Essentials of C Programming

1. Introduction to C Programming
History & Standards:
C was developed in the early 1970s by Dennis Ritchie at Bell Labs as a system programming language for implementing the UNIX operating system. It evolved from earlier languages like B and BCPL, adding types, structures, and a more powerful syntax. Over time, C became standardized:
- **K&R; C:** Original version described in the book "The C Programming Language" by Kernighan and Ritchie.
- **ANSI C (C89/C90):** Formal standardization that ensured portability.
- **C99:** Added inline functions, variable-length arrays, and new data types.
- **C11:** Improved multithreading support, safer functions.
- **C18:** Latest minor revision, mainly bug fixes.

Building a C Program:
1. **Writing:** You write your code in a `.c` file using an editor.
2. **Compiling:** The compiler checks syntax and converts to object code.
3. **Linking:** Combines your object code with required libraries.
4. **Execution:** The executable runs on the target machine.

Language Basics:
- **Keywords:** Reserved words like `int`, `return`, `if`.
- **Variables:** Named memory locations storing data.
- **Naming Rules:** Start with a letter/underscore, no spaces, case-sensitive.
- **Identifiers:** Names for variables, functions, arrays, etc.
- **Data Types:** Basic (int, float, char, double), Derived (arrays, pointers, structures), Void.
- **Qualifiers:** signed, unsigned, short, long — modify storage size or sign.
- **Constants:** Fixed values: numeric (`42`), char (`'A'`), string (`"Hello"`), enum.
- **Type Conversion:** Implicit (automatic promotion), Explicit (type casting, e.g., `(float)x`).

2. Operators & Flow Control
Operators:
- Arithmetic: +, -, *, /, %
- Relational: ==, !=, >, <, >=, <=
- Assignment: =, +=, -=, *=, /=
- Increment/Decrement: ++, --
- Logical: &&, ||, !
- Bitwise: &, |, ^, ~, <<, >>
- Conditional (Ternary): (condition) ? value1 : value2
- Special: sizeof, comma operator, pointer operators.

Precedence: Determines order of evaluation; associativity decides direction (left-to-right or right-to-left).

Flow Control:
- Conditionals: if, if-else, nested if, else-if ladder, switch-case.
- Loops: for, while, do-while.
- Jump statements: break, continue, goto (rarely used due to readability issues).

3. Preprocessors, Arrays, Strings & Functions
Preprocessors: Run before compilation; directives start with `#`.
- `#define` for constants and macros.
- `#include` to add header files.
- Conditional compilation: `#ifdef`, `#ifndef`.

Arrays:
- 1D: int arr[5] = {1, 2, 3, 4, 5};
- 2D: int mat[3][3];
- Access via index, e.g., arr[0].

Strings:
- Null-terminated character arrays.
- Functions: strlen, strcpy, strcmp, strcat.

Functions:
- Prototype, definition, call.
- Parameters passed by value or by address.
- Can return a value or void.
- Recursion: calling itself until base case.

4. Pointers in C
- Store memory addresses of variables.
- Types: to int, char, float, structures, pointer to pointer.
- NULL pointer: points nowhere.
- Pointer arithmetic: p++, p--.
- Pointer to structure: access with ->.
- Dynamic memory: malloc, calloc, realloc, free.

5. Structures, Unions, File Handling & Best Practices
Structures: Group different data types under one name.
- Access members using dot or arrow.
- Array of structures and nested structures.
- typedef to create type aliases.

Unions: Similar to structures but share memory among members.

File Handling:
- printf, scanf for formatted I/O.
- getchar, putchar for single char I/O.
- File functions: fopen, fclose, fprintf, fscanf, fread, fwrite.
- Command-line arguments: main(int argc, char *argv[])

Best Practices:
- Proper indentation.
- Meaningful names.
- Avoid magic numbers.
- Modular code.
- Check function return values.
- Comment important sections.