CSE141 Introduction to Programming

Fall'25

Lab #4

Instructor: Ms Sadaf Alvi , Teaching Assistant: Ali Hamza

Instructions for Lab Submission

1. **File Naming and Submission:**

    a. Submit your solution as a zip file named `Lab3_<ERP>.zip`, where <ERP> is your ERP number. For example, if your ERP number is 12345, the file should be named `Lab3_12345.zip`.

    b. Ensure that the zip file:

        i. Contains your source code files named `Task1.cpp`, `Task2.cpp`.

        ii. Does not include unnecessary files like executables.

        iii. Includes a .txt file containing explanations where asked.

    c. Do not submit code in TEXT files.

2. **Code Organization:**

    a. Ensure that the code structure is modular and adheres to the principles of good software development (e.g., separation of concerns).

3. **Code Neatness:**

    a. Ensure your code is well-formatted and easy to read. Use proper indentation and meaningful variable names.

    b. Include appropriate comments to explain the logic where necessary (especially for functions and complex sections of the code).

    c. Add clear explanations where required(or specified) to make the logic of your program easy to follow.

4. **Code Compilation and Functionality:**

    a. Ensure your code compiles without errors or warnings. You should be able to compile and run your code on any system with a standard C++ compiler (e.g., GCC, Clang, MSVC).

    b. Double-check that the functionality of the program meets the requirements outlined in the task.

5. **Additional Instructions:**

    a. Do not use any libraries or features that have not been covered in the course material, unless specifically instructed.

    b. Make sure your code adheres to the principles of good software development (e.g., modular functions, minimal repetition, and no hardcoding).

    c. If you encounter any issues or require clarifications, reach out before the submission deadline.

6. **Deadline:**

    a. Ensure that you submit your lab on time, before the deadline. Late submissions will not be accepted.

    b. Double-check that the zip file is correctly named and contains all necessary files before submitting.

7. **Plagiarism Policy:**

   a. The work submitted must be entirely your own. Any code or content copied from others (including online sources, classmates or AI) will result in an automatic zero for the task.

Lecture Summary

The if statement

Until last week, we have been writing programs that execute statements in sequence. However, in many cases, we need to execute a statement only if a certain condition is true. For example, we may want to print a message only if the user enters a negative number. In such cases, we use the **if** statement. The syntax of the `if` statement is as follows:

```
if (condition) {
    statement;
}
```

The condition is a Boolean expression that evaluates to either `true` or `false`. If the condition is `true`, the statement is executed. Otherwise, the statement is skipped. For example, the following program prints Hello only if the user enters a negative number.

```
#include <iostream>
int main() {
    int n;
    std::cin >> n;
    if (n < 0) {
        std::cout << "Hello\n";
    }
}
```

The if-else statement

Often, we want to execute one statement if the condition is `true` and another statement if the condition is `false`. For example, we may want to print Hello if the user enters a negative number and Hi otherwise. In such cases, we use the **if-else** statement. The syntax of the `if-else` statement is as follows:

```
if (condition) {
    statement1;
} else {
    statement2;
}
```

If the condition is `true`, `statement1` is executed. Otherwise, `statement2` is executed. For example, the following program prints Negative if the user enters a negative number and Positive otherwise.

```
#include <iostream>
int main() {
    int n;
    std::cin >> n;
    if (n < 0) {
        std::cout << "Negative\n";
    } else {
        std::cout << "Non-negative\n";
    }
}
```

The while statement

Often, we want to execute a statement repeatedly as long as a certain condition is true. For example, we may want to print Hello five times. In such cases, we use the **while** statement. The syntax of the `while` statement is as follows:

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

The condition is a Boolean expression that evaluates to either `true` or `false`. If the condition is `true`, the statement is executed. Then, the condition is evaluated again and if it is still `true`, the statement is executed again. This process continues until the condition becomes `false`. For example, the following program prints Hello five times.

```
#include <iostream>
int main() {
    int i = 0;
    while (i < 5) {
        std::cout << "Hello\n";
        i = i + 1;
    }
}
```

The for statement

The **for** statement is a more compact way to write loops. It combines the initialization, condition, and increment/decrement into a single line. The syntax of the `for` statement is as follows:

```
for (initialization; condition; increment/decrement) {
    statement;
}
```

The initialization is executed only once at the beginning. Then, the condition is evaluated. If it is `true`, the statement is executed. After that, the increment/decrement is executed. Then, the condition is evaluated again and the process continues until the condition becomes `false`. For example, the following program prints Hello five times.

```
#include <iostream>
int main() {
    for (int i = 0; i < 5; i = i + 1) {
        std::cout << "Hello\n";
    }
}
```

C++ common mistakes

1. *Gotcha 1.* Will the following code fragment compile? If so, what will it do?

