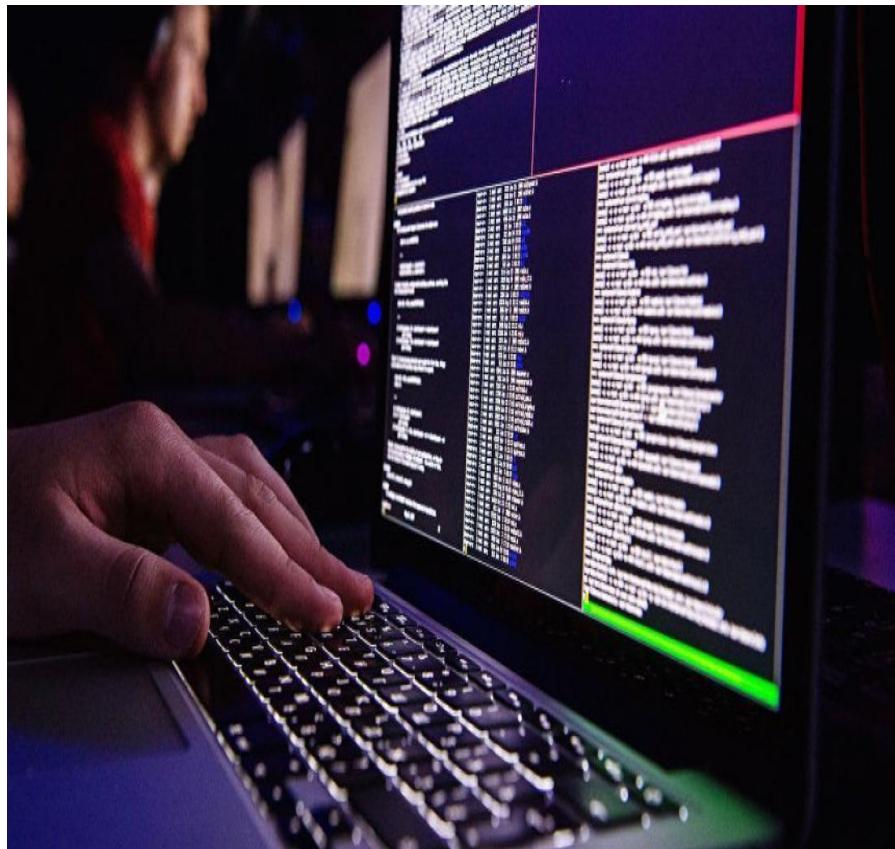


# TOPS Technologies



Software  
Engineering  
Overview



# Topics:

Module – 1 – Software

Module – 2 – Web Development

Module – 3 – Programming Language

Module – 4 – Programming OOPS

Module – 5 – Database

Module – 6 – Linux Scripting



# Module 1 Topics:

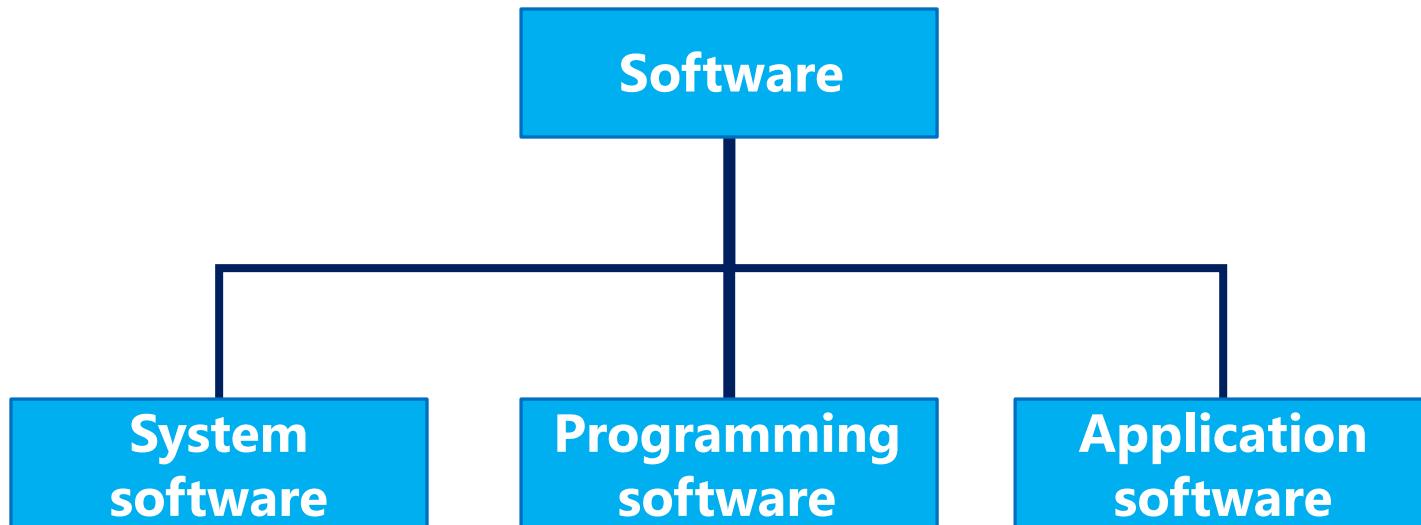
- **What is Software?**
  - Types of software
- **Software Development Process**
  - Software Requirement
  - Software Analysis
  - Designing
  - DFD , flow Chart, Class Diagram, Canvas
- **Model Design with UML**
  - Class Model, State Model and Interaction Model
  - Data Dictionary
  - Class Diagram
  - Development



## What is Software ?

Software, is a collection of computer programs and related data that provide the instructions for telling a computer what to do and how to do it.

## Types of Software



## ❖ **System Software**

System software provides the basic functions for computer usage and helps run the computer hardware and system.

## ❖ **Programming Software**

Programming is the process of designing, writing, testing, debugging, and maintaining the source code of computer programs. This source code is written in a programming language. The purpose of programming is to create a program that exhibits a certain desired behavior.

## ❖ **Application Software**

Application software is the general designation of computer programs for performing user tasks.



## Types of Application Software:

- ❖ **Mobile App:**

Application that runs on mobile platform. Example: Instagram app,

- ❖ **Desktop App:**

Application that runs stand-alone in a desktop or laptop computer.  
Example: Microsoft Word, Web Browser.

- ❖ **Web App:**

Apps that run on a web browser(Mozilla, Google Chrome etc.)  
Example: [www.facebook.com](http://www.facebook.com), [www.google.com](http://www.google.com)

# What is Software Engineering?

---

A naive view: Problem Specification Final Program

But ...

Where did the specification come from?

How do you know the specification corresponds to the user's needs?

How did you decide how to structure your program?

How do you know the program actually meets the specification? How  
do you know your program will always work correctly?

What do you do if the users' needs change?

How do you divide tasks up if you have more than a one-person team?

# What is Software Engineering?

---

## Some Definitions and Issues

“Software engineering is the art of developing quality software on time and within budget.”

Trade-off between perfection and physical constraints

- SE has to deal with real-world issues

State of the art!

- Community decides on “best practice” + life-long duration

# What is Software Engineering?

---

## What is Software Engineering? (II)

“Software engineering is the multi-person construction of multi-version software” - Parnas

Team-work

- Scale issue (“program well” is not enough) + Communication Issue

Successful software systems must evolve or perish

- Change is the norm, not the exception

# What is Software Engineering?

---

## What is Software Engineering? (III)

“software engineering is different from other engineering disciplines” — Summerville

Not constrained by physical laws

- limit = human mind

It is constrained by political forces

- balancing stake-holders

# Software Development Process

---

- Software development process is a set of steps that a software program goes through when developed.
- Alternatively referred to as **software life cycle** and **software development phases**.
- SDLC is a structure imposed on the development of a software product that defines the process for **planning, implementation, testing, documentation, deployment, and ongoing maintenance and support**. There are a number of different development models.
- A Software Development process is essentially a series of steps, or phases, that provide a model for the development and lifecycle management of an application or piece of software.
- The methodology within the SDLC process can vary across industries and organizations, but standards such as ISO/IEC 12207 represent processes that establish a lifecycle for software, and provide a mode for the development, acquisition, and configuration of software systems.

# SDLC Phases

---

Requirements Collection/Gathering	Establish Customer Needs
Analysis	Model And Specify the requirements-“What”
Design	Model And Specify a Solution – “Why”
Implementation	Construct a Solution In Software
Testing	Validate the solution against the requirements
Maintenance	Repair defects and adapt the solution to the new requirements

# Requirement Gathering

---

- Features
- Usage scenarios
- Although requirements may be documented in written form, they may be incomplete, unambiguous, or even incorrect. Requirements will Change!
- Inadequately captured or expressed in the first place
- User and business needs change during the project
- Validation is needed throughout the software lifecycle, not only when the “final system” is delivered.
- Build constant feedback into the project plan
- Plan for change
- Early prototyping [e.g., UI] can help clarify the requirements
- Functional and Non-Functional



# Requirement Gathering(Cont...)

---

Requirements definitions usually consist of natural language, supplemented by (UML) diagrams and tables.

Three types of problems can arise:

- **Lack of clarity:** It is hard to write documents that are both precise and easy-to-read.
- **Requirements confusion:** Functional and Non-functional requirements tend to be intertwined.
- **Requirements Amalgamation:** Several different requirements may be expressed together.



# Requirement Gathering(Cont...)

---

Types of Requirements:

- **Functional Requirements:** describe system services or functions.
  - Compute sales tax on a purchase
  - Update the database on the server
- **Non-Functional Requirements:** are constraints on the system or the development process.

Non-functional requirements may be more critical than functional requirements.

If these are not met, the system is useless!



# Analysis Phase

---

The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished.

This phase defines the problem that the customer is trying to solve.

The deliverable result at the end of this phase is a requirement document.

Ideally, this document states in a clear and precise fashion what is to be built.

This analysis represents the “**what**” phase.

The requirement documentaries to capture the requirements from the customer's perspective by defining goals.



# Analysis Phase

---

This phase starts with the requirement document delivered by the requirement phase and maps the requirements into architecture.

The architecture defines the components, their interfaces and behaviors.

The deliverable design document is the architecture.

This phase represents the “**how**” phase.

Details on computer programming languages and environments, machines, packages, application architecture, distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces, and many other engineering details are established.

The design may include the usage of existing components.



# Design Phase

---

- Implementation Plan
- Critical Priority Analysis
- Performance Analysis
- Test Plan
- The Design team can now expand upon the information established in the requirement document.
- Design Architecture Document
- The requirement document must guide this decision process.
- Analyzing the trade-offs of necessary complexity allows for many things to remain simple which, in turn, will eventually lead to a higher quality product.
- The architecture team also converts the typical scenarios into a test plan.



