# Compiler stages in C++

**1. Preprocessing**

* The preprocessor handles all directives such as #include, #define, and #ifdef.
* It expands macros, includes header files, and removes comments.
* The output is a "**pure" source code file ready for compilation**.

**2. Compilation**

* The compiler converts the preprocessed code into assembly code for the target machine.
* This stage checks for syntax errors, performs semantic analysis, and generates the corresponding machine-level instructions.
* Output: Assembly language file **(.s file**).

**3. Assembly**

* The assembler converts the assembly code **(.s file**) into machine code (binary instructions that the CPU can execute).
* The result is an object file **(.o or .obj**), which is still not a complete program because it may have unresolved references to external libraries or functions.

**4. Linking**

* The linker combines all object files **(.o or .obj**) and resolves references to external libraries or functions (like cout from iostream).
* It produces the final executable file **(.exe or a.out**).
* Any unresolved references at this stage will cause a linker error.

**5. Loading (Execution)**

* Once compiled and linked, the operating system loads the executable into memory and begins execution.
* This stage involves runtime initialization, such as setting up the stack, heap, and global variables.

**Note:** An **interpreter** is a program that executes code directly line by line or statement by statement without the need to first compile the entire source code into machine language. This is different from a compiler, which translates the entire source code into an executable file before running it.

**Examples**: Python, Ruby, JavaScript, PHP, Per.

# Diff B/W Segmentation Fault and Core Dump

1. **Segmentation Fault (Segfault):**

A segmentation fault occurs when a program tries to access memory that it is not allowed to access.

**It usually happens due to:**

* Dereferencing null or invalid pointers.
* Accessing memory beyond the allocated boundaries (buffer overflows).
* Writing to read-only memory.
* Using Uninitialized Pointers

**Example:**

 Dereferencing null or invalid pointers

#include <iostream>

**using** **namespace** std**;**

int main**()** **{**

int**\*** ptr **=** **nullptr;** // Null pointer

**\***ptr **=** 42**;** // Dereferencing a null pointer causes a segmentation fault

**return** 0**;**

**}**

 Accessing Memory Out of Bounds

int arr**[**3**]** **=** **{**1**,** 2**,** 3**};**

int x **=** arr**[**5**];** // Segfault: accessing invalid memory

 Writing to Read-Only Memory

const char**\*** str **=** "Hello"**;**

str**[**0**]** **=** 'h'**;** // Segfault: modifying read-only memory

 Using Uninitialized Pointers

int**\*** ptr**;**

**\***ptr **=** 10**;** // Segfault: pointer points to an unknown location

**2.Core Dump:**

A core dump is a file that contains a snapshot of the program's memory at the time of the crash (e.g., when a segmentation fault occurs).

The operating system generates this file to help developers debug the issue.

**It typically includes:**

* The values of variables.
* The program counter (where the program crashed).
* The call stack.
* Core dumps are optional and depend on the system configuration (e.g., Linux may disable them by default).

**Segmentation Fault Without Core Dump:**

#include <iostream>

**using** **namespace** std**;**

int main**()** **{**

int arr**[**5**]** **=** **{**1**,** 2**,** 3**,** 4**,** 5**};**

cout **<<** arr**[**10**];** // Accessing out-of-bound memory

**return** 0**;**

**}**

**Cause:** Accessing memory outside the bounds of the array.

**Output:** Segmentation fault (if core dump is not enabled).

**Segmentation Fault With Core Dump:**

#include <iostream>

**using** **namespace** std**;**

int main**()** **{**

int**\*** ptr **=** **nullptr;** // Null pointer

**\***ptr **=** 10**;** // Dereferencing null causes a segmentation fault

**return** 0**;**

**}**

**Cause:** Dereferencing a null pointer.

**Output:** Segmentation fault (core dump).

**How to Enable Core Dumps:**

**Check the current core dump limit:**

ulimit **-**c

If the result is 0, core dumps are disabled.

**Enable core dumps:**

ulimit -c unlimited

Run the program again. If a crash occurs, a core file will be created in the current directory.

**Note:** A core file is generated if core dumps are enabled. This can be analyzed using tools like **gdb**

gdb ./a.out core

# GitHub Workflow

**GitHub Workflow:** Clone → Create Branch → Edit → Commit → Push → Pull Request → Merge.

**Git Squash:** Combine multiple commits into one for cleaner history.

**Example of Git Squash:**

**Scenario:**

You have made three commits, each adding or modifying code. Now you want to consolidate these into a single commit before merging.

**View the Commit History:**

git log --oneline

**Output:**

f1a3b2c Add function **for** calculating sum

d3e2a1b Fix a bug in sum function

a9d8c7e Add comments **for** sum function

**Start an Interactive Rebase:**

git rebase -i HEAD~3

**Interactive Rebase Editor:**You will see something like this:

pick f1a3b2c Add function **for** calculating sum

pick d3e2a1b Fix a bug in sum function

pick a9d8c7e Add comments **for** sum function

**Change the second and third pick to squash (or just s):**

pick f1a3b2c Add function **for** calculating sum

squash d3e2a1b Fix a bug in sum function

squash a9d8c7e Add comments **for** sum function

**Edit Commit Messages:** Git will prompt you to combine commit messages. You can "comment out" unnecessary lines using #:

# Commit messages for squash

Add function **for** calculating sum

# Fix a bug in sum function

# Add comments for sum function

**Final message might be:**

Add function for calculating sum and fix a bug.

**Finalize the Rebase:** Save and close the editor. The commits will now be squashed into one.

**Before and After Git Squash:**

**Before Squash:**

git log --oneline

**Output:**

f1a3b2c Add function **for** calculating sum

d3e2a1b Fix a bug in sum function

a9d8c7e Add comments **for** sum function

**After Squash:**

git log --oneline

**Output:**

f1a3b2c Add function **for** calculating sum **and** fix a bug**.**

**Git Pull vs. Fetch:**

**Pull:** Fetch + Merge. (Fetches changes from the remote and merges them automatically into the current branch.)

**Fetch:** Only fetch changes. (Fetches changes from the remote but doesn’t merge them automatically into the current branch.)

# Diff B/W Shared Pointer and Weak Pointer in C++

**Shared Pointer (std::shared\_ptr):**

* A shared pointer is a smart pointer that shares ownership of an object with other std::shared\_ptr instances.
* The object is destroyed when the last std::shared\_ptr managing it is destroyed.
* **Reference Counting:** Each std::shared\_ptr increases the reference count of the object, ensuring proper cleanup when no references remain.

**Example:**

#include <iostream>

#include <memory>

int main**()** **{**

std**::**shared\_ptr**<**int**>** ptr1 **=** std**::**make\_shared**<**int**>(**42**);** // Shared ownership

std**::**shared\_ptr**<**int**>** ptr2 **=** ptr1**;** // Reference count increases

std**::**cout **<<** "Value: " **<<** **\***ptr1 **<<** "\n"**;**

std**::**cout **<<** "Reference Count: " **<<** ptr1**.**use\_count**()** **<<** "\n"**;** // Output: 2

**return** 0**;**

**}**

**Weak Pointer (std::weak\_ptr):**

* A weak pointer is a non-owning reference to an object managed by a std::shared\_ptr.
* It does not contribute to the reference count, so it won't prevent the object from being destroyed.
* **Use Case:** Commonly used to break Cyclic Dependencies in scenarios like doubly-linked lists or trees.

**Example:**

#include <iostream>

#include <memory>

struct Node **{**

std**::**shared\_ptr**<**Node**>** next**;**

std**::**weak\_ptr**<**Node**>** prev**;** // Prevent circular reference

**};**

int main**()** **{**

auto node1 **=** std**::**make\_shared**<**Node**>();**

auto node2 **=** std**::**make\_shared**<**Node**>();**

node1**->**next **=** node2**;**

node2**->**prev **=** node1**;** // `prev` is a weak pointer

std**::**cout **<<** "Reference Count of node1: " **<<** node1**.**use\_count**()** **<<** "\n"**;** // Output: 1

std**::**cout **<<** "Reference Count of node2: " **<<** node2**.**use\_count**()** **<<** "\n"**;** // Output: 1

**return** 0**;**

**}**

**Key Points:**

1. **Cyclic Dependencies:** std::weak\_ptr is used to avoid memory leaks caused by **Cyclic Dependencies** in std::shared\_ptr.
2. **Object Access:** Before using a std::weak\_ptr, it must be converted to a std::shared\_ptr to check if the object is still valid.

