C Programming Notes UNIT 1

Q1) Define C Language?

The C programming language is a procedural and general-purpose language, C was developed by **Dennis Ritchie** at **Bell Labs** in the early **1970s**

C language is very **powerful** it has been used to develop operating systems, databases, applications, etc. It provides low-level access to system memory.

As C is a **compiled language** a program written in C language must be run through a C compiler to convert it into an executable that a computer can run.

Some pros of C are: Structured, Portable, Rich libraries, Dynamic memory allocation, Pointers Speed.

Q2) History of C Language?

C was developed by **Dennis Ritchie** at Bell Labs in the early **1970s**

It was designed to be a **high-level language** for system programming, such as writing operating systems and device drivers

C quickly gained popularity due to its **efficiency** and ability to produce low-level machine code

The **first official version of C, C89**, was standardized in **1989** by the **American National Standards Institute (ANSI)**

C remains widely used today, particularly for systems programming and embedded systems, as well as for applications that need to be highly optimized for performance

---

Q3) Basic Structure of C Program?

The basic structure of a C program includes the following elements:

Pre-processor Directives: These are instructions to the pre-processor to include or define certain files or symbols before the compilation process begins. These are denoted by the "#" symbol and are typically used for including header files and defining constants.

Main function: The main function is the starting point of the program execution. It must be defined as "int main ()" and is where the program logic is implemented.

Variable Declarations: Variables are used to store data in a program and must be declared before they are used. The declaration typically includes the data type (e.g., int, float, char, etc.) and the variable name.

Statements and Expressions: These are the instructions that are executed in the program. They include assignment statements, control structures (e.g., if-else, for, while), and function calls.

Comments: Comments are used to explain the code and make it more readable for others. They are ignored by the compiler and are denoted by "//" for single-line comments and "/* */" for multi-line comments.

Q4) Process of compiling and running a C program?

The process of compiling and running a C program involves the following steps:

Writing the source code: This involves using a text editor or an Integrated Development Environment (IDE) to write the program in the C programming language. The source code is saved with a .c file extension.

Compiling the source code: The source code is then passed through a compiler, which converts the code into machine-readable instructions (i.e., object code). The command for compiling a C program typically looks like "gcc -o programName programName.c" on Linux or "cl programName.c" on Windows.

Linking the object code: The object code generated by the compiler is then passed through a linker, which combines the object code with any necessary libraries to produce an executable file. This step is typically done automatically by the compiler.

Running the executable: The final step is to run the executable file, which will cause the program to execute. The command for running the executable on Linux is "./programName" and on Windows is "programName.exe".

Some IDEs like Code::Blocks, Visual Studio or Eclipse have a built-in compiler and build system, so the developer only needs to press "run" button to compile, link and execute the program in one step.

Q5) Give brief explanation of C tokens, keywords, identifiers, constants, strings, special symbols, variables, data types, I/O statements. Interconversion of variables.

C Tokens: are the basic building blocks of a C program. They include keywords, identifiers, constants, strings, and special symbols.

Keywords: are predefined words in C that have a specific meaning, such as "int" or "while."

Identifiers: are user-defined words used to name variables, functions, and other objects.

Constants: are fixed values that cannot be changed, such as numbers or characters.

Strings: a string is a sequence of characters that are stored in an array. The last character in the array must be a null character ('\0') which denotes the end of the string.

Special symbols: include punctuation marks like parentheses, braces, and semicolons.

Variables: are used to store data in a program, and can be of different data types such as int, float, char etc.

I/O statements: are used to input and output data from a program, such as scanf and printf.

It is possible to convert one variable from one data type to another, this process is called **Interconversion of variables.**

Q6) What is Computer system, list some components of a computer system.

A computer system is a **combination of hardware and software components** that work together to perform computational tasks. Computer system can range from a simple smartphone to a complex supercomputer, but they all share the same basic components and function in a similar way.

Central Processing Unit (CPU): The brain of the computer, responsible for executing instructions and performing calculations.

Memory: A temporary storage area where the computer stores data and instructions for the CPU to access.

Storage: A long-term storage area where the computer stores user data and programs.

Input devices: Allow the user to input data and instructions into the computer, such as a keyboard or mouse.

