Module 1 – overview of IIT Industry

What is the program

A **program** is a set of instructions written in a programming language that tells a computer **what to do** and **how to do it**.

In Simple Words:

A program is like a recipe. Just as a recipe gives step-by-step instructions to make a dish, a program gives step-by-step commands that a computer follows to complete a task.

What is the programming?

In my words, **programming** is the process of **giving instructions to a computer** to make it do what you want. It's like teaching a machine step-by-step how to solve a problem or perform a task using a special language the computer understands, like C, Python, or Java.

Explain in your own words what a program is and how it functions.

In my own words, a **program** is a **set of instructions written by a person to make a computer do something specific**. It's like giving the computer a recipe that it will follow step by step to complete a task.

How it works:

1. **You write a program** using a programming language (like Python, C, or Java).
2. The program contains **instructions** like: "add these numbers," "show this message," or "ask the user for input."
3. These instructions are sent to the **computer's processor**, which follows them exactly, in order.
4. The computer **does what you told it to do**—whether it's calculating something, showing a game, or saving data.

What are the key steps involved in the programming process?

The **programming process** is like solving a puzzle using logic and code. It involves several clear steps to go from a problem or idea to a working program. Here are the **key steps** involved:

---

1. Understanding the Problem

Before writing any code, you need to clearly understand **what the program is supposed to do**.

- What is the input?
- What is the desired output?
- What is the logic or process?

🔍 *Example:* If you're building a calculator, you need to know what operations it should support (add, subtract, etc.).

---

2. Planning the Solution (Algorithm Design)

Create a **step-by-step plan** (algorithm) for solving the problem. This might be done using:

- Flowcharts
- Pseudocode (plain English code-like steps)

Types of Programming Languages.

Programming languages can be classified in several ways based on their purpose, level, or how they are executed. Here are the **main types of programming languages**:

---

1. Low-Level Languages

These are close to machine language and are harder for humans to understand but fast and efficient.

- **Machine Language** (binary – 0s and 1s)
- **Assembly Language** (uses mnemonics like `MOV`, `ADD`, etc.)

---

2. High-Level Languages

These are closer to human language, easier to write and understand.

Examples:

- **C**
- **C++**
- **Java**

- **Python**
- **Ruby**
- **C#**
- **JavaScript**

---

3. Procedural Languages

Follow step-by-step instructions or procedures.

Examples:

- **C**
- **Pascal**
- **Fortran**

---

4. Object-Oriented Languages

Based on objects and classes; good for complex programs.

Examples:

- **Java**
- **C++**
- **Python**
- **C#**

---

5. Functional Languages

Focus on mathematical functions and avoid changing states.

Examples:

- **Haskell**
- **Lisp**
- **Scala**

What are the main differences between high-level and low-level programming languages?

The **programming process** is like solving a puzzle using logic and code. It involves several World Wide Web & How Internet WorkSclear steps to go from a problem or idea to a working program. Here are the **key steps** involved:

---

1. Understanding the Problem

Before writing any code, you need to clearly understand **what the program is supposed to do**.

- What is the input?
- What is the desired output?
- What is the logic or process?

🔍 *Example:* If you're building a calculator, you need to know what operations it should support (add, subtract, etc.).

---

2. Planning the Solution (Algorithm Design)

Create a **step-by-step plan** (algorithm) for solving the problem. This might be done using:

- Flowcharts
- Pseudocode (plain English code-like steps)

---

3. Writing the Code (Implementation)

Now, you write the actual **program** using a programming language like Python, C, or Java.

- Follow the logic from your plan
- Use proper syntax and structure

World Wide Web & How Internet Works

World Wide Web (WWW)

What is it?

The **World Wide Web** (WWW or simply "the Web") is a **system of interlinked web pages** that you can access using the internet and a browser (like Chrome, Firefox, etc.).

It was invented by **Tim Berners-Lee** in 1989.