# Can we use delete operator in shared pointer

If you manually use **delete** on an object managed by a std::shared\_ptr, it can lead to undefined behavior, such as **double deletion** or **memory corruption**, because the std::shared\_ptr will still attempt to delete the object when it goes out of scope (which has already been deleted manually).

**Example of Incorrect Usage:**

#include <iostream>

#include <memory>

int main**()** **{**

std**::**shared\_ptr**<**int**>** ptr **=** std**::**make\_shared**<**int**>(**10**);**

// Incorrect: Do not manually delete the object managed by shared\_ptr

**delete** ptr**;** // This causes undefined behavior

**return** 0**;**

**}**

**Example of correct Usage:**

#include <iostream>

#include <memory>

int main**()** **{**

std**::**shared\_ptr**<**int**>** ptr **=** std**::**make\_shared**<**int**>(**10**);**

// Correct: No need to manually delete the object

std**::**cout **<<** "Value: " **<<** **\***ptr **<<** "\n"**;**

// When ptr goes out of scope, the memory will be automatically freed.

**return** 0**;**

**}**

# Diff B/W Binary and Executable

**Binary:**

* **Definition**: A binary file contains **data** in a **machine-readable format**, typically compiled code or data.
* **Purpose:** Binary files **can contain compiled code, data, or libraries**, which may not be ready to run directly.
* **Extension:** Common extensions include **.obj**, **.bin**, **.a**, **.lib**, etc.

**Executable:**

* **Definition**: An executable is a binary file specifically designed to be **executed by the operating system.**
* **Purpose:** Executable files contain compiled and linked code that is ready to be run by the operating system.
* **Extension:** Common extensions are **.exe** (Windows), or no extension on Unix-based systems.

# Diff B/W Signal and Services

**Signals:**

* Signals represent specific pieces of data, like **sensor readings** **or status flags**.
* Used for data communication between ECUs or within an ECU.
* Signals are part of communication protocols like CAN, Ethernet, or FlexRay.
* Typically follow the **Communication Stack**.
* **Example Signal:** Vehicle speed, Engine temperature.

**Example of Signal:**

A signal could represent the **engine temperature** sent from an ECU to a dashboard display.

Signal**:** EngineTemp

Description**:** Sends the current engine temperature**.**

Type**:** Integer **(**0**-**150 °C**)**

**Service:**

* **Services** represent **functional operations** or actions that are provided by a **component** or a **module**.
* Services are typically based on **service-oriented architecture (SOA**) and can be invoked remotely by other software components.
* Used for **remote communication** and invoking operations.
* **Example Service:** Vehicle location service, Navigation service.

**Example of Service:**

A service could be a **vehicle location service** that provides real-time location data to other components.

Service**:** VehicleLocationService

Description**:** Provides the current GPS coordinates of the vehicle**.**

Operations**:**

**-** GetCurrentLocation

**-** GetDestination

Type**:** SOME**/**IP **or** other service protocols**.**

# How to Convert a Signal into a Service

In an Adaptive AUTOSAR system, signals can be abstracted into services by wrapping the signal's functionality within a service interface. This allows the data (previously a signal) to be accessed through operations defined in the service interface.

**Example:**

class EngineMonitoringServiceImpl **:** public EngineMonitoringService **{**

public**:**

float GetEngineTemperature**()** override **{**

// Retrieve the signal value from the communication layer

**return** ReadSignalFromCAN**(**"signalName (EngineTemperature)"**);**

**}**

**};**

Consumers now call the **GetEngineTemperature**() operation to access the data:

auto engineTemp **=** engineMonitoringService**->**GetEngineTemperature**();**

# Explain All Modules in Adaptive Autosar

## COM

The **COM (Communication)** module in Adaptive AUTOSAR facilitates **Service-Oriented Communication** between different software components or applications. Unlike the Classic AUTOSAR COM module, which is signal-based, the Adaptive AUTOSAR COM module is designed for **service-oriented communication** using protocols like **SOME/IP**.

**Key Responsibilities of the COM Module:**

1. **Service-Oriented Communication**:
   * Manages communication between services and applications on different ECUs.
   * Uses service descriptions to enable discovery and interaction between components.
2. **Protocol Abstraction**:
   * Provides a protocol-agnostic API for applications to use.
   * Internally supports protocols like **SOME/IP** for communication.
3. **Service Discovery**:
   * Handles **service registration** and **discovery** using mechanisms like **SOME/IP Service Discovery (SD)**.
   * Ensures that applications can dynamically discover and connect to available services.
4. **Data Serialization/Deserialization**:
   * Serializes data (application-specific data structures) into a format suitable for transmission.
   * Deserializes received data back into the original structure.
5. **Runtime Binding**:
   * Supports dynamic binding of services and clients during runtime.
   * Allows components to communicate flexibly in a dynamic system environment.

**How the COM Module Works:**

1. **Service Description**:
   * Each service is defined by a **Service Interface**, which specifies:
     + **Methods:** Functions exposed by the service (e.g., GetSpeed()).
     + **Events:** Notifications sent by the service (e.g., SpeedExceeded).
     + **Fields:** Data values that represent the state of the service (e.g., currentSpeed).
2. **Service Discovery**:
   * When an application (client) starts, it queries the network to discover available services.
   * The COM module uses **SOME/IP-SD (Service Discovery)** to locate and bind to the desired service.
3. **Communication**:
   * Once the service is discovered, the COM module facilitates communication between the client and service provider:
     + **Request/Response** for method calls.
     + **Publish/Subscribe** for event notifications.
4. **Data Transmission**:
   * The COM module serializes the data using a serialization protocol (e.g., **SOME/IP serialization**) before sending it over the network.
   * At the receiver's end, the data is **deserialized** back into a usable format.

## EM

The **Execution Management (EM)** is responsible for managing **lifecycle** and **execution states** (**starting, stopping, and monitoring**) of applications and services on the Adaptive AUTOSAR platform.

**Key Points:**

1. **Lifecycle Management**:
   * EM controls when applications start and stop.
   * It ensures services are running before dependent applications start.
2. **State Management**:
   * Tracks and manages the states of individual applications, such as **Running**, **Stopped**, **Suspended**, etc..
3. **Error Handling**:
   * Restarts applications or services if they fail.
4. **Dependency Handling**:
   * Starts applications in the right order, based on their dependencies.
5. **Dynamic Management**:

* Supports dynamic addition, removal, or update of applications at runtime in some implementations.

**How EM Works:**

1. **Application Configuration**:
   * Applications declare their lifecycle requirements in the **Manifest File** (part of Adaptive AUTOSAR configuration).
2. **Startup**:
   * At system boot, the EM reads the configuration, initializes platform services, and starts the applications based on dependencies.
3. **State Transitions**:
   * EM manages transitions between states like **Stopped**, **Initializing**, **Running**, and **Terminated**.
4. **Fault Monitoring and Recovery**:
   * If an application fails, the EM can attempt to restart it or notify the platform of a critical failure.

## SM

The **State Management (SM)** module in Adaptive AUTOSAR manages the **system's operational** **modes** and **states** to keep everything running smoothly. It ensures coordination between applications, services, and platform states to keep the system functioning efficiently.

**Key Points:**

1. **Manages System States:**
   * Handles states like Startup, Running, Shutdown, and Error.
2. **Synchronizes Applications:**
   * Ensures all applications follow the system state.
   * **Example:** Apps start only when the system is in Running state.
3. **Manages Modes:**
   * Handles modes like Normal Mode or Low Power Mode for efficient operation.
4. **Handles State Transitions:**
   * Smoothly transitions between states (e.g., from Startup to Running).
5. **Error Handling:**
   * Switches to Error state when something fails.

**How EM Works:**

**1. Startup:**

* SM initializes the system by starting necessary services and applications.

**2. Running:**

* Once ready, the system moves to the Running State where applications and services operate normally.

**3. Error Handling:**

* If something goes wrong (e.g., a service fails), SM detects it and takes action like restarting the service or notifying the system.

**4. Shutdown:**

* When it's time to turn off, SM ensures all services and applications stop safely.

## Core

The **ARA::CORE module** in Adaptive AUTOSAR is responsible for managing core functionalities such as **memory management**, **thread scheduling**, and **resource allocation** across the system. It acts as the underlying framework that enables applications and services to interact with the hardware and system resources.

