used before the loop to get the first value, allowing the loop to terminate immediately if the first input is the sentinel.
- The sentinel value itself is not included in the running total.

Checkpoint

5.12 Write a `for` loop that repeats seven times, asking the user to enter a number. The loop should also calculate the sum of the numbers entered.

5.13 In the following program segment, which variable is the counter variable and which is the accumulator?

```
int a, x, y = 0;
for (x = 0; x < 10; x++)
{
   cout << "Enter a number: ";
   cin >> a;
   y += a;
}
cout << "The sum of those numbers is " << y << endl;
```

5.14 Why should you be careful when choosing a sentinel value?

5.15 How would you modify Program 5-13 so any negative value is a sentinel?

5.9 Focus on Software Engineering: Deciding Which Loop to Use

Concept:

Although most repetitive algorithms can be written with any of the three types of loops, each works best in different situations.

- **The `while` loop**:
  - A conditional, pretest loop.

    - Ideal when you don't want the loop to iterate if the condition is initially false.
    - Good for input validation and reading data lists terminated by a sentinel.
- **The `do-while` loop**:
    - A conditional, posttest loop.
    - Ideal when you always want the loop to iterate at least once.
    - A good choice for repeating a menu.
- **The `for` loop**:
    - A pretest loop with built-in initialization, testing, and updating expressions.
    - Ideal for count-controlled situations where the exact number of iterations is known.

5.10 Nested Loops

Concept:

A loop that is inside another loop is called a *nested loop*.

- A **nested loop** is a loop that is contained within the body of another loop.
- The inner loop executes all of its iterations for each single iteration of the outer loop.
- A clock is a good analogy: for every one-hour tick (outer loop), the minute hand ticks 60 times (inner loop).
- Key points about nested loops:
    - An inner loop completes its full cycle for each iteration of an outer loop.
    - Inner loops iterate more rapidly than outer loops.
    - The total number of iterations is the product of the iterations of all the loops (e.g., an outer loop of 10 and inner loop of 5 gives 50 total inner loop iterations).

Program 5-14

```
1   #include <iostream>
2   #include <iomanip>
3   using namespace std;
4
5   int main()
6   {
7       int numStudents,
```

```
 8           numTests;
 9       double total,
10              average;
11
12       cout << fixed << showpoint << setprecision(1);
13
14       cout << "This program averages test scores.\n";
15       cout << "For how many students do you have scores? ";
16       cin >> numStudents;
17
18       cout << "How many test scores does each student have? ";
19       cin >> numTests;
20
21       for (int student = 1; student <= numStudents; student++)
22       {
23           total = 0;
24           for (int test = 1; test <= numTests; test++)
25           {
26               double score;
27               cout << "Enter score " << test << " for ";
28               cout << "student " << student << ": ";
29               cin >> score;
30               total += score;
31           }
32           average = total / numTests;
33           cout << "The average score for student " << student;
34           cout << " is " << average << ".\n\n";
35       }
36       return 0;
37   }
```

Program Output

5.11 Using Files for Data Storage

Concept:

When a program needs to save data for later use, it writes the data in a file. The data can then be read from the file at a later time.

- Data stored in variables (in RAM) is lost when a program stops running.
- To retain data, programs save it in a **file**, which is usually stored on a computer's disk.
- This allows data to be retrieved and used at a later time.
- **Writing data**: The process of saving data from a variable in RAM to a file. The file being written to is an **output file**.
- **Reading data**: The process of retrieving data from a file and copying it into a variable in RAM. The file being read from is an **input file**.
- The three fundamental steps of file processing are:
    1. **Open** the file: This creates a connection between the program and the file.
    2. **Process** the file: Data is either written to or read from the file.
    3. **Close** the file: This disconnects the file from the program.

Types of Files

- **Text file**: Contains data encoded as text (e.g., ASCII or Unicode). It can be viewed in a simple text editor.
- **Binary file**: Contains data that has not been converted to text. It cannot be viewed correctly with a text editor.

File Access Methods

- **Sequential access**: Data is accessed from the beginning of the file to the end, in order. To get to data at the end, you must read all the data before it.
- **Random access** (or direct access): You can jump directly to any piece of data in the file without reading the preceding data.

Filenames and File Stream Objects

- Files on a disk are identified by a **filename**, which may include an extension (e.g., `.txt`, `.jpg`).

- To work with a file, a C++ program uses a **file stream object**.
- This object is associated with a specific file and acts like `cin` and `cout` but for files instead of the console.

Setting Up a Program for File Input/Output

- To perform file operations, you must include the `<fstream>` header file.

```
#include <fstream>
```

- The `<fstream>` header defines several data types for file stream objects.