What it includes:

- **Websites** (like Google, YouTube, Wikipedia)
- **Webpages** (each page you visit)
- **Links (URLs)** that connect these pages

How you use it:

1. You type a web address (URL) like `www.google.com`
2. Your browser asks the server for that page
3. The server sends the page back
4. The browser shows the page to you

How the Internet Works

What is the Internet?

The **internet** is the **physical network** of computers, cables, satellites, and wireless systems that connects people and devices all over the world.

Think of the **internet** as the **roads**, and the **World Wide Web** as the **cars and shops** you visit on those roads.

Step-by-Step: How it Works

1. **You send a request**
   You type `www.example.com` in your browser.
2. **DNS (Domain Name System)**
   Converts the website name into an IP address (like 142.250.190.78).
3. **Your computer sends data**
   The request goes to the nearest **router** and is sent through a path of other computers, undersea cables, satellites, etc.
4. **Web server receives the request**
   The website's server (a special computer) gets the request and prepares the web page.
5. **Response travels back**
   The page's content (HTML, images, videos) is sent back to your browser.
6. **Browser displays the page**
   You see the page appear on your screen.

Key Components:

Term	Role
Router	Directs your data to the right path
Modem	Connects your home to the internet
IP Address	Unique ID for each device or website on the internet
Server	Stores and sends website data
Client	Your device/browser requesting info from servers
DNS	Translates website names to IP addresses

Summary:

- The **Internet** is the **network of networks**
- The **Web** is just one **service** on the internet (like email, chat, or file transfer)
- When you visit a website, you're using both: the **internet to connect** and the **web to view** content

Describe the roles of the client and server in web communication.

Client

What it is:

A **client** is the **device or software** (like a browser) that **makes a request** for information.

Examples:

- Your **web browser** (Chrome, Firefox)
- Your **smartphone app** (YouTube app, Instagram)

What it does:

1. **Sends a request** to the server (e.g., "Give me the Google homepage")
2. **Waits for the response**
3. **Displays the result** to the user (like a webpage or video)

Server

What it is:

A **server** is a **powerful computer** that stores **websites, files, databases**, or services.

Examples:

- A computer that stores Facebook's data
- The system that runs `www.amazon.com`

What it does:

1. **Listens for requests** from clients
2. **Finds and prepares the correct response** (like a webpage)
3. **Sends the data** back to the client

Client-Server Communication Example

Imagine you're visiting **www.wikipedia.org**:

1. **Client (Your Browser):**
   You type the URL and hit Enter → sends a request to Wikipedia's server
2. **Server (Wikipedia's Computer):**
   Receives your request → finds the homepage data → sends it back
3. **Client (Your Browser):**
   Receives the data → displays the webpage

Summary Table:

Role	Client	Server
Action	Sends request	Receives request
Purpose	Ask for data (e.g., a webpage)	Provide data (e.g., HTML, images)
Example	Your browser	Website host like Google or Facebook

Network Layers on Client and Server.

t question! When a **client and server** communicate over a network (like the internet), their messages pass through multiple **network layers**. These layers are part of a system that makes sure data is **sent, routed, and received correctly**.

The two main models used to explain this are:

---

1. OSI Model (7 Layers)

The **OSI (Open Systems Interconnection)** model divides networking into **7 layers**, from physical wires to the application.

Here's how it works on **both client and server sides**:

Layer	Name	Role in Communication
7	**Application Layer**	Interface for user (e.g., browser, web app)
6	**Presentation Layer**	Data formatting, encryption, compression
5	**Session Layer**	Manages sessions between client-server
4	**Transport Layer**	Ensures reliable delivery (e.g., TCP)
3	**Network Layer**	Chooses path for data (routing, IP address)
2	**Data Link Layer**	Handles local delivery (MAC address, switches)
1	**Physical Layer**	Wires, Wi-Fi, hardware (actual transmission)

Client vs. Server Using Layers

When a **client sends a request** (like visiting a website):

Client:

- **Application Layer**: User types URL in browser
- **Transport Layer**: Breaks into small packets (adds TCP headers)
- **Network Layer**: Adds IP address of the server
- **Data Link + Physical Layers**: Send packets via cable/Wi-Fi

Server:

- Receives data from physical layer upwards
- **Network Layer**: Reads client's IP, routes it
- **Transport Layer**: Reassembles packets
- **Application Layer**: Sends back the webpage

Then the **response goes back** through the same layers from **server to client**.