**Key Functions:**

1. **Memory Management**:
   * **Allocates and frees memory** for applications to use, ensuring there's no memory wastage.
2. **Thread Scheduling**:
   * **Decides which tasks** (threads) should run and when, ensuring fair use of the system’s processing power.
3. **Resource Allocation**:
   * **Manages system resources** (like CPU, memory) to avoid conflicts between applications.
4. **Task Synchronization**:
   * Ensures that tasks or applications can work together without interfering with each other.
5. **Inter-Process Communication (IPC)**:
   * Allows different applications to exchange information, like sending signals or data.

**How ARA::CORE Works:**

1. **Application Initialization**:
   * When the system starts, **ARA::CORE initializes** memory, resources, and threads for running applications.
2. **Running Applications**:
   * During operation, **ARA::CORE manages** tasks by deciding which thread gets CPU time.
   * It also manages **resource allocation** to ensure efficient operation of both foreground and background tasks.
3. **Handling Communication**:
   * ARA::CORE facilitates **communication** between apps, ensuring they can exchange necessary data (like audio or navigation commands).
4. **System Shutdown**:
   * Before shutting down, ARA::CORE **frees memory** and **releases resources**, ensuring a clean exit and preventing memory leaks.

## OS

In **Adaptive AUTOSAR**, the **Operating System (OS)** is responsible for managing the **hardware resources** and **running the applications** in the system. It provides the environment in which all the software applications and services run.

**Key Functions:**

1. **Task Scheduling**: Decides which application or task should run at any given time, ensuring that everything gets a chance to run without interruptions.

2. **Memory Management**: Allocates and frees memory for applications, making sure there’s enough space for them to run properly.

3. **Interrupt Handling**: Reacts to urgent events from hardware, like sensor data or user inputs, and makes sure the system responds quickly.

4. **Communication**: Allows different applications to send messages or data to each other, ensuring smooth interaction between them.

5. **Resource Protection**: Ensures that multiple applications don’t try to use the same resources at the same time, preventing crashes or conflicts.

6. **Time Management**: Keeps track of time and makes sure tasks that need to run periodically (like checking sensors) happen on time.

**How OS Works:**

1. **System Initialization**:
   * The OS initializes hardware components, allocates memory for system tasks, and configures scheduling policies.
2. **Task Execution**:
   * The OS starts execution of various applications, scheduling them according to their priorities. Applications like navigation, media, diagnostics, and communication services run concurrently but are allocated CPU time as per OS rules.
3. **Handling Events**:
   * The OS reacts to events such as hardware interrupts, sensor data, or user inputs by prioritizing and managing responses (e.g., launching safety features in response to sensor data).
4. **Shutdown**:
   * Before the system shuts down, the OS gracefully stops all running tasks, frees resources, and ensures that no critical data is lost.

## DIAG

The **Diagnostic (DIAG) Module** in **Adaptive AUTOSAR** is responsible for **detecting** and **managing faults** in the vehicle's electronic systems. It helps in finding issues, reporting them, and allowing repairs.

**Key Functions:**

1. **Fault Detection**: It identifies problems in vehicle systems, like a sensor failure, and generates fault codes (DTCs).
2. **Communication with Diagnostic Tools**: It allows external tools (like OBD-II scanners) to read fault codes and interact with the vehicle’s systems for troubleshooting.
3. **Fault Memory**: It stores detected faults, and can clear them once the issue is fixed.
4. **Read and Clear Fault Codes (DTCs)**: It lets technicians check and reset the fault codes in the system after repairs.

## PHM

**PHM (Platform Health Management)** in **Adaptive AUTOSAR** is a system that **helps monitor the health** of vehicle components and **predict possible failures** before they happen. It enables proactive maintenance, reducing the chances of unexpected breakdowns.

**Key Functions of PHM:**

1. **Health Monitoring**: Continuously monitors the condition of critical vehicle components (like sensors, actuators, etc.) to detect any signs of degradation or abnormal behavior.
2. **Fault Prediction**: Uses data and algorithms to predict potential future failures or performance issues in components, helping to schedule maintenance before a failure occurs.
3. **Data Analysis**: Collects and analyzes data from vehicle sensors to assess the overall health and performance of the system.
4. **Maintenance Recommendations**: Provides recommendations or alerts about when a component may need servicing or replacement based on predicted failures or wear.
5. **Lifecycle Management**: Helps in tracking the lifecycle of components, ensuring they are replaced or serviced at the right time to maintain vehicle performance.

**How PHM Works:**

1. **Monitoring**: PHM constantly checks the condition of important vehicle parts, like sensors or the engine, to ensure they are working correctly.
2. **Data Collection**: It collects data from sensors that measure things like temperature, pressure, and speed to assess the health of different components.
3. **Analysis and Prediction**: Using algorithms, PHM analyzes the collected data to detect patterns that might indicate a future problem, such as wear or degradation in a part.
4. **Alerts and Recommendations**: If PHM predicts a potential failure, it sends alerts or maintenance recommendations, so the issue can be addressed before it leads to a breakdown.

**Example:**

If PHM detects that the brake system's wear is reaching a critical level, it might predict that the brakes will soon need replacing. It will send a warning to the driver or maintenance team to replace the brakes, preventing a failure.

## Crypto

The **Crypto module** in **Adaptive AUTOSAR** is used **to protect data** and ensure secure communication between vehicle systems. It performs tasks like encryption, decryption, and verifying data integrity to keep sensitive information safe.

**Key Functions:**

1. **Encryption/Decryption**: Protects data by converting it into unreadable form and then back into readable form only for authorized systems.
2. **Hashing**: Creates a unique code for data to check if it has been changed during transmission.
3. **Digital Signatures**: Verifies the source of data to ensure it comes from a trusted sender.
4. **Key Management**: Handles and stores keys used for encryption and decryption securely.

**How it Works:**

* When two vehicle systems need to communicate, the Crypto module encrypts the data to keep it secure.
* The receiving system decrypts the data using a key to read it.
* The module also ensures that data is not altered during transmission using hashing and can verify the sender’s identity with digital signatures.

**Example:**

If the car’s infotainment system needs to send a secure message to another system, the Crypto module encrypts the message before sending it. The other system decrypts it to read, ensuring the data is secure and hasn’t been tampered with.

## UCM

**Update and Configuration Management (UCM)** in **Adaptive AUTOSAR** is responsible for managing **software updates** and ensuring the vehicle’s software is correctly configured.

**Key Functions:**

1. **Software Updates**: Keeps vehicle software up-to-date by managing updates for ECUs (Electronic Control Units).
2. **Configuration Management**: Ensures the correct configuration of software and settings for different vehicle systems.
3. **Version Control**: Tracks and manages software versions to ensure compatibility.
4. **Integrity Checking**: Verifies that software updates are not tampered with and are applied correctly.
5. **Error Handling**: Deals with issues during updates and ensures recovery if something goes wrong.

**How it Works:**

* When an update is needed, UCM downloads, checks, and applies it to the system.
* It ensures only compatible software is used and can roll back to a previous version if there are any issues.

**Example:**

If a new version of the vehicle’s navigation system is available, **UCM** will update the system, verify it’s safe, and ensure it’s working correctly.

## LOG

**Logging in Adaptive AUTOSAR** is a way to record important information about the system’s activities, errors, and events. This helps developers and engineers to track what’s happening in the vehicle’s software, identify problems, and improve system performance.

**Key Functions:**

1. **Record Events**: Logs actions or events happening in the system.
2. **Error Tracking**: Records errors or issues that occur.
3. **Monitor Performance**: Tracks how well the system is performing.
4. **Security**: Logs security-related events to protect the system.

**How it Works:**

* The system creates logs whenever something important happens, such as an error or a change in state.
* These logs are stored and can be reviewed later to understand what happened or fix problems.

**Example:**

If a sensor fails in the vehicle, the system will log an error with details about the failure. This helps engineers fix the issue quickly.

## PER

**Persistency in Adaptive AUTOSAR** means **saving important data** so that it stays even when the **system is turned off** or **restarted**. This ensures that things like settings, configurations, or user preferences are not lost and can be used again when the system is back on.

**Key Points:**

1. **Data Saving**: Data is saved in memory (like flash or EEPROM) so it remains after a reboot.
2. **System Continuity**: It keeps important settings or states intact across reboots.
3. **User Experience**: It helps restore user settings, like volume or language, when the system restarts.

**How it Works:**

* Data is saved to non-volatile memory before the system shuts down.
* When the system restarts, the saved data is loaded back into memory to continue from where it left off.

