

Software Engineering Course

Content

Module 1 - [Overview of IT Industry]

Module 2 - [Fundamentals of Programming]

Module 3 - [OOP Concept]

Module 4 - [HTML & CSS]

Module 5 - [Database]

Module - 1

[Overview of IT Industry]

What is Program

- Program- It is a set of Instructions

Ex. 1- Instructions to your Pet Ex. 2- Starting your Computer

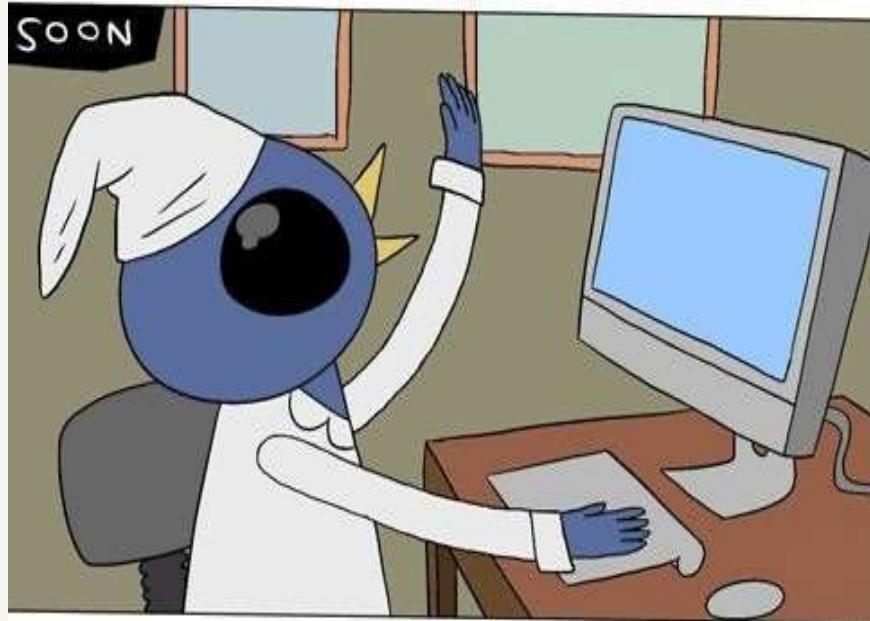


```
>Hello world!  
>_
```

What is Programming

- Programming- To create a Program.

Ex. 1- Using Keyboard & Mouse



Types of Programming Language

- Procedural Programming
 - Ex.- C Language



Types of Programming Language

- Object Oriented Programming
 - Ex.- C++ Language



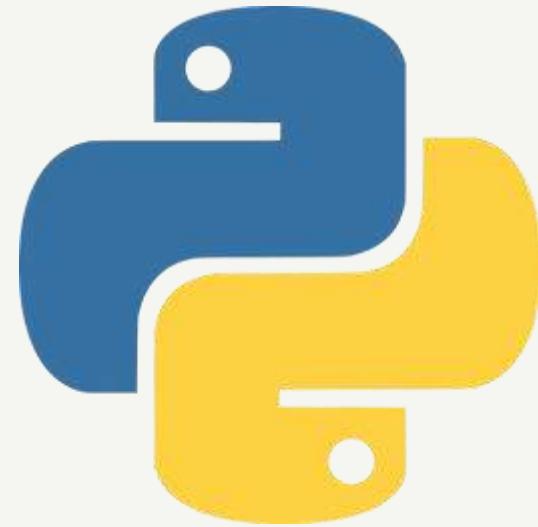
Types of Programming Language

- Logical Programming
 - Ex.- Prolog Language



Types of Programming Language

- Functional Programming
 - Ex.- Python Language



World Wide Web - WWW

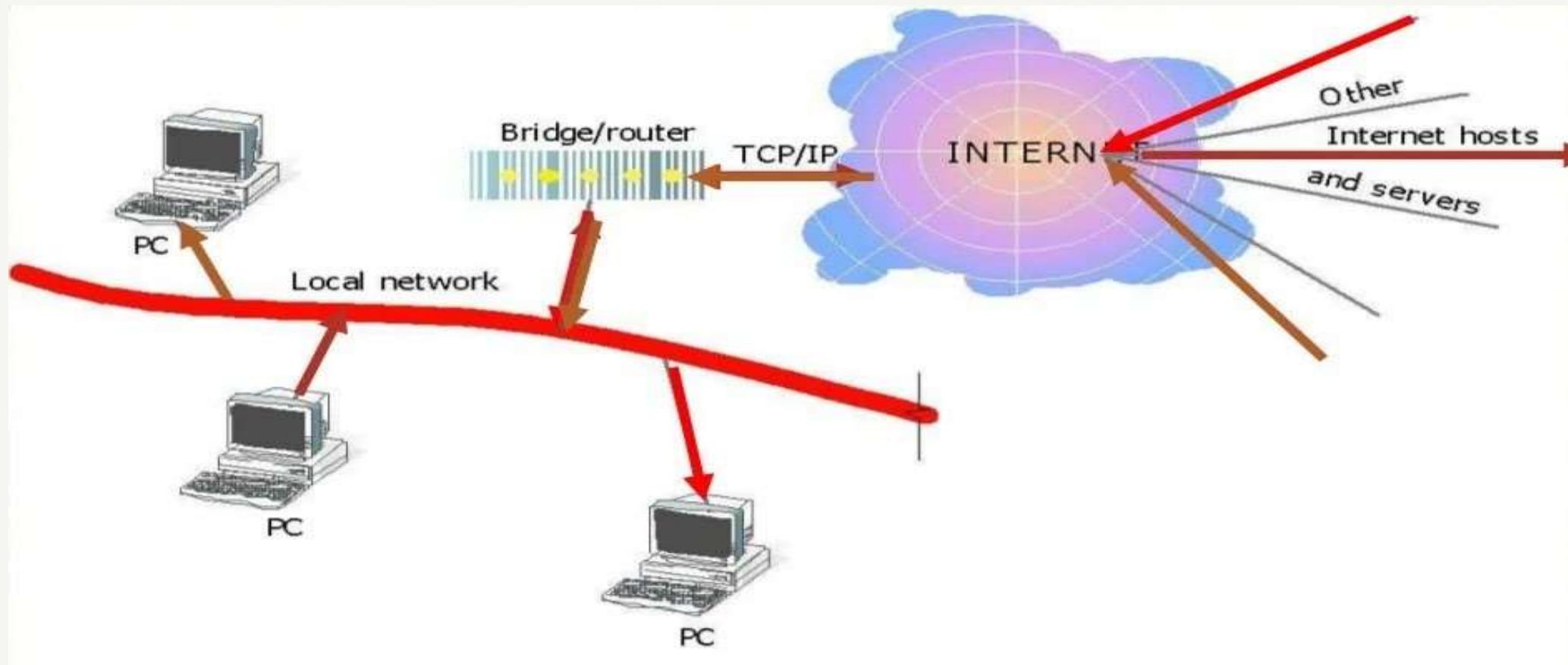
- known as a Web
- is a collection of websites or web pages stored in web servers
- connected to local computers through the internet
- These websites contain text pages, digital images, audios, videos, etc.
- Users can access the content of these sites from any part of the world over the internet using their devices such as computers, laptops, cell phones, etc.
- The WWW, along with internet, enables the retrieval and display of text and media to your device.



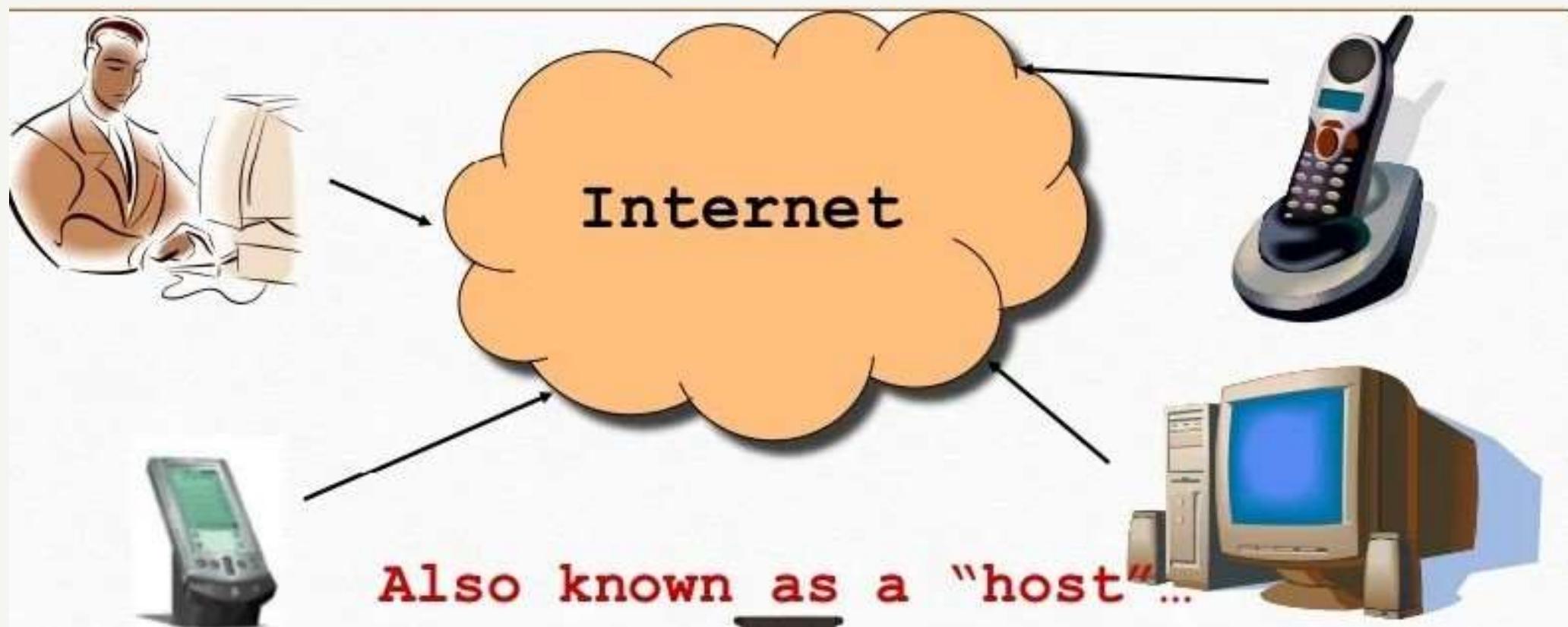
WWW – How Internet Works?

- A server is where websites are stored, and it works a lot like your computer's hard drive.
- Once the request arrives, the server retrieves the website and sends the correct data back to your computer.
- One of the best features of the Internet is the ability to communicate almost instantly with anyone in the world.
- **Email** is one of the oldest and most universal ways to communicate and share information on the Internet, and billions of people use it. **Social media** allows people to connect in a variety of ways and build communities online.

WWW – How Internet Works?



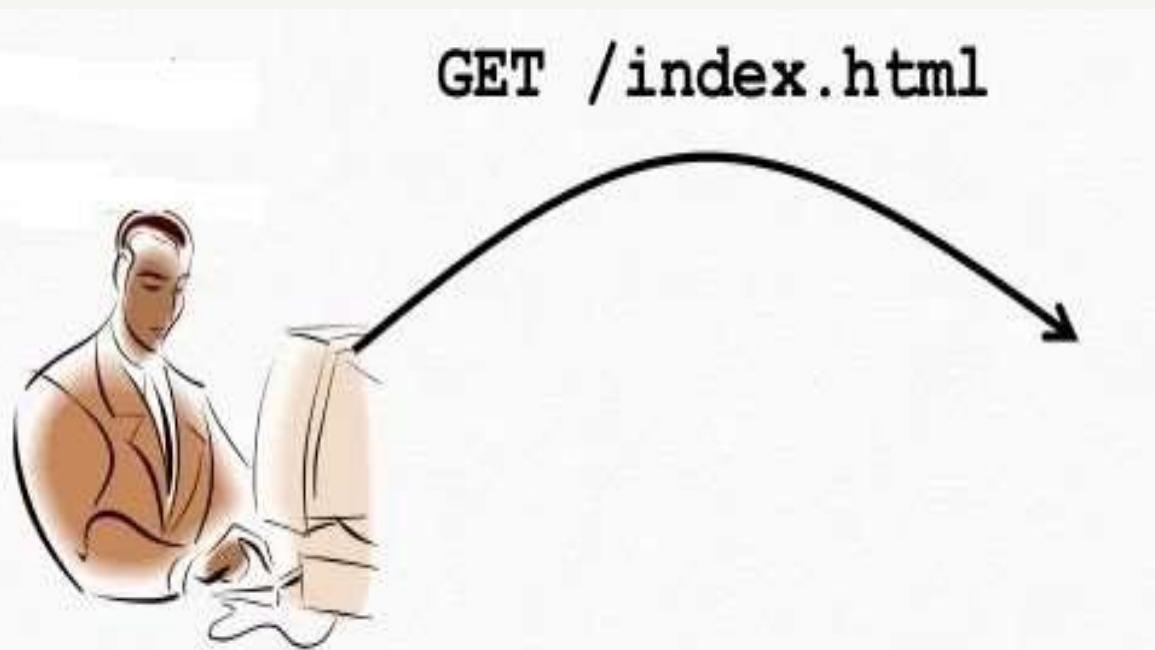
Network layers on Client & Server



Client & Servers

1. Client Program

- Running on end host
- Requests Services
- E.g. Web Browser



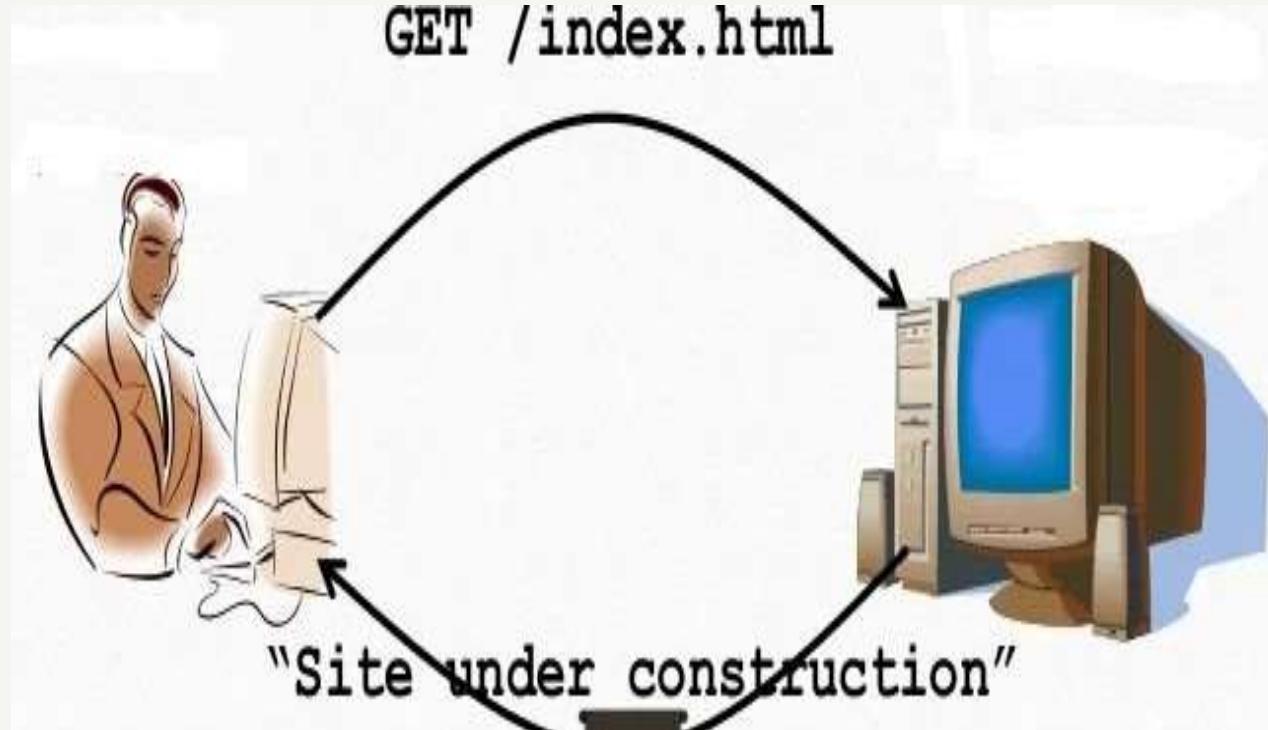
Client & Servers

1. Client Program

- Running on end host
- Requests Services
- E.g. Web Browser

2. Server Program

- Running on end host
- Provides Services
- E.g. Webserver



Client – Servers Communication

1. Client “sometimes on”

- Initiates a request to the server when interested
- E.g. Web Browser on your laptop or cell phone
- Doesn't communicate directly with other clients
- Needs to know the server's address

2. Server is “always on”

- Services requests from many client hosts
- E.g. Web Server for the www.example.com web site
- Doesn't initiate contact with the clients
- Needs a fixed, well-known address

Types of Internet connection

1. Digital subscriber line(DSL)
2. Cable Internet
3. Fiber Optic
4. Satellite Internet
5. Wireless
6. Broadband over Power lines(BPL)

Major Applications of the Internet

- 4 major applications of internet, which are given below.

1. Social Media Internet Applications

e.g .Facebook,Instagram,Twitter,LinkedIn...

2. Communications Internet Applications

e.g. Email,Skype,Zoom,Whatsapp...

3. Entertainment Internet Applications

e.g. Netflix,Hotstar,Youtube, Amazon prime video...

4. Travel Internet Applications

e.g Google Trips,Google Map,trivago...

What are protocols?

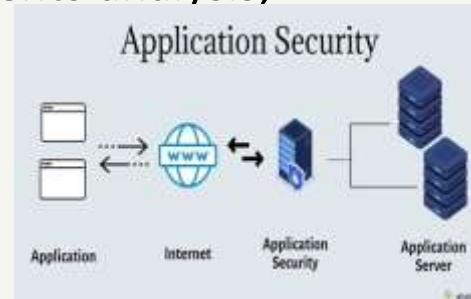
- A Network Protocol is a group of rules accompanied by the network.
- Network protocols will be formalized requirements and plans composed of rules, procedures, and types that describe communication among a couple of devices over the network.
- The protocol can be described as an approach to rules that enable a couple of entities of a communication program to transfer information through any type of variety of a physical medium.
- The protocol identifies the rules, syntax, semantics, and synchronization of communication and feasible error managing methods. In this article, we will discuss the different types of networking protocols.

Types of Protocols

1. HTTP or HTTPS
2. FTP(File Transfer protocols)
3. Email Protocols(POP3,SMTP)
4. TCP(Transmission control protocol) and UDP(User Datagram Protocol)

What is Application Security?

- Application security refers to security precautions used at the application level to prevent the theft or hijacking of data or code within the application.
- It includes security concerns made during application development and design, as well as methods and procedures for protecting applications once they've been deployed.
- All tasks that introduce a secure software development life cycle to development teams are included in application security shortly known as AppSec.
- Its ultimate purpose is to improve security practices and, as a result, detect, repair, and, ideally, avoid security flaws in applications.
- It covers the entire application life cycle, including requirements analysis, design, implementation, testing, and maintenance..



What is Application Security?

- All tasks that introduce a secure software development life cycle to development teams are included in application security shortly known as AppSec.
- Its ultimate purpose is to improve security practices and, as a result, detect, repair, and, ideally, avoid security flaws in applications.
- It covers the entire application life cycle, including requirements analysis, design, implementation, testing, and maintenance..

Software Applications

What is Application Software :

- It is a type of software application that helps in the automation of the task based on the Users Input.
- It can perform single or multiple tasks at the same period of time.
- There are the different application which helps us in our daily life to process our instructions based on certain rules and regulations.
- Application Software helps in providing a graphical user interface to the user to operate the computer for different functionality.
- The user may use the computer for browsing the internet, accessing to email service, attending meetings, and playing games.
- Different high-level languages are used to build application software.

Types of Application Software

- Application software
- System software
- Driver software
- Middleware
- Programming software

Application Software

- The most common type of software, application software is a computer software package that performs a specific function for a user, or in some cases, for another application.
- An application can be self-contained, or it can be a group of programs that run the application for the user.
- Examples of Modern Applications include office suites, graphics software, databases and database management programs, web browsers, word processors, software development tools, image editors and communication platforms.

Example:Microsoft Office, Paint, Powerpoint etc..

System Software

- These software programs are designed to run a computer's application programs and hardware.
- System software coordinates the activities and functions of the hardware and software.
- It controls the operations of the computer hardware and provides an environment or platform for all the other types of software to work in.
- The OS is the best example of system software; it manages all the other computer programs.
- Other examples of system software include the firmware, computer language translators and system utilities..

Example:Notepad ,Calculator etc..

Driver Software

- Also known as device drivers, this software is often considered a type of system software.
- Device drivers control the devices and peripherals connected to a computer, enabling them to perform their specific tasks.
- Every device that is connected to a computer needs at least one device driver to function.
- Examples include software that comes with any nonstandard hardware, including special game controllers, as well as the software that enables standard hardware, such as USB storage devices, keyboards, headphones and printers.

Example: Audio Driver, Video Driver etc..

Middleware

- The term *middleware* describes software that mediates between application and system software or between two different kinds of application software.
For example, middleware enables Microsoft Windows to talk to Excel and Word.
- It is also used to send a remote work request from an application in a computer that has one kind of OS, to an application in a computer with a different OS. It also enables newer applications to work with legacy ones.

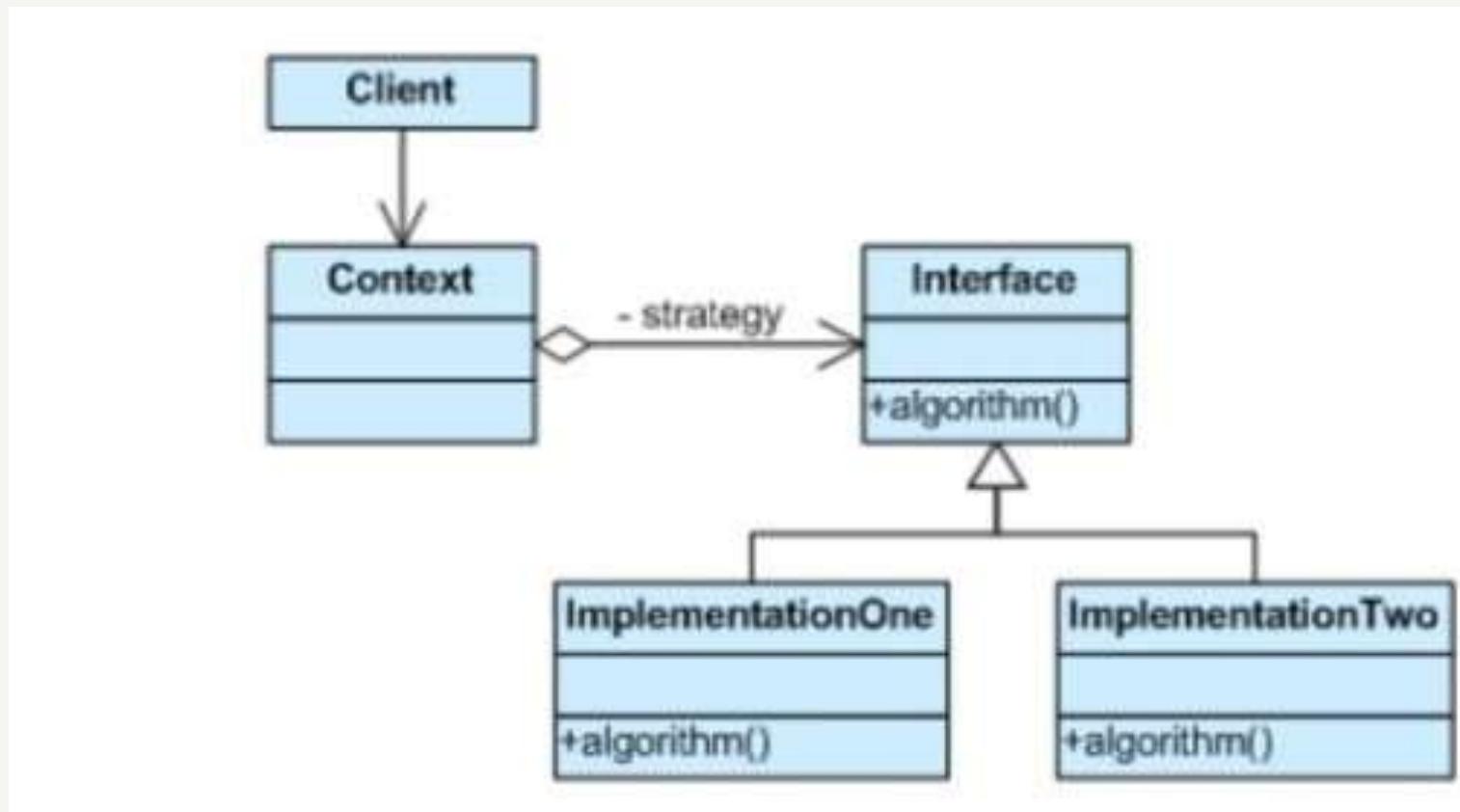
Example: database middleware, application server middleware

Programming Software

- Computer programmers use programming software to write code.
Programming software and programming tools enable developers to develop, write, test and debug other software programs.
- Examples of programming software include assemblers, compilers, debuggers and interpreters.

Examples : Turbo c,Eclipse,Sublime etc..

Architecture



Applicaton and Examples

- Shopping mall Example
 - 1. Accept Customer Details
 - 2. calculate the bill
 - 3. Apply Discount Based on day of the week
 - (a) Monday Low discount -10%
 - (b) Thursday High discount-50%



- Shopping Mall

- 1. Accept customer detail
- 2. Calculate bill amount
- 3. Apply discount based on day of week
 - 1. Monday - Low discount - 10%
 - 2. Thursday - High discount - 50%



- Customer Detail
- CalculateBill
- GetFinalBill

Shopping Mall will contain discount logic.



Shopping Mall

Discount
Logic

Open closed principle - software entities should be open for extension,
but closed for modification.

