Module 2 – Introduction to Programming

o Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today

* The C programming language is one of the most influential and enduring programming languages in computer science. Developed in the early 1970s, C has played a crucial role in shaping modern computing, serving as the foundation for operating systems, embedded systems, and high-performance applications. This essay explores the history and evolution of C, its significance in the development of computing, and the reasons why it remains relevant today.

Importance of C

C has had a profound impact on computing for several reasons:

* **Foundation for Operating Systems**: Unix, Linux, Windows kernels, and macOS components are written in C (and C++). Its ability to interact closely with hardware makes it ideal for system programming.
* **Influence on Other Languages**: Many modern languages, including C++, Java, Python, and JavaScript, borrow syntax and concepts from C.
* **Embedded Systems and IoT**: Due to its efficiency, C is widely used in embedded systems, microcontrollers, and IoT devices where performance and memory management are critical.
* **High-Performance Applications**: C is used in game engines, real-time systems, and high-frequency trading due to its speed and predictability.

Why C Is Still Used Today

Despite being over 50 years old, C remains indispensable for several reasons:

1. **Performance**: C compilers produce highly optimized machine code, making it faster than many higher-level languages.
2. **Hardware Interaction**: C allows direct memory manipulation via pointers, making it essential for firmware and driver development.
3. **Portability**: C programs can run on almost any platform with minimal modifications.
4. **Legacy Code**: Many critical systems (e.g., databases, networking tools) are written in C, requiring ongoing maintenance.
5. **Educational Value**: Learning C helps programmers understand memory management, data structures, and how computers work at a fundamental level.

2. Setting Up Environment

o Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like DevC++, VS Code, or CodeBlocks.

**For Windows:**

* 1. **Download MinGW** (Minimalist GNU for Windows)
     + Go to <https://www.mingw-w64.org/downloads/>
     + Click "MingW-W64-builds" → Download the installer
  2. **Run the Installer**
     + Select:
       - Version: **latest**
       - Architecture: **x86\_64** (for 64-bit Windows)
       - Threads: **posix**
       - Exception: **seh**
     + Set install location (e.g., C:\mingw64)
  3. **Add to System PATH**
     + Open Start Menu → type "Environment Variables"
     + Click "Edit the system environment variables"
     + Click "Environment Variables" → Under "System variables", find "Path" → Edit
     + Add new path: C:\mingw64\bin
  4. **Verify Installation**

-> Open Command Prompt (Win+R → type cmd)

-> Type:

-> gcc --version

-> If you see version info, it's working!

3.Basic Structure of a C Program

o Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.

1. Documentation Section (Comments)

Comments are used to explain the code and are ignored by the compiler. They help make the code more readable.

/\* This is a multi-line comment

It can span several lines\*/

// This is a single-line comment

2. Header Files

Header files contain declarations of functions and macros that are used in the program. They are included using the `#include` directive.

#include <stdio.h> // Standard Input/Output functions

#include <math.h> // Math functions

3. Main Function

Every C program must have a `main()` function - it's the entry point where program execution begins.

**int main() {**

**// Program code goes here**

**return 0; // Indicates successful execution**

**}**

4. Data Types and Variables

C has several basic data types for storing different kinds of values.

**int age = 25; // Integer (whole number)**

**float price = 19.99; // Floating point number**

**double pi = 3.14159265; // Double precision floating point**

**char grade = 'A'; // Single character**

**Complete Example Program**

/\*

Simple C Program

Demonstrates basic structure

\*/

#include <stdio.h> // For printf() function

// Main function - program entry point

int main() {

// Variable declarations

int num1 = 10;

int num2 = 20;

int sum;

float average;

// Calculation

sum = num1 + num2;

average = sum / 2.0;

// Output results

printf("The sum is: %d\n", sum);

printf("The average is: %.2f\n", average);

return 0; // End program

}

Key Components Explained:

1. \*\*Comments\*\*: Document what the code does

2. \*\*Header Files\*\*: Provide necessary function declarations

3. \*\*main() function\*\*: Mandatory entry point

4. \*\*Variables\*\*: Store data of different types

5. \*\*Statements\*\*: Perform operations and control flow

6. \*\*return 0\*\*: Indicates successful program termination

Common Data Types in C:

- `int`: Integer numbers (e.g., 5, -3, 0)

- `float`: Single-precision floating point (e.g., 3.14, -0.001)

- `double`: Double-precision floating point (more precise than float)

- `char`: Single character (e.g., 'A', 'z', '9')

- `void`: Represents absence of type

4. Operators in C

o Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators

**1. Arithmetic Operators**

Used to perform mathematical calculations.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| + | Addition | a + b |
| - | Subtraction | a - b |
| \* | Multiplication | a \* b |
| / | Division | a / b |
| % | Modulus (Remainder) | a % b |

**2. Relational Operators**

Used to compare two values and return 1 (true) or 0 (false).