# Implementation Phase

---

- In the implementation phase, the team builds the components either from scratch or by composition.
- Given the architecture document from the design phase and the requirement document from the analysis phase, the team should build exactly what has been requested, though there is still room for innovation and flexibility.
- For example, a component may be narrowly designed for this particular system, or
- the component may be made more general to satisfy a reusability guideline.
  - Implementation - Code
  - Critical Error Removal
- The implementation phase deals with issues of quality, performance, baselines, libraries, and debugging.
- The end deliverable is the product itself. There are already many techniques associated with implementation.



# Testing Phase

---

Simply stated, quality is very important. Many companies have not learned that quality is important and deliver more claimed functionality but at a lower quality level.

It is much easier to explain to a customer why there is a missing feature than to explain to a customer why the product lacks quality.

A customer satisfied with the quality of a product will remain loyal and wait for new functionality in the next version.

Quality is a distinguishing attribute of a system indicating the degree of excellence.

Regression Testing

Internal Testing

Unit Testing Application Testing Stress Testing



# Testing Phase(Cont...)

---

The testing phase is a separate phase which is performed by a different team after the implementation is completed.

There is merit in this approach; it is hard to see one's own mistakes, and a fresh eye can discover obvious errors much faster than the person who has read and re-read the material many times.

Unfortunately, delegating (alternate) testing to another team leads to a lack (dull) attitude regarding quality by the implementation team.

If the teams are to be known as craftsmen, then the teams should be responsible for establishing high quality across all phases.

An attitude change must take place to guarantee quality. Regardless if testing is done after-the-fact or continuously, testing is usually based on a regression technique split into several major focuses, namely internal, unit, application, and ~~external~~.



# Maintenance Phase

---

Software maintenance is one of the activities in software engineering, and is the process of enhancing and optimizing deployed software (software release), as well as fixing defects.

Software maintenance is also one of the phases in the System Development Life Cycle (SDLC), as it applies to software development. The maintenance phase is the phase which comes after deployment of the software into the field.

The developing organization or team will have some mechanism to document and track defects and deficiencies.

configuration and version management reengineering (redesigning and refactoring)

updating all analysis, design and user documentation

Repeatable, automated tests enable evolution and refactoring



# Maintenance Phase(Cont...)

---

Maintenance is the process of changing a system after it has been deployed.

- **Corrective maintenance:** identifying and repairing defects
- **Adaptive maintenance:** adapting the existing solution to the new platforms.
- **Perfective Maintenance:** implementing the new requirements

In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered “maintenance”!

Software just like most other products is typically released with a known set of defects and deficiencies.

The software is released with the issues because the development organization decides the utility and value of the software at a particular level of quality outweighs the impact of the known defects and deficiencies.



# DFD

---

DFD- Data Flow Diagrams

Graphical representation of flow of data inside application.

Used for visualization and data processing.

DFD elements are:

- External Entity
- Process
- Data Flow
- Data Store



### **1) External entity:**

Can be user or external system that performs some process or activity in project Symbolized with rectangle.

If we have entity 'admin' then symbol will be

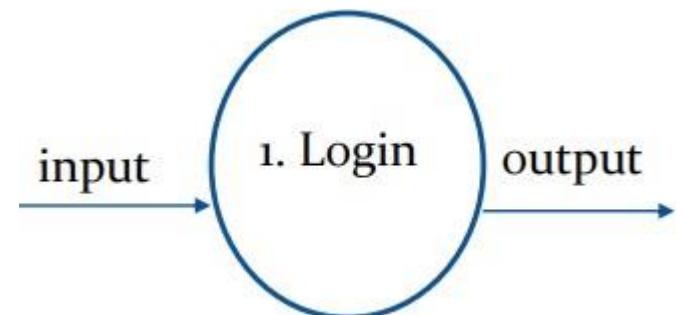
Admin

### **2) Process:**

Work or action taken on incoming data to produce output

Each process must have input and output

Symbolized as



### 3) Data Flow

Can be used to show input and output of data

Should be named uniquely and don't include word 'data'

Names can be 'payment', 'order', 'complaint' etc.

Symbolized as



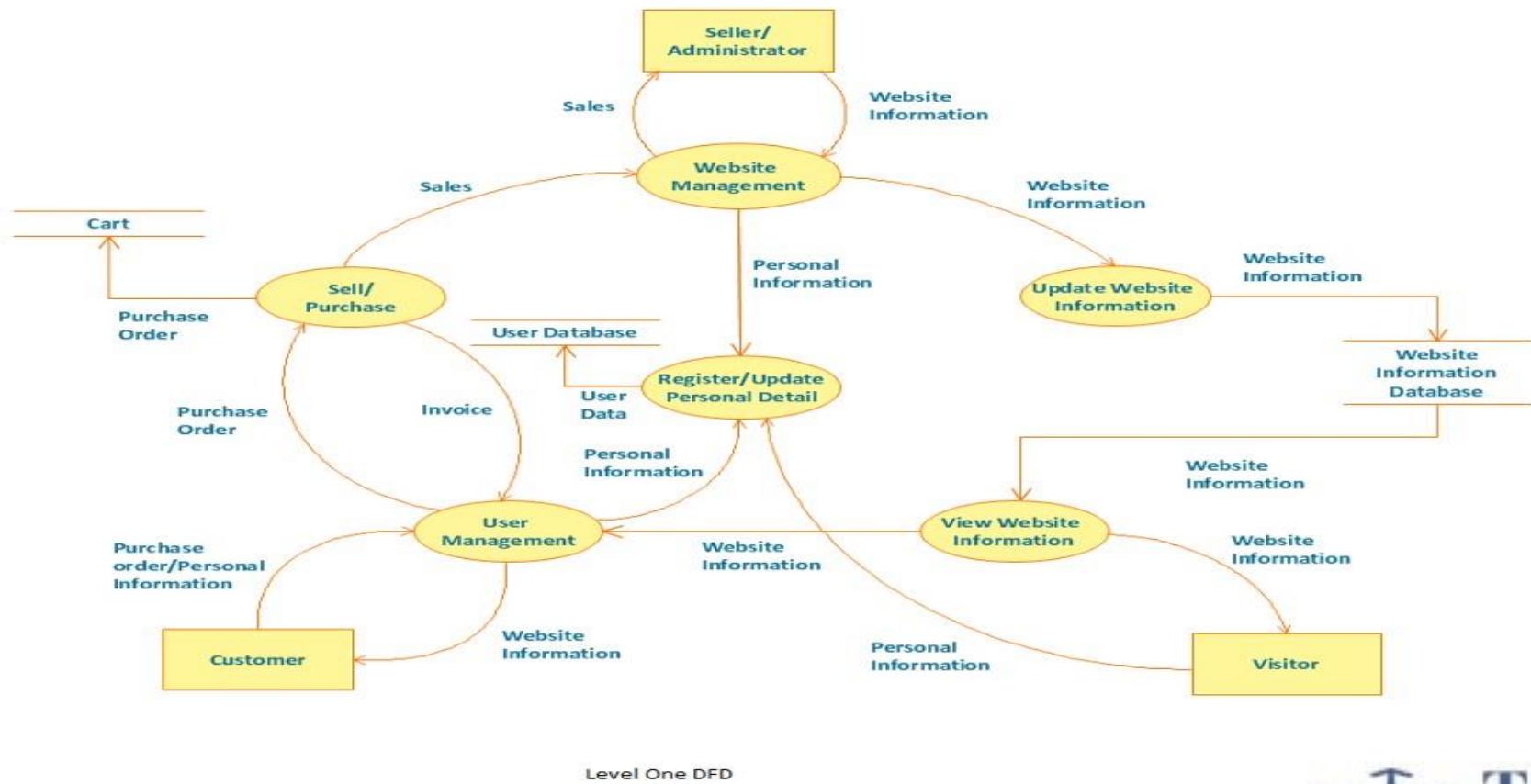
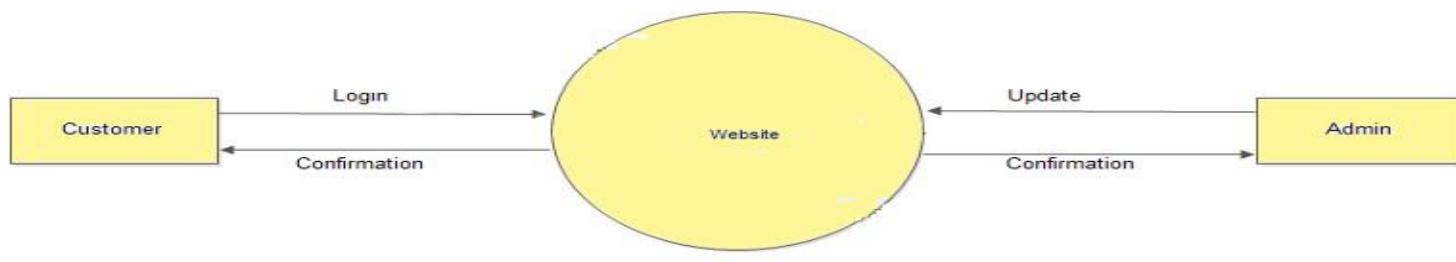
### 4) Data Store

Can be used to show database tables Only process  
may connect data stores

There can be two or more process sharing same data  
store

Symbolized as





# DFD RULES

---

## **1. Rule 1 :**

Each process must have data flowing into it and coming out from it

## **2. Rule 2:**

Each data store must have data going inside and data coming outside

## **3. Rule 3:**

A data flow out of a process should have some relevance to one or more of the data flows into a process

#### **4. Rule 4:**

Data stored in system must go through a process.

#### **5. Rule 5:**

Two data stores can't communicate with each other unless process is involved in between .

#### **6. Rule 6**

The Process in DFD must be linked to either another process or a data store A process can't be exist by itself, unconnected to rest of the system

## **CONTEXT LEVEL (DFD level-0: )**

It's also context level DFD

Context level diagrams show all external entities. They do not show any data stores.

The context diagram always has only one process labelled 0

## **DFD Level-1(or 2)**

Include all entities and data stores that are directly connected by data flow to the one process you are breaking down

show all other data stores that are shared by the processes in this breakdown

Like login process will linked to users & database in further levelling.