New discount strategy may be applied in future



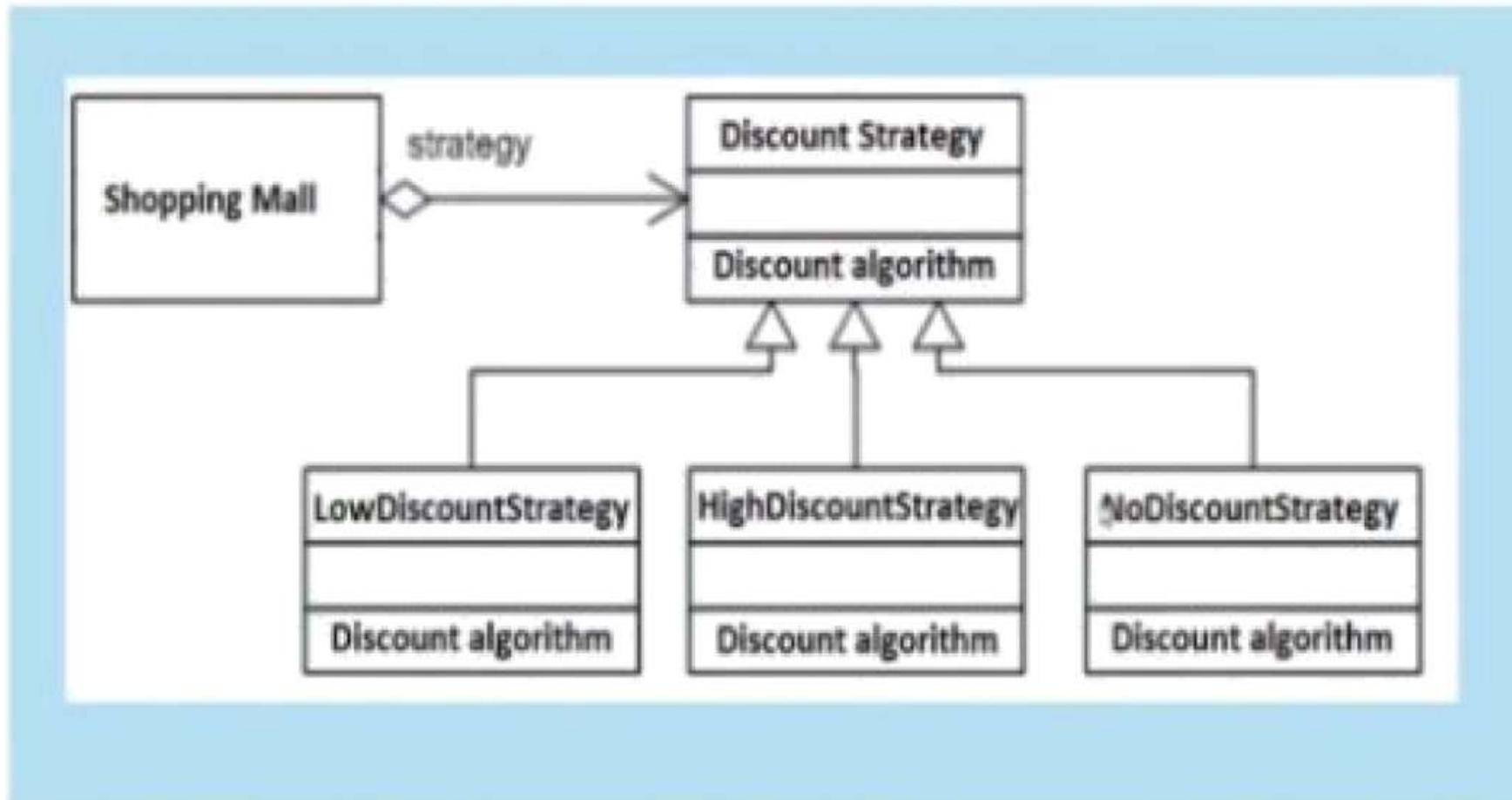
Open closed principle – software entities should be open for extension,
but closed for modification.



TOP#TECHNOLOGIE

S

Training | Outsourcing | Placement | Study Abroad



Software Architecture

- Software architecture is the blueprint of building software. It shows the overall structure of the software, the collection of components in it, and how they interact with one another while hiding the implementation.
- This helps the software development team to clearly communicate how the software is going to be built as per the requirements of customers.
- There are various ways to organize the components in software architecture. And the different predefined organization of components in software architectures are known as software architecture patterns.

- A lot of patterns were tried and tested. Most of them have successfully solved various problems. In each pattern, the components are organized differently for solving a specific problem in software architectures.
- Well, I hope you don't want to bore yourself by reading the endless types of software architecture patterns.

Layers in Software Architecture

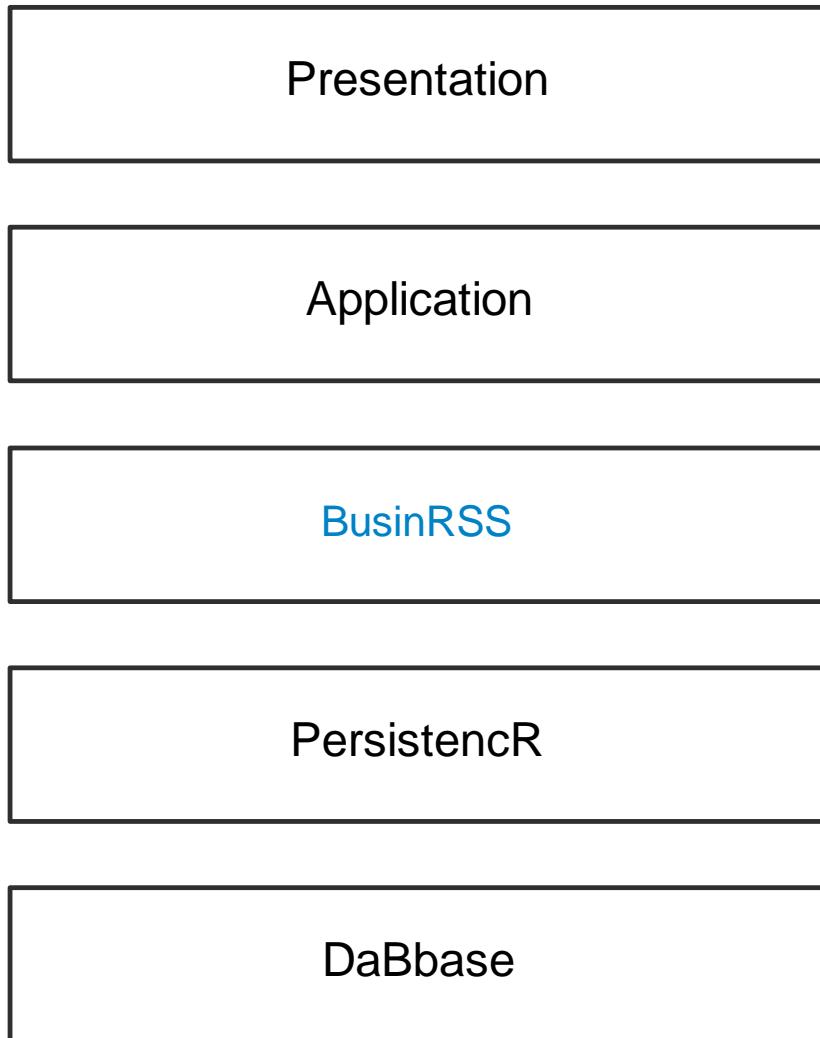
1. Presentation layer

1. Application layer

1. Business layer

1. Persistence layer

1. Database layer



Calls beh'een layers
flo\b/ do\b'inwards

1. Presentation layer

The presentation layer, also called the UI layer, handles the interactions that users have with the software. It's the most visible layer and defines the application's overall look and presentation to the end-users. This is the tier that's most accessible, which anyone can use from their client device, like a desktop, laptop, mobile phone or tablet.

2. Application layer

The application layer handles the main programs of the architecture. It includes the code definitions and most basic functions of the developed application. This is the layer that programmers spend most of their time in when working on the software.

You can use this layer to implement specific coordination logic that doesn't align exactly with either the presentation or business layer.

3. Business layer

The business layer, also called the domain layer, is where the application's business logic operates. Business logic is a collection of rules that tell the system how to run an application, based on the organization's guidelines. This layer essentially determines the behavior of the entire application. After one action finishes, it tells the application what to do next.

4. Persistence layer

The persistence layer, also called the data access layer, acts as a protective layer. It contains the code that's necessary to access the database layer. This layer also holds the set of codes that allow you to manipulate various aspects of the database, such as connection details and SQL statements.

5. Database layer

The database layer is where the system stores all the data. It's the lowest tier in the software architecture and houses not only data but indexes and tables as well. Search, insert, update and delete operations occur here frequently. Application data can store in a file server or database server, revealing crucial data but keeping data storage and retrieval procedures hidden.

Environments in industry

There are different Types of environments in Industry.

1. The analysis and design environment
2. The development environment
3. The common build environment
4. The testing environment
5. The production environment



Environments in industry

1. Analysis & Design Environment

- The analysis and design environment is aligned to the **planning and analysis phases** of the SDLC. In this environment, the main processes that take place include carrying out an in-depth examination of the current system and the proposed system. The system architecture is also defined and includes developing the design of the hardware, software, and network requirements for the system. Within this environment, systems and business analysts work closely with software engineers.

2. The development environment

- The development environment can also be a physical space where development takes place and where software engineers interact. Another example of the development environment is the **integrated development environment** (IDE). The IDE provides a platform where tools and development processes are coordinated in order to provide software engineers a convenient way of accessing the resources they require during the development process.

Environments in industry

3. The common build environment

- The **common build environment** is closely aligned to the development phase of the SDLC. In this environment, software engineers merge the work done in the development environment. Within this environment, software engineers build systems. These are used to automate the process of software compilation.

4. The testing environment

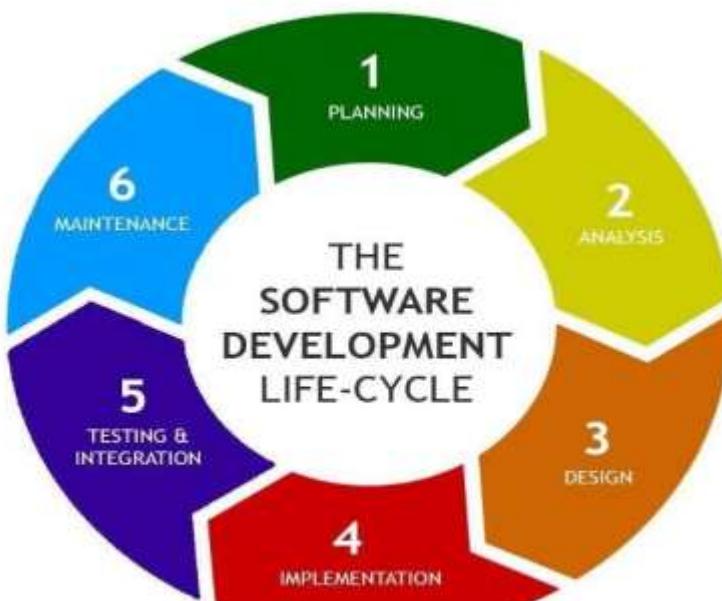
- The test environment is where testing teams evaluate the application/quality. program's This also allows computer programmers to find out and solve any defects that may interfere with the application's smooth operation or degrade the user experience.

5. The production environment

- When the end-user use a web/mobile application, the program is operating on a production server. It's been created in the production environment.

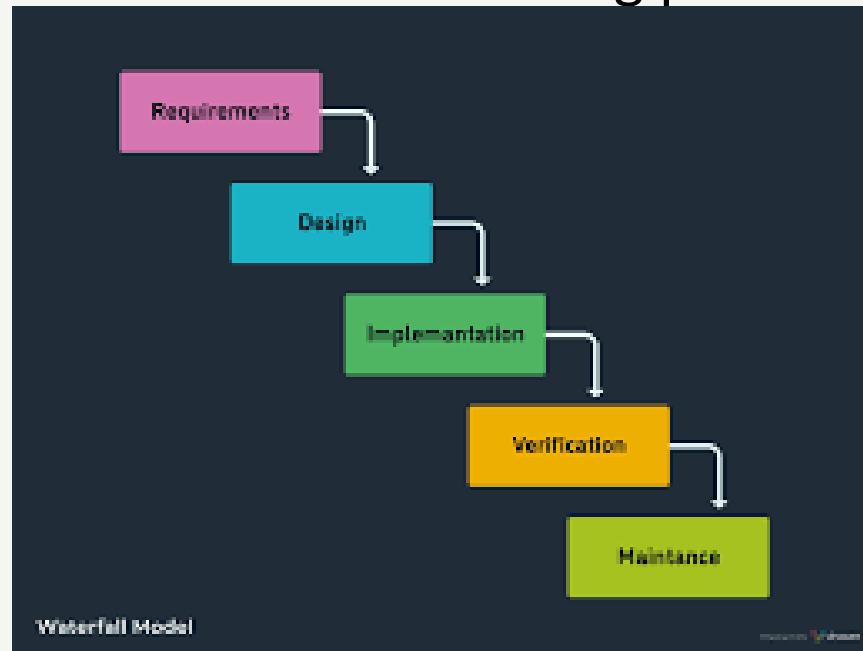
SDLC

- The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software.



SDLC Methodology

- The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. In detail, the SDLC methodology focuses on the following phases of software development:
 - Requirement Gathering
 - Analysis
 - Designing
 - Implementation
 - Testing
 - Maintenance



Programming

There are countless definitions of what computer programming is, but here is Ours.

“Programming is how you get computers to solve problems.”

There are two key phrases here that are important:

- **You:** without the programmer (you), the computer is useless. It does what **you** tell it to do.
- **Solve problems:** computers are tools. They are complex tools, admittedly, but they are not mysterious or magical: they exist to make tasks easier.

Programming

Computer programs make computers work

Computer programs (or software) are what makes computers work. Without software, modern computers are just complicated machines for turning electricity into heat. It's software on your computer that runs your operating system, browser, email, games, movie player – just about everything.

Programming

Programming is

Programming is a creative task: there is no right or wrong way to solve a problem, in the same way, that there is no right or wrong way to paint a picture.

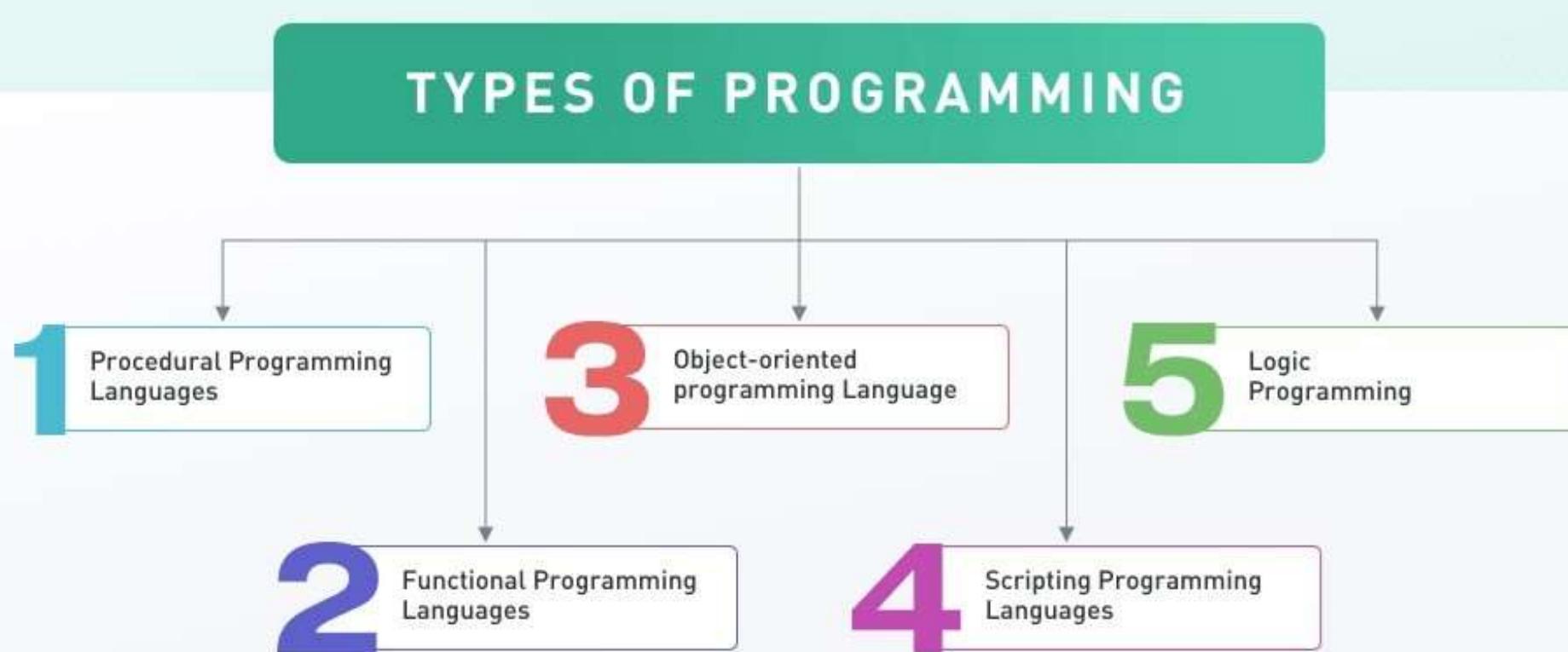
There are choices to be made, and one way may seem better than another, but that doesn't mean the other is wrong! With the right skills and experience, a programmer can craft software to solve an unlimited number of problems – from telling you when your next train will arrive at playing your favourite music.

The possibilities are constrained only by your imagination. That's why I love programming. Programming having a multiple Programming Languages.

Programming Languages

- A **programming language** is a computer language programmers use to develop software programs, scripts, or other sets of instructions for computers to execute.
- Although many languages share similarities, each has its own syntax. Once a programmer learns the languages rules, syntax, and structure, they write the source code in a text editor or IDE. Then, the programmer often compiles the code into machine language that can be understood by the computer. Scripting languages, which do not require a compiler, use an interpreter to execute the script.
- A programming language is a set of instructions that can be used to interact with and control a computer. These languages are used to design websites, create apps, develop operating systems, control spacecraft, and analyze data. Programming languages are necessary because computers can't understand English. Programming languages bridge this gap by helping programmers translate

Types of Programming Languages



Types Of Programming Languages

1. Low-level programming language

Low-level language is **machine-dependent** (0s and language. The processor runs low- level programs need of a compiler or interpreter, so the programs language can be run very fast.

Low-level language is further divided into two parts -

i. Machine Language

Machine language is a type of low-level programming language. It is also called as **machine code or object code**. Machine language is easier to read because it is normally displayed in binary or hexadecimal form (base 16) form. It does not require a translator to convert the programs because computers directly understand the machine language programs.

The advantage of machine language is that it helps the programmer to programs faster than the high-level programming language.

ii. Assembly Language

Assembly language (ASM) is also a type of low-level programming language that is designed for specific processors. It represents the set of instructions in a **symbolic and human-understandable form**. It uses an assembler to convert the assembly language to machine language.

The advantage of assembly language is that it requires less memory and less execution time to execute a program.

2. High-level programming language

High-level programming language (HLL) is designed for **developing user-friendly software programs and websites**. This programming language

requires a compiler or interpreter to translate the program into machine language (execute the program).

The main advantage of a high-level language is that it is **easy to write, and maintain**.

High-level programming language includes **Python, Java, JavaScript, PHP, C++, Objective C, Cobol, Perl, Pascal, LISP, FORTRAN, and Swift**
languagening

A high-level language is further divided into three parts -

i. **Procedural Oriented programming language**

Procedural Oriented Programming (POP) language is derived from structured programming and based upon the procedure call concept. It divides a program into small procedures called **routines or functions**.

Procedural Oriented programming language is used by a software programmer to create a program that can be accomplished by using a programming editor like IDE, Adobe Dreamweaver, or Microsoft Visual Studio.

The advantage of POP language is that it helps programmers to track the program flow and code can be reused in different parts of the program.

Example: C, FORTRAN, Basic, Pascal,
etc

ii. Object-Oriented Programming language

Object-Oriented Programming (OOP) language is **based upon the objects**. In this **programming language**, **programs** are **divided** into **small parts** **called objects**. It is used to implement real-world entities like inheritance, polymorphism, abstraction, etc in the program to makes the program reusable, efficient, and easy-to-use.

The main advantage of object-oriented programming is that OOP is faster and easier to execute, maintain, modify, as well as debug.

Example: C++, Java, Python, C#,

etc

iii. Natural language

Natural language is a **part of human languages** such as English, Russian, German, and Japanese. It is used by machines to understand, manipulate, and interpret human's language. It is used by developers to **perform tasks such as translation, automatic summarization, Named Entity Recognition**

(NER), relationship extraction, and topic

The main advantage of natural language is that it can answer questions in any subject and directly respond within seconds.

3. Middle-level programming language

Middle-level programming language **lies between the programming language and high-level programming language.** It is also known as the intermediate programming language and pseudo-language.

A middle-level programming language's advantages are that it has the features of high-level programming, it is a user-friendly language, and is closely related to machine language and human language.

Example: C, C++,
language

Important Programming Language

- **Development is Mainly Divided into two Parts**

Front end Development

HTML

CSS

JAVASCRIPT

React

Angular

Important Programming Language

Backend Development

In Backend you have as many languages you can choose anyone language

If you are Choosing web development platform then you have a Mainly Three Languages

PHP

JAVA

PYTHON

Dot Net

Important Programming Language

HTML:

HTML—“HyperText Markup Language”—is **the language used to tell your web browser what each part of a website is**. So, using HTML, you can define headers, paragraphs, links, images, and more, so your browser knows how to structure the web page you're looking at.

CSS:

CSS makes the front-end of a website shine and it creates a great user experience. Without CSS, websites would be less pleasing to the eye and likely much harder to navigate. In addition to layout and format, CSS is responsible for font color and more.

Important Programming Language

JAVASCRIPT:

JavaScript has evolved over the past 25 years **to become a versatile and accessible programming language for working with web browsers.** Developers use JavaScript to build complex interactive websites and browser games, and to connect servers to websites and web applications

Important Programming Language

Programming Language in

The backend (or “server-side”) is the portion of the website you don't see. It's **responsible for storing and organizing data, and ensuring everything on the client-side actually works.** The backend communicates with the frontend, sending and receiving information to be displayed as a web page.

Writing Source Code

What is Source Code?

Source code is the source of a computer program. It contains declarations, instructions, functions, loops and other statements, which act as instructions for the program on how to function.

Programs may contain one or more source code text files, which can be stored on a computer's hard disk, in a database, or be printed in books of code snippets.

Important Notes For Source Code

- Coding Must Have Comments
- Coding Must Have a Proper Architecture
- Coding Must Have a
- Declaring Coding Versions
- Divide your code in packages

Running Code

When you will run your code two things are Most Important

1.Com piler

2.Interpreter

1.Com piler:

It is a translator which takes input i.e., High-Level Language, and produces an output of low-level language i.e. machine or assembly language.

- A compiler is more intelligent than an assembler it checks all kinds of limits, ranges, errors, etc.

- But its program run time is more and occupies a larger part of memory.
It has a slow speed because a compiler goes through the entire program
and then translates the entire program into machine codes.



2. Interpreter :

An interpreter is a program that translates a programming language into a comprehensible language. –

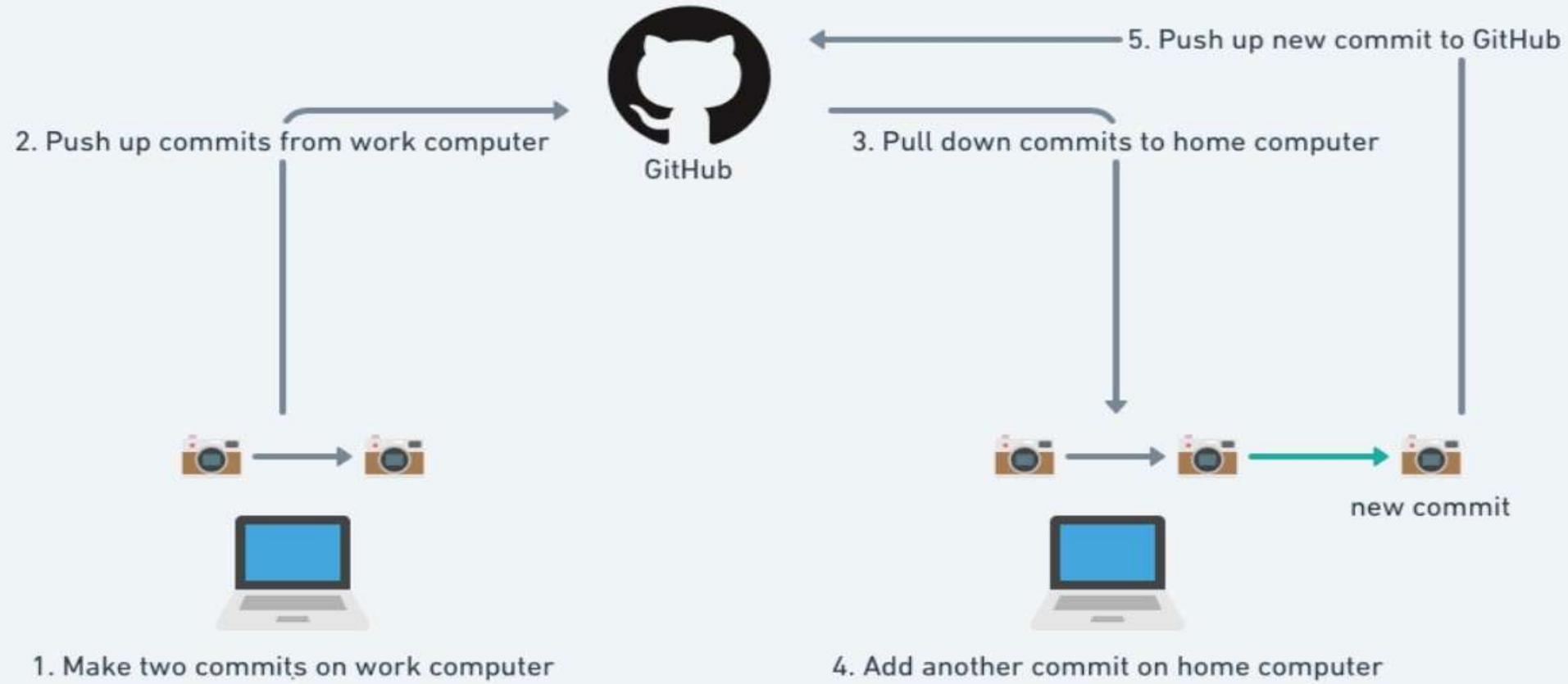
- It translates only one statement of the program at a time.
- Interpreters, more often than not are smaller than compilers.



GitHub

- GitHub is one of the most popular resources for developers to share code and work on projects together. It's free, easy to use, and has become central in the movement toward open-source software.
- Git is used for managing the changes to a project over time. A project might be just a single file, a handful of files, or thousands of files. Those files can be anything from plain text to images or videos.
- Because Git is focused on managing changes, it is often used as a collaboration tool allowing people to work on the same project at the same time. By tracking their individual changes, Git can bring everything together to the final version.

Working with GitHub



Module - 2

[Fundamentals of Programing]

Basic Concepts of Programming Languages

1. Syntax
2. Data Structures
3. Variables
4. Operators
5. Control & Looping Structures
6. Functions
7. Arrays & Strings
8. File Handling

Overview of C Programming

What is C Programming?

C programming is a high-level, general-purpose programming language developed in the 1970s. It is widely used for system and application software development due to its efficiency and portability. C provides low-level access to memory and allows direct manipulation of hardware, making it ideal for developing operating systems, embedded systems, and other performance-critical applications.

History and Evolution

C programming was developed by Dennis Ritchie at Bell Labs in 1972 as an evolution of the B language, primarily for system programming. It gained popularity through its use in developing the Unix operating system. In the 1980s, it was standardized as ANSI C, and over time, updates like C99 and C11 introduced new features. C remains a foundational language, influencing many modern languages and widely used in system-level and performance-critical applications.