---

2. TCP/IP Model (Real-World Use)

In real networks, we use the **TCP/IP model**, which has **4 layers**:

Layer (TCP/IP)	OSI Equivalent	Example Tasks
Application	Layers 5–7	HTTP, HTTPS, FTP, DNS
Transport	Layer 4	TCP, UDP – ensure reliable or fast delivery
Internet	Layer 3	IP addressing and routing
Network Access	Layers 1–2	Physical connection, Ethernet, Wi-Fi

---

Real Example: Client Loads a Website

1. **Client (You)**
   - Uses **HTTP** (Application Layer) to request a page
   - Uses **TCP** (Transport Layer) to ensure message delivery
   - Uses **IP** (Internet Layer) to find the server
   - Sends data via **Wi-Fi/Ethernet** (Network Layer)
2. **Server**
   - Receives the request
   - Processes it using **HTTP**
   - Sends a response (webpage HTML, images) back the same way

Explain the function of the TCP/IP model and its layers.

What is the TCP/IP Model?

The **TCP/IP model** is a simplified framework that describes **how data is transmitted across a network** like the internet.

It defines **how computers communicate** and ensures that messages sent from one device can **reach another device correctly and reliably**.

TCP/IP stands for:

- **TCP** – Transmission Control Protocol
- **IP** – Internet Protocol

Together, they **control how data is packaged, addressed, sent, and received** across the internet.

Layers of the TCP/IP Model (4 Layers)

The model has **4 layers**, each with specific roles. Here's an overview from top to bottom:

---

1. Application Layer

What it does:

- Interfaces directly with the user or software applications.
- Provides services like email, web browsing, file transfer, etc.

Examples of protocols:

- **HTTP** – for web pages
- **FTP** – for file transfers
- **SMTP/POP3** – for email
- **DNS** – for domain name resolution

 Think of it as: The layer that allows your **browser or app** to talk to the internet.

---

2. Transport Layer

What it does:

- Ensures **reliable data transfer** between devices.
- Breaks data into **segments**, numbers them, checks for errors, and ensures all parts arrive correctly.

Main protocols:

- **TCP** – Reliable (used for web browsing, email, etc.)
- **UDP** – Fast but not guaranteed (used for streaming, gaming)

Think of it as: The **delivery guy** that makes sure your message arrives safely, in order, and asks for re-delivery if anything is missing.

Client and Servers.

They are **two sides of a communication system** in a network.

Client

What it is:

A **client** is any **device or software** that **requests services or resources** from another computer.

Examples:

- Web browser (like Chrome or Firefox)
- Mobile app (like Instagram or WhatsApp)
- Email client (like Outlook)

What it does:

- Sends **requests** to a server (e.g., "Show me a webpage")
- Waits for and **receives responses** from the server
- Displays the result to the user

Server

What it is:

A **server** is a **powerful computer or software** that provides **services, data, or resources** to clients.

Examples:

- Web server (stores websites)
- File server (stores files)
- Email server (manages email)

What it does:

- **Listens** for requests from clients
- **Processes** the request (e.g., retrieves a file or webpage)
- **Sends back** the correct response/data

How Client-Server Communication Works

Example: Visiting `www.wikipedia.org`

1. **Client (Your browser)**
   → Sends a request: "Give me the homepage of Wikipedia."
2. **Server (Wikipedia's Web Server)**
   → Receives the request
   → Finds the homepage file
   → Sends the webpage back to your browser
3. **Client (Browser again)**
   → Receives the data
   → Displays the page to you

Key Differences

Feature	Client	Server
Role	Sends requests	Responds to requests
Control	Controlled by the user	Runs automatically
Resources	Uses resources	Provides resources
Examples	Browser, App, PC	Google server, File server

Real-Life Analogy

Think of a **client** as a **customer** in a restaurant and the **server** as the **kitchen**:

- The customer places an order (request)
- The kitchen prepares the food (response)
- The waiter brings the food to the customer

Explain Client Server Communication.

