CSE141 Introduction to Programming
Fall'25

Lab #3

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";
    }
}
```

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.

These are the kinds of operators and conversions you will likely use while programming.

1. Bitwise Operators

Bitwise operators work at the binary (bit) level of integers.

Operator	Name	Example (in binary)
&	AND	6 & 3 = 2 (110 & 011 = 010)
|	OR	6 | 3 = 7 (110 | 011 = 111)
^	XOR	6 ^ 3 = 5 (110 ^ 011 = 101)
~	NOT	$\tilde{6}$ = -7
«	Left Shift	6 « 1 = 12 (110 → 1100)
»	Right Shift	6 » 1 = 3 (110 → 011)

2. Relational Operators

Relational operators compare values and return a boolean result (`true` or `false`).

```
==   Equal to
!=   Not equal to
>    Greater than
<    Less than
>=   Greater than or equal to
<=   Less than or equal to
```

Example:

```
int a = 5, b = 10;
cout << (a < b);   // true (1)
cout << (a == b);  // false (0)
```

3. Type Casting

Type casting converts a variable from one data type to another.
Implicit Casting (Type Promotion): Done automatically by the compiler.

```
int a = 5;
double b = a;   // int promoted to double
```

Explicit Casting: Manually specified by the programmer.

```
double x = 5.7;
int y = (int)x;  // explicit cast, y = 5
```

4. Increment and Decrement Operators

Used to increase or decrease a variable by 1.

```
++a   Pre-increment: increment first, then use the value
a++   Post-increment: use the value first, then increment
--a   Pre-decrement: decrement first, then use the value
a--   Post-decrement: use the value first, then decrement
```

Example:

```
int a = 5;
cout << ++a;  // 6
cout << a++;  // 6 (a becomes 7 after this)
```

5. Ternary Operator

A shorthand for simple if–else conditions.

```
(condition) ? value_if_true : value_if_false;
```

Example:

```
int age = 18;
string result = (age >= 18) ? "Adult" : "Minor";
```

Lab Questions

1. *Even or Odd* Write a program `even_odd.cpp` that takes an integer as input and prints whether the number is Even, Odd, or Zero. *Sample execution:*

```
Enter an integer: 0
Zero

Enter an integer: 7
Odd

Enter an integer: 10
Even
```

2. *Elevator Floor Control System*

   Imagine you are programming the logic for a simple elevator system in a building with 10 floors (0–10). The elevator can move up or down depending on the passenger's request.

Problem Statement

Write a program `elevator.cpp` that:

1. Starts with the elevator at a given **current floor** (ask the user to enter it).
2. Asks the user for the **direction of movement**:
   - Enter `U` for Up.
   - Enter `D` for Down.
   - If user input anything else, output "Invalid Input".
3. Uses `if - else` statements to update the elevator's position:
   - If the direction is Up, increment the floor (`++`).
   - If the direction is Down, decrement the floor (`-`).
   - Hint: You are expected to make use of increment (`++`) and decrement (`--`) operators here.
4. Checks the building limits:
   - The elevator cannot go above the 10th floor.
   - The elevator cannot go below the ground floor (0).
5. Prints the new floor number or an error message if the move is not allowed.

Example Runs

```
Enter current floor: 5
Enter direction (U/D): U
Elevator moved to floor: 6
```

```
Enter current floor: 10
Enter direction (U/D): U
Error: Cannot go above 10th floor.

Enter current floor: 0
Enter direction (U/D): D
Error: Cannot go below ground floor.
```

3. *XOR Encryption and Decryption*

   As you have learned bitwise operators in class, there is a very useful operator called **XOR** (^). This operator has applications in encryption due to its **reversible property**:

   $$a \oplus b \oplus b = a$$

   In other words, if we encrypt some data by applying XOR with a key, we can get back the original data by applying XOR with the same key again.

   Write a program that:

   1. Asks the user to enter a **single character** (letter) and a **key** (integer).
   2. Encrypts the character by applying XOR (^) with the key.
   3. Decrypts the encrypted value by applying XOR (^) with the same key again.
   4. Make sure to display both the encrypted and decrypted results.

   ### Example Run

   ```
   Enter a character: A
   Enter key: 5
   Encrypted value: D
   Decrypted value: A
   ```

4. *BMI Calculator*

   The Body Mass Index (BMI) is a simple measure of body weight relative to height. It is calculated using the formula:

   $$\text{BMI} = \frac{\text{weight (kg)}}{\text{height(m)*height(m)}}$$

   Based on the BMI value, the following categories are used:

Category	BMI Range
Underweight	< 18.5
Normal	$18.5 - - 24.9$
Overweight	$25.0 - - 29.9$
Obese	≥ 30.0

Write a program `bmi.cpp` that asks the user to enter their weight (in kilograms) and height (in meters), computes the BMI using the formula above and determines the BMI category using the table provided.

1. Implement the solution in two ways:
   - Using `if - else if` statements.
   - Using the `ternary (?:)` operator.
2. Submit both implementation in seperate .cpp files as part 1 and part 2.

Example Run

