WEEK 2

AASMA ABDUL WAHEED (FOIT & CS)

CONTENTS MODULE 2

- **Hello World Program in C++**
  - Processing a C++ program/Execution Flow
  - Syntax of C++ (cout << “literal string \n”)
  - Syntax of C++ (cout << Numeric Constant/Expression)
  - Comments/ Importance of Comments
  - Syntax Errors
  - Syntax vs. Semantics
- **Arithmetic expression**
  - Output Numbers (Literal Constants) (cout << 2 << endl;)
  - Arithmetic Operators (+, -, *, /, %)
  - Defining Expression/Arithmetic Expression
  - Operator Precedence & Associativity
  - Arithmetic Expression evaluation
  - Output value of an Arithmetic Expression (cout << 2*3 << endl;)
- **Problem Solving using Arithmetic Expression (literal constants)**

1. HELLO WORLD PROGRAM IN C++

- The "Hello, World!" program is typically the first program beginners write to understand basic syntax. Here's the simplest version:

```
#include <iostream>   // Enables input/output
using namespace std;  // Allows using cout without std:: prefix

int main() {
    cout << "Hello, World!" << endl;  // Output to the console
    return 0;  // Program ends successfully
}
```

- **#include <iostream>**: This is a preprocessor directive that includes the standard input/output stream library.
- **using namespace std**: This eliminates the need to prefix standard library functions with std::
- **int main()**: This is the entry point of the program. It returns an integer, indicating success or failure of the program.

2. PROCESSING A C++ PROGRAM / EXECUTION FLOW

- The program begins with preprocessing directives (#include), followed by the main() function, which is the starting point.
- Statements inside main() are executed sequentially unless control structures alter the flow.
- Execution flow:

  - **Compilation**: The program is translated to machine language.
  - **Linking**: External libraries are connected.
  - **Execution**: The operating system executes the program.

Programming with the Problem Analysis–Coding–Execution Cycle

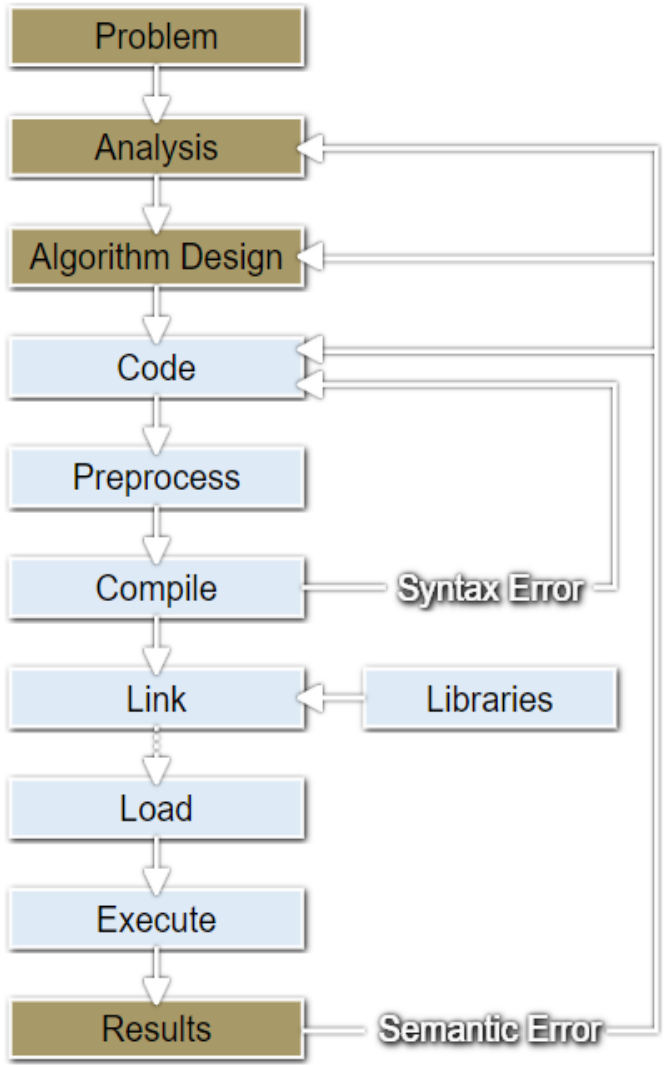

Fig. 1 Development Process

1. *Analyze* the problem
    1. Thoroughly understand the problem and all requirements
        1. Does the program require user interaction?
        2. Does the program manipulate data?
        3. What is the output?
    2. If the problem is complex, divide it into subproblems
        1. Analyze and design *algorithms* for each subproblem
    3. Check the correctness of the algorithm
        1. Can test using sample data
        2. Some mathematical analysis might be required