Client-server communication is the process where a **client (user device)** and a **server (service provider) exchange data** over a network.

The **client** sends a **request**
The **server** sends a **response**

This is how most internet services (like websites, emails, apps) work.

How It Works – Step by Step

Example: Visiting a Website (like `www.example.com`)

1. **Client (Your Browser)**
   - You type the URL and press enter
   - The browser sends an **HTTP request** to the server
2. **Internet Path**
   - The request is routed using IP addresses and DNS
   - Routers and switches direct the message to the correct server
3. **Server (Web Server)**
   - The server receives the request
   - Finds the requested web page
   - Sends back an **HTTP response** with the page data (HTML, images, etc.)
4. **Client Again**
   - Your browser receives the data
   - It **renders the webpage** and shows it to you

What Happens Behind the Scenes (Using TCP/IP Model)

Layer	What Client Does	What Server Does
Application	Sends request using app (e.g., HTTP)	Receives and processes the request
Transport	Uses TCP to package and ensure delivery	Uses TCP to receive and reassemble packets
Internet	Adds IP address to locate the server	Uses IP to know where to send the response
Network Access	Sends data over Wi-Fi or Ethernet	Sends data back over the same route

---

Real-Life Analogy

Imagine a **client** is like a **customer in a restaurant**, and the **server** is like the **kitchen**:

- The customer (client) orders food (request)
- The kitchen (server) prepares the dish (processes request)
- The waiter brings it back (response)
- The customer eats it (views the result)

Common Protocols Used in Client-Server Communication

Protocol	Purpose	Example Use
HTTP/HTTPS	For websites	Browsers loading web pages
FTP	File transfer	Uploading/downloading files
SMTP/IMAP	Email sending/receiving	Sending and reading emails
DNS	Resolving names to IPs	Finding the address of a website

Summary

- **Client-Server communication** is how devices talk over a network.
- The **client asks**, and the **server answers**.
- It works using layers of protocols (like TCP/IP) to ensure the message gets delivered correctly.

Types of Internet Connections.

Internet connections come in various forms, depending on speed, technology, and usage. Here are the **main types**:

---

1. Dial-Up Connection

- Uses a **telephone line** to connect to the internet.
- Very **slow** (up to 56 kbps).
- Cannot use phone and internet at the same time.

Rarely used today due to slow speed.

---

2. DSL (Digital Subscriber Line)

- Uses a **telephone line**, but faster than dial-up.
- Can use **internet and phone at the same time**.
- Speed: 256 kbps to 100+ Mbps.

Still used in some homes and small offices.

---

3. Cable Internet

- Uses **TV cable lines (coaxial)** for internet.
- Faster than DSL, with speeds up to 1 Gbps.
- Shared bandwidth — may slow down with many users.

Common in urban areas.

---

4. Fiber Optic Internet

- Uses **light signals through fiber cables**.
- **Very fast**: up to 10 Gbps or more.
- Reliable and supports heavy usage (like streaming, gaming).

Fastest and most reliable option available today.

---

5. Wireless (Wi-Fi)

- Connects devices **without cables** using a **router**.
- Based on an underlying wired connection (fiber, DSL, etc.).
- Used at homes, schools, cafes, etc.

Great for mobile device access.

---

6. Mobile Data (3G, 4G, 5G)

- Uses **cellular networks**.
- No cables needed – works anywhere with signal.
- Speed varies:
    - **3G**: Slow
    - **4G**: Fast (10–100 Mbps)

- **5G**: Super-fast (up to 10 Gbps)

Perfect for smartphones, tablets, and portable use.