# Flow Chart

---

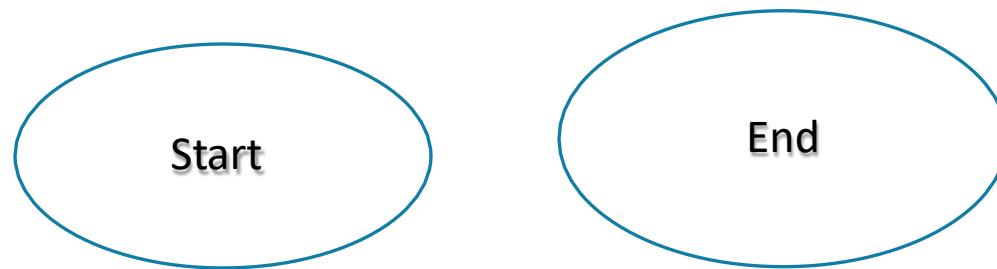
- Used to show algorithm or process . Can give step solution to the problem
- The first flow chart was made by John Von Newman in 1945
- Pictorial view of process
- Flowcharts are generally drawn in the early stages of formulating computer solutions.
- Flowcharts facilitate communication between programmers and business people.
- These flowcharts play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems.
- Once the flowchart is drawn, it becomes easy to write the program in any high level language.
- Often we see how flowcharts are helpful in explaining the program to others. Hence, it is correct to say that a flowchart is a must for the better documentation of a complex program. by step

# Flowchart Symbol

## 1) Start Or End:

Show starting or ending of any flow chart

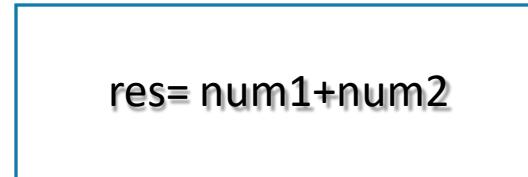
Symbolized as:



## 2. Process:

Defines a process like defining variables or initializing variable or performing any computation

Symbolized as

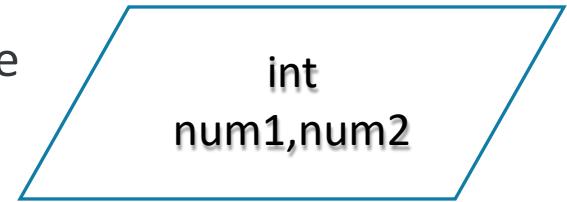


### 3) Input or Output

Used when user have to get or initialize any variable

Like get num1 and num2 from user

Symbolized as

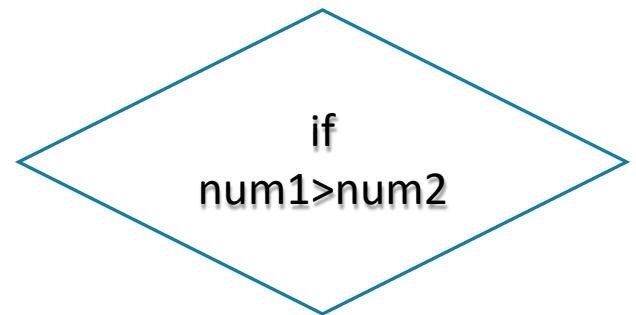


### 4) Decision Making

For checking condition this symbols can be used

Like if num1 is greater than num2

Can be symbolized as



#### 4) Flow lines

Lines showing flow of data and process

Showing flow of instructions also

Can be symbolized as



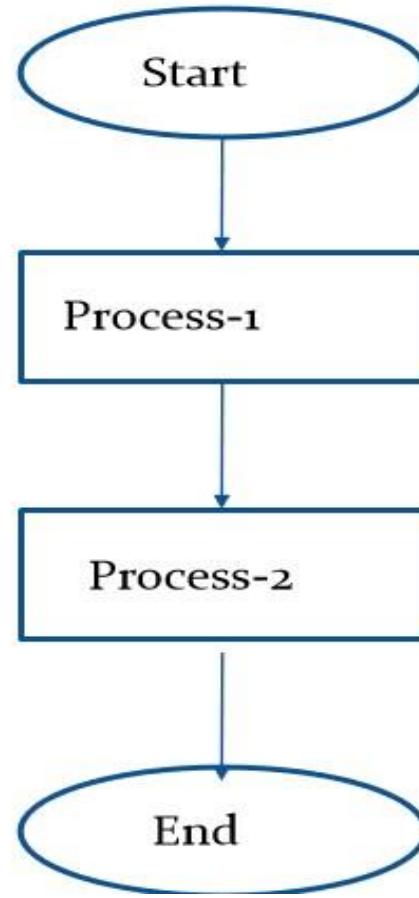
### Programs can be in three format

1. Linear or sequence
2. Branching
3. Looping

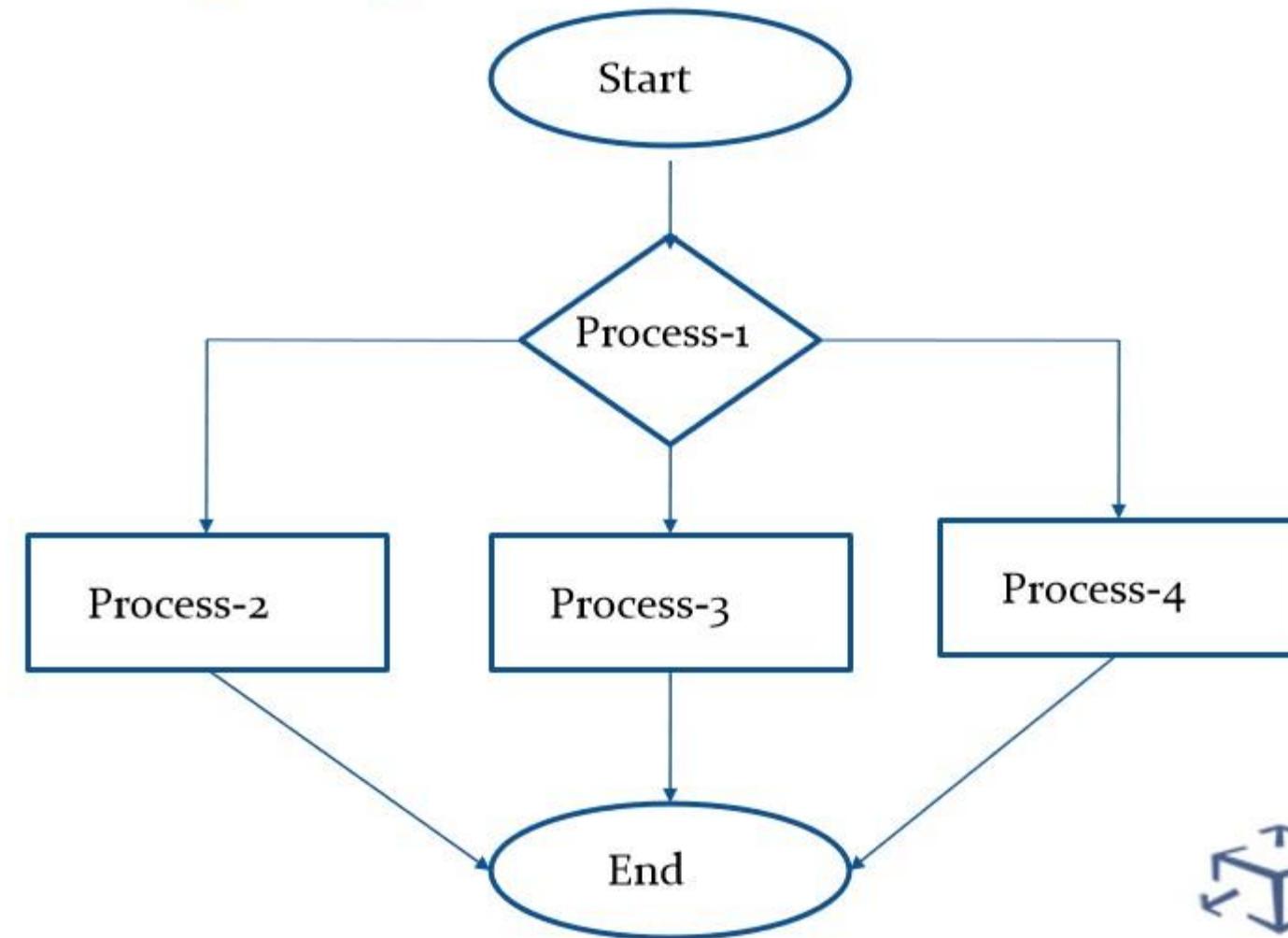
Following are notations can be used to show this format