**Example:**

If you adjust the volume in your car’s infotainment system, **Persistency** ensures that the volume setting stays the same even after turning the car off and on again.

## TSync

**Time Sync** in **Adaptive AUTOSAR** ensures that all parts of a system (like different ECUs in a car) have the same, accurate time. This is important for tasks that need to happen at the right time, such as coordinating systems or recording events.

**Key Points:**

1. **Same Time Across Systems**: Ensures that all ECUs in the vehicle share the same time.
2. **Accurate Timing**: Helps in tasks like controlling systems or triggering actions at the correct time.
3. **Protocols Used**: Uses methods like **NTP** (Network Time Protocol) to synchronize time.

**How it Works:**

* One ECU or device provides the correct time (master time source).
* Other ECUs synchronize their clocks to this master time, ensuring they are all in sync.

**Example:**

If your car’s braking system needs to coordinate with other systems, **Time Sync** makes sure that all ECUs involved use the same time, so everything happens at the right moment.

## Manifest Files In Adaptive Autosar

**Execute Manifest file**

This provides the details of what is the name of executable, when it should be executed, when it should be started and stop all those details are present inside file.

**Example:**

{

"ExecutionManifest": {

"Applications": [

{

"name": "App1",

"Executable": {

"FilePath": "/path/to/app1\_binary",

"StartTime": "Boot", // The app starts when the system boots up

"StopTime": "Shutdown", // The app stops when the system shuts down

"Permissions": "read-write-execute" // Permissions for the app

},

"Resources": {

"CPU": "2 cores",

"Memory": "512MB"

}

}

]

}

}

**Machine Manifest file:** This provides the details of hardware configuration, such as CPU, memory, and network setup.etc.

{

"MachineManifest": {

"MachineConfig": {

"MachineName": "Machine1",

"CPU": {

"CoreCount": 4,

"Type": "ARM Cortex-A72"

},

"Memory": {

"RAM": "8GB",

"ROM": "128GB"

},

"Network": {

"IP\_Address": "192.168.1.100",

"Ports": ["8080", "5000"]

}

}

}

}

**Service Instance Manifest**

* **Purpose**: Configures **service-oriented communication** between applications or components.
* **What it contains**: Details about how services interact with each other and the transport protocols used for communication (e.g., Ethernet, CAN, etc.).

**Example:**

This manifest might define how a specific service communicates over Ethernet or another protocol, including port numbers and communication requirements.

{

"ServiceInstanceManifest": {

"ServiceInstances": [

{

"name": "DataService",

"TransportProtocol": "Ethernet",

"Communication": {

"Port": 5000,

"Protocol": "TCP"

},

"ServiceBehavior": {

"StartOnDemand": **true**,

"Timeout": "30s"

}

}

]

}

}

# Adaptive AUTOSAR Doc Types

**1. MMOD (Module Definition)**

* This document defines the module’s functionality, interfaces, and configuration.

**2. RS (Requirement Specification)**

* This document specifies the requirements of a software module or system, defining what the module needs to accomplish.

**3. SWS (Software Specification)**

* A detailed description of the behavior, functionality, and configuration of a software module or component.

**4. TPS (Test Plan Specification)**

* This document outlines the plan for testing software modules or components.

**5. TR (Test Report)**

* This document provides the results of tests conducted on the software modules or system.

# Docker

**Docker** is an open-source platform used for developing, shipping, and running applications in **containers**. Containers are lightweight, portable, and self-sufficient environments that package up an application and its dependencies (such as libraries, configurations, and runtime) so that it can run consistently **across different computing environments.**

**Advantage:**

The main advantage of Docker is its ability to create consistent and portable environments for applications. This allows developers to **build, test, and deploy applications** across different systems **without worrying about differences in software versions**, **configurations, or dependencies.**

# Handling Cyclic Dependencies in C++

**Breaking Circular References with std::shared ptr and Lambdas**

**Problem**

Using **std::shared ptr** in lambdas can cause memory leaks due to **cyclic dependencies**.

**Goal**

Understand the problem and resolve it using **std::weak** **ptr.**

• **Issue:** Lambda captures a shared ptr, causing a cyclic dependency.

• **Result:** Reference count never drops to zero, and resource is not destroyed.

**Problematic Code: Cyclic Dependency:**

#include <iostream>

#include <memory>

#include <functional>

class MyClass **{**

public**:**

MyClass**(**int value**)** **:** data**(**value**)** **{**

std**::**cout **<<** "Constructed: " **<<** data **<<** "\n"**;**

**}**

**~**MyClass**()** **{**

std**::**cout **<<** "Destroyed: " **<<** data **<<** "\n"**;**

**}**

int data**;**

std**::**function**<**void**()>** action**;**

void perform**()** **{**

**if** **(**action**)** **{**

action**();**

**}**

**}**

**};**

int main**()** **{**

auto resource **=** std **::** make\_shared **<** MyClass **>(**42**)** **;**

resource **->** action **=** **[** resource **]()** **{**

std **::** cout **<<** " Data : " **<<** resource **->** data **<<** "\n"**;**

**};**

resource **->** perform **()** **;**

**}**

**Output:**

Constructed**:** 42

Data**:** 42

**Problem**

**Note:** The destructor of MyClass is not called because of the cyclic dependency,

leading to a memory leak.

**What’s Happening?**

• Lambda captures the std::shared ptr.

• Lambda stored in the action member of MyClass.

• Creates a cyclic dependency:

→ shared ptr keeps lambda alive.

→ Lambda keeps shared ptr alive.

• **Result:** Destructor of MyClass is never called.

**Solution: Using std::weak ptr**

**How to Break Cyclic Dependency Use std::weak ptr:**

• weak ptr does not increase reference count.

• Promotes to shared ptr with lock().

• If destroyed, lock() returns nullptr.

**Fixed Code : Using std::weak ptr**

#include <iostream>

#include <memory>

#include <functional>

class MyClass **{**

public**:**

MyClass**(**int value**)** **:** data**(**value**)** **{**

std**::**cout **<<** "Constructed: " **<<** data **<<** "\n"**;**

**}**

**~**MyClass**()** **{**

std**::**cout **<<** "Destroyed: " **<<** data **<<** "\n"**;**

**}**

int data**;**

std**::**function**<**void**()>** action**;**

void perform**()** **{**

**if** **(**action**)** **{**

action**();**

**}**

**}**

**};**

int main**()** **{**

std**::**shared\_ptr**<**MyClass**>** resource **=** std**::**make\_shared**<**MyClass**>(**42**);**

// Potential circular dependency if `resource` directly captured as a shared\_ptr

std**::**weak\_ptr**<**MyClass**>** weakResource **=** resource**;**

resource**->**action **=** **[**weakResource**]()** **{**

// Lock the weak pointer to access the shared resource safely

**if** **(**auto sharedResource **=** weakResource**.**lock**())** **{**

std**::**cout **<<** "Data: " **<<** sharedResource**->**data **<<** "\n"**;**

**}** **else** **{**

std**::**cout **<<** "Resource no longer available.\n"**;**

**}**

**};**

// Call the lambda through the action

resource**->**perform**();**

// Resource goes out of scope here and is destroyed

**return** 0**;**

**}**

**Output:**

Constructed**:** 42

Data**:** 42

Destroyed**:** 42

**Result Key Observation:** The destructor of MyClass is correctly called because the cyclic dependency was resolved using std::weak ptr

**Breaking Cyclic Dependency with std::weak ptr**

1. **Replace std::shared ptr with std::weak ptr:** → std::weak ptr observes the object without increasing the reference count.

2. **Access with lock():** → Converts std::weak ptr to std::shared ptr. → Ensures safe access only if the object is still valid.

3. **Outcome:** The lambda no longer holds ownership directly, breaking the cycle and enabling proper cleanup.

# How would you integrate a new service in an Adaptive AUTOSAR-based system?

**Steps:**

1. **Define Service Interface:**
   * Specify the service details (e.g., methods, fields) in the Service Interface Manifest.
2. **Implement Service Provider:**
   * Develop the service logic and use ara::com to register it.
3. **Configure Service Discovery:**
   * Enable the service to advertise its availability.
4. **Develop Service Consumer:**
   * Use ara::com to discover and consume the service.
5. **Test Integration:**
   * Verify communication between provider and consumer.

# How does Adaptive AUTOSAR ensure data serialization/deserialization?

Adaptive AUTOSAR uses the **ara::com** API to **serialize** and **deserialize** data for network communication. It supports encoding data into a standard format for transmission and decoding it at the receiver's end.