```
Enter weight (kg): 70
Enter height (m): 1.75
BMI = 22.9
Category: Normal
```

5. *Implicit vs Explicit Conversion*

   In C++, type conversions can be **implicit** (automatic: done automatically by the compiler) or **explicit** (done manually using type casting). These conversions affect not only calculations but also decision-making when used inside `if-else` statements.

   Write a program that:

   1. Asks the user to enter two integers: `totalMarks` and `obtainedMarks`.
   2. Calculates the percentage in two different ways:
      - **Implicit conversion:** Store the result of `(obtainedMarks / totalMarks) * 100` in a `double`.
      - **Explicit conversion:** Store the result of `((double) obtainedMarks / totalMarks) * 100` in another `double`.
   3. Uses an `if-else` statement to check whether the student has passed. A student passes if the percentage is greater than or equal to 50.
      - First, test the condition using the implicit conversion result.
      - Then, test the condition using the explicit conversion result.
   4. Displays the result of both checks to show the difference.
   5. This program will help you build a strong understanding of how type conversion works, the difference between implicit and explicit, and how they can affect a program.

Example Run

```
Enter total marks: 40
Enter obtained marks: 27

Using implicit conversion:
Percentage = 0
```

```
Result = Fail

Using explicit conversion:
Percentage = 67.5
Result = Pass
```

6. *Quadratic Equation Roots* The two roots of a quadratic equation can be obtained using the following formula:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

where $b^2 - 4ac$ is called the discriminant of the quadratic equation. If it is positive, the equation has two real roots. If it is zero, the equation has one root. If it is negative, the equation has no real roots.

Write a program `quadratic.cpp` that prompts the user to enter values for `a`, `b`, and `c` and displays the result based on the discriminant. If the discriminant is positive, display two roots. If the discriminant is 0, display one root. Otherwise, display "The equation has no real roots."

7. *Two-Digit Guessing Game* Write a program `guess.cpp` where the computer generates a random two-digit number (10–99). The user enters a guess. *Award rules:*

   - Exact match → 50,000 rupees
   - All digits match, but in different order → 10,000 rupees
   - One digit matches → 1,000 rupees
   - Otherwise → Better luck next time!

8. *Days in Month* Write a program `days_in_month.cpp` that asks the user to enter a **month number (1–12)** and a **year**. The program should print how many days are in that month.

   ### Rules

   - Months with 31 days: January, March, May, July, August, October, December.
   - Months with 30 days: April, June, September, November.
   - February (month 2) has 28 days normally, but 29 days if the given year is a leap year.
   - A year is a leap year if it is divisible by 4 but not by 100, unless it is also divisible by 400.

     ### Examples of Leap Years

     – 2024 → Leap year
     – 1900 → NOT a leap year
     – 2000 → Leap year
     – 2023 → NOT a leap year

Example Runs

```
Enter month (1-12): 2
Enter year: 2024
February 2024 has 29 days.

Enter month (1-12): 2
Enter year: 2023
February 2023 has 28 days.

Enter month (1-12): 11
Enter year: 2023
November 2023 has 30 days.
```

Bonus Questions (Not marked – just for fun and extra practice!)

9. *Rolling Dice* Write a program `roll_dice.cpp` that simulates rolling a fair six-sided die (random integer between 1 and 6) and prints the result. *Sample execution:*

```
You rolled: 4
```

10. *Hurricane Categories*

    The strength of hurricanes is classified using the **Saffir–Simpson Hurricane Wind Scale**. The scale assigns a category (1–5) based on the wind speed in miles per hour (mph).

Category	Wind Speed (mph)
1	74 – 95
2	96 – 110
3	111 – 130
4	131 – 155
5	156 and above

Problem Statement

Write a program `hurricane.cpp` that:

1. Asks the user to enter the wind speed in mph.
2. Determines and prints the hurricane category based on the table above.
3. If the wind speed is below 74 mph, print `"Not a hurricane"`.
4. Implement the solution in two ways:
   - Using `if - else if` statements.
   - Using the `ternary (?:)` operator.

Example Runs

```
Enter wind speed: 120
Category 3

Enter wind speed: 70
Not a hurricane
```

11. *Lottery Game* Write a program `lottery.cpp` where the computer generates a random three-digit number and the user guesses.

    - Exact match → 1,000,000 rupees
    - All digits match → 500,000 rupees
    - One digit matches → 100,000 rupees