Importance and Applications

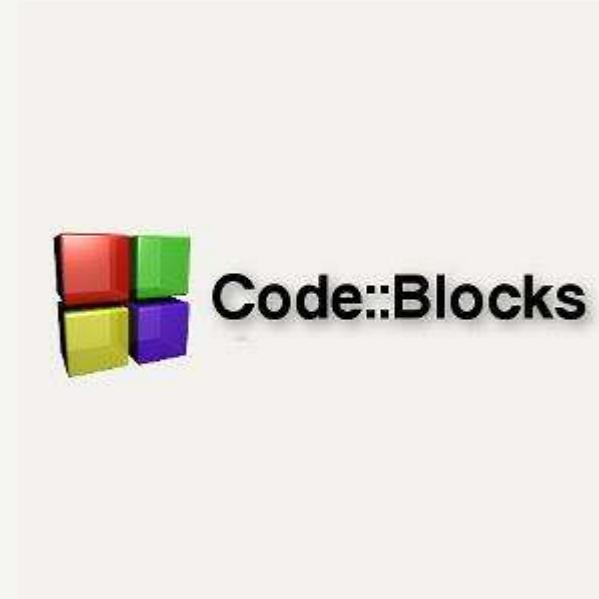
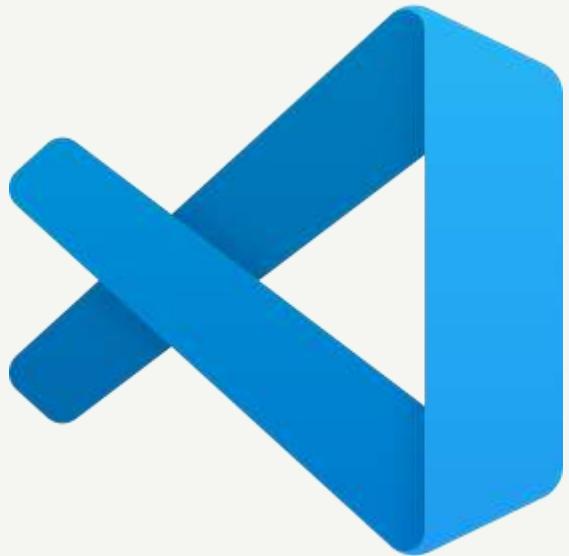
C programming is crucial for system-level programming due to its efficiency, low-level memory access, and portability. It is widely used in developing operating systems, embedded systems, and device drivers. C also forms the foundation for many modern languages like C++, Java, and Python, and is used in performance-critical applications like real-time systems, networking, and gaming. Its role in developing compilers and interpreters further highlights its importance in the software development ecosystem.

Setting Up Environment

Installing a C Compiler (e.g., GCC)

To install a C compiler, first, download and install a compiler like **GCC**. On Linux, you can install GCC using the command `sudo apt-get install gcc`. On Windows, you can use **MinGW** to install GCC. After installation, ensure the compiler is added to the system's PATH variable to compile C programs from the command line.

Choosing an IDE (DevC++. VS Code, Codeblocks, etc)



Writing Your First Program

To write your first C program, follow these steps:

1. Open your text editor or IDE (like Code::Blocks or Visual Studio Code).
2. Write the following code:

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Writing Your First Program

To write your first C program, follow these steps:

1. Open your text editor or IDE (like Code::Blocks or Visual Studio Code).
2. Write the following code:

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Writing Your First Program

Save the file with a .c extension (e.g., hello.c).

Compile the program using a C compiler (e.g., gcc hello.c -o hello on the command line).

Run the program (./hello on Linux/Mac or hello.exe on Windows), and it should display "Hello, World!" on the screen.

Basic Structure of a C Program

Structure of a C Program

A C program typically consists of three main parts:

- 1. Preprocessor Directives:** Instructions like `#include` to include libraries, which are processed before compilation.
- 2. Main Function:** The entry point of the program, written as `int main()`, where execution begins.
- 3. Functions and Statements:** The body of the program containing logic and functions to perform specific tasks, with the main function returning an integer value (`return 0;`).

Comments in C

In C, comments are used to add explanatory notes within the code, which are ignored by the compiler. There are two types of comments:

- 1. Single-line comments:** Use `//` to comment out a single line.

- 2. Multi-line comments:** Enclosed between `/*` and `*/`, used for comments spanning multiple lines.

Comments help improve code readability and provide explanations for complex sections.

Data Types and Variables

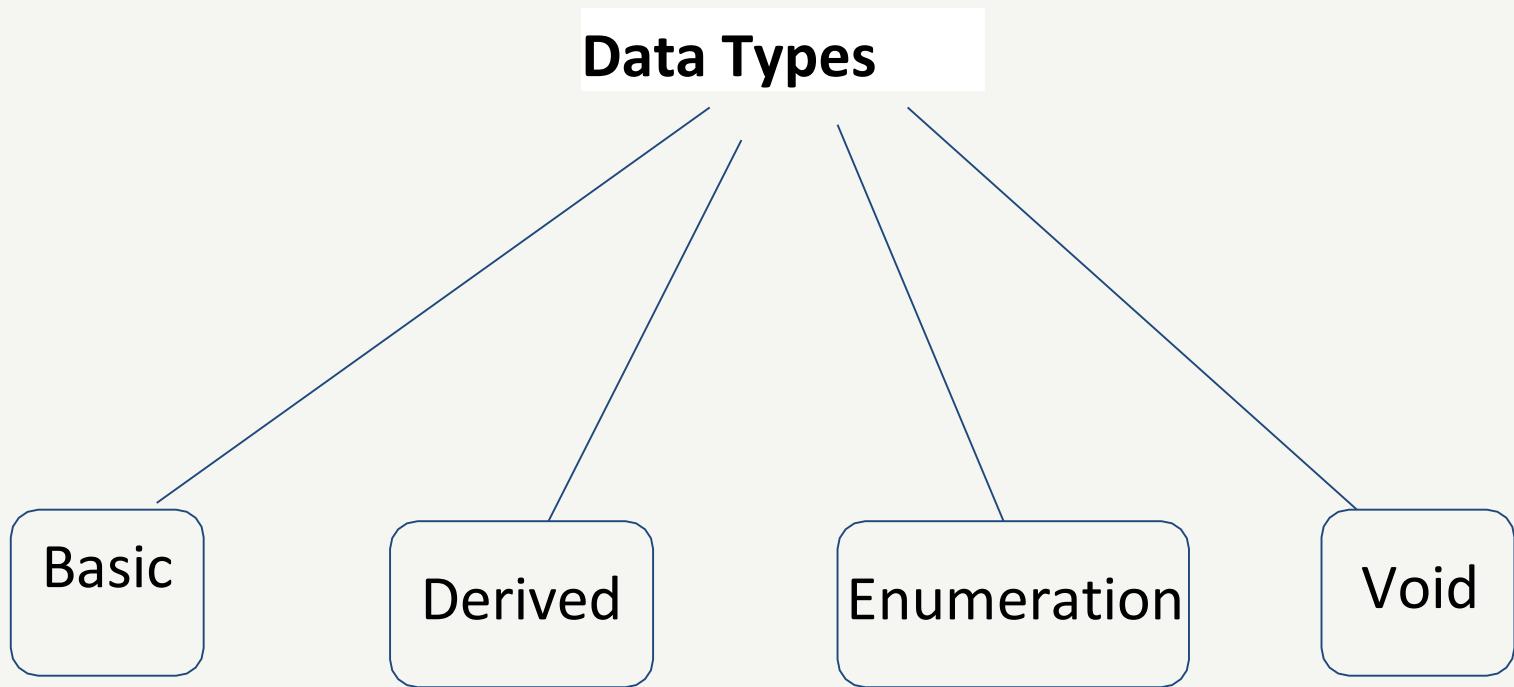
What are Data Types?

- A data type specifies what type of data a variable can store such as integer, floating, character,

etc. **int, float, char**

```
int res =  
5;
```

Types of Data Types



Basic Data Types

- The basic data types are integer- and floating-point based.

float zab1 = 5.34;

int res = 5;

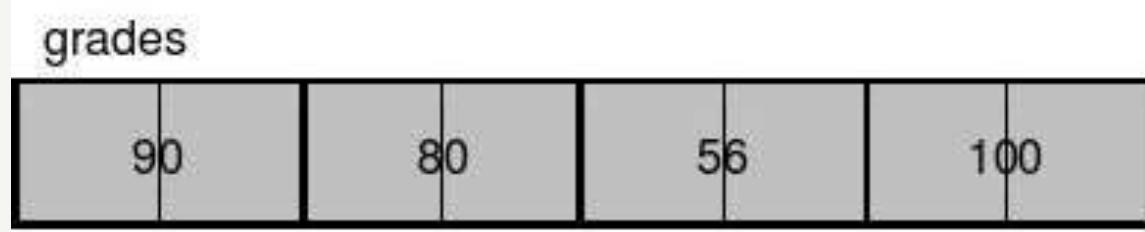
char b = 'G';

Basic Data Types

Data Types Memory Size Range		
short int	2 byte	-32,768 to 32,767
signed short int	2 byte	-32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte	
double	8 byte	
long double	10 byte	

Derived Data Types

- Array
- s
- Pointers
- Union
- Structure



```
int * p = &n;
```

Void

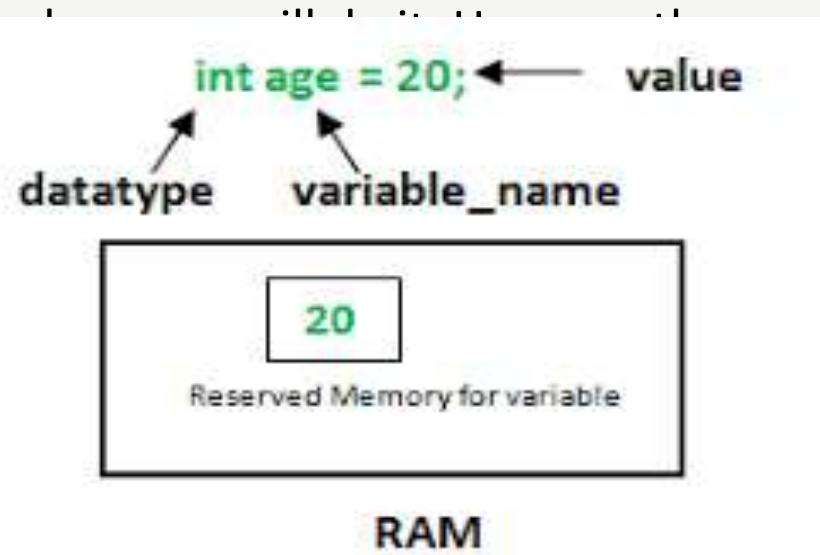
- Void is an empty data type that has no value.

```
void num();
```

Variables

3. Variables

- Variables are the names you give to computer memory locations which are used to store values in a computer program.
- For example, assume you want to store two values 10 and 20 in your program and at a later stage, you want to use these two values. Let's follow three simple steps –
 - i. Create variables with appropriate names.
 - ii. Store your values in those two variables.
 - iii. Retrieve and use the stored values from the variables.

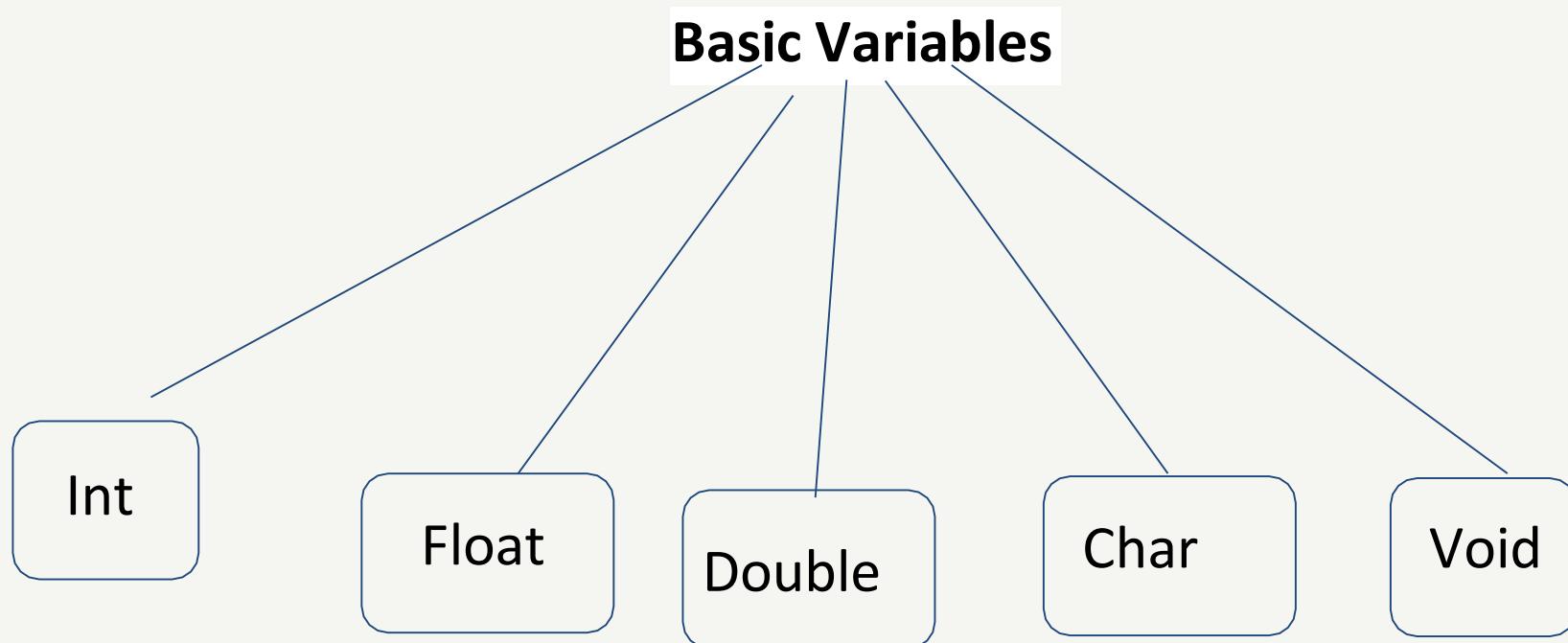


Deep Dive in Variables

- A variable is a name given to a storage area our programs can use to store values.
- The name of a variable can be composed letters, digits, and the underscore character.

```
int res =  
5;
```

Basic Types of Variables



Variable Declaration

- A variable declaration tells the where and how much storage to for the variable.

```
float f,  
salary;  
double d;
```

Variable Definition

- A variable definition assigns a value in variable
 - .

i = 43;

Refer This Example :

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/variable/variables.cpp>

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/variable/globalVariable.cpp>

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/variable/localVariable.cpp>

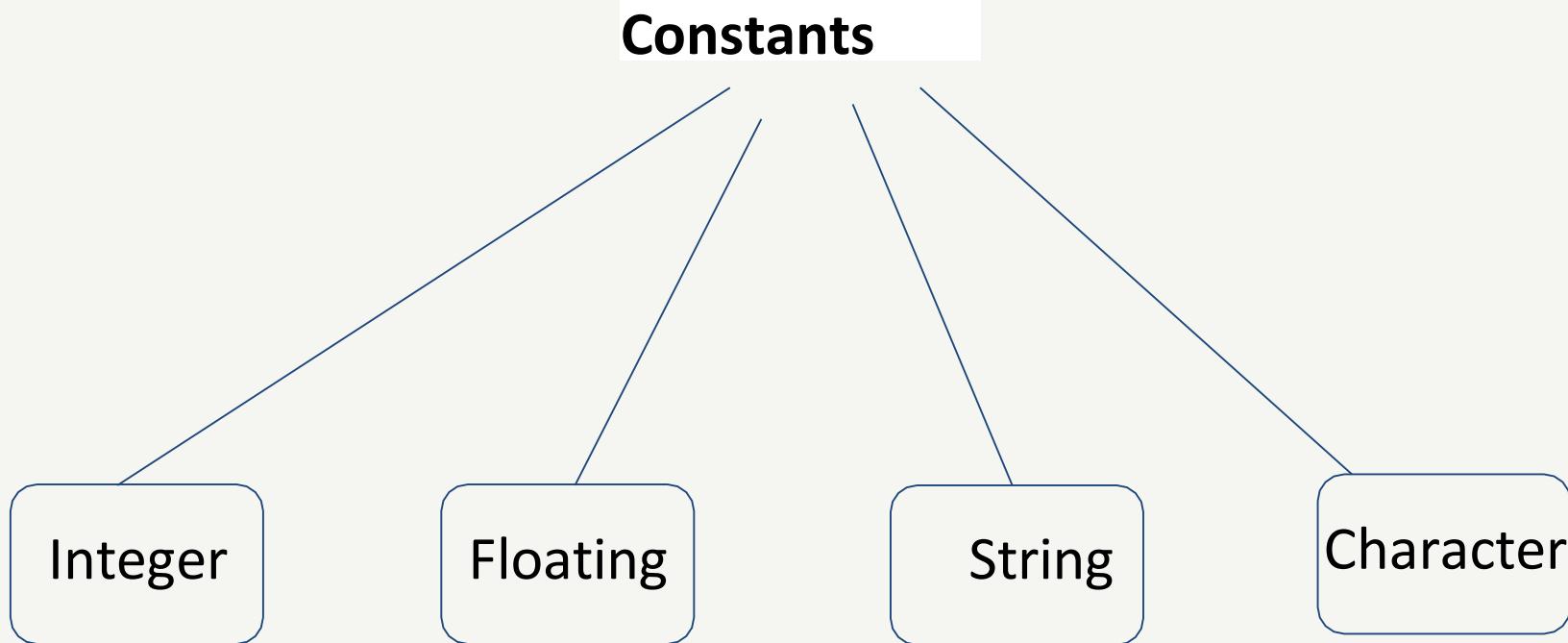
<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/variable/typeCasting.cpp>

What are Constants?

- Constants are the fixed value in program . Which means that we cannot change it's value.

```
const int res = 5;
```

Types of Constants



const keyword

- Using const keyword
- Syntax : const data_type variable_name = value ;

**Ex. - const int a =
10;**

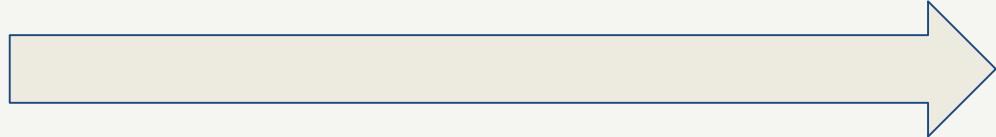
Constants

- A constant is a value assigned to variable which will remain the same throughout the program

```
const int abc = 5;
```

Keywords

- Some Reserved keywords names given in the next slide cannot be used as identifier names.



Keyword

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Identifiers

- Identifier is a name used to identify variable, function, or any user-defined item.

Ex. - num1, getchar(), sum, ab_c etc.

```
num1 = 3;
```

Identifiers

- Rules for writing the names of Identifiers
 - An identifier starts with a letter A to Z, a to z,
 - or an underscore '_'.
Followed by zero or more letters, underscores, and digits (0 to 9).

```
num1 = 3;
```

Refer This Example :

<https://github.com/TopsCode/Software-Engineering/blob/master/C/dataType.c>

<https://github.com/TopsCode/Software-Engineering/blob/master/C/inputOutput1.c>

<https://github.com/TopsCode/Software-Engineering/blob/master/C/simpleProgram.c>

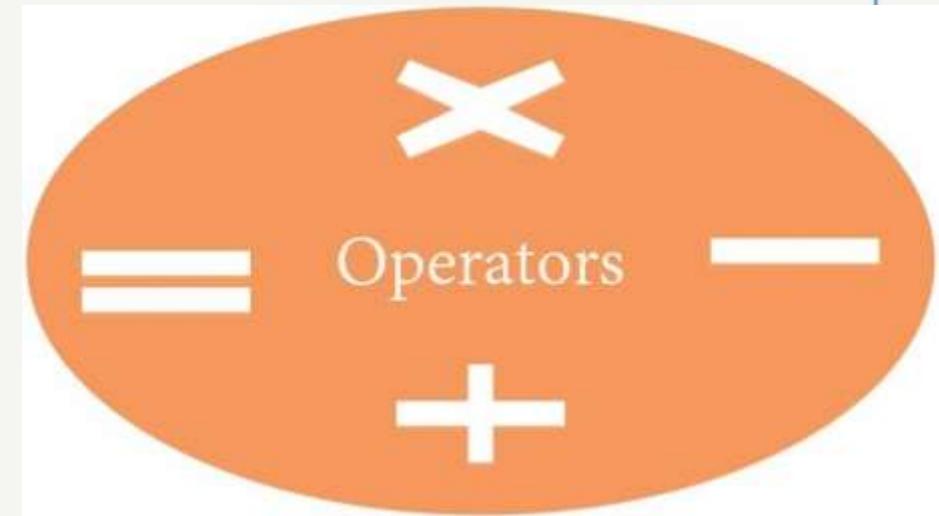
<https://github.com/TopsCode/Software-Engineering/blob/master/C/inputOutput.c>

Operators

What are Operators?

- A symbol that takes one or more operands such as variables, expressions or values and operates on them to give an output.

Ex. - $=, +, -, /, *, ==, ++, --, \%,$
etc.



Operators

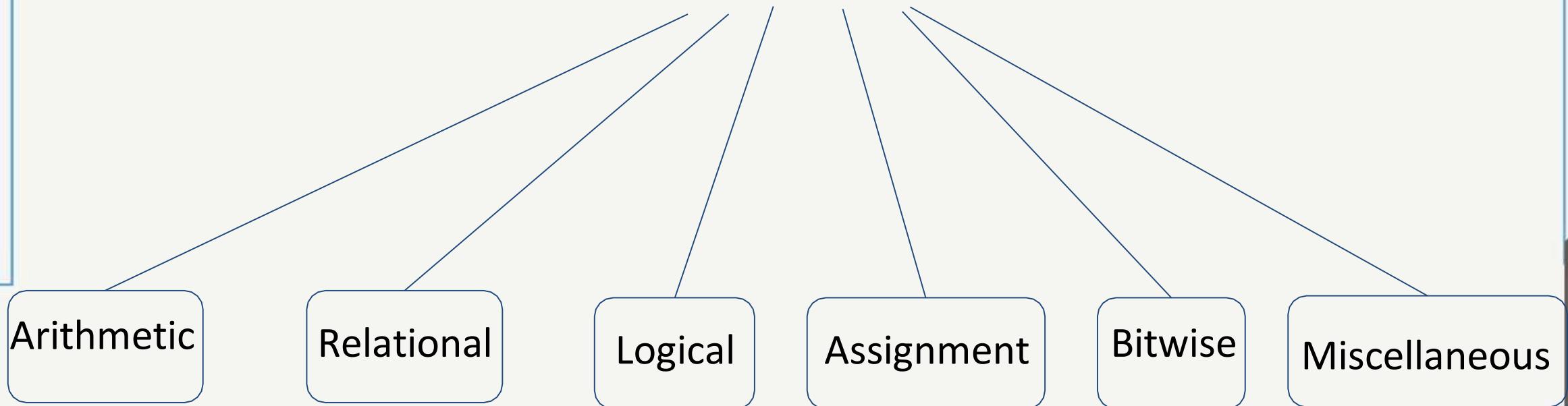
- An operator is a **symbol** that tells the compiler to perform specific mathematical or logical functions.

a= b + c;

Ex. - +, -, *, /, ==, ++, --

Types of Operators

Operators



Arithmetic Operators

Operator	Function	Example
+	Addition	<code>var=a+b</code>
-	Subtraction	<code>var=a-b</code>
*	Multiplication	<code>var=a*b</code>
/	Division	<code>var=a/b</code>
%	Modulo	<code>var=a%b</code>
++	Increment	<code>var++</code>
--	Decrement	<code>var-</code>

Relational Operators

Operator	Meaning of Operator	Example
<code>==</code>	Equal to	<code>5 == 3</code> is evaluated to 0
<code>></code>	Greater than	<code>5 > 3</code> is evaluated to 1
<code><</code>	Less than	<code>5 < 3</code> is evaluated to 0
<code>!=</code>	Not equal to	<code>5 != 3</code> is evaluated to 1
<code>>=</code>	Greater than or equal to	<code>5 >= 3</code> is evaluated to 1
<code><=</code>	Less than or equal to	<code>5 <= 3</code> is evaluated to 0

Logical Operators

Operator	Meaning	Example
<code>&&</code>	Logical AND. True only if all operands are true	If $c = 5$ and $d = 2$ then, expression $((c==5) \&\& (d>5))$ equals to 0.
<code> </code>	Logical OR. True only if either one operand is true	If $c = 5$ and $d = 2$ then, expression $((c==5) \mid\mid (d>5))$ equals to 1.
<code>!</code>	Logical NOT. True only if the operand is 0	If $c = 5$ then, expression $!(c==5)$ equals to 0.

Assignment Operators

- The Basic type of Assignment operator is '='.
- There are other derived operators

Ex. - *=, -=, /=, += etc.

c += 15;

c = c + 15;

Bitwise Operators

Operators	Meaning of operators
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Miscellaneous Operators

Operat or	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
?:	Conditional Expression.	If Condition is true ? then value X : otherwise value

Refer this Example:

- **Assignment Operator**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/AssignmentOperator.c>

- **Arithmetic Operator Example**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/ArthmeticOperator.c>

- **Logical Operator Example**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/logicalOperator.c>

Practical Assignments:

1. Write a C program to print your name.
2. Create a program to calculate the sum of two numbers.
3. Write a program to find the area of a circle given its radius.
4. Develop a program to convert temperature from Celsius to Fahrenheit.

Control Flow Statement

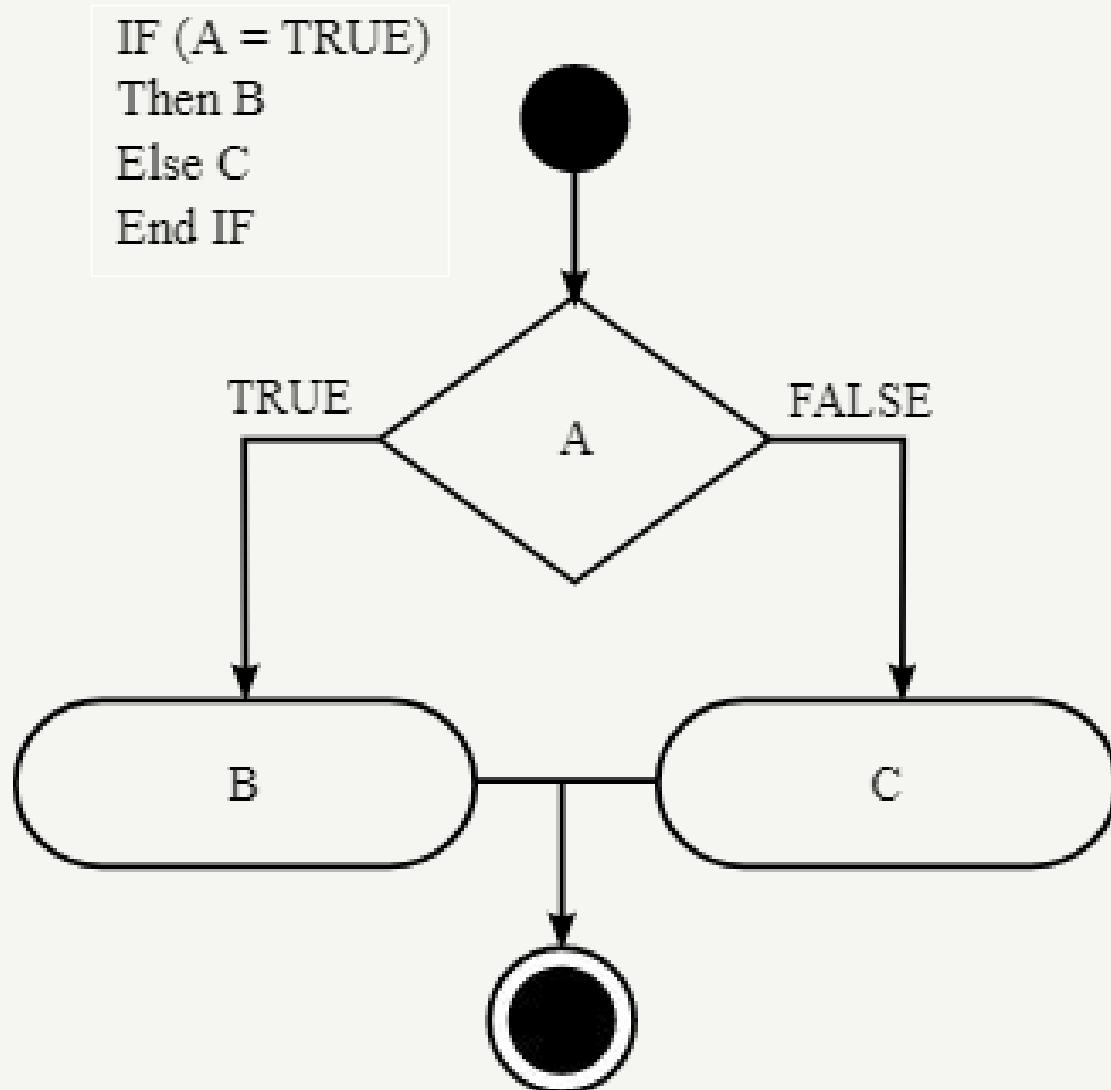
What is Control Structure?