**Example:**

* **Data Exchange:**
  + A camera application captures an image and sends metadata to a display application.
  + The metadata (e.g., resolution, timestamp) is serialized into a **binary format** using ara::com.
  + The display application deserializes the metadata to display the image.

Serialization ensures compatibility and efficient data transfer between different applications.

# SOME/IP Workflow in Adaptive AUTOSAR

SOME/IP (Scalable service-Oriented MiddlewarE over IP) is a middleware protocol used in Adaptive AUTOSAR to enable **service-oriented communication** between applications. It supports dynamic **service discovery**, **communication**, and **data exchange** across the network.

**Service Advertisement:**

* **Service Provider** (e.g., navigation system) makes itself available by advertising its services (e.g., GetRoute).
* This is done through a **Service Discovery** (SD) protocol, where the service sends out an advertisement that it can provide a specific service (e.g., navigation route).
* The advertisement includes information such as the **service name**, **service ID**, and available functions or methods that can be called by other applications.

**Service Discovery:**

* **Service Consumers** (e.g., infotainment system) discover available services in real time by querying the network for advertisements.
* The consumer identifies the available service, like the navigation service, and establishes communication with the service provider.
* The **Service Discovery** protocol enables dynamic and on-the-fly discovery of services in the vehicle network, so new ECUs can join the system and find services automatically.

**Service Request and Response:**

* Once the consumer discovers the service, it can send a **request** to the **service provider**.
* For example, the infotainment system sends a request to the navigation service asking for route data.
* The service provider (navigation system) processes the request and sends back a **response** containing the requested data (e.g., current route).
* Communication can happen using either **unicast** (one-to-one) or **multicast** (one-to-many) messaging depending on the use case.

**Data Serialization and Deserialization:**

* Data exchanged between the service provider and consumer is **serialized** into a **binary format** to ensure efficient transmission over the network.
* The data is deserialized at the receiving end to be understood and processed by the application.

**Communication Methods:**

* **SOME/IP** supports both **unicast** (communication between one consumer and one provider) and **multicast** (communication between one sender and multiple receivers) methods for data transmission.

**Key Points:**

* **Service Advertisement:** Services announce their availability to the network.
* **Service Discovery:** Consumers search for and find available services.
* **Service Binding:** Consumers establish communication links with the service.
* **Request/Response:** Consumers request services, and providers send back responses.
* **Service Withdrawal:** Providers can withdraw their services if unavailable.

In **SOME/IP**, both **TCP** and **UDP** are used, depending on the need for **reliability** or **speed**.

1. **UDP (User Datagram Protocol)**:
   * **Used for Service Discovery** (finding services on the network).
   * Faster and more efficient for sending short messages.
   * **No guarantee of delivery** (some packets may be lost).
   * Used when **speed** is more important than reliability, like in **multicast** service advertisements.
2. **TCP (Transmission Control Protocol)**:
   * **Used for communication** between service consumers and providers (e.g., exchanging data).
   * Slower than UDP but ensures that all messages are delivered correctly and in order.
   * **Reliable** and makes sure no data is lost.
   * Used when **reliability** is important, such as for large data exchanges or critical services. like in **unicast** service advertisements.

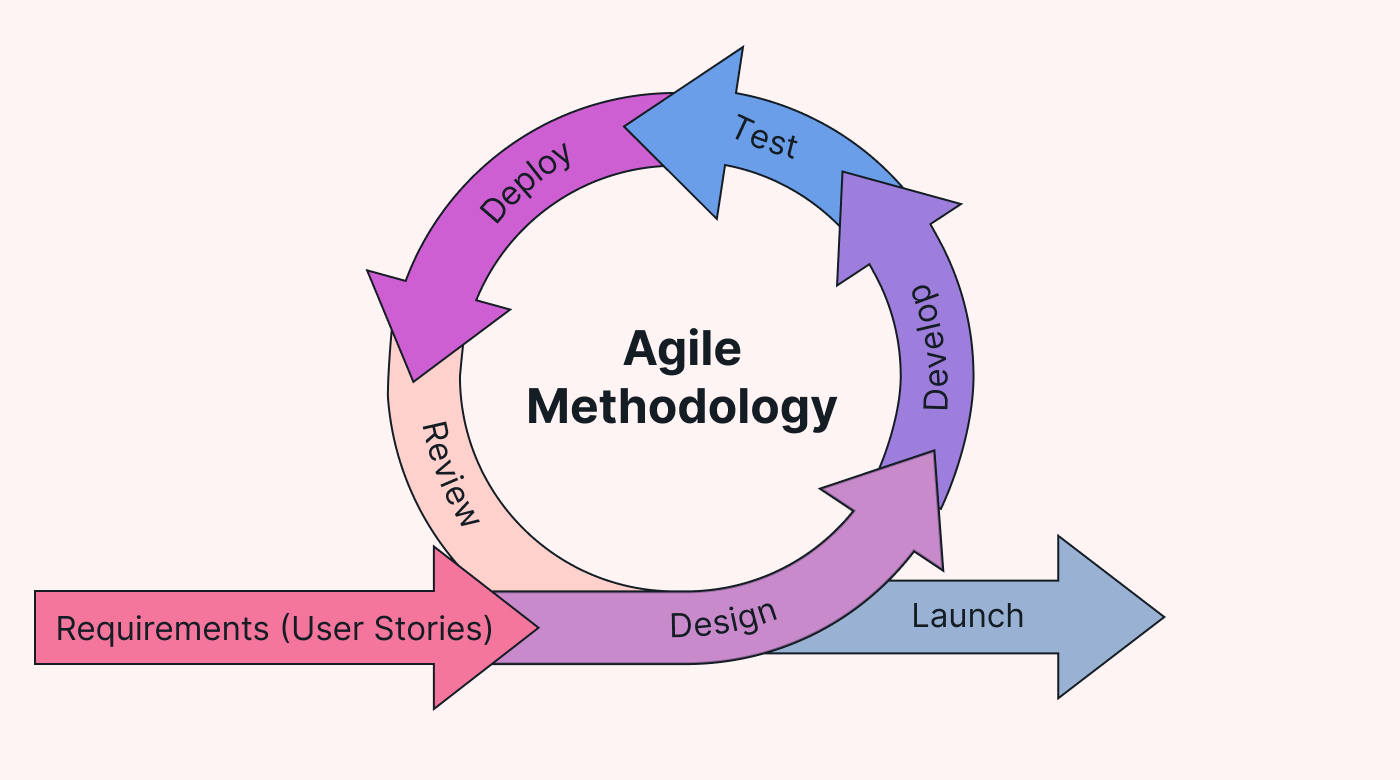
# Printable Characters (ASCII)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **ASCII Value** | | |  | | --- | | **Character** | | |  | | --- | | **Description** | |
| 48 | 1 | Zero |
| 65 | A | Uppercase A |
| 97 | a | Lowercase a |

# What is Agile methodology?

The **Agile methodology** is a project management approach that involves breaking the project into phases and emphasizes continuous collaboration and improvement.

It’s follows the **Planning ->Design->Develop->Test->Deploy->Review->Lunch.**



# What hardware is required to run an Adaptive AUTOSAR stack?

Adaptive AUTOSAR is designed for high-performance computing environments in modern vehicles, focusing on autonomous driving, connected services, and over-the-air updates. The hardware required for an Adaptive AUTOSAR stack typically includes:

**1. ECUs (Electronic Control Units)**

* High-performance **SoCs (System-on-Chip)** with multi-core processors
* Example: Renesas R-Car S4, NXP S32G, Infineon AURIX, NVIDIA Orin

**2. Processor Architectures**

* **x86-64** (Intel, AMD) – Common for development & simulation
* **ARM 64-bit (AArch64)** – Preferred for automotive applications (e.g., R-Car S4, NXP S32G)

**3. Memory & Storage**

* **RAM**: At least **2GB – 8GB** (for applications like ADAS, higher RAM is required)
* **Storage**: eMMC, UFS, NVMe SSD (for logging and large data handling)

**4. Communication Interfaces**

* **Ethernet** (100 Mbps, 1 Gbps, or more) – For SOME/IP & service-oriented communication
* **CAN/CAN FD** – For legacy communication with Classic AUTOSAR ECUs
* **LIN, FlexRay, and SPI** – Depending on specific vehicle requirements

**5. Virtualization & Hypervisors**

* **Hardware-assisted virtualization** (e.g., ARM Cortex A with hypervisor support)
* **Hypervisors**: AGL Xen, QNX Hypervisor, or Jailhouse for mixed-criticality systems

**6. Security Modules**

* **Hardware Security Modules (HSMs)** – Secure boot, encryption, and authentication
* **TPMs (Trusted Platform Modules)** – For cryptographic operations

**7. Debugging & Development Tools**

* **JTAG Debuggers**: Lauterbach, iSystem
* **Tracing tools**: Infineon DAP, ETM (Embedded Trace Macrocell)
* **ECU Test Benches**: dSPACE, Vector CANoe

**8. Virtual Platforms & Cloud-based Simulations**

* **QEMU** – For virtual ECU simulations
* **AWS Graviton-based EC2** – Cloud-based Adaptive AUTOSAR execution

# What is the difference between Inter-ECU and Intra-ECU communication in Adaptive AUTOSAR?

In **Adaptive AUTOSAR**, communication between ECUs can be classified into **Inter-ECU** and **Intra-ECU** communication based on how data is exchanged.