Table 5-1 File Stream Objects

FILE STREAM DATA TYPE	DESCRIPTION
`ofstream`	Output file stream. You create an object of this data type when you want to create a file and write data to it.
`ifstream`	Input file stream. You create an object of this data type when you want to open an existing file and read data from it.
`fstream`	File stream. Objects of this data type can be used to open files for reading, writing, or both.

Creating a File Object and Opening a File

- First, a file stream object must be created.
- Then, the file must be opened and linked to that object using the `open` member function.
- To open a file for input (reading):

```
ifstream inputFile;
inputFile.open("Customers.txt");
```

- To open a file for output (writing):

```
ofstream outputFile;
outputFile.open("Employees.txt");
```

- **Important**: Opening a file with an `ofstream` object will create the file. If the file already exists, its contents will be erased.
- You can also define the object and open the file in a single statement:

```
ifstream inputFile("Customers.txt");
ofstream outputFile("Employees.txt");
```

Closing a File

- It is good practice to explicitly close files using the `close` member function when you are finished with them.

```
inputFile.close();
```

- **Reasons to close files**:
  1. It ensures any data held in an operating system buffer is written to the file.
  2. It frees up operating system resources.

Writing Data to a File

- The stream insertion operator (`<<`) is used with `ofstream` objects to write data to a file, just as it is used with `cout` to write to the screen.

```
outputFile << "Price: " << price << endl;
```

- Using `endl` or the `\n` character writes a newline to the file, which separates items onto different lines.

Program 5-15

```
1   #include <iostream>
2   #include <fstream>
3   using namespace std;
4
5   int main()
6   {
7       ofstream outputFile;
```

```
    outputFile.open("demofile.txt");

    cout << "Now writing data to the file.\n";

    outputFile << "Bach\n";
    outputFile << "Beethoven\n";
    outputFile << "Mozart\n";
    outputFile << "Schubert\n";

    outputFile.close();
    cout << "Done.\n";
    return 0;
}
```

Program Screen Output

Program 5-16

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

int main()
{
    ofstream outputFile;
    outputFile.open("demofile.txt");

    cout << "Now writing data to the file.\n";

    outputFile << "Bach";
    outputFile << "Beethoven";
    outputFile << "Mozart";
    outputFile << "Schubert";

    outputFile.close();
    cout << "Done.\n";
    return 0;
}
```

Program Screen Output

Program 5-17

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

int main()
{
    ofstream outputFile;
    int number1, number2, number3;

    outputFile.open("Numbers.txt");

    cout << "Enter a number: ";
    cin >> number1;
    cout << "Enter another number: ";
    cin >> number2;
    cout << "One more time. Enter a number: ";
    cin >> number3;

    outputFile << number1 << endl;
    outputFile << number2 << endl;
    outputFile << number3 << endl;
    cout << "The numbers were saved to a file.\n";

    outputFile.close();
    cout << "Done.\n";
    return 0;
}
```

Program Screen Output with Example Input Shown in Bold

Program 5-18

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

int main()
{
    ofstream outputFile;
    string name1, name2, name3;

    outputFile.open("Friends.txt");

    cout << "Enter the names of three friends.\n";
    cout << "Friend #1: ";
    cin >> name1;
    cout << "Friend #2: ";
    cin >> name2;
    cout << "Friend #3: ";
    cin >> name3;

    outputFile << name1 << endl;
    outputFile << name2 << endl;
    outputFile << name3 << endl;
    cout << "The names were saved to a file.\n";

    outputFile.close();
    return 0;
}
```

Program Screen Output with Example Input Shown in Bold

Reading Data from a File

- The stream extraction operator (`>>`) is used with `ifstream` objects to read data from a file into variables.

```
inputFile >> name;
```

Reading Data from a File

Program 5-19

```
1   #include <iostream>
2   #include <fstream>
3   #include <string>
4   using namespace std;
5
6   int main()
7   {
8       ifstream inputFile;
9       string name;
10
11      inputFile.open("Friends.txt");
12      cout << "Reading data from the file.\n";
13
14      inputFile >> name;
15      cout << name << endl;
16
17      inputFile >> name;
18      cout << name << endl;
19
20      inputFile >> name;
21      cout << name << endl;
22
23      inputFile.close();
24      return 0;
25  }
```

Program Output

The Read Position

- An `ifstream` object maintains a **read position**, which marks the location of the next byte to be read from the file.
- When a file is first opened, the read position is at the beginning of the file.
- As data is read, the read position automatically advances through the file.
- The `>>` operator reads data up to the next whitespace character (space, tab, or newline).

Reading Numeric Data from a Text File

- Even when a text file contains numbers, they are stored as characters (e.g., "100").
- When you use the `>>` operator to read from a text file into a numeric variable (like an `int` or `double`), it automatically converts the character representation into the appropriate numeric data type.

Program 5-20

