**SOFTWARE ENGINEERING ASSIGNMENTS**

**Module-2**

**Introduction to Programming**

**Over View of C programming**

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

**-> Introduction of C**

C programming is one of the most influential and widely used programming languages in the history of computer science. Developed in the early 1970s, C has played a key role in the development of modern software and operating systems. Its simplicity, efficiency, and close interaction with hardware make it a timeless language in the programming world.

* **History and Evolution**
* · **Origin (Early 1970s)**:  
  C was developed by **Dennis Ritchie** at **Bell Labs** in **1972**. It evolved from earlier languages like **B** and **BCPL (Basic Combined Programming Language)**. The main goal was to create a language for writing operating systems.
* · **Unix and C**:  
  One of C's biggest milestones was that it was used to rewrite the **UNIX operating system**, which was originally written in assembly language. This made UNIX portable, faster to develop, and easier to maintain.
* · **Standardization (1980s)**:  
  In the mid-1980s, the **American National Standards Institute (ANSI)** established a standard version of C, known as **ANSI C** or **C89**. Later versions like **C99**, **C11**, and **C18** added more features like inline functions, variable-length arrays, and better support for multi-threading.

**Importance of C Programming**

1. Foundation for Many Languages:  
   C has influenced many other popular languages like C++, Java, C#, Objective-C, and even Python. Learning C helps understand how these languages work at a deeper level.
2. System-Level Programming:  
   C allows direct manipulation of memory using pointers, making it ideal for writing operating systems, device drivers, and embedded software.
3. High Performance:  
   C code is fast and efficient because it is compiled directly to machine code and gives programmers low-level control over system resources.
4. Portability:  
   Programs written in C can run on different machines with little or no modification, making it perfect for cross-platform development.

**Why C is Still Used Today**

* Despite being over 50 years old, C remains relevant due to:
* **Embedded Systems**:  
  Most embedded devices (like washing machines, routers, sensors) use C due to its speed and efficiency.
* **Operating Systems**:  
  Windows, Linux, macOS, and Android have large portions written in C.
* **Education**:  
  C is widely taught in computer science courses because it helps students understand computer memory, logic, and data structures.
* **Open Source Projects**:  
  Many popular open-source tools and systems like **Git**, **Linux Kernel**, and **MySQL** are written in C.

1. **Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.**

* Here are three real-world applications where C programming is extensively used:
* **1. Embedded Systems**
* **Example:** *Washing machines, microwave ovens, automotive control systems*
* **Why C is used:**
* C provides low-level access to memory and hardware.
* It allows direct control over microcontrollers and sensors.
* It produces fast and efficient code with minimal overhead.
* **Real-World Use:**
* In *automobiles*, C is used for engine control units (ECUs).
* In *medical devices*, C programs monitor and control sensors (like ECG machines).
* **2. Operating Systems**
* **Example:** *Linux, Windows, macOS (kernel parts)*
* **Why C is used:**
* C can interact directly with hardware through memory management and pointers.
* It allows fine-grained control over system resources.
* It is highly portable across hardware platforms.
* **Real-World Use:**
* The **Linux kernel** is almost entirely written in C.
* Many parts of **Windows** and **macOS** system code are developed using C.
* **3. Game Development**
* **Example:** *Game engines, handheld gaming devices*
* **Why C is used:**
* C offers high performance and low-level control required for real-time graphics and input/output.
* It integrates well with graphics libraries like OpenGL and DirectX.
* **Real-World Use:**
* Older and some modern **game engines** (e.g., *id Tech*, *Unreal Engine's early versions*) use C/C++.
* **Nintendo** consoles and handhelds use C in their firmware and SDKs.

**Setting Up Environment**

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

* Step 1: Install GCC Compiler

>Go to the official website: https://jmeubank.github.io/tdm-gcc/

> Download the installer (e.g., tdm-gcc-10.3.0.exe)

> Run the installer and choose:

* "Create" for a new installation
* Select “MinGW/GCC” compiler suite
* Proceed with default settings unless specific needs

> Wait for installation to complete

> Done! GCC is now installed.

Step 2: Choose and Install an IDE

-Option A: Dev-C++ (Easy & Beginner Friendly)

-Option B: Code::Blocks (IDE with built-in GCC)

-option C: Visual Studio Code (Powerful for future use)

* C language “hello world” program.

#include <stdio.h>

int main(){

printf(“hello world”);

}

**Basic Structure of a C Program**

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

1. Header Files

-> These are included at the beginning of a C program using #include. They allow access to standard functions (like printf, scanf etc.).

2. Main Function

-> The main() function is the entry point of every C program. The execution starts from here.

3. Comments

-> Used to explain the code. They are ignored during execution.