**1. Intra-ECU Communication (Within the Same ECU)**

* Communication **inside** a single ECU between different applications or software components.
* Uses **ARA::COM** (AUTOSAR Runtime for Adaptive Applications - Communication).
* Supports **method calls, events, and field communication** between applications.
* Runs on the same OS (e.g., Linux, QNX) and doesn't need network interfaces.
* Uses **shared memory, message queues, or IPC (Inter-Process Communication).**

**Example:**

* An Adaptive AUTOSAR application running on an ECU interacts with another application on the same ECU using **ARA::COM**.

**2. Inter-ECU Communication (Between Different ECUs)**

* Communication **between multiple ECUs** in a vehicle network.
* Uses **SOME/IP** (Scalable Service-Oriented Middleware over IP) for service-based communication.
* Typically runs over **Ethernet**, but can also use CAN, CAN FD, or FlexRay if needed.
* Uses **service discovery** to dynamically find available services.

**Example:**

* An **ADAS ECU** sends camera sensor data to an **Infotainment ECU** using SOME/IP over Ethernet.
* A **Powertrain ECU** communicates with the **Vehicle Control ECU** over CAN FD.

# Can a Classic AUTOSAR application run on Adaptive AUTOSAR?

No, not every application that runs on **Classic AUTOSAR** can be directly supported in **Adaptive AUTOSAR** because both platforms have different architectures and purposes.

# What is the role of the ARA layer in Adaptive AUTOSAR?

In **Adaptive AUTOSAR**, the **ARA (AUTOSAR Runtime for Adaptive Applications) layer** is responsible for providing essential **runtime services and communication mechanisms** for adaptive applications.

The **ARA (AUTOSAR Runtime for Adaptive Applications) layer** in Adaptive AUTOSAR is responsible for:

1. **Communication (ARA::COM)** → Enables service-based communication using SOME/IP.
2. **Execution (ARA::EXEC)** → Manages application startup, shutdown, and scheduling.
3. **State Management (ARA::SM)** → Controls system states and transitions.
4. **Logging (ARA::LOG)** → Handles logging and debugging.
5. **Data Storage (ARA::PERSISTENCY)** → Manages saving and retrieving data.
6. **Security (ARA::SECURE)** → Ensures authentication, encryption, and protection.

It acts as a middleware that helps Adaptive AUTOSAR applications run smoothly and communicate efficiently.

# Key Differences in Memory Usage Between Classic and Adaptive AUTOSAR?

1. **Dynamic Memory Management:**
   * **Adaptive AUTOSAR** uses more **dynamic memory** due to its service-oriented and more flexible nature. It loads and unloads applications during runtime.
   * **Classic AUTOSAR** is more static, with fixed memory configurations defined at compile-time.
2. **Memory Size Requirements:**
   * **Classic AUTOSAR** typically runs on embedded systems with **limited RAM** and uses **ROM** for the majority of the system's code and configuration.
   * **Adaptive AUTOSAR** may require **larger RAM** because it runs on **POSIX-based OS** like Linux or QNX, which manage larger applications and more dynamic memory allocation.

# where all executable files are present in adaptive autosar?

In **Adaptive AUTOSAR**, executable files are typically present in the **runtime environment** where the applications are executed. These executable files can be located in different places depending on the system setup, platform, and deployment strategy.

Here’s a breakdown of where executable files might be found in an Adaptive AUTOSAR system:

If you have an executable for an adaptive application named MyApp, it might be found in one of the following paths:

* **On Linux-based systems**: /opt/ara/MyApp/bin/MyAppExecutable
* **In a containerized system**: /usr/local/bin/MyAppExecutable
* **During the build process**: build/bin/MyAppExecutable

# What is the step-by-step process to fix defects in Adaptive AUTOSAR?

* **Understand the Problem**: Reproduce and analyze logs.
* **Isolate the Defect**: Identify the affected components.
* **Debug the Code**: Use debugging tools to pinpoint the issue.
* **Check Configurations**: Validate configurations (manifest.json, deployment files).
* **Modify Code/Configuration**: Make necessary changes to code or settings.
* **Rebuild and Test**: Rebuild the system and deploy the fix.
* **Validate the Fix**: Ensure the issue is resolved and test for regressions.
* **Document the Fix**: Update the documentation with details of the fix.
* **Monitor the System**: Continuously monitor to ensure the fix is effective.

# Most Important coding programs

## What is size of virtual function in a class

**Code:**

#include <iostream>

**using** **namespace** std**;**

class Base **{**

public**:**

virtual void func**()** **{}**

**};**

class Derived **:** public Base **{**

public**:**

void func**()** override **{}**

**};**

int main**()**

**{**

std**::**cout **<<** "Size of Base: " **<<** **sizeof(**Base**)** **<<** std**::**endl**;**

std**::**cout **<<** "Size of Derived: " **<<** **sizeof(**Derived**)** **<<** std**::**endl**;**

**return** 0**;**

**}**

**Output:**

Size of Base**:** 8

Size of Derived**:** 8

## Constructor will only supports integers data types does not allow any other data types

**Code:**

#include <iostream>

**using** **namespace** std**;**

class A **{**

int a**;** // Data member to store the integer value

public**:**

// Constructor that accepts an integer

A**(**int a1**)** **{**

a **=** a1**;**

cout **<<** "Constructor called with integer: " **<<** a **<<** endl**;**

**}**

// Delete the constructor for all other types

template **<**typename T**>**

A**(**T**)** **=** **delete;** // This will prevent the class from being constructed with non-integer types

**};**

int main**()** **{**

A obj**(**10**);** // Valid, as it uses the constructor with an integer argument

// A obj(10.5); // Invalid, this line will cause a compilation error because 10.5 is a double

**return** 0**;**

**}**

Output:

Constructor called with integer**:** 10

## The Template specialization allows you to provide a specific implementation for int types, overriding the generic template.

**Code:**

#include <iostream>

**using** **namespace** std**;**

template**<**typename T**>**

class A**{**

public**:**

void print**(){**

cout **<<** " specifice for generic data type "**;**

**}**

**};**

template**<>**

class A**<**int**>{**

public**:**

void print**(){**

cout **<<** " specifice for int data type "**;**

**}**

**};**

int main**(){**

A**<**int**>** obj**;**

obj**.**print**();** // specifice for int data type

A**<**float**>** obj2**;**

obj2**.**print**();** // specifice for generic data type

**}**

Output:

specifice for int data type

specifice for generic data type

## Write a program to find the size of a class without using the sizeof operator

**Code:**

#include <iostream>

**using** **namespace** std**;**

class A **{**

int a**;**

**};**

int main**()** **{**

A **\***obj1 **=** 0**;**

obj1**++;**

std**::**cout **<<** "Size of A: " **<<** obj1 **<<** " bytes" **<<** std**::**endl**;**

**return** 0**;**

**}**

**Output:**

Size of A: 0x4 bytes

## Write a program to swap the nibble in hex digits 0xAB to 0xBA

**Code:**

#include <iostream>

**using** **namespace** std**;**

int main**()** **{**

unsigned short hexValue **=** 0xAB**;** // 0xAB in hexadecimal

// Swap the nibbles (4 bits each)

unsigned short swapped **=** **((**hexValue **>>** 4**)** **&** 0x0F**)** **|** **((**hexValue **<<** 4**)** **&** 0xF0**);**

