CSDC101 Fundamentals of Programming: Key Concepts Reviewer

This reviewer summarises essential concepts from "CSDC101 Fundamentals of Programming" lectures, covering computer systems, programming fundamentals, and an introduction to C++.

--------------------------------------------------------------------------------

**I. Introduction to Programming and Problem Solving (Lec-01)**

**A. Computer Systems Basics**

• **Computer Program:** A set of instructions for a computer to follow.

• **Computer Software:** The collection of programs used by a computer.

• **Hardware:** The actual physical machines that make up a computer installation.

• **Main Classes of Computers:**

    ◦ Personal Computers (PCs)

    ◦ Workstation

    ◦ Mainframe

• **Networks:** A number of computers connected to share resources, such as printers and information.

**B. Computer Organization**

• **Question:** Can you list the five main components of a computer?

    ◦ Based on the provided information, the main components include:

 1. **Input/Output:** Allows communication to the computer and allows the computer to communicate to the user.

2. **Main Memory:** Consists of a long list of numbered locations (memory locations) that contain a string of 0s and 1s. Its contents can change, and it stores instructions and data while a program is running.

            • **Binary Digit (Bit):** A digit that can only be zero or one.

            • **Byte:** An 8-bit portion of memory.

            • **Address:** A number that identifies a memory location.

            • **Random Access Memory (RAM):** Main memory allows direct access to any memory location.

3. **Secondary Memory:** Stores instructions and data between sessions. Data is generally found by searching through other items first (Sequential Access).

 4. **Processor (Central Processing Unit - CPU):** Follows program instructions and performs operations like Add, Subtract, Multiply, Divide, and moving data.

  5. **Computer Software (Operating System - OS):** Allows communication with the computer, allocates resources, and responds to user requests (e.g., Windows, UNIX, Linux).

**C. Data Representation and Interpretation**

• **Binary Nature:** Each memory location contains a string of 0s and 1s.

• **Data Meaning:** How the computer interprets a string like 01000001 (which could represent ‘A’, 65, or an instruction) depends on the current instruction.

**D. Programming Languages and Translation**

• **High-Level Languages:** Common programming languages (e.g., C, C++, Java, Python) that resemble human language and are designed to be easy to read and write.

• **Low-Level Languages:** Languages closer to machine instructions, such as assembly language commands (e.g., ADD X Y Z).

• **Question:** Can you describe the work of a compiler?

    ◦ **Compilers:** Translate high-level language into machine language.

• **Question:** Can you define source code? Define object code?

    ◦ **Source Code:** The original program written in a high-level language.

    ◦ **Object Code:** The translated version of the program in machine language.

• **Linkers:** Combine the object code for programs we write with object code for pre-compiled routines into a single machine language program that the CPU can run.

**E. History of Computing**

• **First Programmable Computer:** Designed by Charles Babbage, who began work in 1822.

• **First Programmer:** Ada Augusta, Countess of Lovelace, a colleague of Babbage.

**F. Algorithms and Program Design**

• **Algorithm:** A sequence of precise instructions which leads to a solution.

• **Program:** An algorithm expressed in a language the computer can understand.

• **Program Design Process Phases:**

◦ **Question:** Can you list the two main phases of the program design process?

        1. **Problem Solving Phase:**

            • **Question:** Can you describe the first step to take when creating a program?

◦ **Make sure the task is completely specified:** Define the input, expected output, and how the output is organized.

            • Develop an algorithm before implementation to save time and test for correctness.

        2. **Implementation Phase:**

            • Translate the algorithm into a programming language.

            • Compile the source code to locate errors in using the language.

            • Run the program on sample data; results may require modifying the algorithm and program.

**G. Algorithmic Representation Tools**

• **Flowchart:** A graphical representation of an algorithm using symbols, shapes, and arrows to demonstrate a process.

    ◦ Common Symbols: Terminal (Start/End), Input/Output, Process/Instruction, Decision, Connector/Arrow.

• **Pseudocode:** An artificial and informal language that helps programmers develop algorithms, where statements showing "dependency" are indented.

--------------------------------------------------------------------------------

**II. Introduction to C++ (Lec-02)**

**A. History of C and C++**

• **Dennis Ritchie:** Developed the C programming language at AT&T Bell Labs in the 1970s. C is a high-level language with many features of a low-level language and was used to maintain UNIX systems.

• **Bjarne Stroustrup:** Developed C++ at AT&T Bell Labs in the 1980s to overcome shortcomings of C. He incorporated Object-Oriented Programming (OOP), and C remains a subset of C++.

**B. C++ Standard Library**

• C++ programs consist of pieces called classes and functions.

• Learning C++ involves learning the language itself and how to use the classes and functions in the C++ Standard Library (STL).

**. Six Phases of Creating and Executing a Program**

1. **Creating/editing a program**

2. **Pre-processing:** A pre-processor program executes automatically before compilation, obeying pre-processor directives that indicate manipulations to be performed before compilation.

3. **Compiling**

4. **Linking**

5. **Loading**

6. **Execution**

    ◦ Programs might not work on the first try.

**D. Basic C++ Program Structure (Example Elements)**

• **Comments:** Lines starting with // or /\* ... \*/ for explanation.

• **Pre-processor Directive:** E.g., #include <iostream>.

◦ **Question:** Explain what #include <iostream> does?