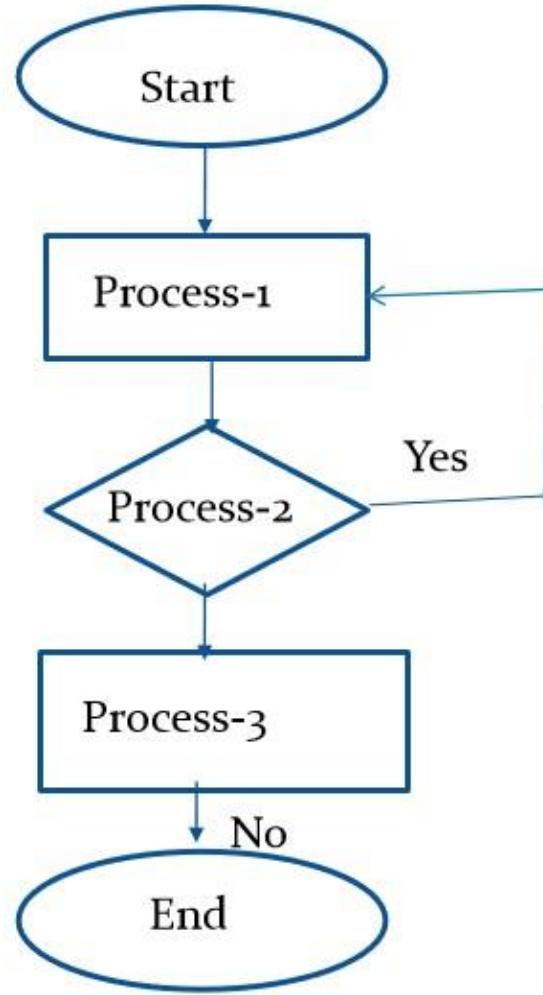
# Linear Program Structure



# Branching Program Structure..

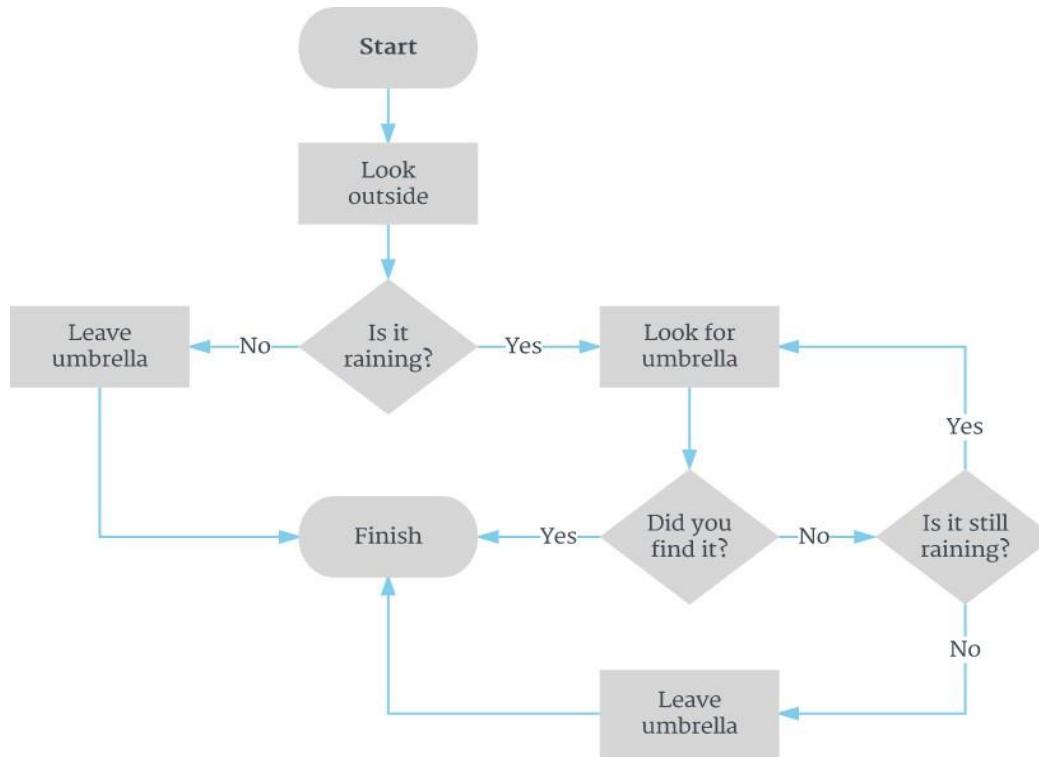


# Looping Program Structure



# Flow Chart For Taking Umbrella

---



# CLASS DIAGRAM

---

The class diagrams are a neat way of visualizing the classes in our system before we actually start coding them up.

They're a static representation of the system structure.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system.

The class diagrams are widely used in the modelling of object oriented systems. They are the only UML diagrams, which can be mapped directly with object-oriented languages.



# NEED OF CLASS DIAGRAM

---

Planning and modelling ahead of time make programming much easier.

Besides that, making changes to class diagrams is easy, whereas coding different functionality after the fact is kind of annoying.

When someone wants to build a house, they don't just grab a hammer and get to work. They need to have a blueprint — a design plan — so they can ANALYZE & modify their system.

You don't need much technical/language-specific knowledge to understand it.



# CLASS DIAGRAM NOTATIONS:

---

## 1). Class:

Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

Class is represented with rectangles with compartments.  
Class name is placed in the first compartment (centred, bolded, and capitalized).

Attributes is placed in the second compartment(left-aligned, not bolded, and lowercase)

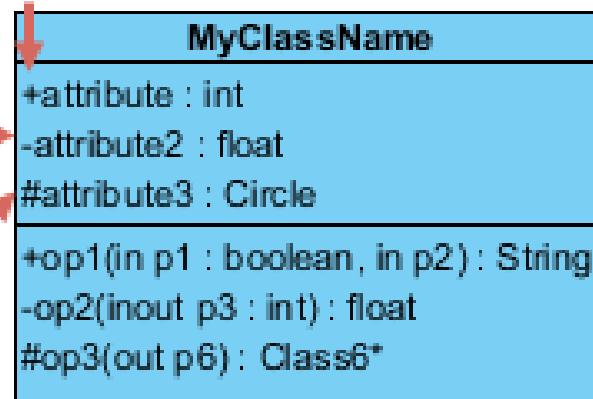
The bottom one lists the class's operations, which represents the behaviour of the class.

<b>Class Name</b>
+ Attribute Name: type - Attribute Name: type # Attribute Name: type
+Operation 1(arg):return +Operation 2(arg):return
+Operation 3(arg):return

## 2). Visibility of class Members

<b>public</b>	+	anywhere in the program and may be called by any object within the system
<b>private</b>	-	the class that defines it
<b>protected</b>	#	(a) the class that defines it or (b) a subclass of that class
<b>package</b>	~	instances of other classes within the same package

### Public Attribute

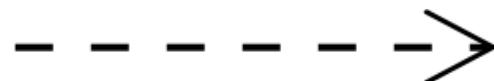


Private Attribute — — ►

Protected Attributes ↗

### 3). Relationship:

#### a). Dependency:



➤ Relation between two or more classes in which a change in one may force changes in the other.

➤ It will always create a weaker relationship

➤ Dependency indicates that one class depends on another.



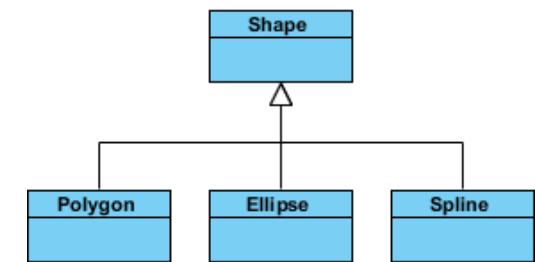
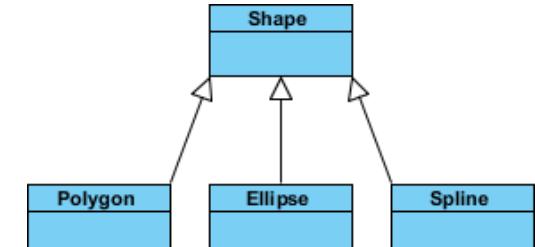
#### b). Generalization(Inheritance):



➤ It refers to a relationship between two classes  
class is a specialized version of another

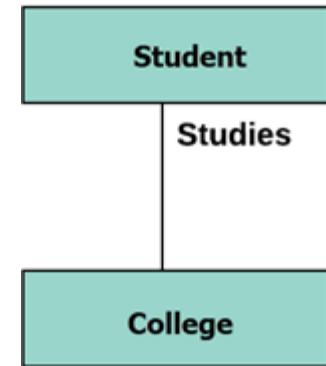
➤ Represents is-a relationship. Example "Honda  
is a type of car."

So the class Honda would have a generalization  
relationship with the class car.

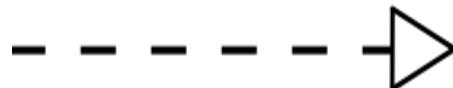


c). Association: \_\_\_\_\_ or \_\_\_\_\_

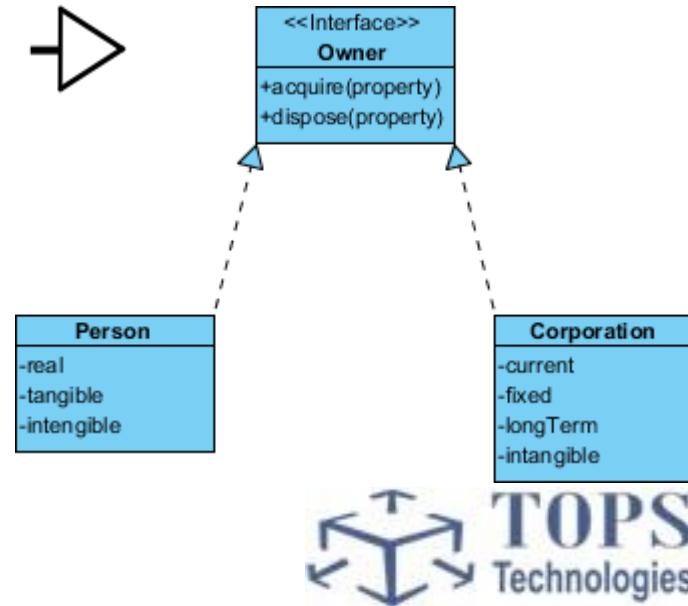
- This type of relationship represents static relationships between classes A and B.
- For example; an employee works for an organization.



d). Realization/Implementation



- Relationship between two class elements, in which one class element implements/executes the behaviour that the other model element specifies.



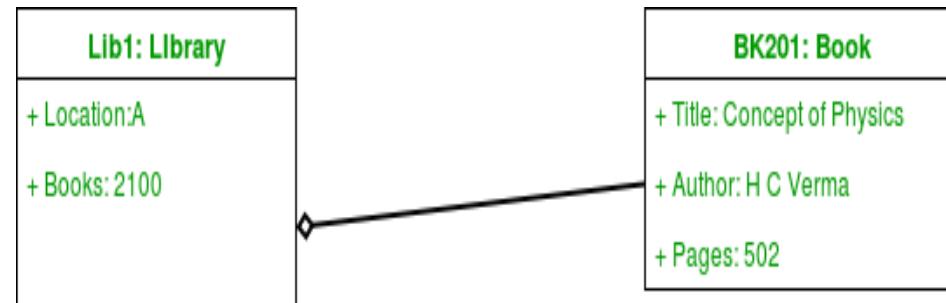
### e) Aggregation



Aggregation is a special type of association that models a whole- part relationship between aggregate and its parts.

Describes has-a relationship.

Child can exist without the parent.



### f) Composition:



Composition is a type of association where the child cannot exist independent of the other.