- Real life situations where we have to condition based decisions by asking 'if' questions.

Ex. - if age is above 18, I am allowed drive vehicles or else not allowed.



Flow of Control Structure



Control structure Statements

- Following types of Statements:
 - if
 - if else
 - Nested if
 - Switch
 - Nested switch

If Statements

- if statement is the basic decision making statement
- Used to decide whether a certain statement or block of statements will be executed or not

If Statements

- Syntax :

```
if( condition )  
{  
    statement_1 ; // true block  
    statements  
}  
statement x ;
```

Example :

<https://github.com/TopsCode/Software-Engineering/blob/master/C/if.c>

If else Statements

- if else statement allows selecting any one of the two available options depending upon the output of the test condition

If else Statements

- **Syntax :**

```
if (condition )  
{  
    statements; // true  
}           statement  
else  
{  
    statements ; // false statement  
}
```

Example :

<https://github.com/TopsCode/Software-Engineering/blob/master/C/ifElse.c>

Nested If Statements

- Nested if statement is simply an if statement embedded with another if statement

Nested If Statements

- Syntax

:

```
if (condition1)
{
    statements ;      // executes when condition1 is
    if ( condition2) true
    {
        statements ;      // executes when condition2 is
        true
    }
}
```

Example :

<https://github.com/TopsCode/Software-Engineering/blob/master/C/ifElseLadder.c>

Refer this

Example:

If statement:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/if.c>

- If-else:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/ifElse.c>

- If else with compound Relational Test:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/compoundRelationalTest.c>

- Nested if-else

<https://github.com/TopsCode/Software-Engineering/blob/master/C/NestedIfElse.c>

- If-else Ladder:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/ifElseLadder.c>

Switch Statements

- Switch case statements are a substitute for long if statements that compare a variable to several integer values

Switch Statements

- Syntax

```
:
```

```
switch ( n )
```

```
{
```

```
case 1 :
```

```
    break;
```

```
case 2 :
```

```
    break ;
```

```
default :
```

```
// executed when n =  
1  
// executed when n =  
2  
// executed when n doesn't match any  
case
```

<https://github.com/TopsCode/Software-Engineering/blob/master/C/switchCase.c>

Example :

Nested Switch Statements

- Nested Switch Statements occurs when a switch statement is defined inside another switch statement.

Nested Switch Statements

- Syntax :

```
switch(ch1)
{
    case 'A': printf ("\n This A is part of outer switch " );
    switch (ch2)
    {
        case 'A' : printf(" \n This A is part of inner switch " );
        break;
        case' B' :
            }
        break ;
        case' B' :
    }
```

Looping in C

What are Loops?

- A loop statement allows us to execute a statement or group of statements multiple times based on a condition

Ex. - printing 1 to 100 on the output screen



Types of Loops

- **Entry Controlled** A condition is checked before executing the loop. It is also called as a pre-checking loop.
- **Exit Controlled** A condition is checked after executing the loop. It is called as a post-checking loop.

Entry controlled Loops

1. **For Loops-** It is a repetition control structure that allows you to efficiently write loop that needs to execute a specific number of times.

2. **While Loops-** It repeatedly executes a target statement as long as the condition is true given.

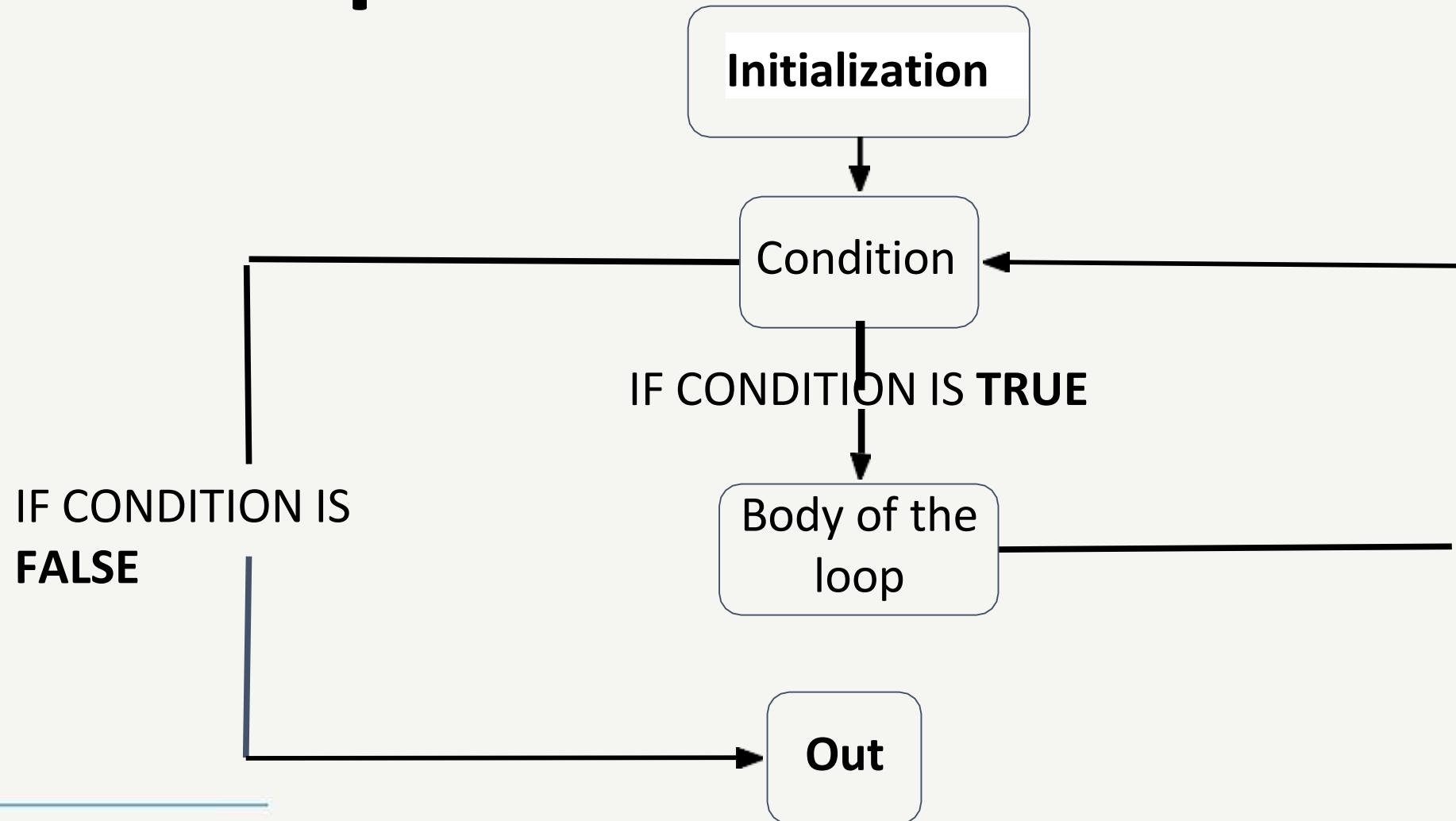
For Loops

- Used to efficiently write a loop that needs to execute a specific number of times.

Syntax :

```
for ( initialization ; test condition ; increment )  
{  
    Body of loop  
}
```

For Loop Flow Chart



For Loop

- All for loops are executed in following sequence
 - : Step 1 : It executes initialization statements
 - Step 2 : It checks the condition;
 - if true then go to Step 3
 - other wise Step 4
 - Step 3: Executes loop and go to Step 2
 - Step 4 : Out of the Loop

Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/ForLoop.c>

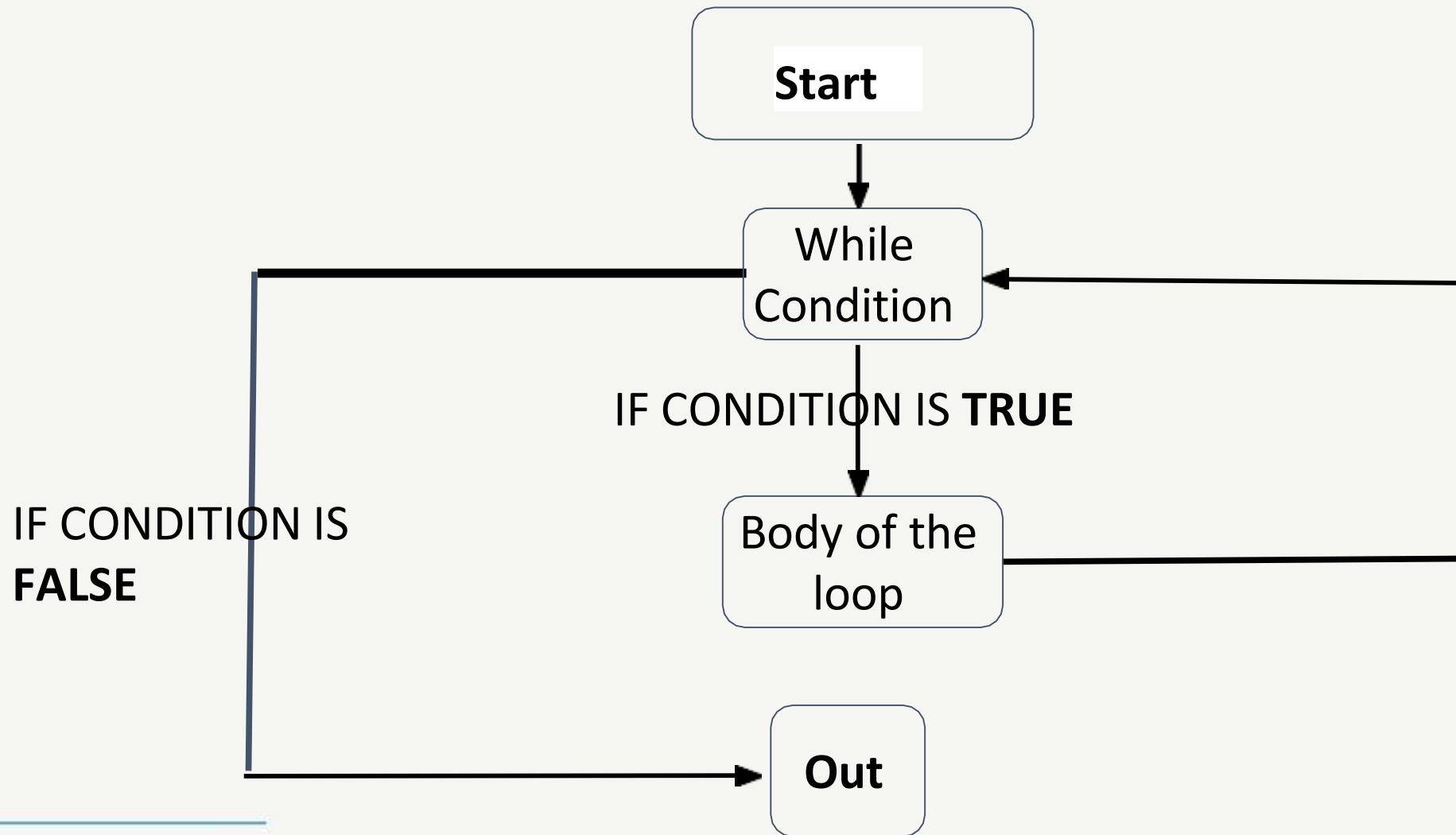
While Loops

- It repeatedly executes a target statement as long as the given condition is true

Syntax :

```
while (condition )  
{  
    Body of loop  
}
```

While Loop Flow Chart



While Loops

- All while loops are executed in following sequence
 - :

Step 1 : It checks the test
condition if true then
go to Step 2 other wise
to Step 3

Step 2 : Executes body of loop and go to Step
1. Step 3 : Other statements of program.

Example :

<https://github.com/TopsCode/Software-Engineering/blob/master/C/while.c>

Exit controlled Loops

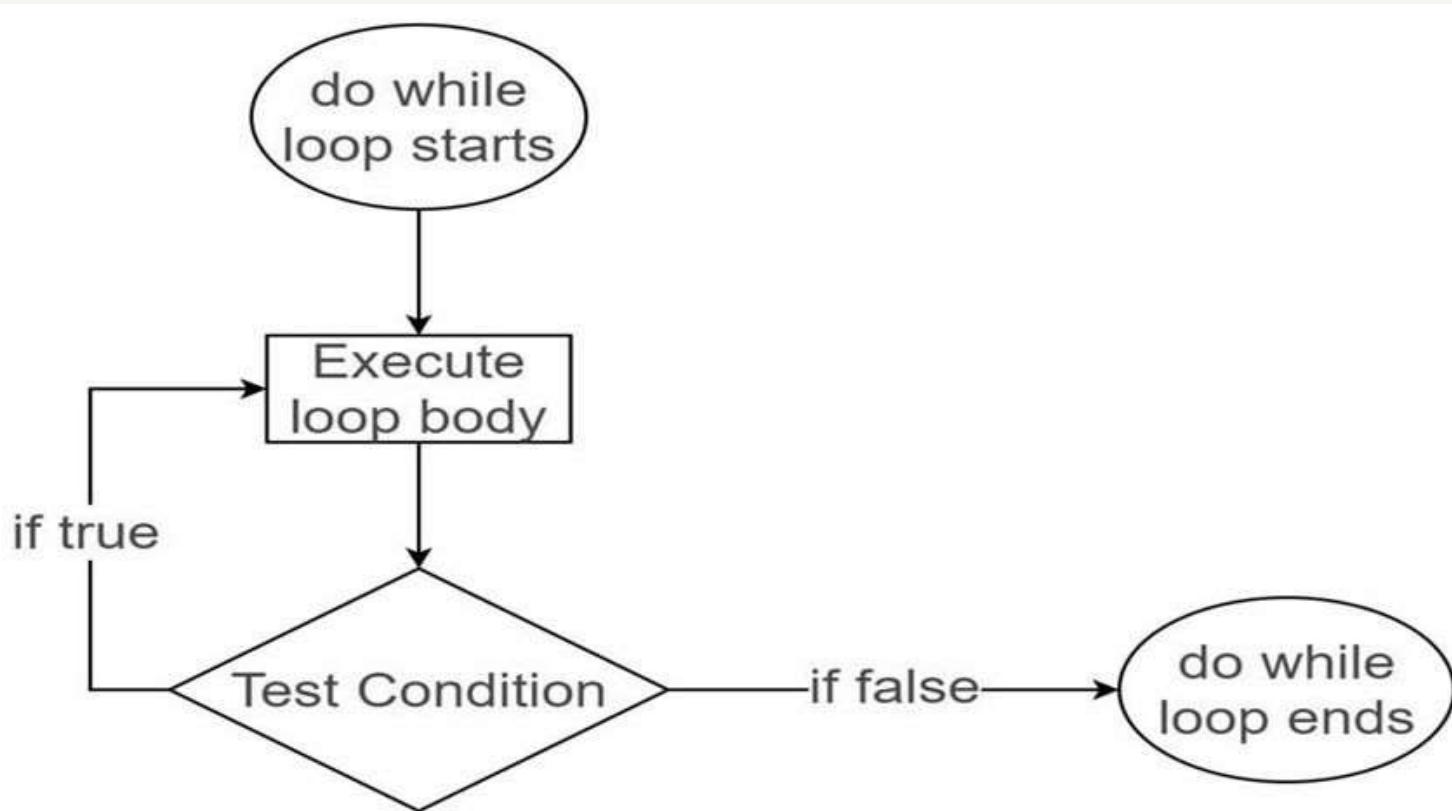
1. **Do-while Loops-** Do-while loop is similar to while loop , except the fact that it execute once even if condition is false.

Do-while Loops

- **Syntax :**

```
do
{
    body of loop
} while (condition);
```

Do-while Loops



Do-while Loops

- All do while loops are executed in following sequence :

Step 1 : Executes the body of loop and go to Step 2

.

Step 2 : It checks the test

condition if true then

go to Step 1 otherwise go

to Step 3.

Step 3: Other statements of the program .

Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/DoWhile.c>

Loops

Jumping Statements:

1. Goto Statement.
2. Break Statement.
3. Countinue Statement

Loops

The GOTO statement:

- By using this goto statements we can transfer the control from current location to anywhere in the program.
-
- To do all this we have to specify a label with goto and the control will transfer to the location where the label is specified.

Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/got.c>

goto label;

label:

Loops

The Break Statement:

- The break statement is used inside loop or switch statement.
- When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.

Syntax: break ;

Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Break.c>

Loops

Continue statement:

- The continue statement is also used inside loop.
- When compiler finds the continue statement inside a loop, compiler will skip all the following statements in the loop and resume the next loop iteration.

Syntax: continue ;

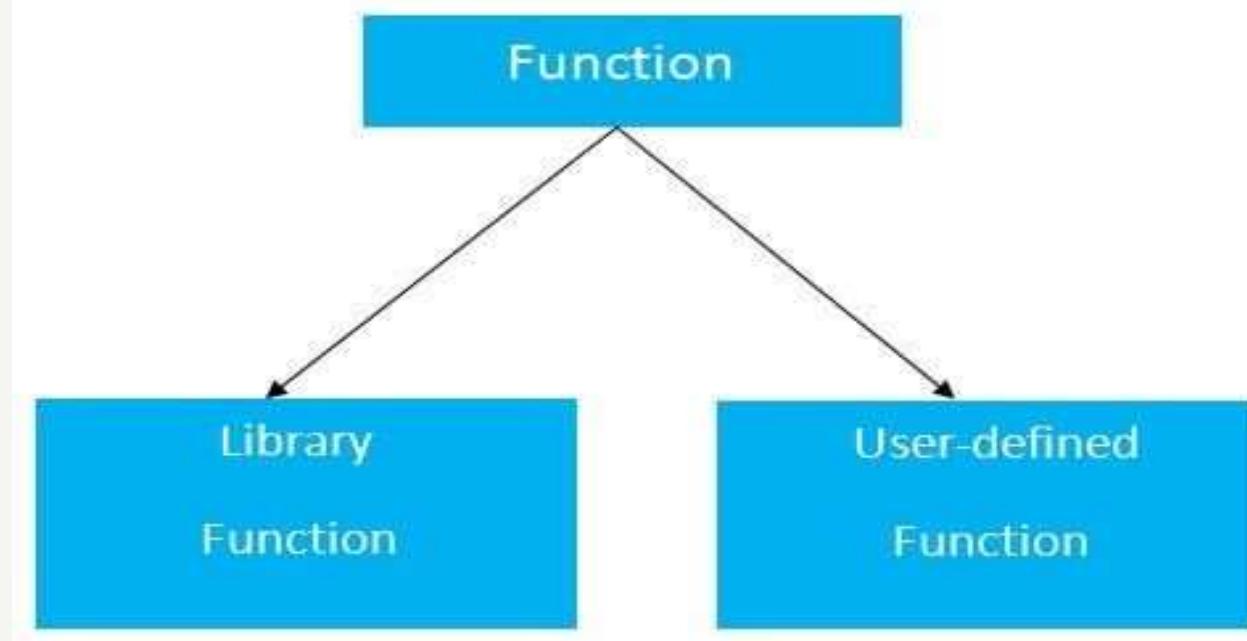
Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/continue.c>

Functions and Arrays

6. Functions

- A function is a block of organized code that is used to perform a single task. They provide better modularity for your application and reuse-ability. Depending on the programming language, a function may be called a subroutine, a procedure, a routine, a method, or a subprogram.



Types of Parameters

**Parameter in
Functions**

Call by
Value

Call by
Reference

Call by Value

- A method of passing parameters, where it copies the actual value into formal parameter
- Changes made to the parameter inside the function have no effect on the actual parameters

Call by Reference

- A method of passing arguments which copies the address of an argument into formal parameter.
- Changes made to the parameter affect the passed argument.

What are Functions?

- A function is a set of statements that take inputs, do some specific computation and produces output

**Ex. - main(), sum(), swap()
etc.**

```
int num1(  
);
```

Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Function1.c>

Parts of Functions

- **FUNCTIONS-**
 - Function Definition
 - Function Call
 - Function Declaration

```
int num1(  
);
```

Functions Definition

- Syntax of Function Definition-

```
return_typefunction_name( parameter list )  
{  
    body of the function  
}
```

Components in Definition

Return_type : It is data type of value which function will return.If function does not return any value data type will be void().

```
return_type  function_name( parameter list )  
{  
    body of the function  
}
```

Components in Definition

Function_name : It is the name given to the function by the programmer.

```
return_type  function_name( parameter list )
{
    body of the function
}
```

Components in Definition

Parameter list : This list refers to type, order and number of parameters of the function. A function can have no parameters also.

```
return_type  function_name( parameter list
)
{
    body of the function
}
```

Components in Definition

Body of function : It is collection of statements that define the working of the function.

```
return_type function_name( parameter list )  
{  
    body of the function  
}
```

Function Declaration

- It tells the compiler about the function name and how to call the function.
- Syntax :
`return_type function_name (parameter_list) ;`

Function Declaration

- Only type is required in function declaration , we can skip the parameter name
- Example:

```
int add ( int , int );
```

Function Call

- To use a function, we have to call that function to perform the given task.

Function Call

- When a program calls a function, the compiler gets redirected towards the function definition
- Function Call simply pass the required parameters along with the function name

What are Function Parameters

- Parameters are the variables that are taken as input to perform the function

```
int max(int num1, int  
        num2);
```

Scope of a Variable

What is Scope?

- Scope is the part of the program where a defined variable can have its existence and beyond that it cannot exist

Three Scopes

- **Three places where variables can be declared-**
 - Local Variables : Declared inside a block
 - Global Variables : Declared outside all functions
 - Formal Parameter : Declared in function definition

Local Variables

```
#include<stdio.h>
> void
{ mainin(t) x , y ,           //      local
  z                               variable
;
x=10 ;
y =20 ;      %d      %d ” , x, y,
}      z =x + y;      z );
printf(“ %d
```

Global Variables

```
#include<stdio.h

>      int z;                  // global variable

void main ()
{
    int x , y ;              // local variable
    x=10 ;
    y =20 ;
    z =x + y ;
    printf(“ %d  %d  %d ” , x, y, z );
}
```

Formal Parameters

```
void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

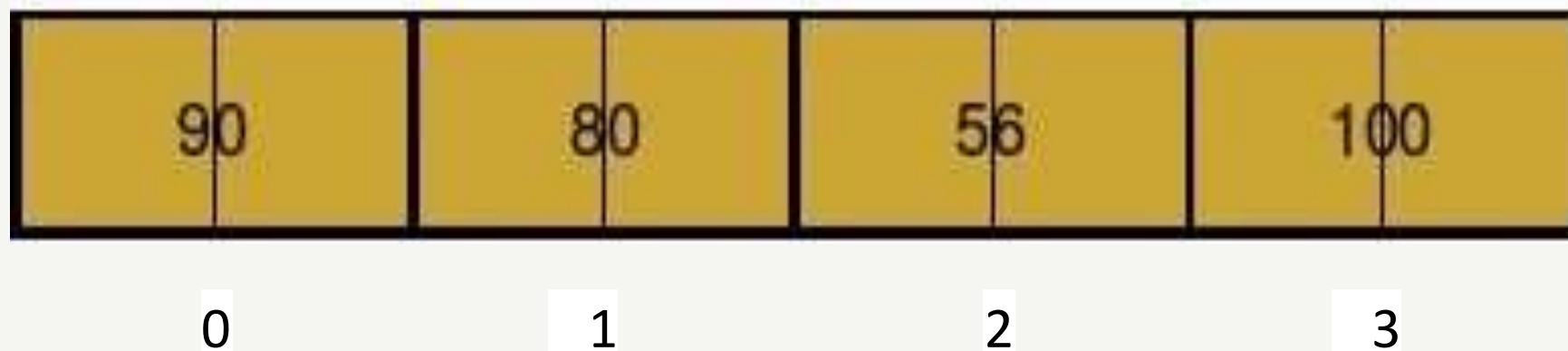
Refer This Example

[https://github.com/TopsCode/Software-
Engineering/blob/master/C/functions/pass_by_reference.c](https://github.com/TopsCode/Software-Engineering/blob/master/C/functions/pass_by_reference.c)

[https://github.com/TopsCode/Software-
Engineering/blob/master/C/functions/pass_by_value.c](https://github.com/TopsCode/Software-Engineering/blob/master/C/functions/pass_by_value.c)

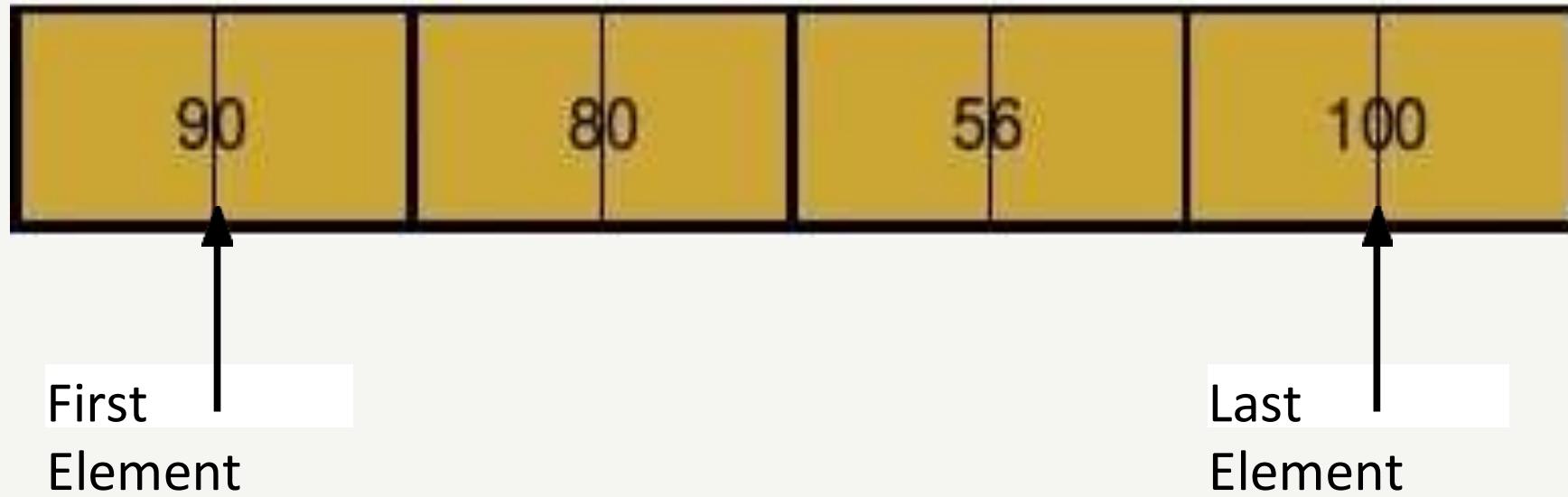
What are Arrays?

- An array is used to store a collection of data, and it is often used as a collection variables of the same data type



What are Arrays?

- All arrays consist of contiguous memory locations.



Declaring Arrays

- In declaration we specify the type of element and size of the array element
- **Syntax :** data_type array_name [
 size] :

Ex. - **int** **roll[20];**

Initializing Arrays

- We can initialize an array in C either one by one or using single statement

Ex. - **double balance [] = { 1000.0, 2.0, 3.4, 7.0, 50.0};**

OR

balance[4] = 50.0 ;

Accessing Arrays

Accessing Array Elements

- An element is accessed by placing the index of the element within the square brackets after the name of the array

Ex. - **double income = balance[9];**

Accessing Struct Members

- The individual members of structure can be accessed using the member access operator (.)
- It connects the structure variable and the structure member.
- **Syntax:**
`structure_variable.structure_member`

<https://github.com/TopsCode/SoftwareEngineering/blob/master/c%2B%2B/Array/1dArray.cpp>

Types of Arrays

Types of Arrays

- Single/ One Dimensional Array
- Multi Dimensional Array

Single Dimensional