How does broadband differ from fiber-optic internet?

Broadband is a **general term** for **high-speed internet** that is **always on** (unlike dial-up). It refers to several types of internet connections, such as:

- **DSL (Digital Subscriber Line)**
- **Cable internet**
- **Fiber-optic**
- **Satellite**
- **Wireless**

So, **fiber-optic is actually one type of broadband**.

What is Fiber-Optic Internet?

Fiber-optic internet uses **thin glass or plastic fibers** to transmit data as **light signals**.

- Extremely **fast and reliable**
- Supports **very high bandwidth** (good for 4K streaming, gaming, video calls)
- Doesn't get affected much by distance or weather

Key Differences: Broadband vs Fiber-Optic

Feature	Broadband	Fiber-Optic
Meaning	General term for fast internet	Specific type of broadband
Medium	Copper wires, coaxial cables, radio	Glass or plastic fiber cables
Speed	Moderate to fast (up to 500 Mbps)	Very fast (up to 1 Gbps or more)
Examples	DSL, Cable, Satellite, Fiber	Only fiber-optic connections
Signal Loss	More over long distances	Very little signal loss
Cost	Usually cheaper	Slightly more expensive (but worth it)

Reliability	Can be affected by weather/electricity	Very reliable and stable

---

Summary:

- **Broadband** = umbrella term for all fast internet types.
- **Fiber-optic** = the **fastest and most advanced** type of broadband.

All fiber is broadband, but not all broadband is fiber.

Protocols.

What are Protocols?

Protocols are a set of rules or standards that define how data is transmitted and received across a network. They ensure proper communication between different devices by specifying **how** data is formatted, sent, received, and acknowledged.

Difference Between HTTP and HTTPS

Feature	HTTP	HTTPS
Full Form	HyperText Transfer Protocol	HyperText Transfer Protocol Secure
Security	Not secure — data is sent in plain text	Secure — data is encrypted using SSL/TLS
Data Encryption	No encryption	Yes, encryption via SSL/TLS
URL Format	`http://`	`https://`
Port Used	Port **80**	Port **443**
SSL/TLS Certificate	Not required	Required
Website Trust	Browsers may mark as "Not Secure"	Shows a padlock icon in address bar
Performance	Slightly faster (no encryption overhead)	Slightly slower (due to encryption process)

Application Security

Application Security refers to the **measures and practices** used to **protect software applications** from threats, vulnerabilities, and attacks throughout their lifecycle — from development to deployment and beyond.

What is the role of encryption in securing applications?

Encryption plays a **critical role** in securing applications by transforming sensitive data into unreadable code that only authorized users can access — keeping the data safe from hackers, eavesdroppers, and unauthorized access.

Software Applications and Its Types.

Software Applications, often called **apps**, are programs designed to perform specific tasks for users — such as writing documents, editing photos, managing data, or browsing the web.

These applications run on top of an operating system and help users interact with the computer to do useful work.

What is the difference between system software and application software?

Feature	System Software	Application Software
Definition	Software that manages and controls hardware and provides a platform for other software.	Software designed to help users perform specific tasks or solve particular problems.
Purpose	Runs the computer and its hardware.	Helps the user do tasks like writing, drawing, browsing, etc.
Interaction	Works in the background. Users don't interact directly with it very often.	Directly used by the user to perform tasks.
Examples	Operating systems (Windows, Linux), device drivers, utilities, firmware.	Microsoft Word, Photoshop, Google Chrome, WhatsApp.
Dependency	Needed for the computer to function.	Runs on top of system software; depends on it.
Installation	Comes pre-ins	

Module 2 – Introduction to Programming

Overview of C Programming

C programming is a powerful, general-purpose programming language developed in the early 1970s by **Dennis Ritchie** at **Bell Labs**. It is known

for its **speed, efficiency, and control over system resources**, making it widely used in system/software development.