Child will never exist independent of a parent

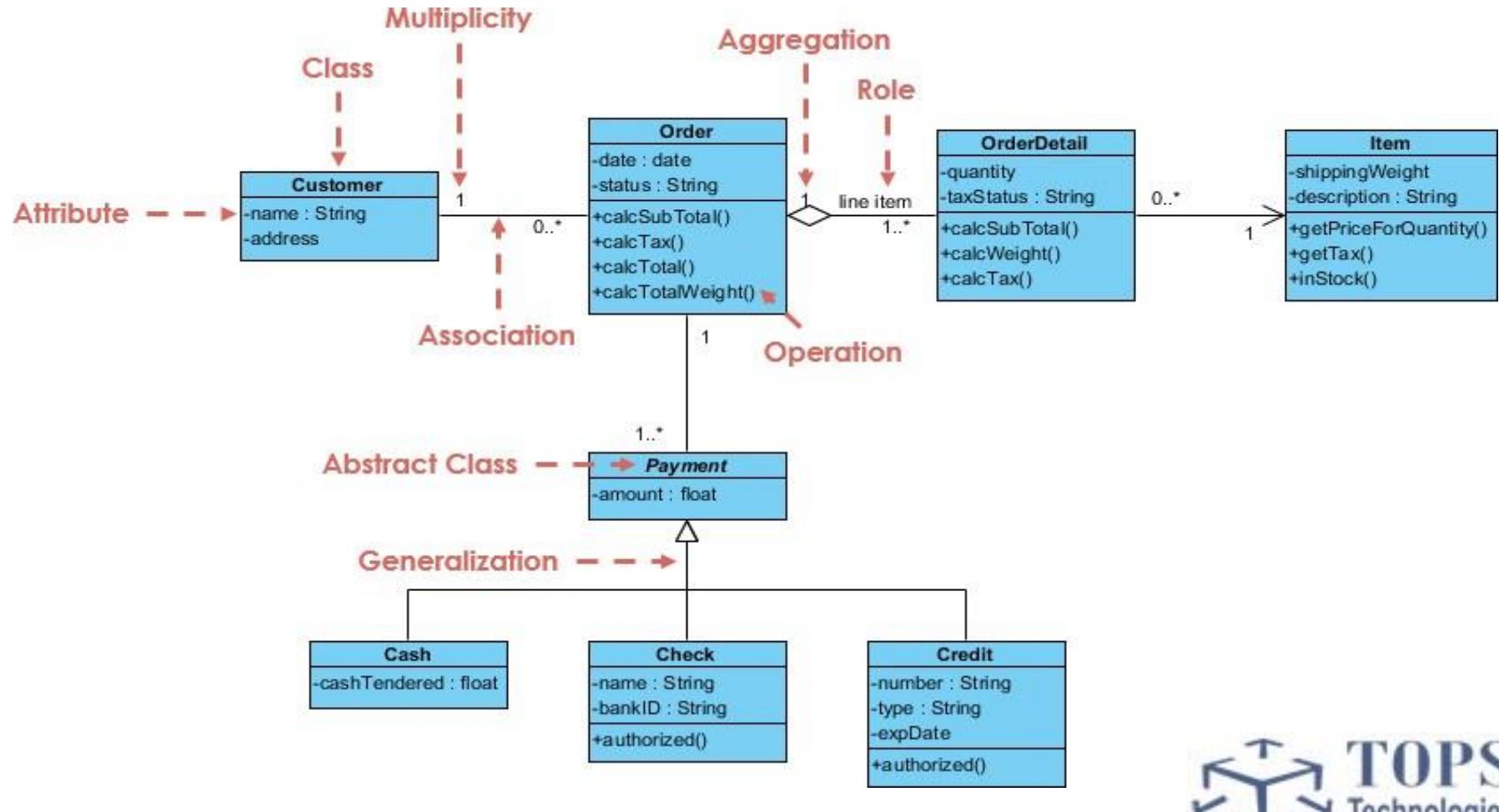


## Multiplicity:

- Represent cardinality between associated entities
- It specifies how many instances of attributes are created when a class is initialized. If a multiplicity is not specified, by default one is considered as a default multiplicity.
- Let's say that there are 100 students in one college. The college can have multiple students.

NOTATION	MEANING
0..1	Zero or one
1	One only
0..*	Zero or more
*	Zero or more
1..*	One or more
7	Seven only
0..2	Zero or two
4..7	Four to seven

# Class Diagram for Payment System



# Model Design With UML

---

Unified Modelling Language (UML) is a general purpose modelling language.

The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

UML is not a programming language, it is rather a visual language. We use UML diagrams to portray the behaviour and structure of a system.



# Model

---

Model is a simplification of reality,. A model may provide

- Blueprint of a system
- Organization of the system
- Dynamic of the system

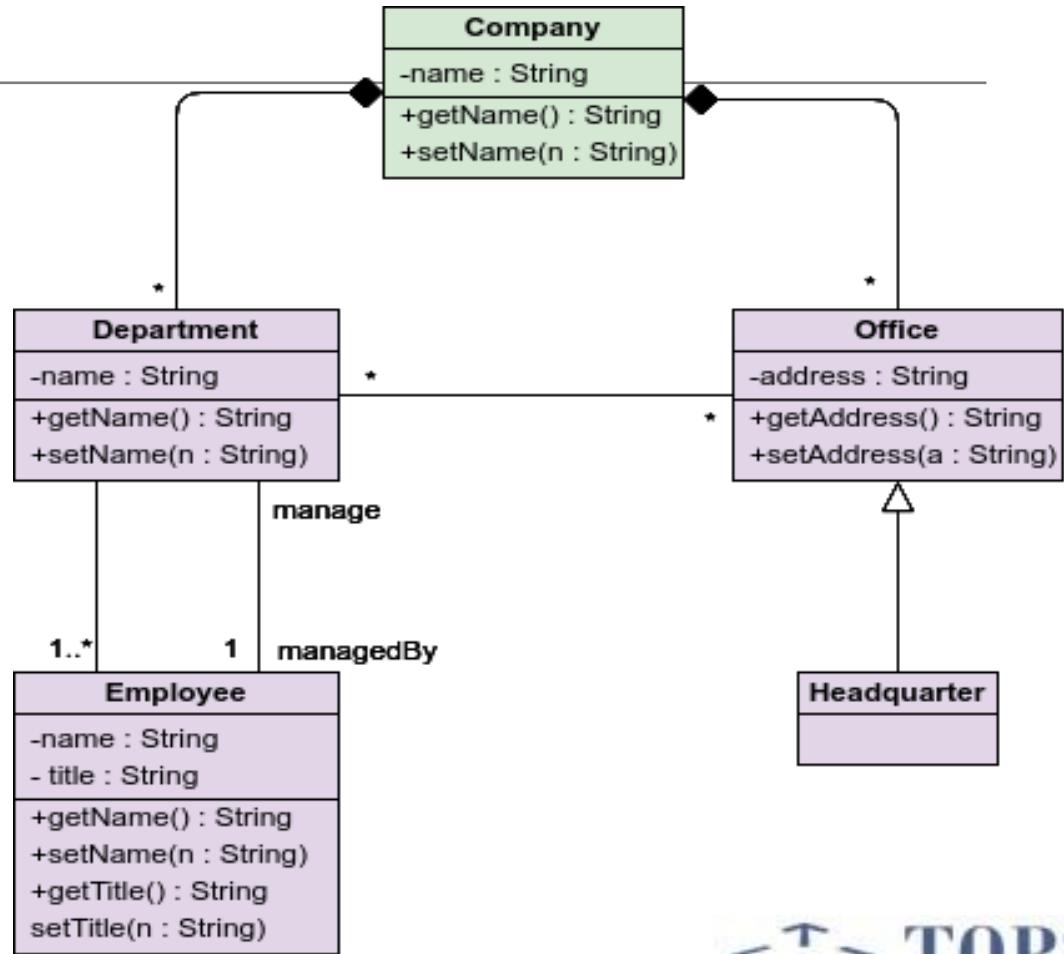
**Three kinds of models describe a system from different viewpoints:**

1. Class Model
2. State Model
3. Interaction Model

A complete description of a system requires models from all 3 viewpoints

# Class Model

- The class model describes the static structure of the objects in a system and their relationships.
- The class model contains class diagrams
- A class diagram is a graph whose nodes are classes and whose arcs are relationships among classes.



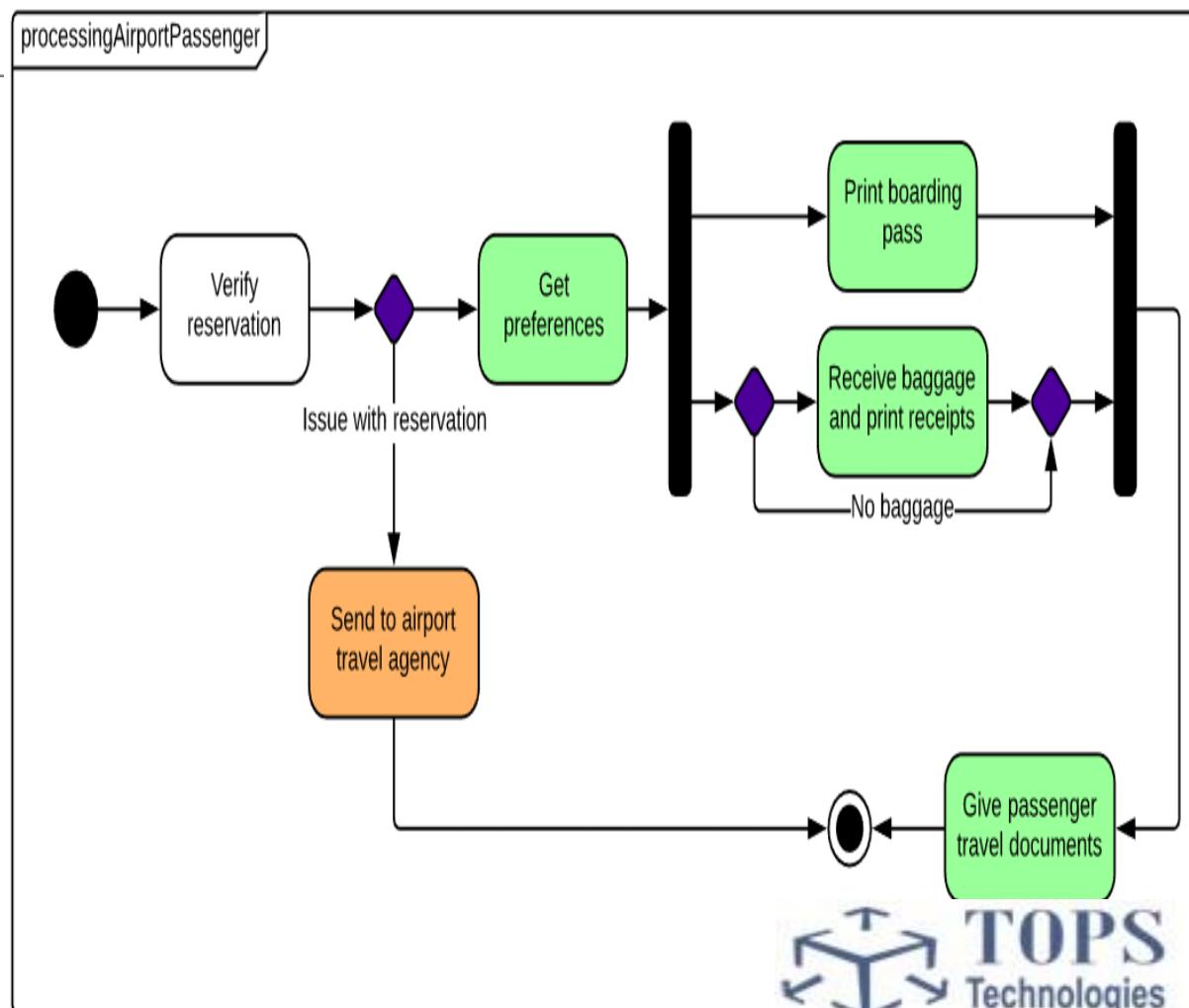
# State Model

The state model describes the aspects of an object that change over time.

State diagrams express the state model.

The state diagram is a graph whose nodes are states and whose arcs are transitions between states caused by events.

State diagram for an airport check-in is shown in figure.