```
int roll[20]
```

Example :

```
;
```

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Array/example1.c>

Multi Dimensional

```
int roll  
[20][12]...;
```

Multi Dimensional

- C programming language provides us multi-dimensional array.
- Out of which we will discuss Two Dimensional Array

Multidimensional Arrays

What are they?

- Multidimensional arrays are arrays of arrays
- **General Form-**
data_type
array_name[size1][size2]....[sizeN];

Example

```
int x[2][3][4] =  
{  
    { {0,1,2,3}, {4,5,6,7}, {8,9,10,11} },  
    { {12,13,14,15}, {16,17,18,19}, {20,21,22,23} }  
};
```

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Array/2DArray.cpp>

Pointers and strings

Pointers in C :

Pointers in C are variables that store the memory address of another variable. They are declared using the `*` symbol, and you can access the value stored at the memory address using the dereference operator `*`. Pointers allow direct memory manipulation, making them powerful for dynamic memory allocation and working with arrays, functions, and structures. The address of a variable is obtained using the address-of operator `&`.

Pointer Declaration and Initialization

In C, pointers are declared using the `*` symbol, and they must be initialized to a valid memory address before use.

Pointer Declaration:

```
int *ptr; // Declares a pointer to an integer
```

Pointer Declaration and Initialization

Pointer Initialization:

You can initialize a pointer by assigning it the address of a variable using the address-of operator &:

```
int x = 10;  
int *ptr = &x; // Pointer ptr holds the address of variable x
```

Now, **ptr** points to the memory location where **x** is stored. You can access the value of **x** through ***ptr**.

Strings

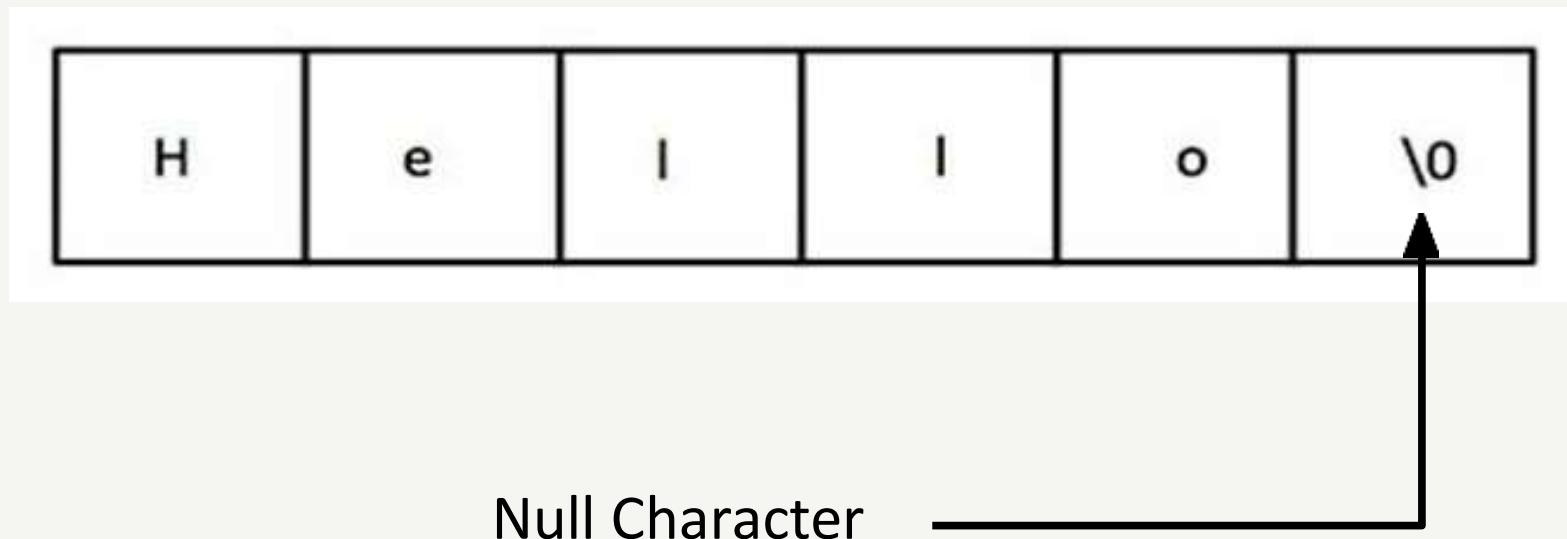
What are Strings?

- A string is an array of characters stored in a consecutive memory locations
- The ending character is always the null character '\0'. It acts as string terminator.
- **Syntax :**

```
char string_name [ length ] ;
```

Strings

- The compiler automatically places the '\0' at the end of the string when we initializes the array



String Functions

Function	What It Does
strcpy()	Copies one string to another.
strlen()	Returns the length of a string, not counting the NULL character

String Functions

Function	What It Does
strcmp()	Compares two strings. If the strings match, the function returns 0.
strcat()	Appends one string to another, creating a single string out of two.

String Functions

Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/strlen.c>

<https://github.com/TopsCode/Software-Engineering/blob/master/C/stringFunctin.c>

Structures and File Handling

Structures in C

Introduction to Structures

In C, a structure is a user-defined data type that allows grouping different types of data under a single name. Each data element in a structure is called a member, and they can be of different types, such as integers, floats, or arrays. Structures are used to represent complex data entities, such as records in databases. They are defined using the `struct` keyword and can be accessed using the dot operator (`.`).

Structure Declaration and Initialization

In **C language**, a **struct** is a user-defined data type that groups different data types under a single name. It is declared using the **struct** keyword, followed by the structure name and member variables. Structure variables can be initialized separately (`s1.age = 20;`), directly (`struct Student s2 = {"Alice", 21, 90.0};`), using **typedef**, or as an array (`struct Student students[2] = {{"Mike", 19, 78.5}, {"Sara", 20, 92.0}};`). Structures help in organizing complex data efficiently.

Array of Structure

An **array of structures** in C is used to store multiple structure variables of the same type. It allows managing structured data in a systematic way.

Example: Array of Structures

```
#include <stdio.h>
struct Student {
    char name[50];
    int age;
    float marks;
};
```

Array of Structure

```
int main() {
    struct Student students[2] = {
        {"Mike", 19, 78.5},
        {"Sara", 20, 92.0}
    };

    for (int i = 0; i < 2; i++) {
        printf("Name: %s\nAge: %d\nMarks: %.2f\n\n", students[i].name,
               students[i].age, students[i].marks);
    }
    return 0;
}
```

Array of Structure

Explanation:

- `students[2]` is an array storing two `Student` structures.
- Each structure has `name`, `age`, and `marks`.
- The `for` loop prints each student's details.

This is useful for handling multiple records efficiently! 🚀

Nested Structures

Nested Structures in C

A **nested structure** is a structure inside another structure, allowing hierarchical data organization.

Example: Nested Structure

```
#include <stdio.h>

struct Address {
    char city[50];
    int zip;};

```

Nested Structures

```
struct Student {  
    char name[50];  
    int age;  
    struct Address addr; // Nested Structure  
};  
int main() {  
    struct Student s1 = {"John Doe", 20, {"New York", 10001}};
```

Nested Structures

```
printf("Name: %s\nAge: %d\nCity: %s\nZIP: %d\n",
      s1.name, s1.age, s1.addr.city, s1.addr.zip);

return 0;
}
```

Nested Structures

Explanation:

- `struct Address` stores city and ZIP code.
- `struct Student` contains an `Address` structure as a member.
- We initialize `Student` with name, age, city, and ZIP.
- This method helps in structuring complex data logically! 🚀

File Handling in C

File Operations (Opening, Closing, Reading, and Writing)

1. Opening a File (**fopen()**)

A file is opened using **fopen(filename, mode)**, where mode can be:

- "r" (read), "w" (write), "a" (append)
- "r+" (read/write), "w+" (write/read), "a+" (append/read)

```
FILE *file = fopen("data.txt", "w"); // Opens file in write mode
if (file == NULL) {
    printf("Error opening file!\n");
    return 1;
}
```

File Operations (Opening, Closing, Reading, and Writing)

2. Writing to a File (**fprintf()** / **fputc()** / **fputs()**)

```
fprintf(file, "Hello, File Handling!"); // Write text to file  
fclose(file); // Close file after use
```

3. Reading from a File (**fscanf()** / **fgetc()** / **fgets()**)

```
FILE *file = fopen("data.txt", "r");  
char str[100];  
fgets(str, 100, file); // Read a line from file  
printf("Read: %s", str);  
fclose(file);
```

File Operations (Opening, Closing, Reading, and Writing)

File Operations in C

File handling in **C** allows reading and writing data to files using standard library functions.

4. Closing a File (**fclose()**)

Always close the file using **fclose(file)** ; to free resources.

These functions enable efficient file operations in C! 🚀

File Pointers

A **file pointer** in C is a pointer of type `FILE*` used to handle file operations. It is created using `FILE *fp;` and initialized with `fopen(filename, mode)`. Functions like `fgetc(fp)`, `fputc(ch, fp)`, `fscanf(fp, format, args)`, and `fprintf(fp, format, args)` use file pointers to read/write data. Always close the file using `fclose(fp);` to prevent memory leaks.

File Handling Functions

File handling in C is done using standard functions:

- `fopen(filename, mode)`: Opens a file in specified mode ("r", "w", "a", etc.).
- `fclose(fp)`: Closes an opened file.
- `fprintf(fp, format, args)`, `fputs(str, fp)`, `fputc(ch, fp)`: Write data to a file.
- `fscanf(fp, format, args)`, `fgets(str, size, fp)`, `fgetc(fp)`: Read data from a file.

These functions enable efficient file operations in C. 

Module 3

[OOP Concepts]

Introduction to C++

Introduction to C++

C++ is a powerful, high-performance programming language that supports **procedural, object-oriented, and generic programming**. It includes fundamental concepts like **variables, data types, operators, loops, conditionals, functions, and classes**. The **#include <iostream>** header is used for input/output, with **cout** for output and **cin** for input. C++ is widely used for **system software, game development, and competitive programming**.

Introduction to C++

Introduction to C++ Language

C++ is a high-level, general-purpose programming language developed by **Bjarne Stroustrup** in **1979** as an extension of C. It supports **procedural, object-oriented, and generic programming**, making it powerful and flexible. Key features include **encapsulation, inheritance, polymorphism, and abstraction**. C++ is widely used in **software development, game engines, operating systems, and competitive programming** due to its speed and efficiency.

Introduction to C++

Introduction to C++ Language

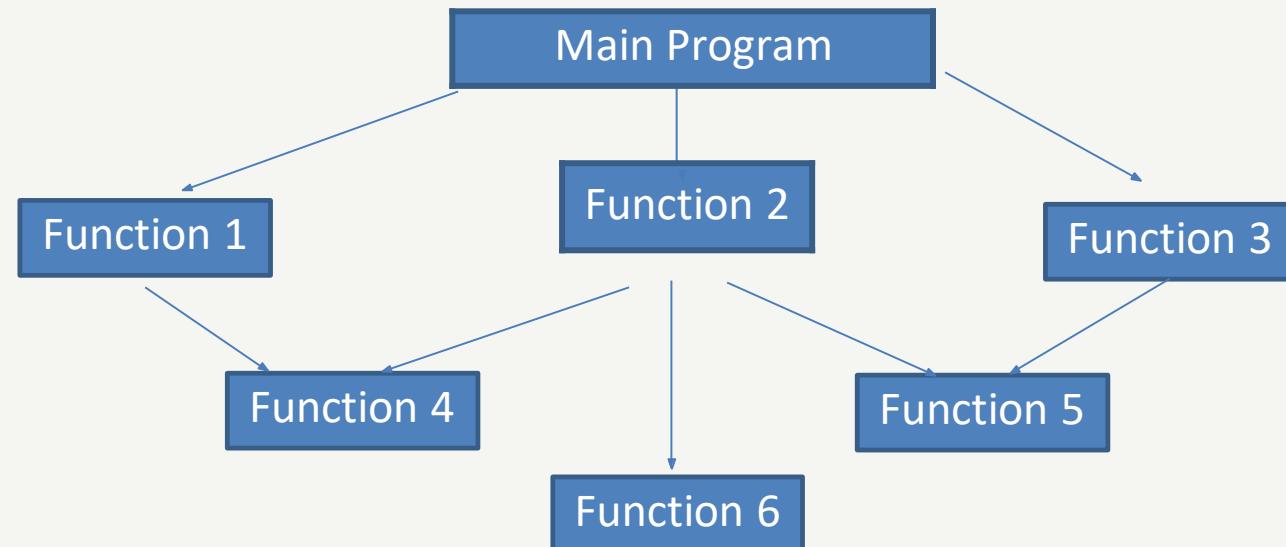
C++ is a high-level, general-purpose programming language developed by **Bjarne Stroustrup** in **1979** as an extension of C. It supports **procedural, object-oriented, and generic programming**, making it powerful and flexible. Key features include **encapsulation, inheritance, polymorphism, and abstraction**. C++ is widely used in **software development, game engines, operating systems, and competitive programming** due to its speed and efficiency.

Procedure Oriented Programming

In Procedure Oriented programming , the problem is viewed as sequence of things to be done such as reading, calculating and printing.

The number of functions are return to accomplish such task, i.e. the focus is on functions.

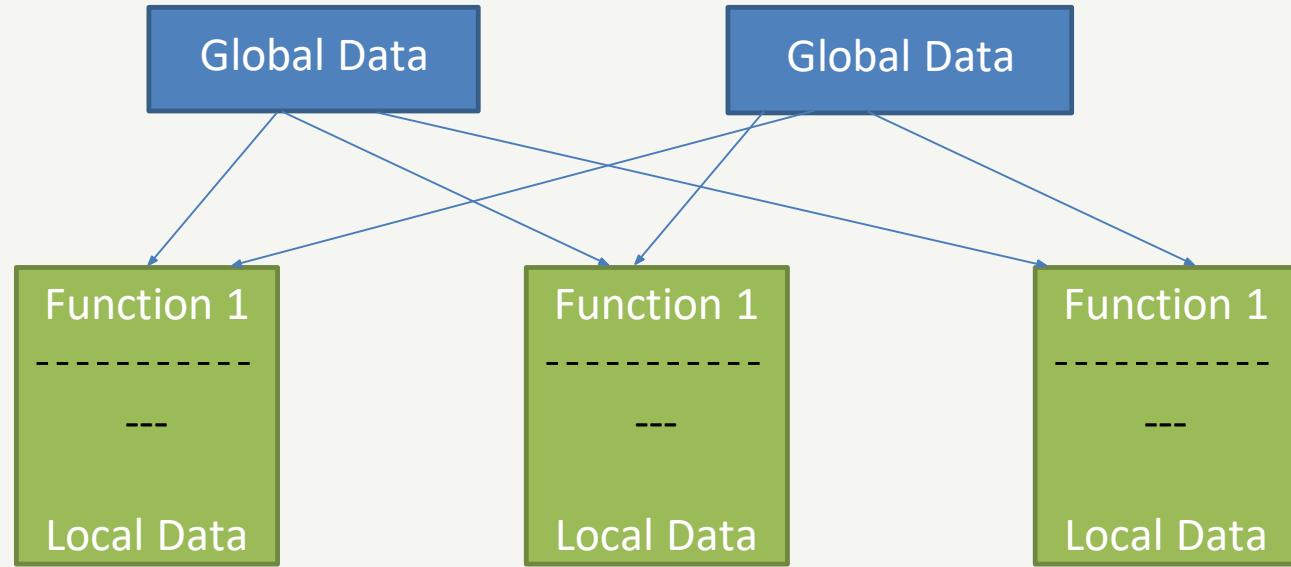
The typical program structure for procedure programming is shown below :



Procedure Oriented Programming

Characteristics of Procedure Oriented Programming.

- ❖ Emphasis is on doing things
- ❖ Large programs are broken into small known as functions
- ❖ Most of the function share global data.
- ❖ Data move openly around the system from function to function.
- ❖ Follows Top Down approach in program design



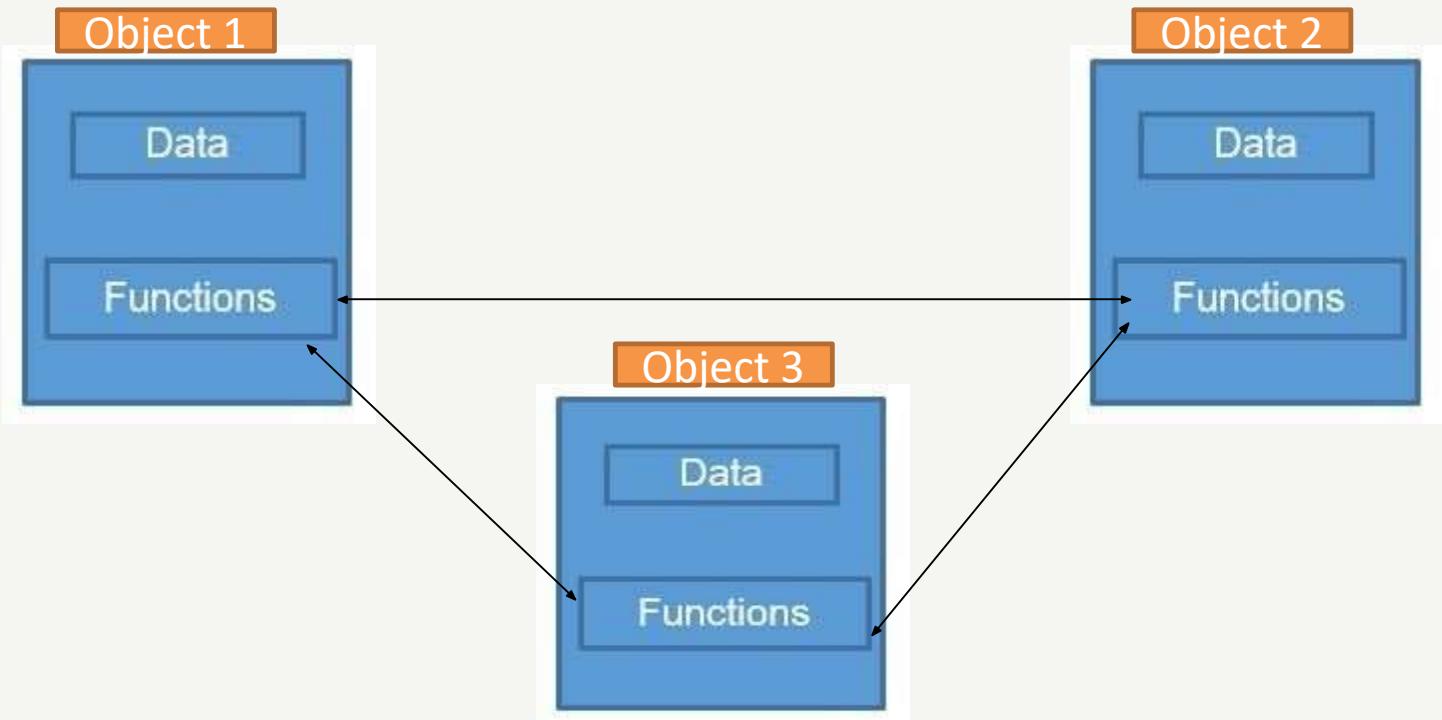
Object Oriented Programming

In order to remove some of the flaws of POP, OOP came into existence.

OOP treats data as critical element in program development and does not allow it to flow freely around the system.

It ties the data more closely to the function that operates on it.

Object Oriented Programming allows decomposition of program into a number of entities called objects and then builds data and function around these objects.



Object Oriented Programming

Characteristics of Object Oriented Programming:

- ❖ Emphasis on data rather than procedure.
- ❖ Program are divided into objects.
- ❖ Data is hidden and cannot be accessed by external functions.
- ❖ Objects may communicate with each other through functions.
- ❖ Follows bottom up approach in program design.

Setting Up C++ Development Environment

Setting Up C++ Development Environment

To start programming in C++, follow these steps:

1. Install a C++ Compiler

- Use **MinGW** (for Windows) or **GCC** (for Linux/macOS).
- Download **MinGW-w64** from mingw-w64.org or install **g++** via:
 - **Windows**: Install **MinGW** or use **MSYS2**.
 - **Linux/macOS**: Run `sudo apt install g++` (Linux) or `brew install gcc` (macOS).

Setting Up C++ Development Environment

Install an IDE or Code Editor

- **Code::Blocks, Dev-C++, Visual Studio, or Eclipse** (for full IDEs).
- **VS Code or Sublime Text** (for lightweight coding).

1. Write and Compile Code

Create a **.cpp** file, write C++ code, and compile using:

```
g++ filename.cpp -o output  
./output
```

Setting Up C++ Development Environment

1. Run the Program

- After compiling, execute the output file to see the result. 

Writing and Running Your First C++ Program

Writing and Running Your First C++ Program

1. Create a C++ File

Open your text editor or IDE and create a new file called `main.cpp`.

Writing and Running Your First C++ Program

Write the C++ Code

Add the following simple C++ code to print "Hello, World!" to the console:

```
#include <iostream> // Include the input-output library

int main() { // Main function where execution begins
    std::cout << "Hello, World!" << std::endl; // Output to console
    return 0; // Exit status
}
```

Writing and Running Your First C++ Program

Compile and Run

Open your terminal/command prompt, navigate to the folder containing your `main.cpp`, and run the following commands:

On Windows/Linux/macOS (with **GCC or **MinGW**):**

```
g++ main.cpp -o main // Compile the code  
.main // Run the executable
```

The output will be:

Hello, World!

Done!

You've successfully written and executed your first C++ program!

Variables, Data Types, and Operators

Variables, Data Types, and Operators

In C++, **variables** are used to store data values and must be declared with a specific data type (e.g., `int`, `float`, `char`). For example, `int x = 10;`. **Constants** are fixed values that cannot be changed during program execution and are declared using the `const` keyword, like `const int MAX = 100;`. Variables and constants help manage and control the flow of data in a program.

Data Types and Size Specifiers

In C++, **data types** define the type of data a variable can hold, such as `int`, `float`, `char`, and `double`. **Size specifiers** (like `short`, `long`, `long long`) modify the size of data types to allocate more or less memory. For example, `int` typically stores a 4-byte integer, while `long` stores a larger integer. The `sizeof()` function can be used to determine the size of a data type or variable.

Assignments, Arithmetic, Relational, Logical, and Bitwise Operators

In C++, **assignment operators** like `=` assign values to variables. **Arithmetic operators** (e.g., `+`, `-`, `*`, `/`, `%`) perform mathematical operations. **Relational operators** (e.g., `==`, `!=`, `>`, `<`, `>=`, `<=`) compare values and return a boolean result. **Logical operators** (e.g., `&&`, `||`, `!`) evaluate logical expressions, while **bitwise operators** (e.g., `&`, `|`, `^`, `~`, `<<`, `>>`) perform operations on individual bits of integer types. These operators allow for efficient control and manipulation of data.

Type Conversion in C++

Type Conversion in C++

Type conversion refers to converting one data type to another. There are two types of type conversion in C++:

Implicit Type Conversion (Automatic)

The compiler automatically converts a lower data type to a higher one when needed (e.g., `int` to `float`).