   ```
   int a = 10, b = 18;
   if (a = b) std::cout << "equal";
   else std::cout << "not equal";
   ```

   Solution: It uses the assignment operator `=` instead of the equality operator `==` in the conditional. This code fragment will set the variable `a` to 18 and the result of this statement is an integer, which the C++ compiler converts to a `bool` value `true`. So, it prints `"equal"` without an error.

2. *Gotcha 2.* What does the following code fragment do?

   ```
   bool a = false;
   if (a = true) std::cout << "yes";
   else std::cout << "no";
   ```

   Solution: It prints `"yes"`. Note that the conditional uses `=` instead of `==`. This means that `a` is assigned the value `true`. As a result, the conditional expression evaluates to `true`. For this reason, it is much better style to use `if (a)` or `if (!a)` when testing `bool` values.

3. *Gotcha 3.* What does the following code fragment do?

   ```
   int a = 17, x = 5, y = 12;
   if (x > y);
   {
       a = 13;
       x = 23;
   }
   std::cout << a;
   ```

   Solution: Always prints 13 since there is a spurious semicolon after the `if` statement. Thus, the assignment statement `a = 13;` will be executed even though `(x <= y)`. It is legal (but uncommon) to have a block that does not belong to a conditional statement, loop, or method.

4. *Gotcha 4.* What does the following code fragment do?

   ```
   for (int x = 0; x < 100; x += 0.5) {
       std::cout << x << std::endl;
   }
   ```

   Solution: It goes into an infinite loop printing 0. The compound assignment statement `x += 0.5` is equivalent to `x = (int)(x + 0.5)`

Nested `if-else` statements

It is possible to nest **`if`** statement within another **`if`** statement.

```cpp
#include <iostream>
int main()
{
    std::cout << "Enter the values of x and y: ";
    int x{}, y{};
    std::cin >> x >> y;
    if (x > 0) { // outer if statement
        if (y > 0) { // inner if statement
            std::cout << "Both x and y are positive\n";
        }
        else {
            std::cout << "x is positive but y is not\n";
        }
    }
    else {
        std::cout << "x is not positive\n";
    }
}
```

An **if-else-if** ladder is a series of **if-else** statements where each **else** is followed by another **if** statement.

```cpp
#include <iostream>
int main()
{
    int x{};
    std::cout << "Enter (x, y) coordinates: ";
    std::cin >> x >> y;
    if (x > 0 && y > 0) {
        std::cout << "Quadrant 1\n";
    }
    else if (x < 0 && y > 0) {
        std::cout << "Quadrant 2\n";
    }
    else if (x < 0 && y < 0) {
        std::cout << "Quadrant 3\n";
    }
    else if (x > 0 && y < 0) {
        std::cout << "Quadrant 4\n";
    }
    else {
        std::cout << "On an axis\n";
    }
}
```

The **if** statements are evaluated from top to bottom. The first **if** statement that evaluates to **true** is executed and the rest are skipped. To understand this better, see how the compiler translates the above code:

```
if (x > 0 && y > 0) {
    std::cout << "Quadrant 1\n";
}
else {
    if (x < 0 && y > 0) {
        std::cout << "Quadrant 2\n";
    }
    else {
        if (x < 0 && y < 0) {
            std::cout << "Quadrant 3\n";
        }
        else {
            if (x > 0 && y < 0) {
                std::cout << "Quadrant 4\n";
            }
            else {
                std::cout << "On an axis\n";
            }
        }
    }
}
```

That is, each **else** contains a single **if** statement. This is why we can skip curly braces after each **else** in a ladder.

Nested loops

A loop inside another loop is called a nested loop. The inner loop is executed fully for each iteration of the outer loop.

```
#include <iostream>
int main()
{
    for (int i=1; i <= 5; ++i) {
        for (int j=1; j <= 5; ++j) {
            std::cout << "(" << i << "," << j << ")\t";
        }
        std::cout << '\n';
    }
}
```

The fist loop will iterate from 1 to 5. For each iteration of the outer loop, the inner loop will iterate from 1 to 5. The above code will print the following output:

```
(1,1)   (1,2)   (1,3)   (1,4)   (1,5)
(2,1)   (2,2)   (2,3)   (2,4)   (2,5)
(3,1)   (3,2)   (3,3)   (3,4)   (3,5)
(4,1)   (4,2)   (4,3)   (4,4)   (4,5)
(5,1)   (5,2)   (5,3)   (5,4)   (5,5)
```