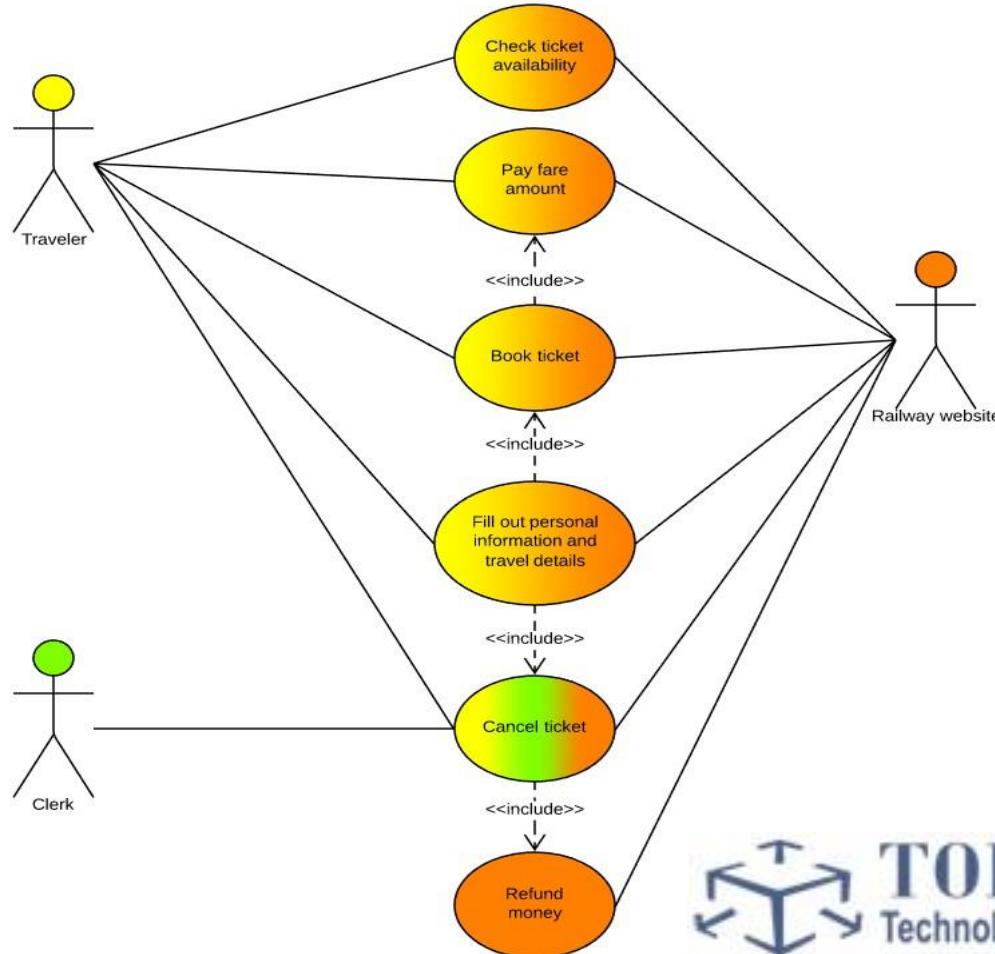


# Interaction Model

The interaction model describes how the objects in a system cooperate to achieve broader results.

The interaction model starts with use case that are then elaborated with sequence and activity diagrams.

A use case focuses on the functionality of a system i.e, what a system does for users.



# Class Modelling

---

Class Modelling is the task of specifying classes using a specific language in which the properties are called **attributes**, the relations are called **associations** and behaviour is defined as **operations**.

UML class modelling is platform independent, so it is not about Java, nor C#. At some point classes are transformed to a platform specific technology, which may be Java classes, Enterprise Java Beans, ADF Business Components and so on.

Class Diagram is used for class modelling. Class Modelling describes:

- Objects and class concepts
- Link and association concept
- Generalization and Inheritance

# Data Dictionary

---

Data dictionary is referred to as meta-data. Meaning, it is a repository of data about data.

It is a repository that contains description of all data objects consumed or produced by the system .

These data objects are often stored in a spreadsheet, word processing tool or even a purpose built meta-data repository.

A data dictionary can be created that defines data elements, including details such as names, aliases descriptions and allowable values, and including whether multiple values are permissible.

.

# Example

---

Field Name	Data Type	Constraint	Description
pID	int	PK	Its unquied product id
pName	Varchar(20)	NotNull	Its product name
pImage	Varchar(20)	NotNull	Its name of image
pQty	Int	NotNull	Its product Quantity



# Module 2 Topics:

- **About Websites**
  - Website Functionality
  - Web Programming
- **HTML**
  - Structure
  - HTML Tags
- **CSS**
  - Types of CSS
  - Property of CSS3
  - CSS Property



# What is Website

---

A website is a collection of publicly accessible, interlinked Web pages that share a single domain name.

Websites provide information from anywhere in the world.

Follows some rules & regulations i.e. client-server architecture standard.

WWW – world wide web

WWW commonly known as the Web, is an information system where documents and other web resources are identified by Uniform Resource Locators (URLs, such as <https://www.google.com/>), which may be interlinked by hypertext, and are accessible over the Internet.



# Protocols

---

1. A user uses his or her Web browser to initiate a request for a web server resource.
2. HTTP is used to send a GET request to the web server.
3. The web server processes the GET request on the server (typically locating the requested code and running it).
4. The Web server then sends a response back to the Web browser. The HTTP protocol is used to send the HTTP response back to the Web browser.
5. The user's web browser then processes the response (typically HTML and JavaScript) and renders the web page for display to the user.
6. The user may then enter data and perform an action such as clicking a submit button that causes his or her data to be sent back to the Web server for processing.



7. HTTP is used to POST the data back to the Web server.
8. The Web server then processes the POST request (again, calling your code in the process).
9. The Web server then sends a response back to the Web browser. HTTP is used to send the HTTP response to the Web browser.
10. The Web browser again processes the response and displays the Web page to the user.

This process is repeated over and over during a typical Web application session.

HTTP messages are typically sent between the Web server and Web browser using port 80, or port 443 when using Secure HTTP (HTTPS).



# The Web Browser's Role

---

- The Web browser provides a platform-independent means of displaying Web pages that were written in HTML.
- HTML was designed to be able to render information on any operating system while placing no constraint on the window size .This is why Web pages are considered platform independent.
- The Web browser also displays images and responds to hyperlinks to other pages.
- Many new client-side technologies enable today's Web browsers to execute code such as JavaScript and to support plug-ins. Technologies such as Asynchronous JavaScript and XML (AJAX) and Microsoft Silver light allow Web browsers to communicate with Web servers without clearing the existing Web page from the browser window. These technologies make the user experience more dynamic and interactive

# The Web Server's Role

---

- The first Web servers were responsible for receiving and processing simple user requests from browsers via HTTP.
- The Web server handled its request and sent a response back to the Web browser.
- The Web server then closed any connection between it and the browser and released all resources that were involved with the request. These resources were easy to release as the Web server was finished processing the request.
- This type of Web application was considered to be stateless because no data was held by the Web server between requests and no connection was left open.
- These applications typically involved simple HTML pages and were therefore able to handle thousands of similar requests per minute.

# Understanding the Web Page and Home Page

---

## **Web Page**

Web site contain the many pages that's called Web Page

Web page contain the information related to our business that user can read easily and from that he can get best experience.

## **Home Page**

When we open any kind of web site that show the first page and it contain all web pages links and it also show the primary contain in a page



# Web Designing

---

A web designer, spend a lot of time creating pages and tweaking them until they look good in the browser.

The web pages may look different to different people when opened in different browser.

The way your site looks and performs is at the mercy of a number of variables such as browser version, platform, monitor size, and the preferences or special needs of each individual user.

Your page may also be viewed on a mobile device like a cell phone, or using an assistive device like a screen magnifier or a screen reader.

Having a solid understanding of the web environment allows you to anticipate and plan for these shifting variables.



# 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.



## What is an HTML File?

An HTML file is a text file containing small markup tags

The markup tags tell the Web browser how to display the page An  
HTML file must have an htm or html file extension

An HTML file can be created using a simple text editor

### Creating a simple web page:

If you are running Windows, start Notepad. IF you are on a Mac, start Simple Text. Type in the following text and save the file as “mypage.html”.

```
<html>
<head>
<title>Title of Page</title>
</head>
<body>
This is my first homepage.<b> This text is bold</b>
</body>
</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 <b> and </b>

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**, <b> means the same as <B>



The browser interprets the HTML tags in the source code and displays your content according to those tags.

**Here's a brief explanation of each HTML tags...**

**<html>** Identifies the language used (file type) for the browser, in this case, a web page written in HTML language.

**<head>** Acts as a container for the page title and meta data. The title is necessary, but meta data is not. You can learn more about meta data later, I don't want to clutter this up with too much information.

**<title>** The title is the name of the page. The title shows up in the title bar at the top of your browser, as the text for bookmarks unless it's changed, and in some search engines as the link text



`</title>` Closes the title. The forward slash ( / ) in front of the HTML element means that command is now canceled.

`</head>` Closes the head section.

`<body>` Between the opening and closing body tags is where you place the actual content you want displayed to the public.

`</body>` Closes the body section

`</html>` Closes the html element, end of page.

comment: `<!--This can be viewed in the HTML part of a document-->`

bold: `<b>Example</b>`

big (text): `<big>Example</big>`



# BASIC HTML TAG

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

# Refer this example:

---

**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/horizontalLine.html>



# ANCHOR TAG

---

- The `<a>` tag defines a hyperlink, which is used to link from one page to another.
- The most important attribute of the `<a>` element is the **href** attribute, which indicates the link's destination.

`<a href="detination">...</a>`

Here is an example that creates a link to the O'Reilly Media web site:

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

**Make Image link:**

`<a href="http://www.oreilly.com"></a>`



# Images - The img Element

The `<img>` tag defines an image in an HTML page.

The `<img>` tag has two required attributes: `src` and `alt`.

`src`: specifies the source of image

`alt`: specifies an alternate text for an image

**Refer This Example:**

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



# Images - The img Element

**Providing width and height dimensions :**

`width="number"` Image width in pixels

`height="number"` Image height in pixels

**Refer This Example:**

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

# Image Map

The `<map>` tag is used to define a client-side image-map. An image-map is an image with clickable areas.

The required name attribute of the `<map>` element is associated with the `<img>`'s `usemap` attribute and creates a relationship between the image and the map.

The `<map>` element contains a number of `<area>` elements, that defines the clickable areas in the image map.

**Refer This Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/HTML/imagemap.html>

# HTML List

- 
1. Ordered List
  2. Unordered List
  3. Nested List
  4. Definition List

**Ordered List    Unordered list**

- |             |          |
|-------------|----------|
| 1. Item 1   | • Item 1 |
| 2. Item 2   | • Item 2 |
| 3. Item 3   | • Item 3 |
| <br>        |          |
| I. Item 1   | ▪ Item 1 |
| II. Item 2  | ▪ Item 2 |
| III. Item 3 | ▪ Item 3 |

## Nested List

- |           |
|-----------|
| 1. Item 1 |
| 2. Item 2 |
| 3. Item 3 |
| <br>      |
| o Item 31 |
| o Item 32 |
| o Item 33 |
| 3. Item 3 |

Tag	Description
<ol>	Defines an ordered list
<ul>	Defines an unordered list
<li>	Defines a list item
<dl>	Defines a definition list
<dt>	Defines a definition term
<dd>	Defines a definition description

## Definition list

- Def 1 :** Explanation  
**Def 2 :** Explanation



# Refer this example:

---

**Ordered List:**

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

**Unordered List:**

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

**Definition List:**

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

**Nested List:**

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



# HTML Tables

## Table Tags

Tag	Description
<table>	Defines a table
<th>	Defines a table header
<tr>	Defines a table row
<td>	Defines a table cell
<caption>	Defines a table caption
<colgroup>	Defines group of table columns
<col>	Defines the attribute values of one or more columns in a table
<thead>	Defines a table head
<tbody>	Defines a table body
<tfoot>	Defines a table footer

# BASIC TABLE MARKUP

## Spanning Cells

Spanning is the stretching of a cell to cover several rows or columns.

Spanning is done using **colspan** or **rowspan** attributes.

### Column spans

Column spans, created with the colspan attribute in the td or th element, stretch a cell to the right to span over the subsequent columns . Here a column span is used to make a header apply to two columns.

`colspan="2"`

Month	Savings
January	\$100
February	\$80
Sum:	\$180

### Row spans

Row spans, created with the rowspan attribute, work just like column spans, except they cause the cell to span downward over several rows.

Month	Savings	Savings for holiday!
January	\$100	
February	\$80	\$50



**Refer This Example:**

**Colspan:**

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

**Rowspan:**

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

<https://github.com/TopsCode/Software-Engineering/blob/master/HTML/Table/Exampe1.html>



# BASIC TABLE MARKUP

## Cell Padding and Spacing

**Cell padding** is the amount of space held between the contents of the cell and the cell border. If you don't specify any cell padding, the cells will have the default value of one pixel of padding.