```
int x = 5;
```

```
float y = x; // Implicit conversion from int to float
```

Type Conversion in C++

Explicit Type Conversion (Casting)

The programmer manually converts one type to another using **casting**.

C++ casting:

```
int y = static_cast<int>(x); // Using C++ static_cast
```

Type conversion ensures compatibility between different data types during operations.

Constants and Literals

Constants and Literals in C++

1. Constants

Constants are fixed values that cannot be modified during program execution. They are declared using the `const` keyword or with `#define` preprocessor directive.

Example using `const`:

```
const int MAX = 100; // Constant integer
```

Example using `#define`:

```
#define PI 3.14 // Define a constant
```

Constants and Literals

Literals

Literals are constant values directly used in the code. They represent fixed values of specific types, such as:

- **Integer literal:** 100
- **Floating-point literal:** 3.14
- **Character literal:** 'A'
- **String literal:** "Hello"
- **Boolean literal:** true or false

Constants and literals help ensure that specific values remain unchanged and are directly usable in expressions.

Control Flow Statement

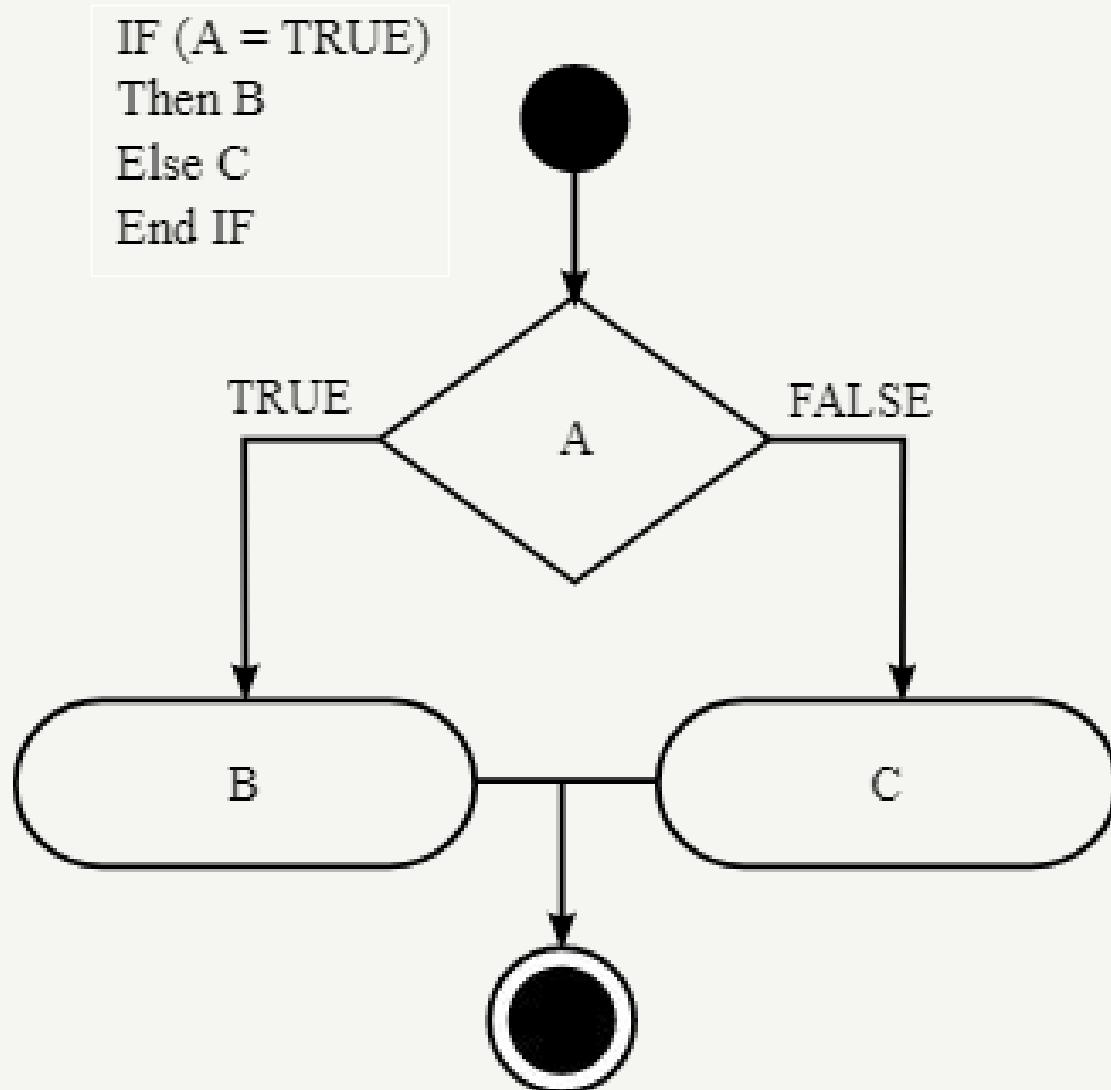
What is Control Structure?

- Real life situations where we have to condition based decisions by asking 'if' questions.

Ex. - if age is above 18, I am allowed drive vehicles or else not allowed.



Flow of Control Structure



Control structure Statements

- Following types of Statements:
 - if
 - if else
 - Nested if
 - Switch
 - Nested switch

If Statements

- if statement is the basic decision making statement
- Used to decide whether a certain statement or block of statements will be executed or not

If Statements

- Syntax :

```
if( condition )  
{  
    statement_1 ; // true block  
    statements  
}  
statement x ;
```

If else Statements

- if else statement allows selecting any one of the two available options depending upon the output of the test condition

If else Statements

- Syntax :

```
if (condition )  
{  
    statements; // true  
}  
else  
{  
    statements ; // false statement  
}
```

Nested If Statements

- Nested if statement is simply an if statement embedded with another if statement

Nested If Statements

- Syntax

:

```
if (condition1)
{
    statements ;      // executes when condition1 is
    if ( condition2) true
    {
        statements ;      // executes when condition2 is
        }
    }
}
```

Switch Statements

- Switch case statements are a substitute for long if statements that compare a variable to several integer values

Nested Switch Statements

- Nested Switch Statements occurs when a switch statement is defined inside another switch statement.

Nested Switch Statements

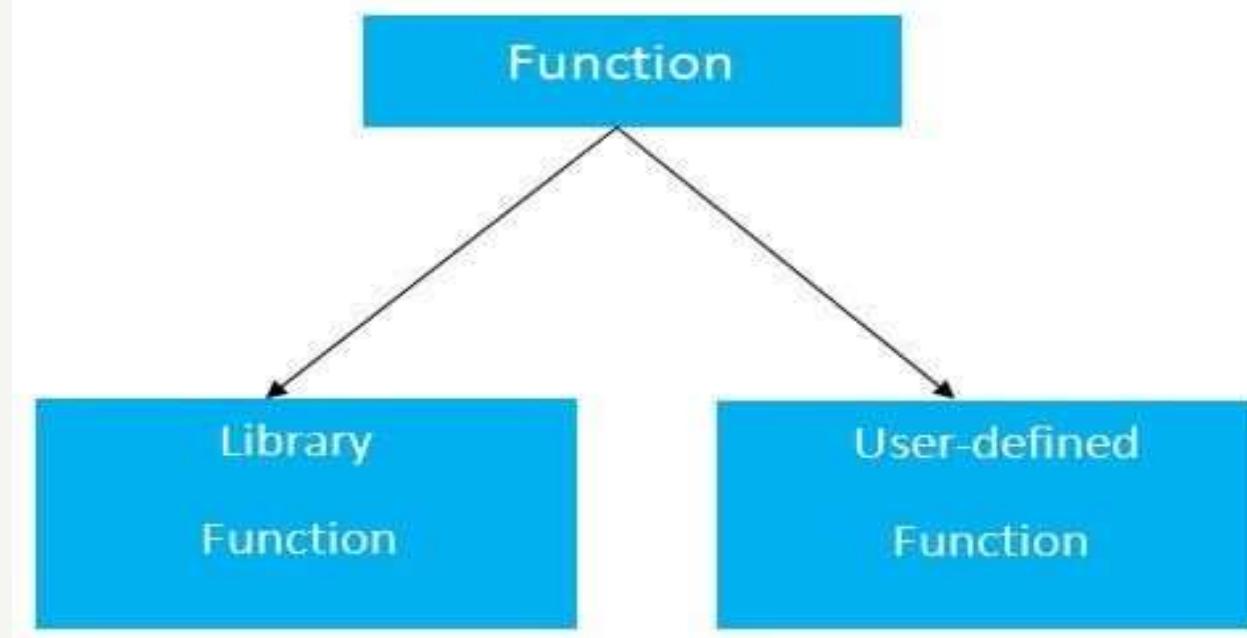
- Syntax :

```
switch(ch1)
{
    case 'A':
        switch (ch2)
        {
            case 'A ':
                break;
            case' B' :
                }
            break ;
        case' B' :
            }
```

Functions and Arrays

6. Functions

- A function is a block of organized code that is used to perform a single task. They provide better modularity for your application and reuse-ability. Depending on the programming language, a function may be called a subroutine, a procedure, a routine, a method, or a subprogram.



Types of Parameters

**Parameter in
Functions**

Call by
Value

Call by
Reference

Call by Value

- A method of passing parameters, where it copies the actual value into formal parameter
- Changes made to the parameter inside the function have no effect on the actual parameters

Call by Reference

- A method of passing arguments which copies the address of an argument into formal parameter.
- Changes made to the parameter affect the passed argument.

What are Functions?

- A function is a set of statements that take inputs, do some specific computation and produces output

**Ex. - main(), sum(), swap()
etc.**

```
int num1(  
);
```

Parts of Functions

- **FUNCTIONS-**
 - Function Definition
 - Function Call
 - Function Declaration

```
int num1(  
);
```

Functions Definition

- Syntax of Function Definition-

```
return_typefunction_name( parameter list )  
{  
    body of the function  
}
```

Components in Definition

Return_type : It is data type of value which function will return.If function does not return any value data type will be void().

```
return_type  function_name( parameter list )  
{  
    body of the function  
}
```

Components in Definition

Function_name : It is the name given to the function by the programmer.

```
return_type  function_name( parameter list )
{
    body of the function
}
```

Components in Definition

Parameter list : This list refers to type, order and number of parameters of the function. A function can have no parameters also.

```
return_type  function_name( parameter list
)
{
    body of the function
}
```

Components in Definition

Body of function : It is collection of statements that define the working of the function.

```
return_type function_name( parameter list )  
{  
    body of the function  
}
```

Function Declaration

- It tells the compiler about the function name and how to call the function.
- Syntax :
`return_type function_name (parameter_list) ;`

Function Declaration

- Only type is required in function declaration , we can skip the parameter name
- Example:

```
int add ( int , int );
```

Function Call

- To use a function, we have to call that function to perform the given task.

Function Call

- When a program calls a function, the compiler gets redirected towards the function definition
- Function Call simply pass the required parameters along with the function name

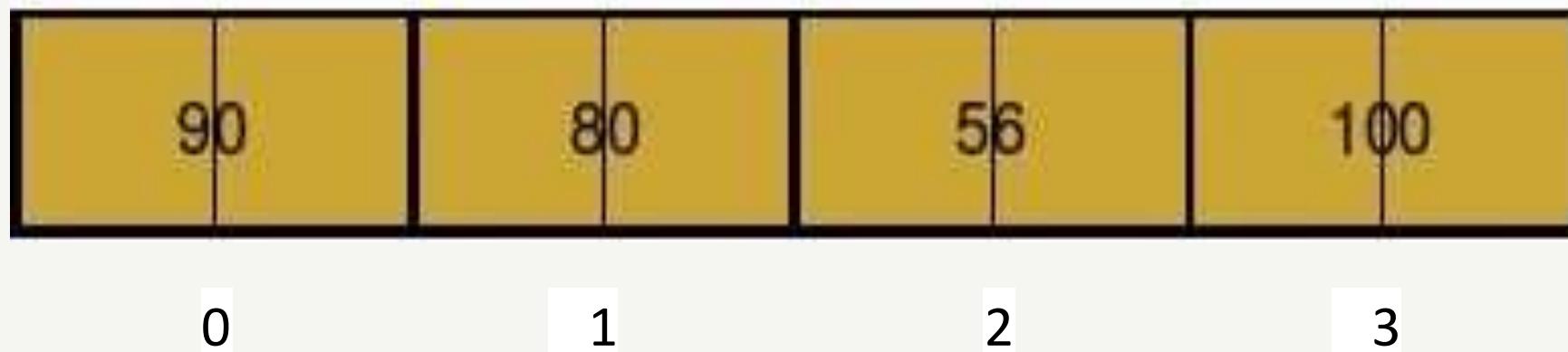
What are Function Parameters

- Parameters are the variables that are taken as input to perform the function

```
int max(int num1, int  
        num2);
```

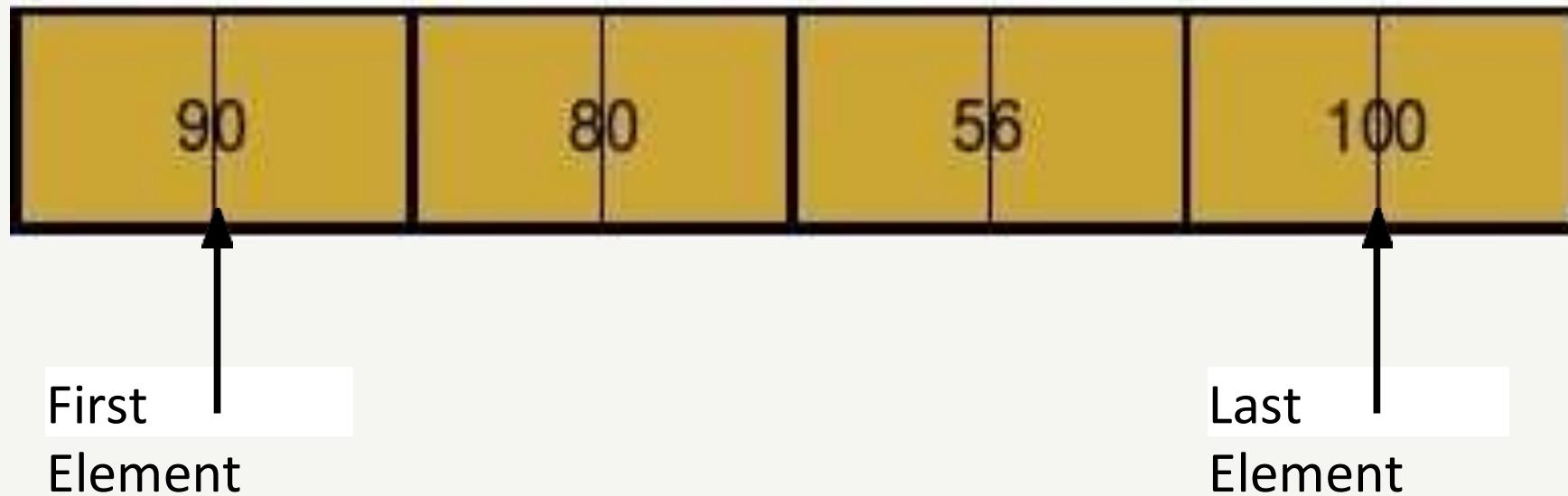
What are Arrays?

- An array is used to store a collection of data, and it is often used as a collection variables of the same data type



What are Arrays?

- All arrays consist of contiguous memory locations.



Declaring Arrays

- In declaration we specify the type of element and size of the array element
- **Syntax :** data_type array_name [
 size] :

Ex. - **int** **roll[20];**

Initializing Arrays

- We can initialize an array in C either one by one or using single statement

Ex. - double balance [] = { 1000.0, 2.0, 3.4, 7.0,
50.0};

OR

balance[4] = 50.0 ;

Accessing Arrays

Accessing Array Elements

- An element is accessed by placing the index of the element within the square brackets after the name of the array

Ex. - `double income = balance[9];`

Types of Arrays

Types of Arrays

- Single/ One Dimensional Array
- Multi Dimensional Array

Single Dimensional

```
int roll[20]
```

```
;
```

Multi Dimensional

```
int roll  
[20][12]...;
```

Multi Dimensional

- C programming language provides us multi-dimensional array.
- Out of which we will discuss Two Dimensional Array

Multidimensional Arrays

What are they?

- Multidimensional arrays are arrays of arrays
- **General Form-**
data_type
array_name[size1][size2]....[sizeN];

Example

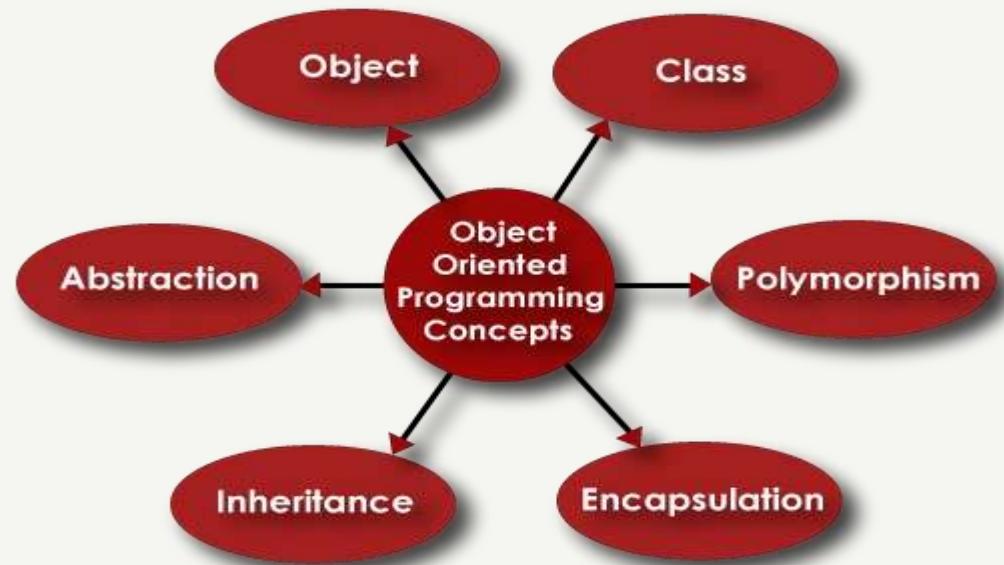
```
int x[2][3][4] =  
{  
    { {0,1,2,3}, {4,5,6,7}, {8,9,10,11} },  
    { {12,13,14,15}, {16,17,18,19}, {20,21,22,23} }  
};
```

Introduction to Object-Oriented Programming

Basic Concepts of OOP

Some of the basic concepts of object oriented programming are:

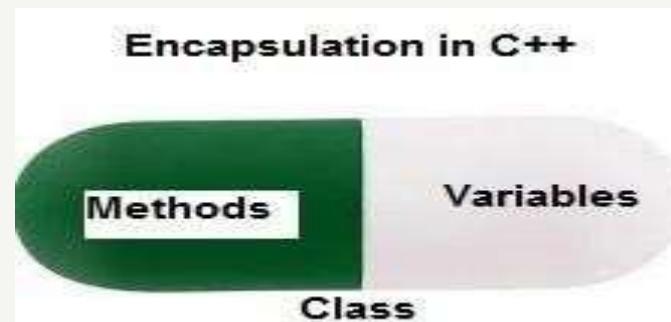
- ❖ Objects
- ❖ Classes
- ❖ Data abstraction and encapsulation
- ❖ Polymorphism
- ❖ Inheritance
- ❖ Dynamic Binding



Some of the basic concepts of object oriented programming are:

❖ **Encapsulation:**

In normal terms, Encapsulation is defined as wrapping up of data and information under a single unit. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.



❖ **Abstraction:**

Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

❖ Polymorphism:

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

C++ supports operator overloading and function overloading.

❖ Inheritance:

- The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.
- Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Classes and Object

❖ Classes

- These contain data and functions bundled together under a unit. In other words class is a collection of similar objects. When we define a class it just creates template or Skelton. So no memory is created when class is created. Memory is occupied only by object. for eg. :

Fruit is class of apple.

❖ Objects

- In other words object is an instance of a class.

NOTE: In other words classes acts as data types for objects.

❖ Member functions

- The functions defined inside the class as above are called member functions.

Example:

```
class classname
{
    variable
    declarations; Data
    Functions;
};

main ( )
{
    classname
    objectname1,objectname2,..
};
```

Types of accessifiers

Private	Public	Protected
Only for that class can access	Other class can also access.	Only immediate inheritance class can access.

	Direct-access scope	Class/object scope
Private	Declaring class	Declaring class
Protected	All derived classes	Declaring class
Friend	Derived in-project classes	Declaring project
Protected Friend	All derived classes	Declaring project
Public	All derived classes	All projects

Example of accessifiers:

```
Class classname
{
    private:
        datatype data;

    public:
        Member functions
};

main ()
{
    classname objectname1,objectname2,..;
}
```

- **How to Access Class Members**

It is possible to access the class members after a class is defined and objects are created.

General syntax to access class member:

```
Object_name.function_name (arguments);
```

```
class exforsys  
{  
    int a,  
    b;  
    public:  
        void sum(int,int);  
} e1;  
e1.sum(5,6);
```

Refer This Example:

Simple Class:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/class/class1.cpp>

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/class/class2.cpp>

Constructor

- It is a member function which initializes a class.
- A constructor has:
 - (i)the same name as the class itself
 - (ii)no return type
- A constructor is called **automatically** invoked whenever a new instance of a class is created.
- You must supply the arguments to the constructor when a new instance is created.
- If you do not specify a constructor, the compiler generates a default constructor for you (expects no parameters and has an empty body).

Types of Constructor

- Simple(Default) Constructor
- Parameterized Constructor
- Copy Constructor



Refer this Example:

Default Constructor:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Constructor/DefaultConstructor.cpp>

Parameterised Constructor:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Constructor/parameterisedConstructor.cpp>

Copy Constructor:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Constructor/copyConstructoar.cpp>

Dynamic Constructors

- The constructor can also be used to allocate memory while creating objects. This will enable the system to allocate the right amount for each object when the objects are not of the same size, thus resulting in the saving of memory.
- Allocation of memory to objects at the time of their construction is known as dynamic constructor of objects. The memory is allocated with the help of the new operator.

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Constructor/DynamicConstructor.cpp>

What is a Destructor?

- It is a member function which deletes an object.
- A destructor function is called automatically when the object goes out of scope:
 1. the function ends
 2. the program ends
 3. a block containing temporary variables ends
 4. a delete operator is called

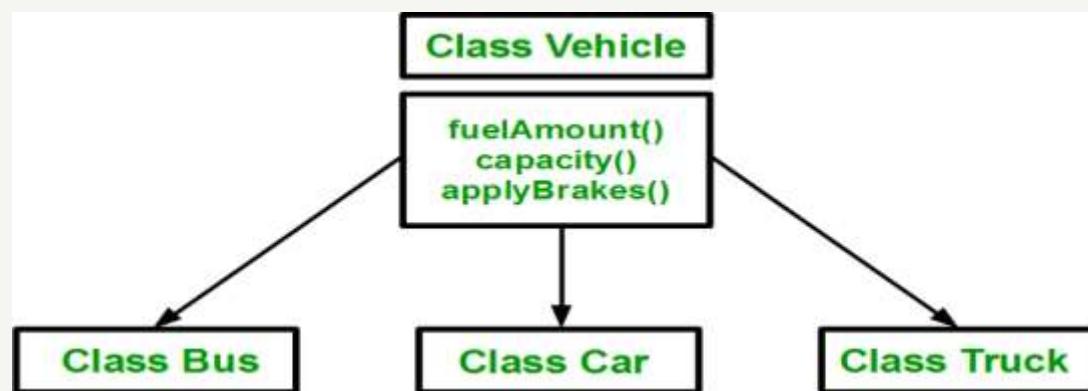
A destructor has:

1. the same name as the class but is preceded by a tilde (~)
2. no arguments and return no values

Inheritance

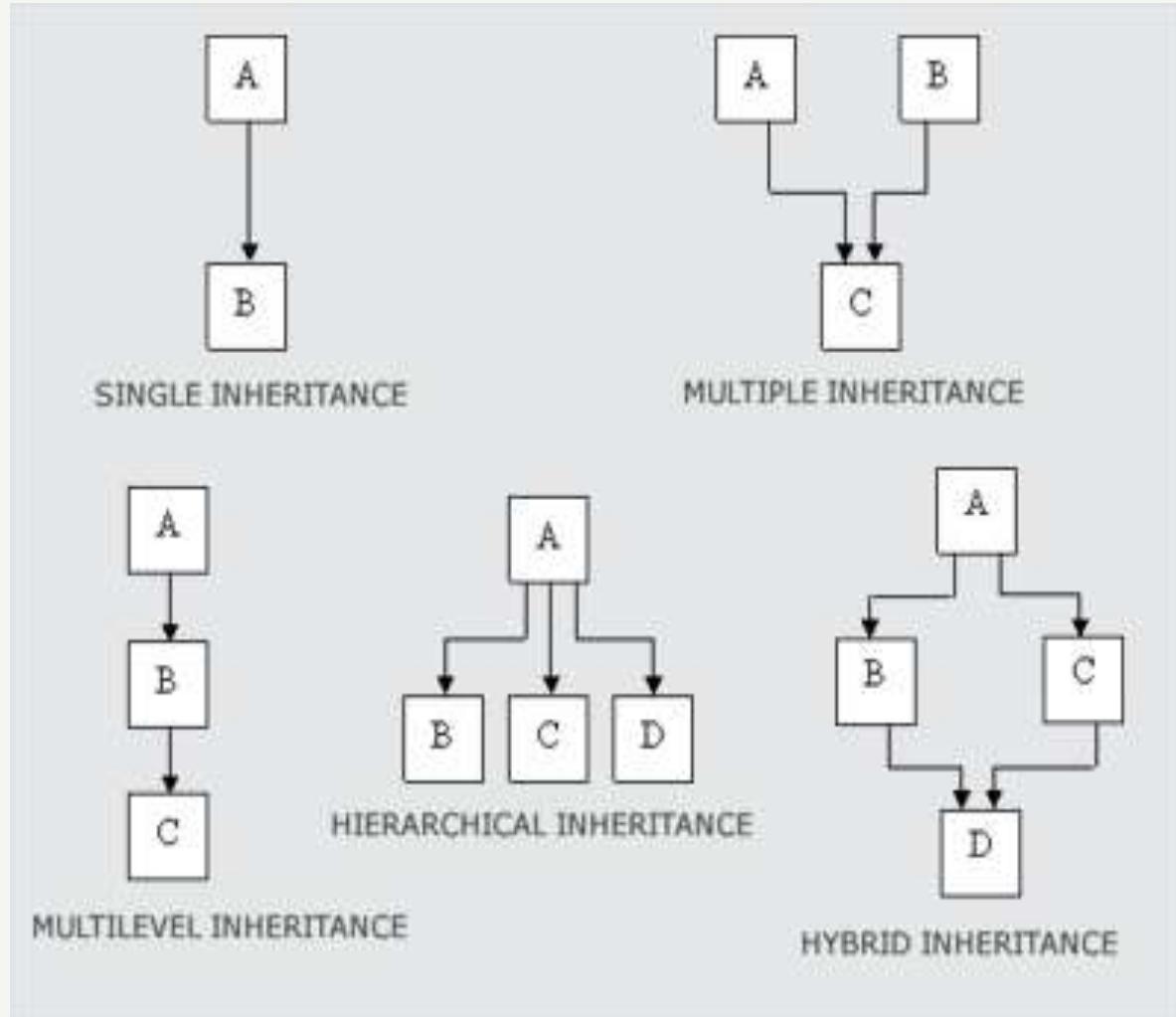
Inheritance

- Inheritance is a mechanism of reusing and extending existing classes without modifying them, thus producing hierarchical relationships between them.
- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.



Types of Inheritance

- Single Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Multilevel Inheritance
- Hybrid Inheritance



Refer This Examples:

Single Inheritance:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Inheritance/single.cpp>

Multiple Inheritance:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Inheritance/multiple.cpp>

Multilevel Inheritance:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Inheritance/multilevel.cpp>

Hierarchical Inheritance

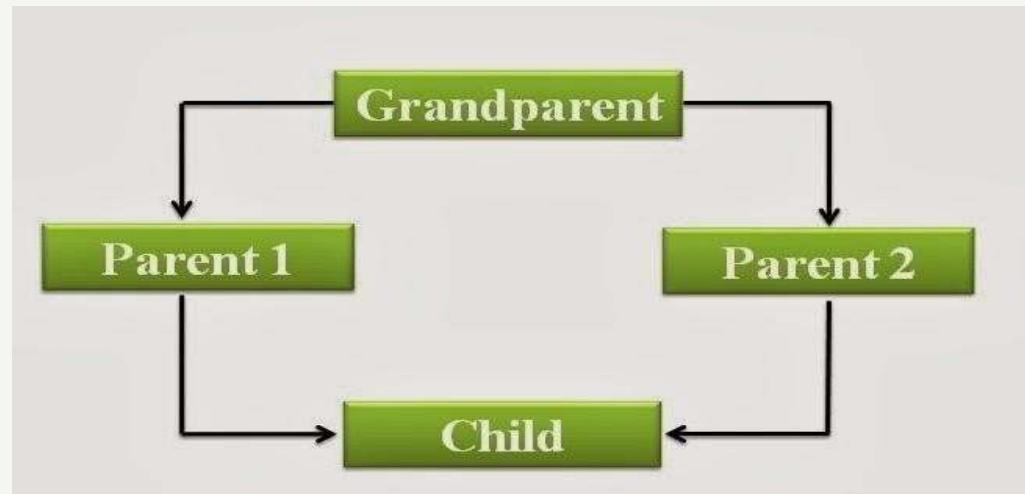
<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Inheritance/hierarchical.cpp>

Hybrid Inheritance

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Inheritance/hybrid.cpp>

Virtual Base Classes

- Virtual base class is used in situation where a derived have multiple copies of base class.
- When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class.



- Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/class/virtualClass.cpp>

Constructor in Derived Class

- Base class constructors are automatically called for you if they have no argument.
- If you want to call a superclass constructor with an argument, you must use the subclass's constructor initialization list.
- Unlike Java, **C++ supports multiple inheritance** (for better or worse), so the base class must be referred to by name, rather than "super()".

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Constructor/constructorInheritance.cpp>

Access Control and Inheritance

In C++, access control refers to the use of access specifiers (**public**, **protected**, **private**) to define the visibility and accessibility of class members. Inheritance allows a class to derive properties and behaviors from another class. Inherited members' access levels depend on the access specifier used during inheritance (**public**, **protected**, **private**), which determines whether the base class members remain accessible in the derived class.

Polymorphism

Polymorphism and Overloading

- Poly refers many.
- "single interface having multiple implementations."
- That is making a function or operator to act in different forms depending on the place they are present is called Polymorphism. Overloading is a kind of polymorphism.
- 2 Types of polymorphism:
 1. **Static** : compile time
 2. **Dynamic** : run time

Polymorphism and Overloading

2 Types:

- **Function overloading** which is the process of using the same name for two or more functions.

Refer This Example: <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/FunctionOverloading/example1.cpp>

- **Operator overloading** which is the process of using the same operator for two or more operands.

Polymorphism and Overloading

Unary Operator Overloading:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/OperatorOverloading/Unary.cpp>

Binary Operator Overloading:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/OperatorOverloading/binary.c>

Polymorphism and Overloading

This refers to the entity which changes its form depending on circumstances at runtime.

Virtual Function:

A virtual function can be defined as the member function within a base class which you expect to redefine in derived classes.

For creating a virtual function, you have to precede your function's declaration within the base class with a **virtual** keyword.

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Function/virtualfunction.cpp>

What is Pure Virtual Function

Pure Virtual Function is a Virtual function with no body.