-> Single-line comment: // comment here

-> Multi-line comment:\*/ comment here /\*

4. Data Types

* Define the type of data a variable can store.

| **Data Type** | **Description** | **Example Values** |
| --- | --- | --- |
| int | Integer | 10, -5, 0 |
| float | Floating point | 3.14, -2.5 |
| char | Character | 'A', 'z' |
| double | Double-precision float | 3.14159 |

5. Variables

-> Used to store data. You must declare the data type before using a variable.

Example code :

#include <stdio.h> // Header file

int main() { // Main function

// Variable declaration

int age = 18;

float marks = 89.5;

char grade = 'A';

// Printing values

printf("Age: %d\n", age);

printf("Marks: %.2f\n", marks);

printf("Grade: %c\n", grade);

return 0; // End of program

}

**Operators in C**

1. **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 basic mathematical operations.

| **Operator** | **Description** | **Example (a=10, b=5)** | **Result** |
| --- | --- | --- | --- |
| + | Addition | a + b | 15 |
| - | Subtraction | a - b | 5 |
| \* | Multiplication | a \* b | 50 |
| / | Division | a / b | 2 |
| % | Modulus (remainder) | a % b | 0 |

**2. Relational Operators**

Used to compare two values. Returns either **true (1)** or **false (0)**.

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| == | Equal to | a == b | 0 |
| != | Not equal to | a != b | 1 |
| > | Greater than | a > b | 1 |
| < | Less than | a < b | 0 |
| >= | Greater or equal to | a >= b | 1 |
| <= | Less or equal to | a <= b | 0 |

**3. Logical Operators**

Used to combine multiple relational expressions.

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| && | Logical AND | (a > 5 && b < 10) | 1 |
| ` |  | ` | Logical OR |
| ! | Logical NOT (negation) | !(a == b) | 1 |

**4. Assignment Operators**

Used to assign values to variables.

| **Operator** | **Description** | **Example** | **Equivalent To** |
| --- | --- | --- | --- |
| = | Simple assignment | a = 10 | — |
| += | Add and assign | a += 5 | a = a + 5 |
| -= | Subtract and assign | a -= 3 | a = a - 3 |
| \*= | Multiply and assign | a \*= 2 | a = a \* 2 |
| /= | Divide and assign | a /= 4 | a = a / 4 |
| %= | Modulus and assign | a %= 2 | a = a % 2 |

**5. Increment/Decrement Operators**

Used to increase or decrease a variable by 1.

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| ++ | Increment (prefix/postfix) | ++a or a++ | a = a + 1 |
| -- | Decrement (prefix/postfix) | --a or a-- | a = a - 1 |

**6. Bitwise Operators**

Work on bits and perform bit-by-bit operations.

| **Operator** | **Description** | **Example (a=5, b=3)** | **Result** |
| --- | --- | --- | --- |
| & | AND | a & b (0101 & 0011) | 1 |
| ` | ` | OR | `a |
| ^ | XOR | a ^ b | 6 |
| ~ | NOT (1's comp.) | ~a | -6 |
| << | Left shift | a << 1 | 10 |
| >> | Right shift | a >> 1 | 2 |

**7. Conditional (Ternary) Operator**

Used as a shorthand for if-else.

c

Copy code

condition ? expr1 : expr2;

**Control Flow Statements in C**

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

* Decision making statements in c allow the program to execute certain parts of code based on specific conditions. These are crucial for implementing logic in a program.
* **1.if statement**

-Used to execute a block of code only if a specified condition is true.

Syntax:

if (condition) {

// Code to execute if condition is true

}

Example:

int num = 10;

if (num > 5) {

printf("Number is greater than 5\n");

}

**2. if-else statement**

-Execute one block of code if the condition is true, another block if the condition is false.

Syntax:

if (condition) {

// Code if condition is true

} else {

// Code if condition is false

}

Example:

int num = 3;

if (num % 2 == 0) {

printf("Even number\n");

} else {

printf("Odd number\n");

}

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

An if or else if statement inside another if or else.

**Syntax:**

if (condition1) {

if (condition2) {

// Code if both condition1 and condition2 are true

}

} else {

// Code if condition1 is false

}

**Example:**

int num = 25;

if (num > 0) {

if (num % 5 == 0) {

printf("Positive and divisible by 5\n");

} else {

printf("Positive but not divisible by 5\n");

}

} else {

printf("Number is not positive\n");

}

1. **switch Statement**

Used to execute different blocks of code based on the value of an expression. It's an alternative to multiple if-else statements when checking a single variable.

**Syntax:**

switch (expression) {

case value1:

// Code

break;

case value2:

// Code

break;

default:

// Code if no case matches

}

**Example:**

int day = 3;

switch (day) {

case 1:

printf("Monday\n");

break;

case 2:

printf("Tuesday\n");

break;

case 3:

printf("Wednesday\n");

break;

default:

printf("Invalid day\n");

}