```
1   #include <iostream>
2   #include <fstream>
3   using namespace std;
4
5   int main()
6   {
7       ifstream inFile;
8       int value1, value2, value3, sum;
9
10      inFile.open("NumericData.txt");
11
12      inFile >> value1;
13      inFile >> value2;
14      inFile >> value3;
15
16      inFile.close();
17
18      sum = value1 + value2 + value3;
19
20      cout << "Here are the numbers:\n"
21           << value1 << " " << value2
22           << " " << value3 << endl;
23
24      cout << "Their sum is: " << sum << endl;
25      return 0;
26  }
```

Program Output

Using Loops to Process Files

- Loops are essential for processing files that contain large amounts of data.
- A loop can be used to read or write multiple items to a file without writing repetitive code.

Program 5-21

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

int main()
{
    ofstream outputFile;
    int numberOfDays;
    double sales;

    cout << "For how many days do you have sales? ";
    cin >> numberOfDays;

    outputFile.open("Sales.txt");

    for (int count = 1; count <= numberOfDays; count++)
    {
        cout << "Enter the sales for day "
             << count << ": ";
        cin >> sales;

        outputFile << sales << endl;
    }

    outputFile.close();
    cout << "Data written to Sales.txt\n";
    return 0;
}
```

Program Output

Detecting the End of the File

- A program must know when it has reached the end of a file to avoid errors from trying to read past it.
- The stream extraction operator `>>` can be used to detect the end of a file. When used as a Boolean expression, it returns `true` if a value was successfully read, and `false` if it failed (e.g., at the end of the file).
- This allows for a simple `while` loop structure to read all the data in a file:

```
while (inputFile >> number)
{
}
```

- The loop will automatically terminate when there is no more data to read.

Program 5-22

```
1   #include <iostream>
2   #include <fstream>
3   using namespace std;
4
5   int main()
6   {
7       ifstream inputFile;
8       int number;
9
10      inputFile.open("ListOfNumbers.txt");
11
12      while (inputFile >> number)
13      {
14          cout << number << endl;
15      }
```

```
16
17          inputFile.close();
18          return 0;
19      }
```

Program Output

Testing for File Open Errors

- The `open` function can fail (e.g., if an input file does not exist).
- You should always test whether a file was opened successfully before attempting to process it.
- You can test the file stream object itself in an `if` statement. It evaluates to `true` if the last operation (like `open`) was successful and `false` if it failed.

```
inputFile.open("info.txt");
if (inputFile)
{
}
else
{
}
```

- Alternatively, you can use the `.fail()` member function, which returns `true` if an operation failed.

Program 5-23

```
1   #include <iostream>
2   #include <fstream>
3   using namespace std;
4
5   int main()
```

```
{
    ifstream inputFile;
    int number;

    inputFile.open("BadListOfNumbers.txt");

    if (inputFile)
    {
        while (inputFile >> number)
        {
            cout << number << endl;
        }

        inputFile.close();
    }
    else
    {
        cout << "Error opening the file.\n";
    }
    return 0;
}
```

Program Output

Letting the User Specify a Filename

- Instead of hard-coding a filename as a string literal, you can prompt the user to enter a filename.
- The user's input can be stored in a `string` object, which can then be passed to the `open` member function (in C++11 and later).

Program 5-24

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

int main()
{
    ifstream inputFile;
```

```
 9       string filename;
10       int number;
11
12       cout << "Enter the filename: ";
13       cin >> filename;
14
15       inputFile.open(filename);
16
17       if (inputFile)
18       {
19           while (inputFile >> number)
20           {
21               cout << number << endl;
22           }
23
24           inputFile.close();
25       }
26       else
27       {
28           cout << "Error opening the file.\n";
29       }
30       return 0;
31   }
```

Program Output

Using the `c_str` Member Function in Older Versions of C++

- In versions of C++ prior to C++11, the `open` function requires a C-style, null-terminated string, not a `string` object.
- The `string` object's `c_str()` member function can be used to get a C-string representation of its contents.
- This provides compatibility with older compilers.