```
<table cellpadding="15">
```

Cell padding adds space between the edge of the cell and its contents.

CELL 1	CELL 2
CELL 3	CELL 4

**Cell spacing** is the amount of space held between cells, specified in number of pixels. If you don't specify anything, the browser will use the default value of two pixels of space between cells.

```
<table cellpadding="15" cellspacing="15">
```

CELL 1	CELL 2
CELL 3	CELL 4

## Refer This Example:

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

# BASIC TABLE MARKUP

## Captions

The caption element is used to give a table a title or brief description. The caption element must be the first thing within the table element, as shown in this example that adds a caption to the nutritional chart from earlier.

**Refer This Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/HTML/Table/caption.html>



# HTML FRAME

---

**Refer This Example:**

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



# HTML Forms And Inputs

---

HTML Forms are used to select different kinds of user input.

- A form is an area that can contain form elements.
- Form elements are elements that allow the user to enter information (like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.) in a form.
- A form is defined with the <form> tag.

<form>

<input>

<input>

</form>

**Input:**

- The most used form tag is the <input> tag. The type of input is specified with the type attribute.



# HTML FORM TAG

Tag	Description
<form>	Defines a form for user input
<input>	Defines an input field
<textarea>	Defines a text-area (a multi-line text input control)
<label>	Defines a label to a control
<fieldset>	Defines a fieldset
<legend>	Defines a caption for a fieldset
<select>	Defines a selectable list (a drop-down box)
<option>	Defines an option in the drop-down box
<button>	Defines a push button



# HTML Forms And Inputs

## Text Fields

- Text fields are used when you want the user to type letters, numbers, etc. in a form. In most browsers, the width of the text field is 20 characters by default.

## Radio Buttons

- Radio Buttons are used when you want the user to select one of a limited number of choices. One option can be chosen.

## Checkboxes

- Checkboxes are used when you want the user to select one or more options of a limited number of choices.

## Submit:

- When the user clicks on the "Submit" button, the content of the form is sent to another file. The form's action attribute defines the name of the file to send the content to. The file defined in the action attribute usually does something with the received input.



# Refer This Example

---

**Text:**

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

**Radio Button:**

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

**Check Box:**

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

# Refer This Example:

---

## Text Area:

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

## Select Example:

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

## Reset And Submit:

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



# Design Customization with Div and Span

## Div Tag :

The `<div>` tag defines a division or a section in an HTML document.

The `<div>` element is often used as a container for other HTML elements to style them with CSS or to perform certain tasks with JavaScript.

**Refer This Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/HTML/div/div1.html>



# Design Customization with Div and Span

## Span Tag:

The HTML `<span>` tag is used for grouping and applying styles to inline elements.

There is a difference between the span tag and the div tag. The span tag is used with inline elements whilst the div tag is used with block-level content.

**Refer This Example** <https://github.com/TopsCode/Software-Engineering/blob/master/HTML/div/span.html>



```
table {  
border-collapse: collapse;  
border:  
td  
th
```



# CSS- Cascading Style Sheet

---

- Styles define how to display HTML elements
- Styles are normally stored in Style Sheets
- Styles were added to HTML 4.0 to solve a problem
- External Style Sheets can save you a lot of work
- External Style Sheets are stored in CSS files
- Multiple style definitions will cascade into one



# CSS Fundamental

- CSS is a mark-up language that describes how web pages should be displayed.
- In a typical web page, HTML is used to define the structure of the page. It creates headers, paragraphs, lists, tables, links, divisions, and more.
- A collection of CSS is normally contained in a stylesheet which is then applied to the HTML page.
- Different CSS stylesheet can be applied to different devices. This enables you to change the display for printers or mobile devices.

## Separation of presentation and structure

- CSS define the appearance of a web page separately from the structure of the document (the HTML). This makes it much easier to update your design site-wide, since all the design information is in a few files.

## The CSS Specifications

- CSS was created and maintained by the [World Wide Web Consortium](#) (W3C).



# Three Types of CSS

1. **External CSS**- In a separate CSS file
  2. **Inline CSS** – At the top of a web page document. In the <head> part
  3. **Internal CSS** – As a attribute in the tag it decorates
- Even multiple external style sheets can be referenced inside a single HTML document.

## Applying CSS inline

To apply a CSS property to an element inline, simply include a style attribute inside the HTML tag:

```
<p style="color: black; font-family: Georgia;">This is the paragraph</p>
```



## Applying CSS in the document <head>

The second method of applying CSS to an HTML document is to include it in the <head> of the document, like so:

```
<head>
<title>This is the title</title>
<meta http-equiv ... charset />
<style type="text/css">
body {
background-color: white;
color: black;
font-family: Georgia;
}
</style>
</head>
```



## **Applying CSS in a separate linked stylesheet**

A third method of including CSS in your page is to put your CSS in a separate file and reference it from the HTML.

A CSS file is a simple text file with a .css extension (just as an HTML file is a simple text file with a .html extension).

You can reference it from your HTML:

**Using the HTML link tag:**

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



# Apply CSS

---

## Parts of a Style Rule :

A CSS rule has the equivalents of the subject, object, verb, and adjective that you would find in any English sentence:

The **subject** (what we are describing) is the **selector**.

The **object** (what is being described about the subject) is the **property**.

The **verb** (always the verb “to be” in CSS) is represented by a **colon**.

The **adjective** (the description) is the **value**



# Apply CSS

## What CSS rule looks like

A CSS rule contains three basic components:

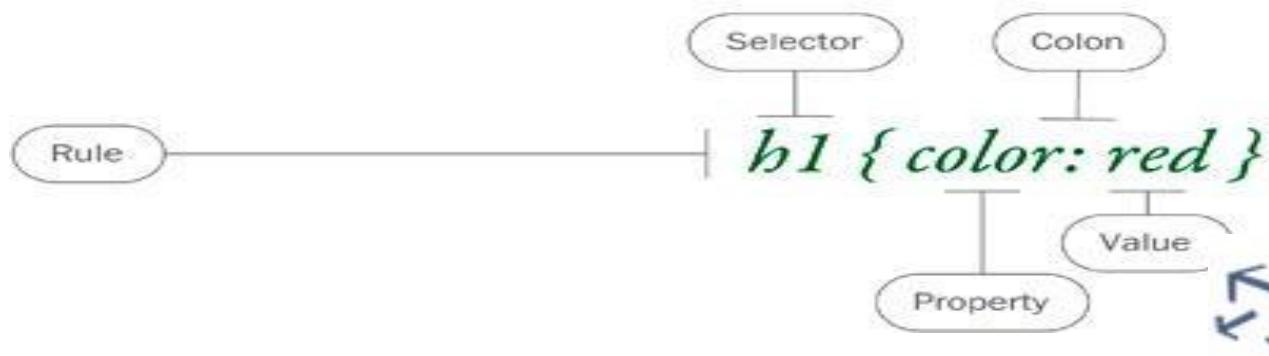
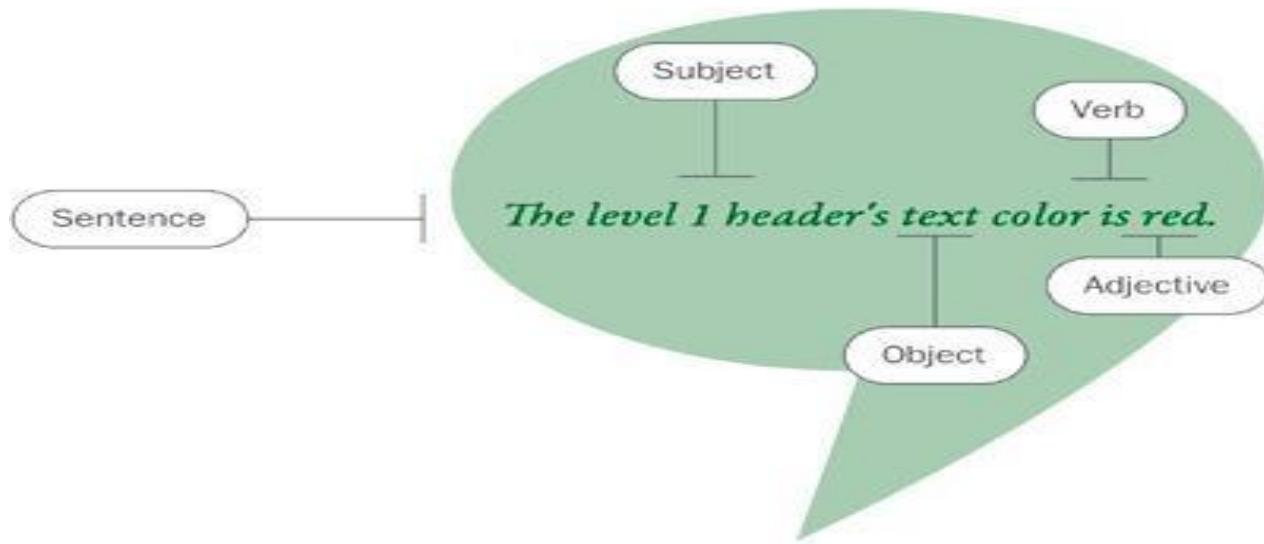
- 1.A selector, which chooses HTML elements to style, and
- 2.Some properties, which describe which aspect of the element's appearance to change (e.g. colour, font, margin etc.)
- 3.Values for those properties

A typical CSS declaration might look like this: [Selector](#)