2. *Implement* the algorithm

1. Once the algorithm is designed and correctness verified, write the equivalent code in a high-level language.
    1. Enter the program using a text editor. This is called the *implementation* of the algorithm.
2. Compile code
3. If the compiler generates errors
    1. Look at the code and remove errors
    2. Run code again through the compiler
4. If there are no syntax errors, the compiler generates equivalent machine code.
    1. The compiler guarantees that the program follows the rules of the language. It does **not** guarantee that the program will run correctly.
5. Linker links machine code with system resources

3. Execution (run the compiles program)
    1. Once compiled and linked, the loader can place the program into the main memory for execution.
    2. The final step is to execute the program.
4. *Maintenance*
    1. Use and modify the program if the problem domain changes.

3. SYNTAX OF C++ (COUT << “LITERAL STRING \N”)

- **cout** is the standard output stream in C++.
- The **<<** operator directs data to cout.
- **"literal string"** is a sequence of characters enclosed in double quotes.
- **\n** is the newline escape sequence that moves the cursor to the next line.
- Example:

```
cout << "This is a string.\n";
```

4. SYNTAX OF C++ (COUT << NUMERIC CONSTANT/EXPRESSION)

- You can output numerical constants or expressions using cout.
- Example:

```
cout << 25;     // Output the number 25
cout << 5 + 3; // Outputs 8, as it evaluates the expression
```

5. COMMENTS/ IMPORTANCE OF COMMENTS

- Single-line comments start with //.
- Multi-line comments are enclosed within /* ... */.
- Comments are essential for:
  - **Code clarity:** Help others (or future you) understand the purpose of code.
  - **Debugging:** Sections of code can be "commented out" to test behaviour.
- Example:

```
// This is a single-line comment
/* This is a multi-line comment */
```

6. SYNTAX ERRORS

- Syntax errors occur when the rules of the language are violated.
- Examples:
  - Missing semicolon (;).
  - Incorrect function definition.
  - Unmatched brackets.

7. SYNTAX VS. SEMANTICS

- **Syntax:** Refers to the correct structure (grammar) of the code.
  - **Example:** Forgetting a semicolon is a syntax error.
- **Semantics:** Refers to the meaning behind the code.
  - **Example:** Correct syntax but incorrect logic, such as adding where you meant to subtract.

8. ARITHMETIC EXPRESSIONS

- **Output Numbers (Literal Constants)**
- You can print literal constants directly:

```
cout << 2 << endl; // Outputs: 2
```

- Arithmetic Operators
  - Addition +
  - Substraction –
  - Multipication *
  - Division /
  - Modulus % (returns the reminder of division)
- Example:

```
cout << 5 + 3 << endl;    // Outputs: 8
cout << 7 % 2 << endl;    // Outputs: 1 (remainder of 7/2)
```

9. DEFINING EXPRESSION/ARITHMETIC EXPRESSION

- An **arithmetic expression** combines numbers and operators:

```
int result = 5 + 2 * 3;
```

10. OPERATOR PRECEDENCE & ASSOCIATIVITY

- Operators have **precedence** determining their order of execution. Multiplication (*), division (/), and modulus (%) have higher precedence than addition (+) and subtraction (-).
- **Associativity** determines how operators of the same precedence level are grouped in the absence of parentheses. Most operators are left-associative.
- Example:

```
int result = 5 + 2 * 3; // Multiplication is done first, result is 11
```

- To change the order of operations, parentheses can be used
- Example:

```
int result = (5 + 2) * 3; // Addition is done first, result is 21
```

11. ARITHMETIC EXPRESSION EVALUATION

- C++ evaluates expressions based on operator precedence and associativity:

```
cout << 5 + 3 * 2 << endl;  // Outputs: 11 (multiplication first)
cout << (5 + 3) * 2 << endl;  // Outputs: 16 (addition first)
```

- You can output the result of an arithmetic expression directly:

```
cout << 2 * 3 << endl;  // Outputs: 6
```

12. PROBLEM SOLVING USING ARITHMETIC EXPRESSION (LITERAL CONSTANTS)

- Arithmetic expressions are often used in solving basic problems. For instance:
  - **Temperature conversion (Celsius to Fahrenheit):**

```
cout << (9.0 / 5.0) * 20 + 32 << endl;  // Convert 20°C to Fahrenheit, outputs 68
```

  - **Length conversion (Feet into Inches):**

```
cout << 6 * 12 << endl;  // Convert 6 feet to inches, outputs 72
```

THANKYOU