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

Introduction

The **C programming language** is one of the most influential and widely used programming languages in the history of computer science. Developed more than five decades ago, C has played a crucial role in shaping modern computing. Despite the rise of newer programming languages, C remains an essential language due to its efficiency, portability, and close relationship with hardware.

History and Evolution of C Programming

The roots of C trace back to the late **1960s** and early **1970s**, during the early days of computer systems. Here's a brief timeline of its evolution:

1. **BCPL and B (1966–1969):**
    - The story begins with **BCPL (Basic Combined Programming Language)**, developed by **Martin Richards**.
    - Later, **Ken Thompson** at Bell Labs created a simplified version called **B**, which was used in early Unix development.
2. **Birth of C (1972):**
    - **Dennis Ritchie**, also at **Bell Labs**, built upon B to create a new language called **C** in 1972.
    - C was developed to rewrite the **Unix operating system**, which was initially written in assembly language. This was a major milestone—**Unix became the first OS written in a high-level language**.
3. **Standardization and Spread (1980s):**
    - C quickly gained popularity in academia and industry.
    - In **1983**, the **American National Standards Institute (ANSI)** started working on standardizing C.
    - The first official standard, known as **ANSI C or C89**, was released in **1989**. It was later adopted as **ISO C (ISO/IEC 9899)**.
4. **Further Versions:**
    - **C99 (1999):** Added features like inline functions, variable-length arrays, and new data types.
    - **C11 (2011):** Improved multithreading support and safer programming features.

- **C18 (2018):** Minor revisions and bug fixes.

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.

To write and run C programs, you need two things:

1. A **C compiler** (like **GCC**) to compile your code.
2. An **IDE** (like **DevC++**, **VS Code**, or **Code::Blocks**) to write and manage your code easily.

🔽 Step-by-Step Guide

Option 1: Install DevC++ (Beginner Friendly)

DevC++ is an IDE that comes with a built-in compiler (GCC). Ideal for beginners.

Steps:

1. Go to: https://sourceforge.net/projects/orwelldevcpp/
2. Click **Download** and install the `.exe` file.
3. Follow the setup wizard (default settings are fine).
4. Launch DevC++.
5. Go to **File → New → Source File**, write your C code, and save with `.c` extension.
6. Press **F9** to compile and run the code.

Option 2: Install Code::Blocks with GCC

Code::Blocks is another beginner-friendly IDE that can include GCC.

Steps:

1. Go to: https://www.codeblocks.org/downloads/
2. Download the **"codeblocks-XXmingw-setup.exe"** version *(includes GCC compiler)*.
3. Run the installer and select default options.
4. Open Code::Blocks.
5. Go to **File → New → Project → Console Application → C**.
6. Write your code and press **F9** to build and run.

Option 3: Install VS Code with GCC (Advanced Setup)

Visual Studio Code (VS Code) is a modern, powerful code editor. You need to manually install GCC and some extensions.

▾ *Steps:*

1. Install GCC Compiler (via MinGW):

- Go to: https://www.mingw-w64.org/
- Download and install the version for Windows.
- During installation:
  - Architecture: `x86_64`
  - Threads: `posix`
  - Exception: `seh`
- Add the `bin` folder (e.g., `C:\Program Files\mingw-w64\...\bin`) to your **System PATH**:
  - Right-click **This PC → Properties → Advanced System Settings → Environment Variables → PATH → Edit** and paste the path.

2. Verify GCC Installation:

- Open **Command Prompt** and type:

```
css
CopyEdit
gcc --version
```

If installed correctly, it will show the version.

3. Install VS Code:

- Download from: https://code.visualstudio.com/
- Install and launch it.

4. Install C/C++ Extension:

- Go to **Extensions** (Ctrl+Shift+X), search for **C/C++**, and install Microsoft's C/C++ extension.