Output devices: Allow the computer to display information to the user, such as a monitor or printer.

Q7) Define Computer Languages?

Computer languages are languages used to write instructions for a computer to execute. They are used by programmers to create software, applications, and systems. There are several types of computer languages, each with their own strengths and purposes.

Low-level languages: such as assembly language, are close to the machine code that the computer's processor can execute directly. They are difficult to write and understand but are close to the hardware and provide a high level of control over the computer's resources.

High-level languages: such as C, C++, Java, Python, and JavaScript, are easier for humans to read and write, and are designed to be more portable across different systems. They provide higher-level abstractions and are closer to human languages, making the coding process more intuitive.

Scripting languages: such as Perl, Python, and JavaScript, are used to automate repetitive tasks and glue together different components of a system.

Markup languages: such as HTML, XML, and JSON, are used to describe the structure and layout of documents and data.

Query languages: such as SQL, are used to access and manipulate data stored in databases.

Each language has its own syntax, semantics, and libraries, and is suitable for different types of tasks, such as web development, mobile app development, data analysis and scientific computing.

Q8) Define computing environments?

Computing environments refer to the **different platforms and operating systems** on which a computer program can run. The term "platform" typically refers to the underlying hardware and software infrastructure, while "operating system" refers to the software that manages the hardware resources of a computer.

Examples of computing environments include:

Windows, MacOS and Linux are examples of operating systems that provide a platform for running programs.

Mobile operating systems such as **Android and iOS**, which are used on smartphones and tablets.

Cloud computing platforms such as **Amazon Web Services**, **Microsoft Azure**, and **Google Cloud** Platform, which provide remote access to computing resources and services over the internet.

Each computing environment has its own set of unique features and capabilities, and developers need to take into account the specific requirements of the environment when writing and deploying their code. Some languages and frameworks are more suited to certain environments than others, such as Java for Android or .Net for Windows.

Q9) What is Pre-processor, role of linker, idea of invocation and execution of a programme?

Pre-processor is a program that processes the source code before it is passed to the compiler. It performs tasks such as including header files, expanding macros, and resolving pre-processor directives. The pre-processor examines the code, looks for lines starting with "#" and replaces the macros or performs other operations like conditional inclusion or exclusion of code.

Role of the linker is to link the object code generated by the compiler with the necessary libraries and other files. The linker is responsible for resolving external references, such as function calls, within the object code and linking them with the appropriate library or object file.

Invocation of a program refers to the process of starting or running a program. When a user runs a program, the operating system loads the program into memory, allocates the necessary resources, and starts its execution. This process is known as invocation.

Once a program is invoked, it is executed by the CPU, which retrieves the instructions from memory and carries out the operations specified in the code. Execution of a program refers to the process of the computer carrying out the instructions specified in the program. The program uses the various resources of the computer system, such as memory and input/output devices, as specified in the code.

Q10) Define in brief Operators, arithmetic Operators, relational and logical, assignment operators, increment and decrement operators, bitwise and conditional operators, special operators, operator precedence and associativity, evaluation of expressions, type conversions in expressions

Operators are symbols or keywords that perform specific operations on one or more operands. There are several types of operators in programming languages, including:

Arithmetic operators: Perform mathematical operations such as addition (+), subtraction (-), multiplication (*), division (/), and modulus (%).

Relational operators: Compare two operands and return a Boolean value based on the comparison, such as equal to (==), not equal to (!=), greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=).

Logical operators: Perform logical operations on Boolean operands such as AND (&&), OR (||), and NOT (!).

Assignment operators: Assign a value to a variable, such as the equal sign (=).

Increment and decrement operators: Increase or decrease the value of a variable by 1, such as ++ and --.

Bitwise operators: Perform bit-level operations on operands, such as AND (&), OR (|), XOR (^), and NOT (~).

Conditional operator: Perform a test and return one of two values based on the outcome, such as (condition) ? value_if_true : value_if_false.

Special operators: Perform specific operations such as the ternary operator (?:) and sizeof operator.

Operator precedence and associativity determine the order in which operators are evaluated in an expression. Precedence refers to the order in which operators are applied, and associativity determines the direction in which an expression is evaluated.