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| == | Equal to | a == b |
| != | Not equal to | a != b |
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater than or equal | a >= b |
| <= | Less than or equal | a <= b |

**3. Logical Operators**

Used to combine multiple conditions.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| && | Logical AND | (a > 0) && (b < 10) |
| ! | Logical NOT | !(a == b) |

**4. Assignment Operators**

Used to assign values to variables.

| **Operator** | **Description** | **Example** | **Equivalent** |
| --- | --- | --- | --- |
| = | Simple assignment | a = 5 | a = 5 |
| += | Add and assign | a += 3 | a = a + 3 |
| -= | Subtract and assign | a -= 2 | a = a - 2 |
| \*= | Multiply and assign | a \*= 4 | a = a \* 4 |
| /= | Divide and assign | a /= 2 | a = a / 2 |
| %= | Modulus and assign | a %= 3 | a = a % 3 |

**5. Increment & Decrement Operators**

Used to increase or decrease a variable's value by 1.

| **Operator** | **Description** | **Example** | **Effect** |
| --- | --- | --- | --- |
| ++ | Increment | a++ | Post-increment |
| ++ | Increment | ++a | Pre-increment |
| -- | Decrement | a-- | Post-decrement |
| -- | Decrement | --a | Pre-decrement |

**6. Bitwise Operators**

Used to perform operations on individual bits.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| & | Bitwise AND | a & b |
| ^ | Bitwise XOR | a ^ b |  |  |
| ~ | Bitwise NOT | ~a |  |  |
| << | Left shift | a << 2 |  |  |
| >> | Right shift | a >> 2 |  |  |

**7. Conditional (Ternary) Operator**

A shorthand for if-else statements.

**Syntax:**  
 condition ? expression1 : expression2

5.Control Flow Statements in C

o Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

**1.**if**Statement**

* Executes a block of code **only if** the condition is true.
* **Syntax:**

if (condition) {

// code to execute if condition is true

}

**Example:**

#include <stdio.h>

#include<conio.h>

int main() {

Int age = 18;

if (age >= 18) {

printf("You are eligible to vote.\n");

}

return 0;

}

**2.**if-else**Statement**

* Executes one block if the condition is **true**, otherwise executes the else block.
* **Syntax:**

if (condition) {

// code if condition is true

} else {

// code if condition is false

}

**Example:**

#include <stdio.h>

#include<conio.h>

int main() {

int num = 10;

if (num % 2 == 0) {

printf("%d is even.\n", num);

} else {

printf("%d is odd.\n", num);

}

return 0;

}

**3. Nested**if-else**Statement**

* Used when multiple conditions need to be checked sequentially.
* **Syntax:**

if (condition1) {

// code if condition1 is true

} else if (condition2) {

// code if condition2 is true

} else {

// code if all conditions are false

}

**Example:**

#include <stdio.h>

#include<conio.h>

int main() {

int marks = 85;

if (marks >= 90) {

printf("Grade: A\n");

} else if (marks >= 80) {

printf("Grade: B\n");

} else if (marks >= 70) {

printf("Grade: C\n");

} else {

printf("Grade: D\n");

}

return 0;

}

**4.**switch**Statement**

* Used when multiple conditions are based on the **same variable**.
* More efficient than multiple if-else statements for fixed values.
* **Syntax:**

switch (expression) {

case value1:

// code if expression == value1

break;

case value2:

// code if expression == value2

break;

default:

// code if no case matches

}

**Example:**

#include <stdio.h>

#include<conio.h>

int main() {

char choice = 'B';

switch (choice) {

case 'A':

printf("You chose option A.\n");

break;

case 'B':

printf("You chose option B.\n");

break;

case 'C':

printf("You chose option C.\n");

break;

default:

printf("Invalid choice.\n");

}

return 0;

}