```
inputFile.open(filename.c_str());
```

Checkpoint

5.16 What is an output file? What is an input file?

5.17 What three steps must be taken when a file is used by a program?

5.18 What is the difference between a text file and a binary file?

5.19 What is the difference between sequential access and random access?

5.20 What type of file stream object do you create if you want to write data to a file?

5.21 What type of file stream object do you create if you want to read data from a file?

5.22 Write a short program that uses a `for` loop to write the numbers 1 through 10 to a file.

5.23 Write a short program that opens the file created by the program you wrote for Checkpoint 5.22, reads all of the numbers from the file, and displays them.

5.12 Optional Topics: Breaking and Continuing a Loop

Concept:

The `break` statement causes a loop to terminate early. The `continue` statement causes a loop to stop its current iteration and begin the next one.

Warning!

Use the `break` and `continue` statements with great caution. Because they bypass the normal condition that controls the loop's iterations, these statements make code difficult to understand and debug. For this reason, you should avoid using `break` and `continue` whenever possible. However, because they are part of the C++ language, we discuss them briefly in this section.

- The `break` **statement** causes a loop to terminate immediately.
- Program execution jumps to the statement immediately following the loop.
- The `continue` **statement** causes the current iteration of a loop to stop immediately.

- Execution then jumps to the start of the next iteration.
    - In a `while` or `do-while` loop, it jumps to the test expression.
    - In a `for` loop, it jumps to the update expression, and then the test expression is evaluated.

Program 5-25

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

int main()
{
    double value;
    char choice;

    cout << "Enter a number: ";
    cin >> value;
    cout << "This program will raise " << value;
    cout << " to the powers of 0 through 10.\n";
    for (int count = 0; count <= 10; count++)
    {
        cout << value << " raised to the power of ";
        cout << count << " is " << pow(value, count);
        cout << "\nEnter Q to quit or any other key ";
        cout << "to continue. ";
        cin >> choice;
        if (choice == 'Q' || choice == 'q')
            break;
    }
    return 0;
}
```

Program Output

Using break in a Nested Loop

- In a nested loop, a `break` statement only terminates the inner loop it is placed in.
- The outer loop will continue its iterations as normal.

The `continue` Statement

- The `continue` statement skips the remainder of the current loop iteration and proceeds to the next one.
- Any statements in the loop body that appear after the `continue` statement are ignored for that iteration.

Program 5-26

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

int main()
{
    int dvdCount = 1;
    int numDVDs;
    double total = 0.0;
    char current;

    cout << "How many DVDs are being rented? ";
    cin >> numDVDs;

    do
    {
        if ((dvdCount % 3) == 0)
        {
            cout << "DVD #" << dvdCount << " is free!\n";
            continue;
        }
        cout << "Is DVD #" << dvdCount;
        cout << " a current release? (Y/N) ";
        cin >> current;
        if (current == 'Y' || current == 'y')
```

```
            total += 3.50;
        else
            total += 2.50;
    } while (dvdCount++ < numDVDs);

        cout << fixed << showpoint << setprecision(2);
        cout << "The total is $" << total << endl;
        return 0;
}
```

Program Output

Case Study: See the Loan Amortization Case Study on the Computer Science Portal at www.pearsonhighered.com/gaddis.

Review Questions and Exercises

Short Answer

1. Why should you indent the statements in the body of a loop?
2. Describe the difference between pretest loops and posttest loops.
3. Why are the statements in the body of a loop called conditionally executed statements?
4. What is the difference between the `while` loop and the `do-while` loop?
5. Which loop should you use in situations where you wish the loop to repeat until the test expression is false, and the loop should not execute if the test expression is false to begin with?
6. Which loop should you use in situations where you wish the loop to repeat until the test expression is false, but the loop should execute at least one time?

7. Which loop should you use when you know the number of required iterations?
8. Why is it critical that counter variables be properly initialized?
9. Why is it critical that accumulator variables be properly initialized?
10. Why should you be careful not to place a statement in the body of a `for` loop that changes the value of the loop's counter variable?
11. What header file do you need to include in a program that performs file operations?
12. What data type do you use when you want to create a file stream object that can write data to a file?
13. What data type do you use when you want to create a file stream object that can read data from a file?
14. Why should a program close a file when it's finished using it?
15. What is a file's read position? Where is the read position when a file is first opened for reading?

Fill-in-the-Blank

1. To _____ a value means to increase it by one, and to _____ a value means to decrease it by one.
2. When the increment or decrement operator is placed before the operand (or to the operand's left), the operator is being used in _____ mode.
3. When the increment or decrement operator is placed after the operand (or to the operand's right), the operator is being used in _____ mode.
4. The statement or block that is repeated is known as the _____ of the loop.
5. Each repetition of a loop is known as a(n) _____.
6. A loop that evaluates its test expression before each repetition is a(n) _____ loop.
7. A loop that evaluates its test expression after each repetition is a(n) _____ loop.
8. A loop that does not have a way of stopping is a(n) _____ loop.
9. A(n) _____ is a variable that "counts" the number of times a loop repeats.

10. A(n) _____ is a sum of numbers that accumulates with each iteration of a loop.
11. A(n) _____ is a variable that is initialized to some starting value, usually zero, then has numbers added to it in each iteration of a loop.
12. A(n) _____ is a special value that marks the end of a series of values.
13. The _____ loop always iterates at least once.
14. The _____ and _____ loops will not iterate at all if their test expressions are false to start with.
15. The _____ loop is ideal for situations that require a counter.
16. Inside the `for` loop's parentheses, the first expression is the _____, the second expression is the _____, and the third expression is the _____.
17. A loop that is inside another is called a(n) _____ loop.
18. The _____ statement causes a loop to terminate immediately.
19. The _____ statement causes a loop to skip the remaining statements in the current iteration.

Algorithm Workbench

1. Write a `while` loop that lets the user enter a number. The number should be multiplied by 10, and the result stored in the variable `product`. The loop should iterate as long as `product` contains a value less than 100.
2. Write a `do-while` loop that asks the user to enter two numbers. The numbers should be added and the sum displayed. The user should be asked if he or she wishes to perform the operation again. If so, the loop should repeat; otherwise, it should terminate.
3. Write a `for` loop that displays the following set of numbers:

```
0, 10, 20, 30, 40, 50 . . . 1000
```

4. Write a loop that asks the user to enter a number. The loop should iterate 10 times and keep a running total of the numbers entered.
5. Write a nested loop that displays 10 rows of '#' characters. There should be 15 '#' characters in each row.

6. Convert the following `while` loop to a `do-while` loop:

```
int x = 1;
while (x > 0)
{
    cout << "enter a number: ";
    cin >> x;
}
```

7. Convert the following `do-while` loop to a `while` loop:

```
char sure;
do
{
    cout << "Are you sure you want to quit? ";
    cin >> sure;
} while (sure != 'Y' && sure != 'N');
```

8. Convert the following `while` loop to a `for` loop:

```
int count = 0;
while (count < 50)
{
    cout << "count is " << count << endl;
    count++;
}
```

9. Convert the following `for` loop to a `while` loop:

```
for (int x = 50; x > 0; x--)
{
    cout << x << " seconds to go.\n";
}
```

10. Write code that does the following: Opens an output file with the filename Numbers.txt, uses a loop to write the numbers 1 through 100 to the file, then closes the file.

11. Write code that does the following: Opens the Numbers.txt file that was created by the code you wrote in Question 44, reads all of the numbers from the file and displays them, then closes the file.

12. Modify the code that you wrote in Question 45 so it adds all of the numbers read from the file and displays their total.

True or False

1. T F The operand of the increment and decrement operators can be any valid mathematical expression.
2. T F The `cout` statement in the following program segment will display 5:

```
int x = 5;
cout << x++;
```

3. T F The `cout` statement in the following program segment will display 5:

```
int x = 5;
cout << ++x;
```

4. T F The `while` loop is a pretest loop.
5. T F The `do-while` loop is a pretest loop.
6. T F The `for` loop is a posttest loop.
7. T F It is not necessary to initialize counter variables.
8. T F All three of the `for` loop's expressions may be omitted.
9. T F One limitation of the `for` loop is that only one variable may be initialized in the initialization expression.
10. T F Variables may be defined inside the body of a loop.
11. T F A variable may be defined in the initialization expression of the `for` loop.
12. T F In a nested loop, the outer loop executes faster than the inner loop.
13. T F In a nested loop, the inner loop goes through all of its iterations for every single iteration of the outer loop.
14. T F To calculate the total number of iterations of a nested loop, add the number of iterations of all the loops.

15. T F The `break` statement causes a loop to stop the current iteration and begin the next one.

16. T F The `continue` statement causes a terminated loop to resume.

17. T F In a nested loop, the `break` statement only interrupts the loop in which it is placed.

18. T F When you call an `ofstream` object's `open` member function, the specified file will be erased if it already exists.

Find the Errors

Each of the following programs has errors. Find as many as you can.

1.
```cpp
#include <iostream>
using namespace std;
int main()
{
    int num1 = 0, num2 = 10, result;
    num1++;
    result = ++(num1 + num2);
    cout << num1 << " " << num2 << " " << result;
    return 0;
}
```

2.
```cpp
#include <iostream>
using namespace std;
int main()
{
    int num1, num2;
    char again;
    while (again == 'y' || again == 'Y')
        cout << "Enter a number: ";
        cin >> num1;
        cout << "Enter another number: ";
        cin >> num2;
        cout << "Their sum is << (num1 + num2) << endl;
        cout << "Do you want to do this again? ";
        cin >> again;
     return 0;
}
```

3.
```
#include <iostream>
using namespace std;
int main()
{
    int num, bigNum, power, count;
    cout << "Enter an integer: ";
    cin >> num;
    cout << "What power do you want it raised to? ";
    cin >> power;
    bigNum = num;
    while (count++ < power);
        bigNum *= num;
    cout << "The result is << bigNum << endl;
    return 0;
}
```

4.
```
#include <iostream>
using namespace std;
int main()
{
    int numCount, total;
    double average;
    cout << "How many numbers do you want to average? ";
    cin >> numCount;
    for (int count = 0; count < numCount; count++)
    {
        int num;
        cout << "Enter a number: ";
        cin >> num;
        total += num;
        count++;
    }
    average = total / numCount;
    cout << "The average is << average << endl;
    return 0;
}
```

5.
```
#include <iostream>
using namespace std;
int main()
{
       int choice, num1, num2;
    do
```