When **evaluating expressions**, **type conversions** may be applied to ensure that the operands are of the same type. These type conversions, also known as type casting, can be explicit or implicit and can cause loss of precision or information.

Q11) Define in brief Algorithms, Representation using flowcharts and pseudocode?

Algorithm is a set of instructions that describes a computational process to solve a specific problem or accomplish a specific task. Algorithms can be represented using different techniques, such as flowcharts and pseudocode.

Flowchart is a graphical representation of an algorithm that uses shapes, symbols, and arrows to indicate the flow of the algorithm. Flowcharts are useful for visualizing the logic and structure of an algorithm, and can be used to communicate algorithms to a non-technical audience.

Pseudocode is a simplified representation of an algorithm that uses a combination of natural language and programming constructs. Pseudocode is similar to a programming language but is

not meant to be executed by a computer. It is used to express an algorithm in a way that is easy to understand and can be translated into any programming language.

Both flowcharts and pseudocode are used to represent algorithms at a high level of abstraction, independent of any specific programming language or implementation details. They are useful for designing, analysing, and communicating algorithms before implementing them in code.

Q12) Explain Difference between Application and System software?

Application software:

Refers to software that is designed to perform specific tasks and functions for the user.

Includes software such as word processing, web browsing, and video games.

Focused on providing specific functionality to the end user.

Often installed by the user and can vary depending on their needs.

Examples include Microsoft Word, Google Chrome, and Minecraft.

System software:

Refers to the underlying software that controls and manages the computer's hardware and basic operations.

Includes the operating system, device drivers, and utilities.

Provides a platform for other software to run on.

Typically, pre-installed on a computer and necessary for the proper functioning of the system.

Examples include Windows, Linux, MacOS, and iOS.

Q13) What are Keywords in C, Give some examples.

C is a programming language that has a number of keywords, which are reserved words that have special meaning in the language and cannot be used as variable or function names. Some common keywords in C include:

int: used to declare integer variables
char: used to declare character variables
float: used to declare floating-point variables
double: used to declare double-precision floating-point variables
void: used to declare functions or variables with no type
if: used to make a decision in a program
else: used as a secondary decision in a program
for: used to create a loop
while: used to create a loop
return: used to exit a function and return a value

Q14) What are Identifiers? Give Rules for defining them.

An **Identifier** is **a name given to a variable, function, or any other user-defined item**. Identifiers are used to identify and refer to these items in the code. The rules for defining identifiers in C are as follows:

Identifiers can **only contain letters, digits, and underscores.**

The **first character** of an identifier **must be a letter or an underscore.**

Identifiers are **case-sensitive**, meaning that "myVariable" and "myvariable" would be considered different identifiers.

Identifiers **cannot be the same as a keyword or reserved word** in the C language.

Identifiers can be of **any length**, but it is generally **recommended to keep them short** and meaningful.

Q15) What are different data types in C, Give some Examples?

Data types are used to specify the type of a variable or a value.

The basic data types in C are:

int: used to store integers (whole numbers) such as 1, 2, -3, etc. Example: int age = 25;

float: used to store decimal numbers with a single precision. Example: float price = 12.34;

double: used to store decimal numbers with a double precision. Example: double pi = 3.14159265;

char: used to store a single character, enclosed in single quotes. Example: char letter = 'A';

In addition to these basic data types, C also provides several derived data types, such as arrays, pointers, and structures, which can be used to store more complex data.

Q16) Difference between compiler and interpreter?

Compiler: A compiler is a program that converts source code written in a high-level programming language into machine code that can be executed by a computer.

Compilers typically translate the entire program at once, before the program is executed.

The process of translation happens only once, after that the executable file can be run any number of times.

The output of a compiler is usually an executable file that can be run on any machine that has the same architecture, regardless of whether the compiler is installed on that machine or not.

Examples of compilers include GCC for C and C++.

Interpreter: An interpreter is a program that reads and executes source code, line by line, without first converting it to machine code.

Interpreters typically execute the program as it is being translated, so the program is executed immediately after it is read.

Interpreters need to be installed on the target machine in order to run the program.

The output of an interpreter is not a standalone executable, but the program runs on the fly.

Examples of interpreters include the Python interpreter, and the Java Virtual Machine (JVM) for Java.