// Print the result

cout **<<** "Original: 0x" **<<** hex **<<** hexValue **<<** endl**;**

cout **<<** "Swapped: 0x" **<<** hex **<<** swapped **<<** endl**;**

**return** 0**;**

**}**

**Output:**

Original**:** 0xab

Swapped**:** 0xba

## Write a program to swap the nibble in binary

**Code:**

#include <iostream>

#include <bitset>

**using** **namespace** std**;**

int main**()** **{**

unsigned short hexValue **=** 0x93**;** // 10010011 in binary

// Swap the nibbles

unsigned short swapped **=** **((**hexValue **>>** 4**)** **&** 0x0F**)** **|** **((**hexValue **<<** 4**)** **&** 0xF0**);**

// Print the result in binary

cout **<<** "Original: " **<<** bitset**<**8**>(**hexValue**)** **<<** endl**;**

cout **<<** "Swapped: " **<<** bitset**<**8**>(**swapped**)** **<<** endl**;**

**return** 0**;**

**}**

**Output:**

Original**:** 10010011

Swapped**:** 00111001

## Write a program to swap the 8bits in hex and binary

**Code:**

#include <iostream>

#include <bitset>

**using** **namespace** std**;**

int main**()** **{**

unsigned short hexValue **=** 0xABCD**;** // Original 16-bit value

// Swap the two bytes: high byte with low byte

unsigned short swapped **=** **(**hexValue **>>** 8**)** **|** **(**hexValue **<<** 8**);**

// Print the result in binary

cout **<<** "Original (Binary): " **<<** bitset**<**16**>(**hexValue**)** **<<** endl**;**

cout **<<** "Swapped (Binary): " **<<** bitset**<**16**>(**swapped**)** **<<** endl**;**

// Print the result in hex format

cout **<<** "Original (Hex): 0x" **<<** hex **<<** hexValue **<<** endl**;**

cout **<<** "Swapped (Hex): 0x" **<<** hex **<<** swapped **<<** endl**;**

**return** 0**;**

**}**

**Output:**

Original **(**Binary**):** 1010101111001101

Swapped **(**Binary**):** 1100110110101011

Original **(**Hex**):** 0xabcd

Swapped **(**Hex**):** 0xcdab

## Write a Program to convert Hexadecimal to Decimal

**Code:**

#include <iostream>

#include <string>

#include <cmath>

**using** **namespace** std**;**

int hexToDecimal**(**string hex**)** **{**

int length **=** hex**.**length**();**

int decimalValue **=** 0**;**

// Loop through each character of the hex string

**for** **(**int i **=** 0**;** i **<** length**;** i**++)** **{**

char hexDigit **=** hex**[**i**];**

// Convert the hex digit to its decimal equivalent

int digitValue**;**

**if** **(**hexDigit **>=** '0' **&&** hexDigit **<=** '9'**)** **{**

digitValue **=** hexDigit **-** '0'**;** // Convert '0'-'9' to 0-9

**}** **else** **if** **(**hexDigit **>=** 'A' **&&** hexDigit **<=** 'F'**)** **{**

digitValue **=** hexDigit **-** 'A' **+** 10**;** // Convert 'A'-'F' to 10-15

**}** **else** **if** **(**hexDigit **>=** 'a' **&&** hexDigit **<=** 'f'**)** **{**

digitValue **=** hexDigit **-** 'a' **+** 10**;** // Convert 'a'-'f' to 10-15

**}** **else** **{**

cout **<<** "Invalid hexadecimal character" **<<** endl**;**

**return** **-**1**;** // Return an error value for invalid input

**}**

// Accumulate the decimal value by multiplying by 16^position

decimalValue **+=** digitValue **\*** pow**(**16**,** length **-** 1 **-** i**);**

**}**

**return** decimalValue**;**

**}**

int main**()** **{**

string hexInput**;**

// Take input as hexadecimal string

cout **<<** "Enter a hexadecimal number (e.g., 0xABCD): "**;**

cin **>>** hexInput**;**

// Convert to decimal

int decimalValue **=** hexToDecimal**(**hexInput**);**

cout **<<** "Decimal equivalent: " **<<** decimalValue **<<** endl**;**

**return** 0**;**

**}**

**Output:**

Enter a hexadecimal number **(**e**.**g**.,** 0xABCD**):** A

Decimal equivalent**:** 10

## Write a program to check if the given number is even without using arithmetic or relational operators

**Code:**

#include<iostream>

**using** **namespace** std**;**

int main**(){**

int a **=** 154**;**

**if(**a **&** 0x01**)** **{**

cout**<<**a**<<**" is an odd number"**;**

**}** **else{**

cout**<<**a**<<**" is an even number"**;**

**}**

printf**(**"\n"**);**

**return** 0**;**

**}**

**Output:**

154 is an even number

## Write a program to check the given number is follow litile endian or big enndian

**Code:**

#include<iostream>

**using** **namespace** std**;**

int main**()**

**{**

union**{**

int x**;**

char c**[**2**];**

**}**var**;**

var**.**x**=**1**;**

**if(**var**.**c**[**0**]==**1**)**

cout**<<**"little endian\n" **;**

**else**

cout**<<**"big endian\n"**;**

**return** 0**;**

**}**

**Output:**

little endian

## Write a program to set a bit in given pos of the number

**Code:**

#include <iostream>

**using** **namespace** std**;**

int main**()**

**{**

string st**;**

**do{**

int num**,**pos**;**

cout**<<**"Enter the number : "**;**

cin**>>**num**;**

cout**<<** "Enter the pos : " **;**

cin**>>**pos**;**

int set\_num **=** num **|** **(**1 **<<** pos**);**

//expain:

10 **|** **(**1**<<**2**)**

10 **|** **(**1**\***2 shifts of 2**)**

10 **|** **(**1**\***4**)**

10 **|** 4

OR **:** 1010

0100

**-----**

1110 **=** 14

**-----**

cout **<<** "set\_num : "**<<**set\_num **<<** endl**;**

cout **<<** "Do you want continue : yes/no : "**;**

cin**>>** st**;**

**}while(**st **==**"yes"**);**

cout**<<** "Exit"**;**

**return** 0**;**

**}**

**Output:**

Enter the number **:** 10

Enter the pos **:** 2

set\_num **:** 14

Do you want **continue** **:** yes**/**no **:** no

## Write a program to Clear a bit in given pos of the number

**Code:**

#include <iostream>

**using** **namespace** std**;**

int main**()**

**{**

string st**;**

**do{**

int num**,**pos**;**

cout**<<**"Enter the number : "**;**

cin**>>**num**;**

cout**<<** "Enter the pos : " **;**

cin**>>**pos**;**

int Clear\_num **=** num **&~** **(**1 **<<** pos**);**

//expain:

10 **&~** **(**1**<<**3**)**

10 **&~** **(**1**\***2 shifts of 3**)**

10 **&~** **(**1**\***8**)**

10 **&~** 8**(**1000**)**

10 **&** 0111

AND **:** 1010

0111

**-----**

0010 **=** 2

**-----**

cout **<<** "Clear\_num : "**<<**Clear\_num **<<** endl**;**

cout **<<** "Do you want continue : yes/no : "**;**

cin**>>** st**;**

**}while(**st **==**"yes"**);**

cout**<<** "Exit"**;**

**return** 0**;**

**}**

**Output:**

Enter the number **:** 10

Enter the pos **:** 3

Clear\_num **:** 2

Do you want **continue** **:** yes**/**no **:** no

## Write a program to check the given number how many set and clear are present

**Code:**

#include <iostream>

**using** **namespace** std**;**

#define number\_of\_bits 8

int main**()**

**{**

string st**;**

**do{**

int num**,**count**=**0**;**

cout**<<**"Enter the number : "**;**

cin**>>**num**;**

**for(**int pos **=** 0 **;** pos **<** number\_of\_bits **;** pos**++)** **{**

**if(**num **&** **(**1 **<<** pos**))**

count**++;**

**}**

//expain:

10 **&** **(**1**<<**3**)**

10 **&** **(**1**\***2 shifts of 2**)**

10 **&** **(**1**\***4**)**

10 **&** 4

10 **&** 0100

AND **:** 1010

0100

**-----**

0000 **=** 0

**-----**

cout **<<** "set\_count : "**<<**count **<<** endl**;**

cout **<<** "clear\_count : "**<<**number\_of\_bits**-**count **<<** endl**;**

cout **<<** "Do you want continue : yes/no : "**;**

cin**>>** st**;**

**}while(**st **==**"yes"**);**

cout**<<** "Exit"**;**

**return** 0**;**

**}**

**Output:**

Enter the number **:** 10

set\_count **:** 2

clear\_count **:** 6

Do you want **continue** **:** yes**/**no **:** no

## Write a program to convet binary formate to decimal formate

**Code:**

#include <iostream>

**using** **namespace** std**;**

#include<cstring>

int main**()**

**{**

string st**;**

**do{**

string binary**;**

int decimal**=**0**;**

cout**<<**"Enter the binary : "**;**

cin**>>**binary**;**

int length **=** binary**.**length**();**

**for(**int i **=** 0 **;** i **<** length**;** i**++)** **{**

**if(**binary**[**i**]** **==** '1'**){**

decimal **=** decimal **+** **(**1 **<<** i**);**

**}**

**}**

cout **<<** "decimal\_value : "**<<**decimal **<<** endl**;**

cout **<<** "Do you want continue : yes/no : "**;**

cin**>>** st**;**

**}while(**st **==**"yes"**);**

cout**<<** "Exit"**;**

**return** 0**;**

**}**

**Output:**

Enter the binary **:** 1010

decimal\_value **:** 10

Do you want **continue** **:** yes**/**no **:** no

Exit

## Write a program to convet decimal formate to binary formate

**Code:**

#include <iostream>

**using** **namespace** std**;**

#include<cstring>

int main**()**

**{**

string st**;**

**do{**

int binaryNum**[**32**],**i**=**0**,**n**;**

cout**<<**"Enter a decimal number: "**;**

cin**>>**n**;**

**while** **(**n **>** 0**)** **{**

binaryNum**[**i**]** **=** n **%** 2**;** // Adding the remainder

n **=** n **/** 2**;** // remove the last value

i**++;**

**}**

cout **<<** "Binary formate: " **;**

**for** **(**int j **=** i **-** 1**;** j **>=** 0**;** j**--)**

cout **<<** binaryNum**[**j**];**

cout **<<** "\nDo you want continue : yes/no : "**;**

cin**>>** st**;**

**}while(**st **==**"yes"**);**

cout**<<** "Exit"**;**

**return** 0**;**

**}**

**Output:**

Enter a decimal number**:** 10

Binary formate**:** 1010

Do you want **continue** **:** yes**/**no **:** no

Exit

## Write a program to print all prime numbers between 1 and 100

**Code:**

#include <iostream>

**using** **namespace** std**;**

int main**(){**

int num**,**i**,**count **=**0**;**

**for(**num **=**1**;** num **<=**100 **;** num**++)**

**{**

count **=**0**;**

**for(**i **=**1**;** i**<=**num**;**i**++){**

**if** **(**num**%**i **==** 0**)**

count **++;**

**}**

**if(**count **==**2 **)**

cout **<<**"Prime number: " **<<** num **;**

**}**

**}**

## Write a program to print all armstrong numbers between 1 and 100

**Code:**

#include <iostream>

#include <math.h>

**using** **namespace** std**;**

int main**(){**

int num**,**num1**,**r**,**sum**;**

**for(**num **=** 1**;** num **<=**153 **;** num**++)**

**{**

int temp **=** num**;**

int digits **=** 0**;**

// Calculate the number of digits

**while(**temp**>**0**){**

temp**/=**10**;**

digits**++;**

**}**

**for(**sum **=**0**,**num1**=**num **;** num1 **;** num1**/=**10 /\* Removing last value\*/**)**

**{**

r**=**num1**%**10**;** //remainder

sum**+=**pow**(**r**,**digits**);**

**}**

**if(**num **==** sum **)**

cout **<<**"Armstrong number: " **<<** sum**;**

**}**

**}**

**Output:**

Armstrong number**:** 1 Armstrong number**:** 2 Armstrong number**:** 3 Armstrong number**:** 4 Armstrong number**:** 5 Armstrong number**:** 6 Armstrong number**:** 7 Armstrong number**:** 8 Armstrong number**:** 9 Armstrong number**:** 153

## Write a program to print all palindrome or reverse numbers between 1 and 100

**Code:**

#include <iostream>

**using** **namespace** std**;**

int main**(){**

int num**,**num1**,**r**,**sum**;**

**for(**num **=** 1**;** num **<=**100 **;** num**++)**

**{**

**for(**sum **=**0**,**num1**=**num **;** num1 **;** num1**/=**10 /\* Removing last value\*/**)**

**{**

r**=**num1**%**10**;** //remainder

sum **=** sum **\*** 10 **+** r**;**

**}**

**if(**num **==** sum **)**

cout **<<**" Palindrome number: " **<<** sum**;**

**}**

**}**

**Output:**

Palindrome number**:** 1 Palindrome number**:** 2 Palindrome number**:** 3 Palindrome number**:** 4 Palindrome number**:** 5 Palindrome number**:** 6 Palindrome number**:** 7 Palindrome number**:** 8 Palindrome number**:** 9 Palindrome number**:** 11 Palindrome number**:** 22 Palindrome number**:** 33 Palindrome number**:** 44 Palindrome number**:** 55 Palindrome number**:** 66 Palindrome number**:** 77 Palindrome number**:** 88 Palindrome number**:** 99

## Write a program to [find factorial of a number](https://www.programiz.com/c-programming/examples/factorial)

**Code:**

#include <iostream>

**using** **namespace** std**;**

int main**(){**

int num **=** 4**,**i**,**fact**;**

**for(**i**=**1**,**fact **=**1 **;** i**<=**num **;** i**++)**

fact **=**fact **\*** i**;**

cout**<<** num **<<**" Factorial number : " **<<** fact**;**

**}**

**Output:**

4 Factorial number**:** 24

## Write a program to [reverse the bits](https://www.programiz.com/c-programming/examples/factorial)

**Code:**

#include <iostream>

#include <bitset>

**using** **namespace** std**;**

int main**(){**

// reverse bits 00001010 == 01010000

int num **=**10**;**

int temp **=**num**;**

int result **=** 0**;**

**for** **(**int i **=** 0**;** i **<** 8**;** i**++)** **{**

result **=** result **<<** 1**;** // Shift result to the left by 1 bit

result **=** result **|** **(**num **&** 1**);** // Add the least significant bit of num to result

num **=** num **>>** 1**;** // Shift num to the right by 1 bit

**}**

cout **<<** "Original: " **<<** bitset**<**8**>(**temp**)** **<<** " (Decimal: " **<<** **(**int**)**temp **<<** ")" **<<** endl**;**

cout **<<** "Reversed: " **<<** bitset**<**8**>(**result**)** **<<** " (Decimal: " **<<** **(**int**)**result **<<** ")" **<<** endl**;**

**}**

**Output:**

Original**:** 00001010 **(**Decimal**:** 10**)**

Reversed**:** 01010000 **(**Decimal**:** 80**)**

## Write a program to [remove the dupicate numbers in a given array](https://www.programiz.com/c-programming/examples/factorial)

**Code:**

#include <iostream>

**using** **namespace** std**;**

int main**()** **{**

int nums**[]** **=** **{**0**,** 0**,** 1**,** 1**,** 1**,** 2**,** 2**,** 3**,** 3**,** 4**};**

int size **=** **sizeof(**nums**)** **/** **sizeof(**int**);**

int j1 **=** 0**;** // Index for placing unique elements

cout **<<** "Before: "**;**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)**

cout **<<** nums**[**i**]** **<<** " "**;**

cout **<<** endl**;**

// Array to store unique elements

int unique**[**size**]** **=** **{**0**};**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)** **{**

bool isDuplicate **=** **false;**

**for** **(**int j **=** 0**;** j **<** i**;** j**++)** **{** // Check against all previous elements

**if** **(**nums**[**i**]** **==** nums**[**j**])** **{**

isDuplicate **=** **true;** // Mark as duplicate

**break;**

**}**

**}**

**if** **(!**isDuplicate**)** **{**

unique**[**j1**]** **=** nums**[**i**];** // Add to unique array

j1**++;**

**}**

**}**

cout **<<** "After (Unique Elements): "**;**

**for** **(**int i **=** 0**;** i **<** j1**;** i**++)**

cout **<<** unique**[**i**]** **<<** " "**;**

cout **<<** endl**;**

**return** 0**;**

**}**

**Output:**

Before**:** 0 0 1 1 1 2 2 3 3 4

After **(**Unique Elements**):** 0 1 2 3 4