```
    {
        cout << "Enter a number: ";
        cin >> num1;
        cout << "Enter another number: ";
        cin >> num2;
        cout << "Their sum is " << (num1 + num2) << endl;
        cout << "Do you want to do this again?\n";
        cout << "1 = yes, 0 = no\n";
        cin >> choice;
     } while (choice = 1)
     return 0;
}
```

6.
```
#include <iostream>
using namespace std;
int main()
{
    int count = 1, total;
    while (count <= 100)
        total += count;
    cout << "The sum of the numbers 1-100 is ";
    cout << total << endl;
    return 0;
}
```

Programming Challenges

1. Sum of Numbers

   Write a program that asks the user for a positive integer value. The program should use a loop to get the sum of all the integers from 1 up to the number entered. For example, if the user enters 50, the loop will find the sum of 1, 2, 3, 4, . . ., 50.

   Input Validation: Do not accept a negative starting number.

2. Characters for the ASCII Codes

   Write a program that uses a loop to display the characters for the ASCII codes 0 through 127. Display 16 characters on each line.

3. Ocean Levels

Assuming the ocean's level is currently rising at about 1.5 millimeters per year, write a program that displays a table showing the number of millimeters that the ocean will have risen each year for the next 25 years.

4. Calories Burned

   Running on a particular treadmill you burn 3.6 calories per minute. Write a program that uses a loop to display the number of calories burned after 5, 10, 15, 20, 25, and 30 minutes.

   Solving the Calories Burned Problem

5. Membership Fees Increase

   A country club, which currently charges $2,500 per year for membership, has announced it will increase its membership fee by 4 percent each year for the next 6 years. Write a program that uses a loop to display the projected rates for the next 6 years.

6. Distance Traveled

   The distance a vehicle travels can be calculated as follows:

```
distance = speed * time
```

   For example, if a train travels 40 miles per hour for 3 hours, the distance traveled is 120 miles.

   Write a program that asks the user for the speed of a vehicle (in miles per hour) and how many hours it has traveled. The program should then use a loop to display the distance the vehicle has traveled for each hour of that time period. Here is an example of the output:

```
What is the speed of the vehicle in mph? 40
How many hours has it traveled? 3
Hour  Distance Traveled
--------------------------------
 1           40
 2           80
 3          120
```

Input Validation: Do not accept a negative number for speed and do not accept any value less than 1 for time traveled.

7. Pennies for Pay

   Write a program that calculates how much a person would earn over a period of time if his or her salary is one penny the first day and two pennies the second day, and continues to double each day. The program should ask the user for the number of days. Display a table showing how much the salary was for each day, and then show the total pay at the end of the period. The output should be displayed in a dollar amount, not the number of pennies.

   Input Validation: Do not accept a number less than 1 for the number of days worked.

8. Math Tutor

   This program started in Programming Challenge 17, of Chapter 3, and was modified in Programming Challenge 11 of Chapter 4. Modify the program again so it displays a menu allowing the user to select an addition, subtraction, multiplication, or division problem. The final selection on the menu should let the user quit the program. After the user has finished the math problem, the program should display the menu again. This process is repeated until the user chooses to quit the program.

   Input Validation: If the user selects an item not on the menu, display an error message and display the menu again.

9. Hotel Occupancy

   Write a program that calculates the occupancy rate for a hotel. The program should start by asking the user how many floors the hotel has. A loop should then iterate once for each floor. In each iteration, the loop should ask the user for the number of rooms on the floor and how many of them are occupied. After all the iterations, the program should display how many rooms the hotel has, how many of them are occupied, how many are unoccupied, and the percentage of rooms that are occupied. The percentage may be calculated by dividing the number of rooms occupied by the number of rooms.

   

   Note:

   It is traditional that most hotels do not have a thirteenth floor. The loop in this program should skip the entire thirteenth iteration.

Input Validation: Do not accept a value less than 1 for the number of floors. Do not accept a number less than 10 for the number of rooms on a floor.

10. Average Rainfall

    Write a program that uses nested loops to collect data and calculate the average rainfall over a period of years. The program should first ask for the number of years. The outer loop will iterate once for each year. The inner loop will iterate 12 times, once for each month. Each iteration of the inner loop will ask the user for the inches of rainfall for that month.

    After all iterations, the program should display the number of months, the total inches of rainfall, and the average rainfall per month for the entire period.

    Input Validation: Do not accept a number less than 1 for the number of years. Do not accept negative numbers for the monthly rainfall.

11. Population

    Write a program that will predict the size of a population of organisms. The program should ask the user for the starting number of organisms, their average daily population increase (as a percentage), and the number of days they will multiply. A loop should display the size of the population for each day.

    Input Validation: Do not accept a number less than 2 for the starting size of the population. Do not accept a negative number for average daily population increase. Do not accept a number less than 1 for the number of days they will multiply.

12. Celsius to Fahrenheit Table

    In Programming Challenge 12 of Chapter 3, you were asked to write a program that converts a Celsius temperature to Fahrenheit. Modify that program so that it uses a loop to display a table of the Celsius temperatures 0–20, and the Fahrenheit equivalents.

13. The Greatest and Least of These

    Write a program with a loop that lets the user enter a series of integers. The user should enter −99 to signal the end of the series. After all the numbers have been entered, the program should display the largest and smallest numbers entered.

14. Student Line Up

    A teacher has asked all her students to line up according to their first name. For example, in one class Amy will be at the front of the line, and Yolanda will be at the end.

Write a program that prompts the user to enter the number of students in the class, then loops to read that many names. Once all the names have been read, it reports which student would be at the front of the line and which one would be at the end of the line. You may assume that no two students have the same name.

Input Validation: Do not accept a number less than 1 or greater than 25 for the number of students.

15. Payroll Report

    Write a program that displays a weekly payroll report. A loop in the program should ask the user for the employee number, gross pay, state tax, federal tax, and FICA withholdings. The loop will terminate when 0 is entered for the employee number. After the data is entered, the program should display totals for gross pay, state tax, federal tax, FICA withholdings, and net pay.

    Input Validation: Do not accept negative numbers for any of the items entered. Do not accept values for state, federal, or FICA withholdings that are greater than the gross pay. If the sum of state tax + federal tax + FICA withholdings for any employee is greater than gross pay, print an error message and ask the user to reenter the data for that employee.

16. Savings Account Balance

    Write a program that calculates the balance of a savings account at the end of a period of time. It should ask the user for the annual interest rate, the starting balance, and the number of months that have passed since the account was established. A loop should then iterate once for every month, performing the following:

    1. Ask the user for the amount deposited into the account during the month. (Do not accept negative numbers.) This amount should be added to the balance.

    2. Ask the user for the amount withdrawn from the account during the month. (Do not accept negative numbers.) This amount should be subtracted from the balance.

    3. Calculate the monthly interest. The monthly interest rate is the annual interest rate divided by 12. Multiply the monthly interest rate by the balance, and add the result to the balance.

    After the last iteration, the program should display the ending balance, the total amount of deposits, the total amount of withdrawals, and the total interest earned.

Note:

If a negative balance is calculated at any point, a message should be displayed indicating the account has been closed and the loop should terminate.

17. Sales Bar Chart

    Write a program that asks the user to enter today's sales for five stores. The program should then display a bar graph comparing each store's sales. Create each bar in the bar graph by displaying a row of asterisks. Each asterisk should represent $100 of sales.

    Here is an example of the program's output:

```
Enter today's sales for store 1: 1000 Enter
Enter today's sales for store 2: 1200 Enter
Enter today's sales for store 3: 1800 Enter
Enter today's sales for store 4: 800 Enter
Enter today's sales for store 5: 1900 Enter
SALES BAR CHART
(Each * = $100)
Store 1: **********
Store 2: ************
Store 3: ******************
Store 4: ********
Store 5: ******************
```

18. Population Bar Chart

    Write a program that produces a bar chart showing the population growth of Prairieville, a small town in the Midwest, at 20-year intervals during the past 100 years. The program should read in the population figures (rounded to the nearest 1,000 people) for 1900, 1920, 1940, 1960, 1980, and 2000 from a file. For each year, it should display the date and a bar consisting of one asterisk for each 1,000 people. The data can be found in the `People.txt` file.

    Here is an example of how the chart might begin:

```
PRAIRIEVILLE POPULATION GROWTH
(each * represents 1,000 people)
1900 **
```

```
1920 ****
1940 *****
```

19. Budget Analysis

    Write a program that asks the user to enter the amount that he or she has budgeted for a month. A loop should then prompt the user to enter each of his or her expenses for the month and keep a running total. When the loop finishes, the program should display the amount that the user is over or under budget.

20. Random Number Guessing Game

    Write a program that generates a random number and asks the user to guess what the number is. If the user's guess is higher than the random number, the program should display "Too high, try again." If the user's guess is lower than the random number, the program should display "Too low, try again." The program should use a loop that repeats until the user correctly guesses the random number.

21. Random Number Guessing Game Enhancement

    Enhance the program that you wrote for Programming Challenge 20 so it keeps a count of the number of guesses the user makes. When the user correctly guesses the random number, the program should display the number of guesses.

22. Square Display

    Write a program that asks the user for a positive integer no greater than 15. The program should then display a square on the screen using the character 'X'. The number entered by the user will be the length of each side of the square. For example, if the user enters 5, the program should display the following:

    ```
    XXXXX
    XXXXX
    XXXXX
    XXXXX
    XXXXX
    ```

    If the user enters 8, the program should display the following:

    ```
    XXXXXXXX
    XXXXXXXX
    XXXXXXXX
    XXXXXXXX
    ```

```
XXXXXXXX
XXXXXXXX
XXXXXXXX
XXXXXXXX
```

23. Pattern Displays

    Write a program that uses a loop to display Pattern A below, followed by another loop that displays Pattern B.

PATTERN A	PATTERN B
+	++++++++++
++	+++++++++
+++	++++++++
++++	+++++++
+++++	++++++
++++++	+++++
+++++++	++++
++++++++	+++
+++++++++	++
++++++++++	+

24. Using Files—Numeric Processing

    If you have downloaded this book's source code from the Computer Science Portal, you will find a file named Random.txt in the Chapter 05 folder. (The Portal can be found at www.pearsonhighered.com/gaddis.) This file contains a long list of random numbers. Copy the file to your system, then write a program that opens the file, reads all the numbers from the file, and calculates the following:

    1. The number of numbers in the file
    2. The sum of all the numbers in the file (a running total)
    3. The average of all the numbers in the file

The program should display the number of numbers found in the file, the sum of the numbers, and the average of the numbers.

25. Using Files—Student Line Up

    Modify the Student Line Up program described in Programming Challenge 14 so it gets the names from a file. Names should be read in until there is no more data to read. If you have downloaded this book's source code you will find a file named LineUp.txt in the Chapter 05 folder. You can use this file to test the program.

26. Personal Web Page Generator

    Write a program that asks the user for his or her name, then asks the user to enter a sentence that describes himself or herself. Here is an example of the program's screen:

```
Enter your name: Julie Taylor Enter
Describe yourself: I am a computer science major, a member of the Jazz club, and
```

    Once the user has entered the requested input, the program should create an HTML file, containing the input, for a simple webpage. Here is an example of the HTML content, using the sample input previously shown:

```
<html>
<head>
</head>
<body>
   <center>
      <h1>Julie Taylor</h1>
   </center>
   <hr />
   I am a computer science major, a member of the Jazz club,
   and I hope to work as a mobile app developer after I graduate.
   <hr />
</body>
</html>
```

27. Average Steps Taken

    A Personal Fitness Tracker is a wearable device that tracks your physical activity, calories burned, heart rate, sleeping patterns, and so on. One common physical activity that most of these devices track is the number of steps you take each day.

If you have downloaded this book's source code, you will find a file named steps.txt in the Chapter 05 folder. The steps.txt file contains the number of steps a person has taken each day for a year. There are 365 lines in the file, and each line contains the number of steps taken during a day. (The first line is the number of steps taken on January 1, the second line is the number of steps taken on January 2, and so forth.) Write a program that reads the file, then displays the average number of steps taken for each month. (The data is from a year that was not a leap year, so February has 28 days.)