5. Create and Run a C File:

- Create a new folder and open it in VS Code.
- Create a file `program.c`, write your code.
- Open the terminal in VS Code (`Ctrl + ~`) and compile:

```
bash
```

```
CopyEdit
gcc program.c -o program
./program
```

Summary

IDE	Compiler Needed	Difficulty	Best For
DevC++	Built-in (GCC)	Easy	Beginners
Code::Blocks	Built-in (with MinGW)	Easy	Beginners to Intermediate
VS Code	Requires GCC setup	Medium	Intermediate to Advanced

Explain the basic structure of a C program, including headers, main function,

comments, data types, and variables. Provide examples.

1. Header Files

Header files contain declarations for functions and macros used in the program. The most common one is:

```
c
CopyEdit
#include <stdio.h>
```

This includes the Standard Input/Output library, which allows functions like `printf()` and `scanf()`.

---

2. Main Function

Every C program must have a `main()` function. It's the entry point where the execution starts.

```
c
CopyEdit
int main() {
    // Code goes here
    return 0;
}
```

---

3. Comments

Comments are used to describe the code. They are ignored by the compiler.

- **Single-line comment:**

```
c
CopyEdit
// This is a single-line comment
```

- **Multi-line comment:**

```
c
CopyEdit
/* This is
   a multi-line comment */
```

---

4. Data Types

C provides several data types, such as:

Data Type	Description	Example
`int`	Integer values	`int age = 18;`
`float`	Decimal values (low precision)	`float temp = 36.5;`
`double`	Decimal values (high precision)	`double pi = 3.14159;`
`char`	Single character	`char grade = 'A';`

---

5. Variables

Variables are used to store data. You must declare the type before using them.

```
c
CopyEdit
int age = 25;
float temperature = 37.5;
char grade = 'A';
```

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

1. Arithmetic Operators

These perform basic mathematical operations.

Operator	Description	Example
`+`	Addition	`a + b`
`-`	Subtraction	`a - b`
`*`	Multiplication	`a * b`
`/`	Division	`a / b`
`%`	Modulus (remainder)	`a % b`

Example:

```
c
CopyEdit
int a = 10, b = 3;
printf("%d", a % b);  // Output: 1
```

---

2. Relational Operators

These compare two values and return true (`1`) or false (`0`).

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

Example:

```
c
```

```
CopyEdit
if (a > b) {
    printf("a is greater than b");
}
```

---

3. Logical Operators

Used to combine multiple conditions.

Operator	Description	Example
`&&`	Logical AND (both true)	`(a > 5 && b < 10)`
`		`
`!`	Logical NOT (negation)	`!(a > 5)`

---

4. Assignment Operators

Assign values to variables.

Operator	Description	Example
`=`	Assign value	`a = 5`
`+=`	Add and assign	`a += 2` → `a = a + 2`
`-=`	Subtract and assign	`a -= 2`
`*=`	Multiply and assign	`a *= 2`
`/=`	Divide and assign	`a /= 2`
`%=`	Modulus and assign	`a %= 2`

---

5. Increment and Decrement Operators

Used to increase or decrease a value by 1.

Operator	Description	Example
`++`	Increment	`a++` or `++a`
`--`	Decrement	`a--` or `--a`

- `a++` → Post-increment (use `a`, then increment)

- `++a` → Pre-increment (increment, then use `a`)

---

6. Bitwise Operators

Work on bits (binary digits) of integers.

Operator	Description	Example
`&`	AND	`a & b`
`` ` ``	`` ` ``	OR
`^`	XOR	`a ^ b`
`~`	NOT	`~a`
`<<`	Left Shift	`a << 2`
`>>`	Right Shift	`a >> 2`

---

7. Conditional (Ternary) Operator

A shorthand for `if-else` statement.

Syntax:

```
c
CopyEdit
(condition) ? expression1 : expression2;
```

Example:

```
c
CopyEdit
int max = (a > b) ? a : b;
```