        ▪ It is a pre-processor directive that includes the iostream standard library, which provides input/output functionalities (like cin and cout) to the program.

• **main Function:** The entry point of a C++ program; int indicates it returns an integer.

• **Braces {}:** Define code blocks.

• **Output Statement:** E.g., cout << "Hello";.

• **Return Statement:** E.g., return 0;.

• **std Namespace:** Many standard C++ components (like cout) belong to the std namespace. using namespace std; allows direct use of these names.

**E. Input and Output with cout and cin**

• **cout:** Used for outputting text and values to the console.

    ◦ **Question:** Describe the output of this line: cout << “C++ is easy to understand!”;

        ▪ It prints the text "C++ is easy to understand!" to the screen.

• **cin:** Used for reading input from the user.

    ◦ **Question:** Explain what this line does: cin >> peas\_per\_pod;

        ▪ It reads a value entered by the user from the standard input (keyboard) and stores it into the variable named peas\_per\_pod.

**F. Testing and Debugging**

• **Bug:** A mistake in a program.

• **Debugging:** The process of eliminating bugs.

• **Question:** Can you describe the three kinds of errors?

1. **Syntax Errors:**

        ▪ Violations of the grammar rules of a language (e.g., missing semicolon).

        ▪ **Question:** Can you tell what kind of errors the compiler catches?

            • Syntax errors are discovered by the compiler.

        ▪ **Question:** Can you tell what kind of error is produced if you forget a punctuation symbol such as a semi-colon?

            • A syntax error.

    2. **Run-time Errors:**

        ▪ Errors detected by the computer at the time the program is running (e.g., division by zero).

3. **Logic Errors:**

        ▪ Errors in the algorithm itself, where the computer runs the program without error but produces incorrect results (e.g., computing a sum before reading input).

        ▪ **Question:** Can you tell what type of error is produced when a program runs but produces incorrect results?

            • A logic error.

--------------------------------------------------------------------------------

**III. C++ Basics (Lec-03)**

**A. Variables and Identifiers**

• **Variable:** A named location in memory used to store a value that can be referred to later.

• **Identifier (Names):** Names for variables or other program items. They are a combination of letters, digits, and underscores, and *must start with a letter or an underscore*.

    ◦ *Examples of Valid Identifiers:* x, x1, \_abc, sum.

    ◦ *Examples of Invalid Identifiers:* 12, 3X, %change, data-1.

• **Keywords (Reserved Words):** Words with pre-defined meanings in C++ that cannot be used as names for variables or other items.

**B. Variable Declaration and Assignment**

• **Variable Declaration:** Informs the compiler what size memory location to use for the variable. Multiple variables in a declaration are separated by commas, and each declaration ends with a semicolon.

• **Assignment Statement:** The most direct way to change a variable's value. It always has a variable on the left-hand side of the equal sign and an expression (variable, number, or complex expression) on the right-hand side.

    ◦ *Examples:* total\_weight = one\_weight + number\_of\_bars;, number\_of\_bars = 37;.

**C. Input and Output (Streams, cout, cin)**

• **Input Stream:** A stream of input being fed into the computer for the program to use.

• **Output Stream:** A stream of output generated by the program.

• **cout:** Used to output values of variables and strings of text to the screen. It uses the **insertion**

**operator (<<)** and statements end with a semicolon.

    ◦ *Escape Sequence:* The backslash \ preceding a character (e.g., \n for a new line) tells the compiler that the character following it has a special meaning.

    ◦ *Displaying double values:* The "magic formula" cout.setf(ios::fixed); cout.setf(ios::showpoint); cout.precision(2); will output double values with two digits after the decimal point.

• **cin:** Used for reading input. When reading into a char variable, it skips blanks and line breaks until the first nonblank character.

**D. Data Types**

• **char:** Short for character; represents single symbols like letters, digits, or punctuation marks. Constants of type char are placed inside single quotes (e.g., 'A').

• **bool:** Boolean expressions evaluate to one of two values: true or false.

• **string Class:** Used to process strings. To use it, you must #include <string>.

    ◦ **Concatenation Operator (+):** Used between strings to join them (e.g., "Monday" + "Tuesday" results in "MondayTuesday").

• **Type Compatibilities:** Generally, you cannot store a value of one type directly into a variable of another incompatible type.

**E. Arithmetic and Expressions**

• **Arithmetic Operators:** Include common operators like addition, subtraction, multiplication, and division.

    ◦ **Modulus Operator (%):** Yields the remainder after integer division and can only be used with integer operands.

    ◦ **Integer Division:** An arithmetic operator normally produces a value of the same type as its operands. For example, 1 / 4 is truncated to 0, while 1.0 / 4.0 is 0.25.

• **Parentheses for Grouping:** Used in C++ expressions just like in algebraic expressions to control the order of operations.

• **Operator Precedence:** Determines the order in which operators are evaluated.

**F. Simple Flow of Control**

• **Flow of Control:** Refers to the order in which statements are executed.

• **if statement:** Allows a program to take an alternative action based on whether a condition is true or false. If the condition is true, the statement in the body of the if is executed; otherwise, it is not.

• **if-else statement:** Allows a program to choose between two alternative actions. If the condition is true, the if body is executed; if false, the else body is executed.

• **Equality and Relational Operators:** Used to form conditions within if and if-else statements.a