```
class classname //This denotes the base class of C++ virtual function
{
public:
    virtual void virtualfunctionname() = 0           //This denotes the pure
                                                    virtual function in C++
};
```

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Function/pureVirtualFunction.cpp>

Abstract classes

- An abstract class is a class that is designed to be specifically used as a base class.
- An abstract class contains at least one pure virtual function.
- You declare a pure virtual function by using a pure specifier (= 0) in the declaration of a virtual member function in the class declaration.
- You cannot use an abstract class as a parameter type, a function return type, or the type of an explicit conversion, nor can you declare an object of an abstract class.

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/Abstract%20Class/abstract.cpp>

Static class

- We can define class members static using **static** keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only **one copy of the static member**.
- A static member is **shared by all objects of the class**. All static data is initialized to zero when the first object is created, if no other initialization is present. We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator :: to identify which class it belongs to.

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c++/OOPS/static/static.c>

Static Function Members:

- By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::.
- A static member function can only access static data member, other static member functions and any other functions from outside the class.
- Static member functions have a class scope and they do not have access to the this pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c++/OOPS/static/staticFunction.c>

Static Function Members:

- By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::.
- A static member function can only access static data member, other static member functions and any other functions from outside the class.
- Static member functions have a class scope and they do not have access to the this pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c++/OOPS/static/staticFunction.c>

• Scope Resolution Operator

Member functions can be defined within the class definition or separately using **scope resolution operator**, `::`. Defining a member function within the class definition declares the function inline, even if you do not use the inline specifier. So either you can define Volume() function as below:

```
class Box
{
    public:

        double length; // Length of a box
        double breadth; // Breadth of a box
        double height; // Height of a box

        double getVolume(void)
        {
            return length * breadth * height;
        }
};
```

Inline Function

An inline function in C++ is a function defined with the `inline` keyword, suggesting to the compiler that it should attempt to replace the function call with the actual function code during compilation, reducing the overhead of function calls. This is typically used for small, frequently called functions to improve performance. However, the compiler may ignore the `inline` request if the function is too complex.

Example:

```
inline int add(int a, int b) {  
    return a + b;  
}
```

Encapsulation

Understanding Encapsulation and Information Hiding

Encapsulation in C++ is the concept of bundling data (attributes) and methods (functions) that operate on the data within a single unit (class). Information hiding is a principle of restricting access to the internal state of an object, exposing only necessary details via public interfaces (methods). This improves security, reduces complexity, and allows for easier maintenance and modification of code without affecting other parts of the program.

Understanding Encapsulation and Information Hiding

Encapsulation in C++ is the concept of bundling data (attributes) and methods (functions) that operate on the data within a single unit (class). Information hiding is a principle of restricting access to the internal state of an object, exposing only necessary details via public interfaces (methods). This improves security, reduces complexity, and allows for easier maintenance and modification of code without affecting other parts of the program.

Types of accessifiers

Private	Public	Protected
Only for that class can access	Other class can also access.	Only immediate inheritance class can access.

	Direct-access scope	Class/object scope
Private	Declaring class	Declaring class
Protected	All derived classes	Declaring class
Friend	Derived in-project classes	Declaring project
Protected Friend	All derived classes	Declaring project
Public	All derived classes	All projects

Encapsulation in C++ Classes

Encapsulation in C++ involves grouping data (member variables) and methods (member functions) that operate on the data into a single class. By using access specifiers (**private**, **protected**, **public**), you control the visibility and modification of the class's data, hiding internal implementation details while providing a controlled interface for interaction. This helps improve security, maintainability, and code organization.

Benefits of Encapsulation in OOP

The benefits of encapsulation in Object-Oriented Programming (OOP) include:

- 1. Data Protection:** Encapsulation protects object data by restricting direct access to it, ensuring that data can only be modified through controlled methods, preventing unauthorized or unintended changes.
- 2. Modularity:** It allows for better code organization by bundling related data and functions into a single unit (class), making the code more modular and easier to understand.

Benefits of Encapsulation in OOP

- 3. Maintainability:** Since internal implementation details are hidden, you can modify the internal structure of a class without affecting the external code that uses the class, promoting easier maintenance and updates.

- 4. Abstraction:** Encapsulation supports abstraction by exposing only relevant functionality through public interfaces, hiding complex implementation details from the user.

Friend Functions and Friend Classes

In C++, a **friend function** is a function declared inside a class but defined outside, which has access to the class's private and protected members. A **friend class** is a class that is granted access to the private and protected members of another class. Friend functions and classes allow controlled access to encapsulated data, which is useful in certain scenarios like operator overloading or mutual cooperation between classes.

File Handling

Introduction to File Handling in C++

File handling in C++ allows reading from and writing to files using file streams. The `<fstream>` library provides classes like `ifstream` (input file stream) for reading files, `ofstream` (output file stream) for writing to files, and `fstream` for both input and output operations. Files are accessed using these streams with methods such as `.open()`, `.close()`, `.read()`, `.write()`, and file state checking with `.eof()`, `.fail()`.

Opening, Closing, Reading, and Writing Files

File handling in C++ involves using file streams from the `<fstream>` library. Here's a breakdown of how to handle files:

- 1. Opening a File:** You can open a file using the `open()` method of file stream objects like `ifstream` (for reading), `ofstream` (for writing), or `fstream` (for both). Files can be opened in different modes, such as read (`ios::in`), write (`ios::out`), or append (`ios::app`).
- 2. Writing to a File:** Data can be written to a file using the `<<` operator with `ofstream` or `fstream` objects. This writes text data to the file.

Opening, Closing, Reading, and Writing Files

Reading from a File: To read data, use the `>>` operator for formatted reading or `.getline()` to read a full line from the file. The data is stored in variables or objects as required.

Closing a File: After finishing file operations, it's crucial to close the file using the `close()` method to ensure data is saved and system resources are freed.

These file operations enable C++ programs to handle input/output with files effectively.

Error Handling in File Operations

Error handling in file operations in C++ can be done by:

- 1. Checking File Open Success:** Use `if (!file)` or `file.fail()` after opening a file to ensure it opened correctly.
- 2. End of File (EOF):** Use `.eof()` to check if you've reached the end of the file while reading.
- 3. Write/Read Failures:** Always verify if operations succeed by checking the file stream's state using `.fail()` or `.good()`.

Working with File Streams

Working with file streams in C++ involves using the `<fstream>` library to handle reading from and writing to files. The main classes used are:

1. `ifstream`: Used for reading files.
2. `ofstream`: Used for writing to files.
3. `fstream`: Used for both reading and writing.

Basic Operations:

- **Opening a File:** Use the `.open()` method or provide the filename directly when creating the file stream object.

Working with File Streams

Basic Operations:

- **Opening a File:** Use the `.open()` method or provide the filename directly when creating the file stream object.
- **Reading from a File:** Use the `>>` operator (for formatted data) or `.getline()` for reading lines.
- **Writing to a File:** Use the `<<` operator to write data to a file.
- **Closing a File:** Use `.close()` to close the file when done.

Working with File Streams

Modes:

- **ios::in**: Open for reading.
- **ios::out**: Open for writing (creates a new file if it doesn't exist).
- **ios::app**: Open for appending data to the end of the file.

These operations allow you to perform essential file handling tasks in C++.

[Module 4]

HTML And CSS

What is Internet, HTTP/HTTPS, WWW, Domain name and Top Domain name

- **Internet:** A global network connecting computers and devices, allowing them to share information and communicate.
- **HTTP/HTTPS:** Protocols used for transferring data over the web. HTTP is not secure, while HTTPS includes encryption for security.
- **WWW:** Stands for World Wide Web, a system of websites and pages on the internet that can be accessed through browsers.
- **Domain Name:** The address used to find a website (e.g., google.com).
- **Top-Level Domain (TLD):** The last part of a domain name, like .com, .org, or .net, indicating the type or origin of the website.

What is SEO ?

- SEO (Search Engine Optimization) is the practice of improving a website's visibility in search engine results. It involves optimizing content, keywords, and website structure to make it more attractive to search engines like Google. The goal is to increase organic (non-paid) traffic to the site, helping it rank higher in search results.

What is Text Editor, Web Browser ?

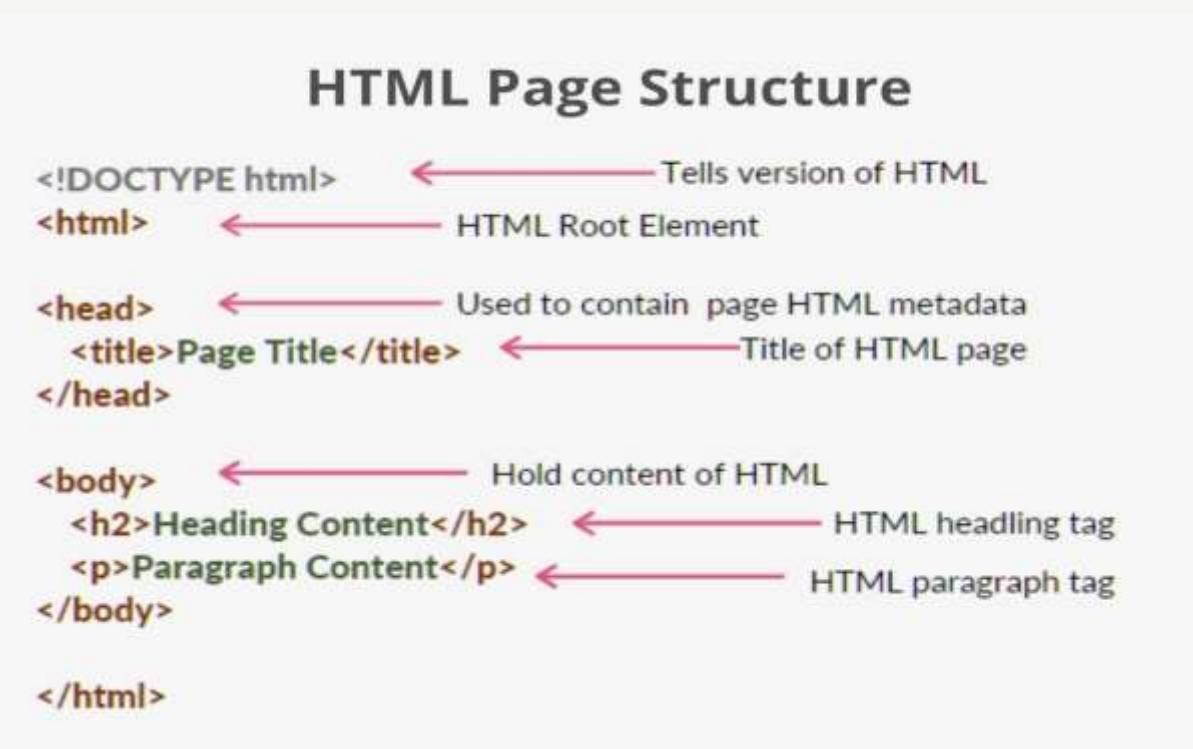
- **Text Editor:** A software used to write and edit plain text, often used by programmers to create code. Examples include Notepad, Sublime Text, and Visual Studio Code.
- **Web Browser:** A software application that allows you to access and view websites on the internet. Examples include Google Chrome, Firefox, Safari, and Microsoft Edge.

HTML

- HTML (Hypertext Markup Language) is the language used to create web page documents.
- The updated version, XHTML (extensible HTML) is essentially the same language with stricter syntax rules.
- (X)HTML is not a programming language; it is a markup language, which means it is a system for identifying and describing the various components of a document such as headings, paragraphs, and lists.
- You don't need programming skills—only patience and common sense—to write (X)HTML.

HTML Structure

- HTML : HTML stands for Hypertext Markup Language. It is a standard markup language for web page creation. It allows the creation and structure of sections, paragraphs, and links using HTML elements (the building blocks of a web page) such as tags and attributes



HTML

HTML Elements:

- HTML documents are text files made up of HTML elements. HTML elements are defined using HTML tags.

HTML Tags:

- HTML tags are used to mark-up HTML elements
- HTML tags are surrounded by the two characters < and > The surrounding characters are called angle brackets HTML tags normally come in pairs like and
- The first tag in a pair is the start tag, the second tag is the end tag The text between the start and end tags is the element content HTML tags are not case sensitive, means the same as
- The browser interprets the HTML tags in the source code and displays your content according to those tags.

HTML

- **Attributes**

HTML attributes provide additional information about HTML elements. They are written within the opening tag of an element and define properties such as behavior, appearance, or content. Here are a few common HTML attributes:

1. **href**: Specifies the URL for a link in an anchor tag ().

Example: `Visit Example`

HTML

1. **src**: Defines the source of an image in an `` tag.

Example: ``

2. **alt**: Provides alternative text for images if they can't be displayed.

Example: ``

3. **class**: Assigns one or more class names to an element, used for styling or scripting.

Example: `<div class="container">Content</div>`

HTML

1. **id**: Provides a unique identifier for an element, often used for styling or JavaScript targeting.

Example: `<button id="submitBtn">Submit</button>`

2. **style**: Allows inline CSS to style an element.

Example: `<p style="color: red;">This is a red text.</p>`

HTML Header

Sure! Here's a simple explanation of HTML layout:

- 1. Header:** The top section of a webpage, often containing the website title, logo, or navigation links.
- 2. Navigation (Nav):** This area typically holds links that guide users to different sections or pages of the website.
- 3. Main Content:** The primary section where most of the content (like articles, images, and text) is displayed.

HTML Header

- 1. Sidebar (Aside):** A secondary section that holds additional, related information such as links, advertisements, or recent posts.

- 2. Footer:** The bottom section of the webpage, often used for copyright information, contact details, or links to privacy policies.

Each of these elements helps structure a webpage in a way that makes it easy to navigate and visually organized. The layout is often styled and arranged using CSS to look appealing and responsive on different devices.

HTML Meta

HTML **meta tags** are used to provide metadata (data about data) for a webpage. This information is not displayed on the page but is used by browsers, search engines, and other services to understand the content and behavior of the page.

Here are some common HTML meta tags:

1. **<meta charset="UTF-8">**: Specifies the character encoding for the webpage (UTF-8 is commonly used for universal character support).

HTML Meta

1. **<meta name="description" content="Description of the webpage">**: Provides a short description of the webpage, which search engines may display in search results.

2. **<meta name="keywords" content="keyword1, keyword2, keyword3">**: Lists keywords related to the page, used for search engine optimization (SEO).

HTML Meta

1. **<meta name="author" content="Author's Name">**: Specifies the name of the author of the webpage.
2. **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Defines the viewable area of the page on different devices, making the page mobile-friendly.
3. **<meta http-equiv="refresh" content="30">**: Refreshes the page after a certain period (in this case, every 30 seconds).

Basic Elements of HTML

- HTML Basic Tags:

Tag	Description
<!DOCTYPE>	Defines the document type
<html>	Defines an HTML document
<head>	Defines information about the document
<title>	Defines a title for the document
<body>	Defines the document's body
<h1> to <h6>	Defines HTML headings
<p>	Defines a paragraph
 	Inserts a single line break
<hr>	Defines a thematic change in the content
<!--...-->	Defines a comment

Tags and Self-Closing Tags

Tags: Tags are used in HTML to define elements on a webpage. They typically come in pairs, with an opening tag and a closing tag, like `<p>` and `</p>` for paragraphs.

- Example: `<h1>This is a heading</h1>`
In this case, `<h1>` is the opening tag, and `</h1>` is the closing tag.

Tags and Self-Closing Tags

Self-Closing Tags: These are tags that don't have a closing tag. They typically end with a slash (/) before the closing angle bracket. Common self-closing tags include those for images, line breaks, and horizontal rules.

- Example:
The tag doesn't need a closing tag because it doesn't have content.

Events

Events are actions that can trigger certain behaviors when a user interacts with an element (like clicking a button or hovering over an image). HTML elements can have event attributes to handle these actions.

1. **onclick**: Triggered when an element is clicked.

- Example: `<button onclick="alert('Hello!')>Click Me</button>`

Events

1. **onmouseover**: Triggered when the mouse is moved over an element.
 - Example: `<div onmouseover="this.style.backgroundColor='yellow'>Hover over me!</div>`
2. **onchange**: Triggered when the value of an element (like a form input) changes.
 - Example: `<input type="text" onchange="alert('Changed!')">`
3. **onload**: Triggered when the page or an image finishes loading.
 - Example: ``

Marquee Tag

The `<marquee>` tag was used to create scrolling text or images in the past, but it is now considered outdated and not recommended for use in modern web design. The `<marquee>` tag automatically scrolls content across the screen.

- `<marquee>Scrolling Text!</marquee>`

Instead of using the `<marquee>` tag, it's better to use CSS animations for creating scrolling effects, as it provides more control and is more accessible.

HTML

Heading Tag Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/HTML/heading.html>

Paragraph Tag Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/HTML/paragraph.html>

Line Break Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/HTML/LineBreak.html>

Horizontal Line Example:

https://github.com/TopsCode/Software-Engineering/blob/master/HTML/horizontTOaPSITLECiHnNOeLOG.IEhS_PtVTm. LDI.

HTML

- HTML : HTML stands for Hypertext Markup Language. It is a standard markup language for web page creation. It allows the creation and structure of sections, paragraphs, and links using HTML elements (the building blocks of a web page) such as tags and attributes

- **HTML Link Tags**

< a href="<http://www.TOPS-int.com>">Go to TOPS-int.com

- **HTML Image**

<https://github.com/TopsCode/Software-Engineering/blob/master/HTML/image1.html>

- **HTML List**

<https://github.com/TopsCode/Software-Engineering/blob/master/HTML/>

- **HTML Table**

<https://github.com/TopsCode/Software-Engineering/blob/master/HTML/Table/c colspan.html>

- **HTML Form**

<https://github.com/TopsCode/Software-Engineering/blob/master/HTML/Form/example1.html>

HTML

- **Formatting Text:**

1. Bold Text: `` and ``
2. Italic Text: `<i>` and ``
3. Marked Formatting: `<mark>`
4. Underlined Text: `<u>` and `<ins>`
5. Strike Text: `<strike>` and ``
6. Monospaced Font: `<tt>`
7. Superscript Text: `<sup>`
8. Subscript Text: `<sub>`
9. Larger Text: `<big>`
10. Smaller Text: `<small>`

Basic Elements of HTML

- **HTML Table Elements:**

```
<table>
<tr>
<th>
<td>
<caption>
<tbody>
<thead>
<tfooter>
```

Attributes:

rowspan
colspan

Basic Elements of HTML

- **HTML FORM Elements:**

- <form>
- <input>
- <textarea>
- <label>
- <fieldset>
- <legend>

CSS

- CSS: CSS stands for Cascading Style Sheets. It is the language for describing the presentation of Web pages, including colors, layout, and fonts, thus making our web pages presentable to the users.
- CSS is designed to make style sheets for the web. It is independent of HTML and can be used with any XML-based markup language. Now let's try to break the acronym:
 - i. Cascading: Falling of Styles
 - ii. Style: Adding designs/Styling our HTML tags
 - iii. Sheets: Writing our style in different documents



CSS

- There are three ways to insert CSS in HTML documents.

1. Inline CSS

exa: <h2 style="color:red;margin-left:40px;">Inline CSS is applied on this heading.</h2>

2. Internal CSS

exa: <style>
body {
background-color: linen;
}
h1 {
color: red; margin-left: 80px;
}
</style>

CSS

3. External CSS

exa:<head>