To select two different numbers from 1 to 5, we can use two nested loops as follows:

```cpp
#include <iostream>
int main()
{
    for (int i=1; i <= 5; ++i) {
        for (int j=i+1; j <= 5; ++j) {
            std::cout << "(" << i << "," << j << ")\t";
        }
        std::cout << '\n';
    }
}
```

The above code will print the following output:

```
(1,2)   (1,3)   (1,4)   (1,5)
(2,3)   (2,4)   (2,5)
(3,4)   (3,5)
(4,5)
```

Lab Questions

0.1 EASY

1. *Fibonacci with While Loop* Write a program `fibonacci.cpp` that asks for a positive integer n and prints the first n numbers of the Fibonacci sequence using a while loop.

   Sample execution:

   ```
   Enter n: 7
   Fibonacci sequence: 0 1 1 2 3 5 8
   ```

2. *Multiplication Table* Write a program `table.cpp` that prints the multiplication table of an integer n up to 10. *Sample execution:*

   ```
   Enter a number: 5
   5 x 1 = 5
   5 x 2 = 10
   ...
   5 x 10 = 50
   ```

3. *Geometric Series* Write a program `geometric.cpp` that takes a base r and integer n and computes the geometric sum like this:

   $$S = 1 + r + r^2 + \cdots + r^n$$

   Sample execution:

   ```
   Enter r and n: 2 4
   Geometric sum: 31
   ```

0.2 MEDIUM

4. *Ball Bounce Simulation* Write a program `bounce.cpp` that reads the initial height from which a ball is thrown and displays after how many bounces the ball reaches a height less than $\frac{1}{4}$ of the initial height.

   The program should prompt the user to enter the decrease height ratio (%). For example, the input value `0.1` means that each time the ball hits the ground it bounces up to a height 10% less than the previous one.

   Sample execution:

```
Enter initial height: 100
Enter decrease ratio: 0.1
Number of bounces: 13
```

5. *GCD and LCM Calculator* Write a program called `gcd_lcm.cpp` that accepts two integers as input and computes both their **Greatest Common Divisor (GCD)** and **Least Common Multiple (LCM)**.

   - The **GCD** of two integers a and b is the largest integer that divides both numbers.
   - The **LCM** of two integers a and b is the smallest positive integer that is divisible by both numbers.
   - One efficient way to compute the GCD of two numbers is to use Euclid's algorithm, which states the following: *If $b = 0$, then the GCD of a and b is absolute value of a. Otherwise, the GCD of a and b is the same as the GCD of b and $(a \bmod b)$.*

     For example, we will compute the GCD of 252 and 105.

$$\begin{aligned} &= GCD(105, 252\%105) = GCD(105, 42) \\ &= GCD(42, 105\%42) = GCD(42, 21) \\ &= GCD(21, 42\%21) = GCD(21, 0) = 21 \end{aligned}$$

   - Once the GCD is found, compute the LCM using the relationship:

$$\text{LCM}(a, b) = \frac{|a \times b|}{\text{GCD}(a, b)}$$

$$\text{LCM}(252, 105) = \frac{252 \times 105}{21} = 1260$$

0.3 HARD

6. *Two Heads Simulation* Write a program `two_heads.cpp` that simulates flipping a fair coin repeatedly until two heads in a row appear.

   - Repeat the experiment many times (e.g., 1000 trials) and compute the average number of flips required.
   - *Hint:* Use a loop to run many trials, keep a running sum of flips, then divide by the number of trials to find the average.

7. *Happy Family Simulation* A couple keeps having children until they have at least one boy and one girl. Assume probability of a boy or girl = $1/2$. Write a program `happy_family.cpp` that runs this experiment many times (e.g., 1000 families) and computes the average number of children required. *Hint:*

   - Use a loop to simulate many families.
   - For each family, simulate births until both sexes appear.
   - Keep track of the number of children.
   - At the end, compute the average:

$$\text{average} = \frac{\text{total children across all families}}{\text{number of families}}$$

Bonus Questions

8. *Power Table* Write a program `power_table.cpp` that prints powers of an integer n from n^1 up to n^{10}. (Similar to multiplication table question no 7).

```
Enter a number: 5
5 ^ 1 = 5
5 ^ 2 = 25
5 ^ 3 = 125
...
5 ^ 5 = 3125
```

9. *Harmonic Number* Write a program `harmonic.cpp` that computes the n-th harmonic number:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

Sample execution:

```
Enter n: 5
Harmonic number: 2.28333
```