```
<link rel="stylesheet" type="text/css" href="mystyle.css">  
</head>
```

- Create Basic layout structure using div tag

CSS

Types of Selectors :

1. Universal Selector
2. HTML selector
3. Id Selector
4. Class Selector
5. Descendant Selector

CSS

Examples :

<https://github.com/TopsCode/Software-Engineering/blob/master/CSS/htmlSelector.html>

<https://github.com/TopsCode/Software-Engineering/blob/master/CSS/universalSelector.html>

<https://github.com/TopsCode/Software-Engineering/blob/master/CSS/classSelector.html>

<https://github.com/TopsCode/Software-Engineering/blob/master/CSS/dependantClassSelector.html>

<https://github.com/TopsCode/Software-Engineering/blob/master/CSS/idSelector.html>

<https://github.com/TopsCode/Software-Engineering/blob/master/CSS/descendantSelector.css>

CSS

Font Formatting with CSS

1. **font-family**

- The font family of a text is set with the font-family property.
- The font-family property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font, and so on.
- Start with the font you want, and end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

Example: <https://github.com/TopsCode/Software-Engineering/blob/master/CSS/Font/font-family.html>

CSS

2. **Serif**

Examples: Times, Times New Roman, Georgia

Serif typefaces have decorative serifs, or slab-like appendages, on the ends of certain letter strokes.

3. **sans-serif**

Examples: Arial, Arial Black, Verdana, Trebuchet MS, Helvetica, Geneva

Sans-serif typefaces have straight letter strokes that do not end in serifs.

They are generally considered easier to read on computer .

4. **Monospace**

Examples: Courier, Courier New, and Andale Mono

In monospace (also called constant width) typefaces, all characters take up the same amount of

4. **Cursive**

Examples: Apple Chancery, Zapf-Chancery, and Comic Sans

Cursive fonts emulate a script or handwritten appearance.

These are rarely specified for professional web pages.

CSS

2. Fantasy

Examples: Impact, Western, or other decorative font

Fantasy fonts are purely decorative and would be appropriate for headlines and other display type.

Fantasy fonts are rarely used for web text due to cross-platform availability and legibility.

Specifying font size:

Use the aptly-named font-size property to specify the size of the text.

font-size

Values: length unit, percentage, xx-small | x-small | small | medium | large | x-large | xx-large | smaller | larger | inherit

Default: medium Applies to: all elements Inherits: yes

<https://github.com/TopsCode/Software-Engineering/blob/master/CSS/Font/font-size.html>

CSS Box Model

- **Margin** - Clears an area around the border. The margin does not have a background color, it is completely transparent
- **Border** - A border that goes around the padding and content. The border is affected by the background color of the box
- **Padding** - Clears an area around the content. The padding is affected by the background color of the box
- **Content** - The content of the box, where text and images appear

In order to set the width and height of an element correctly in all browsers, you need to know how the box model works.

<https://github.com/TopsCode/Software- Engineering/blob/master/CSS/boxModel.html>

<https://github.com/TopsCode/Software- Engineering/blob/master/CSS/boxmodel2.html>

Pseudo Classes and Elements

Pseudo-Classes

Pseudo-classes are used in CSS to define the special state of an element. They allow you to style elements based on user interaction or their position in the document, without needing to add extra classes or IDs in the HTML.

Pseudo Classes and Elements

:hover: Applies styles when the user hovers their mouse over an element.

Example: `a:hover { color: red; }`

(This will change the link color to red when the mouse is hovered over it.)

:focus: Applies styles when an element, like an input field, gains focus (i.e., when the user clicks into it or navigates to it via keyboard).

Example: `input:focus { background-color: lightblue; }`

Pseudo Classes and Elements

:active: Applies styles to an element when it is being clicked or activated.

Example: `button:active { background-color: blue; }`

(This will change the button's background to blue when it's clicked.)

Pseudo Classes and Elements

Pseudo-Elements

Pseudo-elements are used to style specific parts of an element, like the first letter or line of text. They allow for additional styling without adding extra HTML markup.

Here are some common **pseudo-elements**:

1. **::before**: Inserts content before an element's actual content.
Often used for adding decorative content (like icons or quotes) before the content.

Pseudo Classes and Elements

::after: Inserts content after an element's actual content. Similar to **::before**, but adds content after the element.

Differences Between Pseudo-Classes and Pseudo-Elements

- **Pseudo-Classes**: Select elements based on their state or position (e.g., when hovered or focused).
- **Pseudo-Elements**: Target specific parts of an element or add content before or after it (e.g., first letter, first line, or inserting content).

Float and Clear and Alignment

- **Float:** Positions an element to the left or right of its container, with content wrapping around it.
- **Clear:** Used to prevent elements from wrapping around floated elements, forcing them to start below the float.
- **Alignment:** Includes text alignment (left, right, center), vertical alignment, and more advanced methods like Flexbox to align elements in modern layouts.

Float and Clear and Alignment

The **float** property in CSS is used to push elements (like images or text) to the left or right of their containing element, allowing other content to flow around them. It's commonly used for creating layouts, especially for wrapping text around images or positioning elements side by side.

- **Left Float (`float: left`)**: Makes the element float to the left, allowing content to flow to the right of it.
- **Right Float (`float: right`)**: Makes the element float to the right, allowing content to flow to the left of it.

Float and Clear and Alignment

The **clear** property is used to control the behavior of elements that follow a floated element. It ensures that an element is displayed below any floated elements, preventing it from wrapping around the float.

- **clear: left**: The element cannot be next to any floated element on the left side.
- **clear: right**: The element cannot be next to any floated element on the right side.
- **clear: both**: The element cannot be next to any floated element on either side.

Float and Clear and Alignment

Alignment in CSS refers to positioning or aligning elements within their containers. This can be done using several properties depending on the type of alignment you're aiming for:

1. Text Alignment (`text-align`):

- **`text-align: left`**: Aligns text to the left.
- **`text-align: center`**: Centers text within its container.
- **`text-align: right`**: Aligns text to the right.

Opacity and Visibility

- **opacity** affects the **transparency** of the element. When the opacity is set to `0`, the element is completely invisible, but it still responds to events (like clicks).
- **visibility** controls the **visibility** of the element. When set to `hidden`, the element is not visible, but it still takes up space in the layout and can be interacted with depending on its position and size.

Opacity and Visibility

The **opacity** property in CSS controls the transparency of an element. It is a value between **0** (completely transparent) and **1** (completely opaque). The lower the opacity value, the more transparent the element becomes, allowing the background or content behind it to be visible.

- **opacity: 0**: The element is completely transparent (invisible).
- **opacity: 1**: The element is fully visible (no transparency).
- **opacity: 0.5**: The element is 50% transparent.

Opacity and Visibility

The **visibility** property controls whether an element is visible or hidden, but it differs from **display** in that the element still occupies space in the layout, even when hidden.

- **visibility: visible**: The element is visible (default).
- **visibility: hidden**: The element is hidden, but it still occupies space in the layout (it does not affect the flow of other elements).
- **visibility: collapse**: Used for table rows or columns; it will collapse the element and remove it from the layout.

CSS Background

- CSS background properties are used to control the background styling of an element. You can set background colors, images, and even manage the positioning of the background. You can also control how the background image behaves (whether it repeats or not) and its size (to cover the entire area or be contained).

- **4. CSS Links**
- In CSS, you can style hyperlinks (tags). The common properties include setting the color of links (normal, visited, or

CSS Links

In CSS, you can style hyperlinks (tags). The common properties include setting the color of links (normal, visited, or active states), and controlling text decoration (like underlining or removing it). Additionally, you can define hover and active states, allowing you to change the style of a link when the user interacts with it, such as when they hover over or click on the link.

CSS Display

The **display** property in CSS defines how an element is displayed on the page. It controls whether an element behaves like a block-level element (taking up the full width available), an inline element (occupying only the space required), or an inline-block element (a mix of both). You can also use the **display** property to hide elements completely by setting them to **none**.

[Module 5]

Database

DBMS

- DBMS stands for Data Base Management System.
- Data + Management System
- Database is a collection of inter-related data and Management System is a set of programs to store and retrieve those data.
- DBMS is a collection of inter-related data and set of programs to store & access those data in an easy and effective manner.
- For Example, university database organizes the data about students, faculty, and admin staff etc. which helps in efficient retrieval, insertion and deletion of data from it.

DBMS

Here is a list of some popular database.

- MySQL
- Microsoft Access
- Oracle
- PostgreSQL
- dBASE
- FoxPro
- SQLite
- IBM DB2
- LibreOffice Base

Need of DBMS

- Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization: Storage of data and retrieval of data
Storage:
 - According to the principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage.
 - Fast Retrieval of data: Along with storing the data in an optimized and systematic manner, it is also important that we retrieve the data quickly when needed. Database systems ensure that the data is retrieved as quickly as possible.

DBMS-RDBMS

- A **Database Management System (DBMS)** is software that manages and controls access to a database, handling tasks like data storage, retrieval, and manipulation. An **RDBMS (Relational DBMS)** is a type of DBMS that stores data in tables (relations) and uses structured query language (SQL) for data management. RDBMS supports features like data integrity, relationships between tables, and normalization, ensuring efficient data organization and retrieval. Examples include MySQL, PostgreSQL, and Oracle.

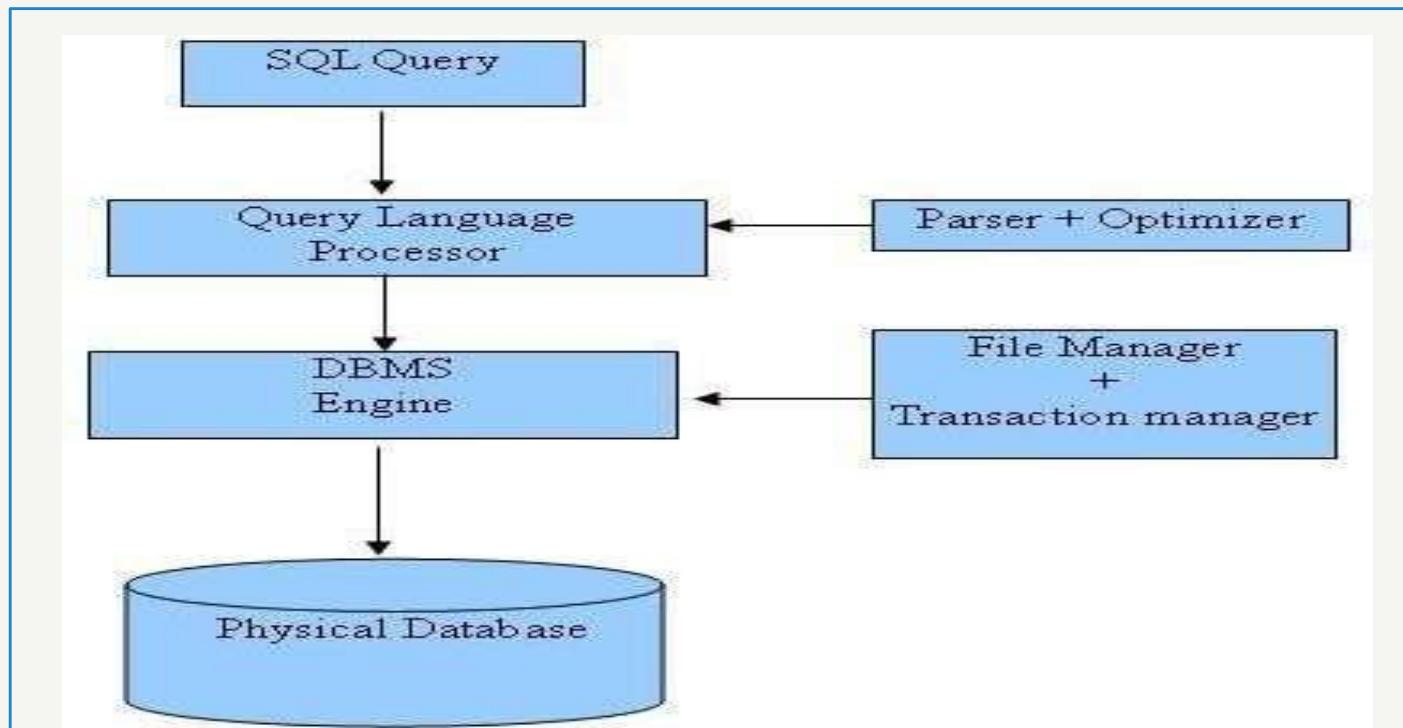
Introduction to SQL

What is SQL?

- SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.
- SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc.
- SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.
- Also, they are using different dialects, such as:
- MS SQL Server using T-SQL, ANSI SQL
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc

Objectives

- “The large majority of today's business applications revolve around relational databases and the SQL programming language (Structured Query Language). Few businesses could function without these technologies...”



What is SQL?

- Allows users to access data in relational database management systems. Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database. Allows users to set permissions on tables, procedures, and views
- SQL stands for Structured Query Language
- SQL allows you to access a database
- SQL is an ANSI standard computer language
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert new records in a database

What is SQL?

- SQL can delete records from a database
- SQL can update records in a database
- SQL is easy to learn
- action queries insert, update & delete data
- select queries retrieve data from DB

SQL constraints

Constraints(Keys)

Primary Key:

- A primary key is a column of table which uniquely identifies each tuple (row) in that table.
- Primary key enforces integrity constraints to the table.
- Only one primary key is allowed to use in a table.
- The primary key does not accept the any duplicate and NULL values.
- The primary key value in a table changes very rarely so it is chosen with care where the changes can occur in a seldom manner.
- A primary key of one table can be referenced by foreign key of another table.

Constraints(Keys)

Unique Key:

- Unique key constraints also identifies an individual table uniquely in a relation or table.
- A table can have more than one unique key unlike primary key.
- Unique key constraints can accept only one NULL value for column.
- Unique constraints are also referenced by the foreign key of another table.

Constraints(Keys)

Foreign Key:

- When, "one" table's primary key field is added to a related "many" table in order to create the common field which relates the two tables, it is called a foreign key in the "many" table.
- In the example given below, salary of an employee is stored in salary table.
- Relation is established via is stored in "Employee" table. To identify the salary of "Jhon" is stored in "Salary" table. But his employee info of "Jhon" is stored in "Salary" table. But his employee info
- For example, salary record of Jhon is stored in salary table. His Employee ID is 1. So, Employee_ID_Ref column in salary table has value 1. This value 1 refers to Employee_ID column in Employee table. So, Employee_ID_Ref column is foreign key in salary table.

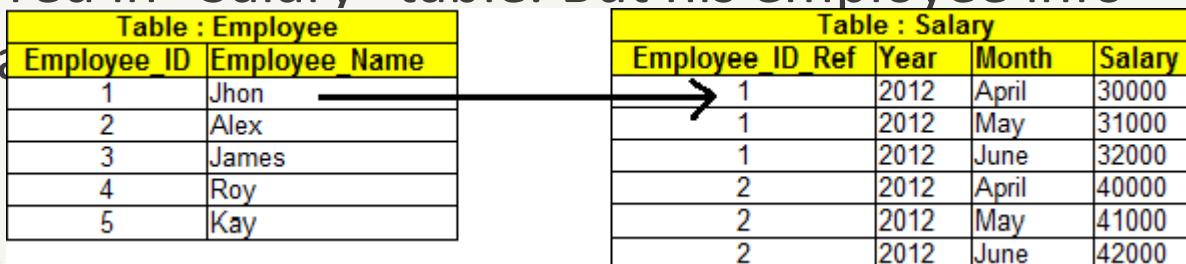


Table : Employee		Table : Salary			
Employee_ID	Employee_Name	Employee_ID_Ref	Year	Month	Salary
1	Jhon	1	2012	April	30000
2	Alex	1	2012	May	31000
3	James	1	2012	June	32000
4	Roy	2	2012	April	40000
5	Kay	2	2012	May	41000
		2	2012	June	42000

SQL syntax

Types

SQL Statement Types

- DDL – Data Definition Language
- DML – Data Manipulation Language
- DCL – Data Control Language
- DQL – Data Query Language
- **SQL Join Types**
- INNER JOIN: returns rows when there is a match in both tables.
- LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN: returns rows when there is a match in one of the tables.

Main SQL Commands and Sub-commands

SQL Statement Types

DDL - Data Definition Language

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

SQL Statement Types

DOL – Data Query Language

Command	Description
SELECT	Retrieves certain records from one or more tables

- **DML – Data Manipulation Language**

Command	Description
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

SQL Statement Types

DCL – Data Control Language

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

SQL Statement Types

SQL CREATE DATABASE STATEMENT

```
CREATE DATABASE database_name;
```

SQL DROP DATABASE Statement:

```
DROP DATABASE database_name;
```

SQL USE STATEMENT

```
USE DATABASE database_name;
```

SQL CREATE TABLE STATEMENT

```
CREATE TABLE table_name( column1 datatype, column2 datatype, column3 datatype,  
..... , columnN datatype, PRIMARY KEY( one or more columns ) );
```

SQL Statement Types

SQL DROP TABLE STATEMENT

```
DROP TABLE table_name;
```

SQL TRUNCATE TABLE STATEMENT

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE STATEMENT (RENAME)

```
ALTER TABLE table_name RENAME TO new_table_name;
```

SQL INSERT INTO STATEMENT

```
INSERT INTO table_name( column1, column2....columnN) VALUES ( value1, value2.  
valueN);
```

SQL Statement Types

SQL UPDATE STATEMENT

UPDATE table_name SET column1 = value1, column2 = value2. columnN=valueN
[WHERE CONDITION];

SQL DELETE STATEMENT

DELETE FROM table_name WHERE {CONDITION}

SQL SELECT STATEMENT

SELECT column1, column2....columnN FROM table_name;

SQL DISTINCT CLAUSE

SELECT DISTINCT column1, column2....columnN FROM table_name;

SQL Statement Types

SQL WHERE CLAUSE

```
SELECT column1, column2....columnN FROM      table_name WHERE CONDITION;
```

SQL AND/OR CLAUSE

```
SELECT column1, column2....columnN FROM      table_name WHERE CONDITION-1  
{AND|OR} CONDITION-2;
```

SQL IN CLAUSE

```
SELECT column1, column2.      column table_name WHERE column_name IN  
FROM (val-1, val-2,      val-N);
```

SQL BETWEEN CLAUSE

```
SELECT column1, column2.      column table_name WHERE column_name  
FROM BETWEEN val-1 AND val-2;
```

SQL Statement Types

SQL LIKE CLAUSE

```
SELECT column1, column2.    column table_name WHERE column_name LIKE {  
FROM PATTERN };
```

SQL ORDER BY CLAUSE

```
SELECT column1, column2.    columnN    table_name WHERE CONDITION  
ORDER FROM BY column_name {ASC|DESC};
```

SQL GROUP BY CLAUSE

```
SELECT SUM(column_name) table_name WHERE CONDITION GROUP BY  
FROM column_name;
```

SQL COUNT CLAUSE

```
SELECT COUNT(column_name)FROM table_name WHERE CONDITION;
```

SQL Statement Types

SQL HAVING CLAUSE

SELECT SUM(column_name) FROM table_name WHERE CONDITION GROUP BY column_name HAVING (arithematicfunction condition)

SQL CREATE INDEX Statement :

CREATE UNIQUE INDEX index_name ON table_name(column1, column2,...columnN);

SQL DROP INDEX STATEMENT

ALTER TABLE table_name DROP INDEX index_name;

SQL DESC Statement :

DESC table_name;

SQL COMMIT STATEMENT

COMMIT

SQL Statement Types

SQL ROLLBACK STATEMENT

```
ROLLBACK;
```

JOIN

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them.

Different types of Joins are:

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN

SQL Statement Types

- **1. Inner Join**

The most frequently used and important of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN.

- The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. T
- he query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax :

```
SELECT table1.column1, table2.column2...FROM table1INNER JOIN table2ON  
table1.common_field = table2.common_field
```

SQL Statement Types

- **Left Join**
- The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result, but with NULL in each column from right table.
- This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.
- **Syntax:**
- `SELECT table1.column1, table2.column2...FROM table1LEFT JOIN table2ON table1.common_filed = table2.common_field;`

SQL Statement Types

- **Right Join**
- The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result, but with NULL in each column from left table.
- This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.
- **Syntax:**
- `SELECT table1.column1, table2.column2...FROM table1RIGHT JOIN table2ON table1.common_field = table2.common_field`

SQL Statement Types

- **Full Join**
- The SQL FULL JOIN combines the results of both left and right outer joins.
- The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.
- **Syntax:**
- `SELECT table1.column1, table2.column2...FROM table1FULL JOIN table2ON table1.common_filed = table2.common_field;`

SQL Statement Types

Function

- SQL has many built-in functions for performing calculations on data. They are divided into 2 categories:
 1. Aggregate Function
 2. Scalar Function

1. Aggregate Function

- These functions are used to do operations from the values of the column and a single value is returned.
- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

SQL Statement Types

1. AVG():

Syntax: `SELECT AVG(column_name) FROM table_name;`

Example:

```
SELECT AVG(AGE) AS AvgAge FROM Students;
```

2. COUNT()

Syntax: `SELECT COUNT(column_name) FROM table_name;`

Example: `SELECT COUNT(*) AS NumStudents FROM Stuents;`

3. FIRST()

Syntax: `SELECT FIRST(column_name) FROM table_name`

Example:

```
SELECT FIRST(MARKS) AS MarksFirst FROM Students;
```

SQL Statement Types

4. LAST()

Syntax: SELECT LAST(column_name) FROM table_name;

Example: SELECT LAST(MARKS) AS MarksLast FROM Students;

5. MAX()

Syntax: SELECT MAX(column_name) FROM table_name;

Example :SELECT MAX(MARKS) AS MaxMarks FROM Students;

6. MIN(): Similar to Max() we can use MIN() function.

7. SUM() :

Syntax: SELECT SUM(column_name) FROM table_name;

Example: SELECT SUM(MARKS) AS TotalMarks FROM Stuents;

SQL Statement Types

Transaction Control

- **The following commands are used to control transactions**
 1. COMMIT – to save the changes.
 2. ROLLBACK – to roll back the changes.
 3. SAVEPOINT – creates points within the groups of transactions in which to ROLLBACK.
- **Commit:**
 - The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.
 - The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

SQL Statement Types

- **Rollback:**
 - The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.
 - This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.
 - The syntax for a ROLLBACK command is as follows – ROLLBACK;
- **Savepoint:**
 - A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.
 - The syntax for a SAVEPOINT command is as shown below.
 - SAVEPOINT SAVEPOINT_NAME;
 - This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.

SQL Statement Types

PROCEDURE

- A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
- So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
- You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.
- To create procedure, use syntax:

```
CREATE PROCEDURE procedure_name AS sql_statement GO;
```

- To execute created procedure, use syntax:

```
EXEC procedure_name;
```

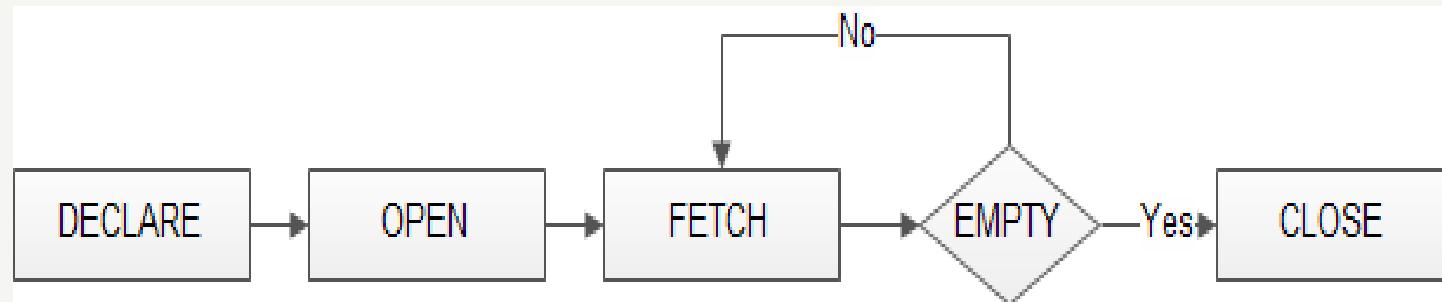
SQL Statement Types

VIEW

- A **SQL View** is a virtual table that is created by querying one or more tables in a database. It doesn't store data itself but provides a way to simplify complex queries by encapsulating them into a single virtual table. Views can be used to simplify data access, enhance security (by restricting access to certain data), and provide a consistent interface. They are created using the **CREATE VIEW** statement.

SQL Statement Types

- **Cursor:**
- It is a temporary area for work in memory system while the execution of a statement is done.
- A Cursor in SQL is an arrangement of rows together with a pointer that recognizes a present row.
- It is a database object to recover information from a result set one row at once.
- It is helpful when we need to control the record of a table in a singleton technique, at the end of the day one row at any given moment. The arrangement of columns the cursor holds is known as the dynamic set.



SQL Statement Types

- **Cursor:**
Syntax
:

```
DECLARE variables;  
records;  
create a cursor;  
BEGIN  
OPEN cursor;  
FETCH cursor;  
process the records;  
CLOSE cursor;  
END;
```

SQL Statement Types

Cursor Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/SQL/Cursor/example>

Stored Procedure with one parameter:

<https://github.com/TopsCode/Software-Engineering/tree/master/SQL/Cursor>

Stored Procedure with multiple parameter:

<https://github.com/TopsCode/Software-Engineering/blob/master/SQL/Cursor/multipleParam>

SQL Statement Types

Trigger

- A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs
- For example, a trigger can be invoked when a row is inserted into a specified table.

Syntax:

```
create trigger [trigger_name] [before | after]
{insert | update | delete} on [table_name] [for each row] [trigger_body]
```

Before Trigger:

<https://github.com/TopsCode/Software-Engineering/blob/master/SQL/Trigger/beforeTrigger>

After Trigger:

<https://github.com/TopsCode/Software-Engineering/blob/master/SQL/Trigger/afterTrigger>

PL/SQL Essentials

Introduction to PL/SQL

Definition and Purpose of PL/SQL

PL/SQL (Procedural Language/SQL) is Oracle's procedural extension to SQL, designed to enhance SQL's capabilities by adding programming constructs like variables, loops, conditionals, and error handling. It allows developers to write complex database operations in the form of procedures, functions, triggers, and packages. The purpose of PL/SQL is to provide more control over data processing, improve performance through batch processing, and enable the creation of reusable, modular database logic.

Introduction to PL/SQL

Benefits of Using PL/SQL

The benefits of using **PL/SQL** include:

- 1. Enhanced Performance:** PL/SQL allows batch processing, reducing the number of database calls by executing multiple SQL statements in a single block, improving performance.
- 2. Error Handling:** It provides robust exception handling, allowing you to manage and respond to errors effectively during database operations.

Introduction to PL/SQL

Benefits of Using PL/SQL

Modularity: PL/SQL supports the creation of reusable code through stored procedures, functions, and packages, promoting code organization and maintainability.

Security: By using stored procedures and functions, you can control access to sensitive data, as users can interact with the database only through these defined interfaces.

Introduction to PL/SQL

Benefits of Using PL/SQL

Portability: PL/SQL is tightly integrated with Oracle databases, allowing for seamless migration of database logic across different applications or environments.

PL/SQL Syntax

Structure of PL/SQL Blocks

The structure of a **PL/SQL block** consists of three main sections:

1. Declaration Section (optional):

- This is where variables, constants, cursors, and types are declared.
It starts with the **DECLARE** keyword and ends before the executable section.

Example:

```
DECLARE  
v_name VARCHAR2(100);  
v_salary NUMBER;
```

Structure of PL/SQL Blocks

2. Executable Section (mandatory):

- This section contains the actual PL/SQL code where logic, SQL statements, and procedures are executed. It starts with the **BEGIN** keyword and ends with **END ;**.

Example:

```
BEGIN
    v_salary := 5000;
    v_name := 'John Doe';
    DBMS_OUTPUT.PUT_LINE('Employee: ' || v_name || ' Salary: ' ||
    v_salary);
```

Structure of PL/SQL Blocks

3. Exception Handling Section (optional):

This section handles errors and exceptions that occur during execution. It starts with **EXCEPTION** and contains **WHEN** clauses for specific error types.

Example:

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No data found');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error occurred');
END;
```

Variables, Constants, and Data Types in PL/SQL

In PL/SQL:

1. **Variables** are used to store temporary data during execution, and their values can be modified. They are declared with a specified data type in the declaration section.

2. **Constants** are similar to variables but cannot have their values changed once assigned. They are declared using the **CONSTANT** keyword and must be assigned a value at the time of declaration.

Variables, Constants, and Data Types in PL/SQL

In PL/SQL:

Data Types in PL/SQL define the kind of data a variable or constant can hold. Common types include:

- **Scalar types** like NUMBER, VARCHAR2, DATE, and CHAR.
- **Composite types** like %TYPE (which allows a variable to inherit a column's data type) and %ROWTYPE (for row-like structures).
- **LOB types** like CLOB and BLOB for storing large objects such as text or binary data.

PL/SQL Control Structures

IF-THEN, IF-THEN-ELSE, CASE Statements

1. IF-THEN

This statement executes a block of code if a condition is true.

- DECLARE
- v_age NUMBER := 25;
- BEGIN
- IF v_age >= 18 THEN
- DBMS_OUTPUT.PUT_LINE('Adult');
- END IF;
- END;

IF-THEN, IF-THEN-ELSE, CASE Statements

2. IF-THEN-ELSE

This statement executes one block of code if the condition is true, and another block if it is false.

- DECLARE
- v_age NUMBER := 16;
- BEGIN
- IF v_age >= 18 THEN
- DBMS_OUTPUT.PUT_LINE('Adult');
- ELSE
- DBMS_OUTPUT.PUT_LINE('Minor');
- END IF;
- END;

IF-THEN, IF-THEN-ELSE, CASE Statements

3. CASE Statement

The **CASE** statement evaluates an expression and executes a block of code based on the match. It can be a **simple** or **searched** case.

- **Simple CASE:**

- DECLARE
- v_grade CHAR := 'A';
- BEGIN
- CASE v_grade
- WHEN 'A' THEN
- DBMS_OUTPUT.PUT_LINE('Excellent');
- WHEN 'B' THEN
- DBMS_OUTPUT.PUT_LINE('Good');
- WHEN 'C' THEN
- DBMS_OUTPUT.PUT_LINE('Average');
- ELSE
- DBMS_OUTPUT.PUT_LINE('Invalid grade');
- END CASE;
- END;

IF-THEN, IF-THEN-ELSE, CASE Statements

Searched CASE:

- DECLARE
- v_age NUMBER := 20;
- BEGIN
- CASE
- WHEN v_age >= 18 THEN
- DBMS_OUTPUT.PUT_LINE('Adult');
- WHEN v_age < 18 THEN
- DBMS_OUTPUT.PUT_LINE('Minor');
- END CASE;
- END;

Searched CASE:

- DECLARE
- v_age NUMBER := 20;
- BEGIN
- CASE
- WHEN v_age >= 18 THEN
- DBMS_OUTPUT.PUT_LINE('Adult');
- WHEN v_age < 18 THEN
- DBMS_OUTPUT.PUT_LINE('Minor');
- END CASE;
- END;

Loops (WHILE, FOR)

1. WHILE Loop

A **WHILE loop** repeatedly executes a block of code as long as the specified condition is true. The condition is evaluated before each iteration.

```
DECLARE
    v_counter NUMBER := 1;
BEGIN
    WHILE v_counter <= 5 LOOP
        DBMS_OUTPUT.PUT_LINE('Counter: ' || v_counter);
        v_counter := v_counter + 1; -- Increment counter
    END LOOP;
END;
```

Loops (WHILE, FOR)

2. FOR Loop

A **FOR loop** is used when the number of iterations is known beforehand. It automatically handles the initialization, condition checking, and incrementing of the loop variable.

```
BEGIN
    FOR v_counter IN 1..5 LOOP
        DBMS_OUTPUT.PUT_LINE('Counter: ' || v_counter);
    END LOOP;
END;
```

Advanced PL/SQL

Concepts

SQL Cursor

In PL/SQL, a **cursor** is a pointer that allows you to retrieve and manipulate data row-by-row from a result set returned by a SQL query. Cursors are used to handle queries that return multiple rows. There are two types of cursors: **implicit cursors**, which PL/SQL handles automatically for single SQL statements, and **explicit cursors**, which you define to handle more complex queries manually. Cursors are essential for working with result sets in a controlled, iterative manner.

Implicit vs. Explicit Cursors

1. Implicit Cursors:

- Automatically created by PL/SQL for single SQL queries (like `SELECT INTO`, `INSERT`, `UPDATE`, or `DELETE`).
- You don't need to declare them or manage their lifecycle; PL/SQL handles them automatically.
- You can access attributes like `%FOUND`, `%NOTFOUND`, `%ROWCOUNT`, and `%ISOPEN` to check the status of the query.

Implicit vs. Explicit Cursors

Example:

```
DECLARE
```

```
    v_name VARCHAR2(100);
```

```
BEGIN
```

```
    SELECT name INTO v_name FROM employees WHERE  
    employee_id = 101;
```

```
-- PL/SQL implicitly handles the cursor for the SELECT statement
```

```
END;
```

Implicit vs. Explicit Cursors

2. Explicit Cursors:

- Defined explicitly by the programmer to handle queries that return multiple rows.
- You must declare, open, fetch, and close the cursor manually, providing more control over the result set.
- Useful when you need to process multiple rows or perform complex queries.

Implicit vs. Explicit Cursors

Example:

```
DECLARE  
CURSOR emp_cursor IS  
    SELECT name FROM employees  
    WHERE department_id = 10;  
    v_name employees.name%TYPE;  
BEGIN  
    OPEN emp_cursor;  
LOOP
```

```
    FETCH emp_cursor  
    INTO v_name;  
  
    EXIT WHEN  
    emp_cursor%NOTFOU  
    ND;  
  
    DBMS_OUTPUT.PUT_L  
    INE(v_name);  
  
    END LOOP;  
  
    CLOSE emp_cursor;  
END;
```

Key difference :

Implicit Cursors: Automatically managed by PL/SQL for simple queries, no explicit declaration needed.

Explicit Cursors: Manually defined and controlled, used for complex queries or when handling multiple rows.

Rollback and Commit Savepoint

Transaction Management in PL/SQL

Transaction management in PL/SQL ensures that a series of SQL operations are executed as a single unit, preserving the integrity and consistency of the database. Key concepts include:

1. **BEGIN**: Marks the start of a transaction.
2. **COMMIT**: Permanently saves all changes made during the transaction to the database. Once committed, changes cannot be undone.

Transaction Management in PL/SQL

- 3. ROLLBACK:** Reverts all changes made during the transaction back to the last committed state, ensuring data is not corrupted in case of errors.
- 4. SAVEPOINT:** Creates a point within a transaction that can be rolled back to, allowing partial rollback.
- 5. SET TRANSACTION:** Configures transaction properties, such as isolation levels (e.g., read-only).

11.Tools

Tools can be used in Programming



THANK YOU