```
{  
property: value;  
another-property: value;  
}
```



## Parts of a Style Rule :



# CSS FOR PRESENTATION

- All the styles will "cascade" into a new "virtual" style sheet by the following rules, where number four has the highest priority:
  - Browser default [**Lowest Priority**]
  - External style sheet
  - Internal style sheet (inside the <head> tag)
  - Inline style (inside an HTML element) [**Highest Priority**]
- So, an inline style (inside an HTML element) has the highest priority, which means that it will override a style declared inside the <head> tag, in an external style sheet, or in a browser (a default value).



# CSS Syntax

The CSS syntax is made up of three parts:

- 1.a selector
- 2.a property and
- 3.a value

**selector {property: value}** Example:

**body {color: black}**

**p {font-family: "sans serif"}**

- If you wish to specify more than one property, you must separate each property with a semicolon. The example below shows how to define a center aligned paragraph, with a red text color:

**p {text-align:center;color:red}**



# CSS Boxes And CSS Selector

## Types of Selectors :

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



# CSS Boxes And CSS Selector

## Universal Selector

-The CSS universal selector (\*) matches elements of any type.

```
*{ color: red; }
```

**Syntax:**

```
*{ style properties }
```

**Refer This Example:**

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



# CSS Boxes And CSS Selector

## HTML Selector

-To set all the level 2 header's text color as red.

```
h2 { color: red; }
```

The styles will be applied to any content tagged with:

```
<h2>...</h2>
```

All header level 2 tags on the page will be red, unless you override its declarations with other declarations.

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



# CSS Boxes And CSS Selector

## Class Selector

- If you don't want all of your tags to appear exactly the same, you need a "free agent" selector that can be applied to any HTML tag. This is the class selector. When defining a class rule, you place a period immediately before the class name to let the browser know, "Hey, this is a class selector, not an HTML or ID selector":

```
.highlight { background-color: yellow;}
```

This says: The hilight class text background color is yellow.

- To apply this class (and thus its styles) to an HTML tag, add the class attribute to a tag with the class name in quotes. You can apply the same class to any HTML tag you choose, as many times as you want:

```
<h2 class="hilight">Chapter I...</h2>
```

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



# CSS Boxes And CSS Selector

- **Class Selector: Dependent Class**

A dependent class allows you to specify the styles a class should have if the class is applied to a particular HTML tag.

- This allows you to create a general style for a class, but then specify more styles for a specific HTML tag within that class.
- For example, you might set up a general class to make the background color yellow, and then set up a dependent class so that when that class is applied to a particular tag, the background color is green instead:

If the hilight class is used with a level 2 header, then its background color is green.

```
h2.hilight { background-color: green; }
```

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



# CSS Boxes And CSS Selector

## ID Selector

- The difference between class selector and ID selector is that you use a hash mark at the beginning, to declare it, rather than a period:

```
#title01 { color: green; }
```

To apply the ID (and thus its styles) to an HTML tag, add the ID attribute to the tag with the name of the ID you want to apply:

```
<div id="title01">Chapter I...</div>
```

Similar to the class selector, you do not add the hash mark with the ID name when it's in the HTML tag. The hash mark is only included when you are setting up the ID rule.

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



## ID Selector

So what's **the difference between a class and an ID?** It isn't so much in how these selectors work, but in what you use them for:

Identifying major page sections (for example, header, content, footer) Identifying unique content or modules (for example, search, navigation, ad) Identifying an element to be used with Javascript

### Refer This Example:

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

# CSS Boxes And CSS Selector

- **Class Selector: Mix and Match Classes**

As if being able to add a single class to an HTML tag wasn't enough, you can also add multiple classes to a single HTML tag, mixing and matching styles as needed. Simply list all of the classes you want applied to a particular HTML tag in the class attribute, separated by spaces:

```
<p class="hilight smallprint">I should...</p>
```

- When applied to an HTML tag, that tag picks up the styles of all of the classes applied to it.

**Refer This Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/CSS/multipleClassSelector.html>



# Rules for Naming class and id name

- Class or ID names could be “bob”, “3423\_jyt”, or “9-8-2009”, but you should always try to name them something meaningful according to what the class or ID is for, rather than what styles it applies.
- For example, I might call a class “redText” to hilight certain text
- . But what happens if later I want to make the hilighted text yellow? The yellow text is now created using a class called redText.
- A better choice would be to call the class something like “hilight” or “smallprint,” which describes what it is for and allows for different versions.

## Descendant Selector

- A descendant selector targets elements that are contained within another element.
- It is an example of a contextual selector, because it selects the element based on its context or relation to another element.
- Descendant selectors are indicated in a list separated by a character space. `li em { color: olive; }`
- Example showing how contextual selectors can be grouped in a comma-separated list.

`h1 em, h2 em, h3 em { color: red; }`

- It is also possible to nest descendant selectors several layers deep. `ol a em { font-variant: small-caps; }`

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



# Specificity in CSS

- If you have two (or more) conflicting CSS rules that point to the same element, there are some basic rules that a browser follows to determine which one is most specific and therefore wins out.
- If the selectors are the same then the latest one will always take precedence.
- For example, if you had:  
`p { color: red; } p { color: blue; }`

p elements would be colored **blue** because **that rule came last.**



# Font Formatting with CSS



# Font Formatting with CSS

## The Font Properties

- In CSS, fonts are specified using a little bundle of font-related properties for **typeface, size, weight, and font. style** .
- The nature of the Web makes specifying type tricky, if not downright frustrating, particularly if you have experience designing for print (or even formatting text in a word processing program). Because you have no way of knowing **which fonts are loaded on users' machines**, you can't be sure that everyone will see text in the font you've chosen.
- And because the **default font size varies by browser and user preferences**, you can't be sure how large or small the type will appear, as well



# Font Formatting with CSS

## The Font Properties

### 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.

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



# Font Formatting with CSS

Generic font families:

## 1. Serif

- Examples: [Times](#), [Times New Roman](#), [Georgia](#)
- Serif typefaces have decorative serifs, or slab-like appendages, on the ends of certain letter strokes.

## 2. 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 monitors.



# Font Formatting with CSS

## 3. Monospace

- Examples: [Courier](#), [Courier New](#), and [Andale Mono](#)
- In monospace (also called constant width) typefaces, all characters take up the same amount of space on a line.

## 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.

## 5. 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.

# Font Formatting with CSS

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



# Font Formatting with CSS

You can specify text in a several ways:

At a specific size using one of the CSS length :

```
h1 { font-size: 1.5em; }
```

When specifying a number of units, be sure the unit abbreviation immediately follows the number, with no extra character space in between:

INCORRECT `h1 { font-size: 1.5 em; } /*space before the em*/`

As a percentage value, sized up or down from the element's default or inherited font size:

```
h1 { font-size: 150%; }
```



# Refer This Example

---

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



# FLOATING AND POSITIONING



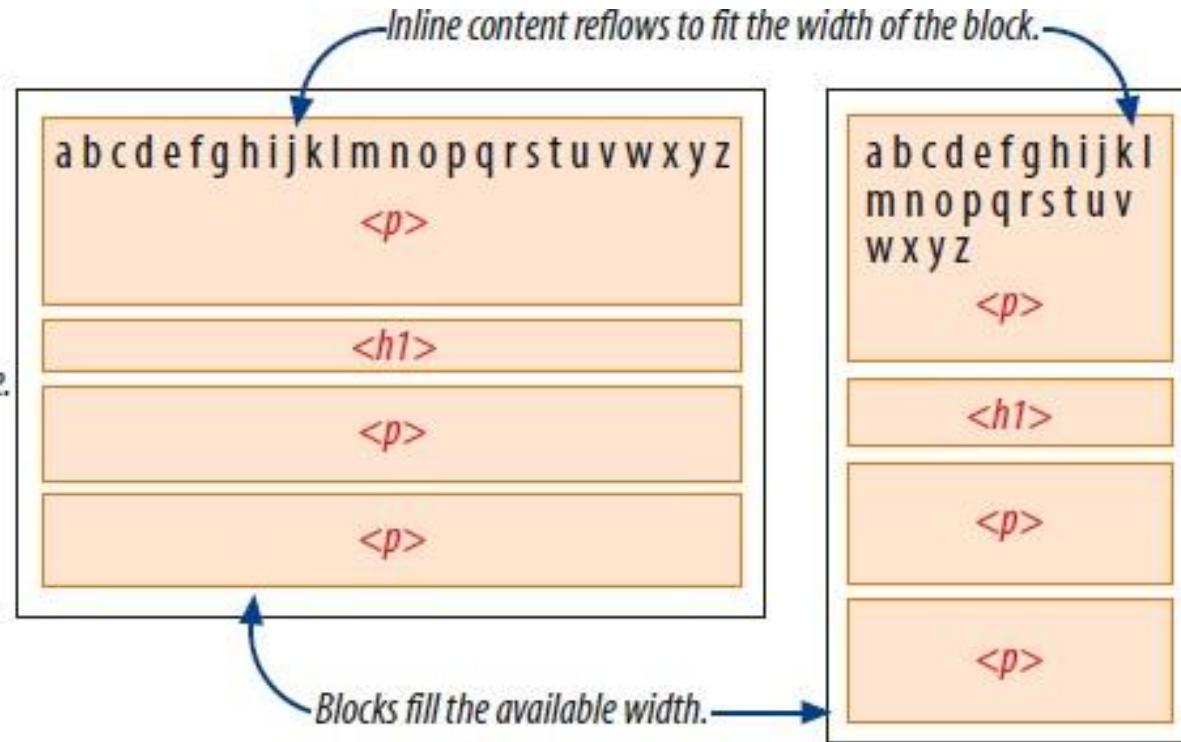
# FLOATING AND POSITIONING

- Floating an element moves it to the left or right, and allows the following text to wrap around it.
- Positioning is a way to specify the location of an element anywhere on the page with pixel precision.
- **CSS layout model:**
  - Text elements are laid out from top to bottom in the order in which they appear in the source, and from left to right (in left-to-right reading languages\*).
- **Block elements**
  - Stack up on top of one another and fill the available width of the browser window or other containing element.
  - Inline elements and text characters line up next to one another to fill the block elements.



*Blocks are laid out in the order in which they appear in the source.*

*Each block starts on a new line.*



Objects in the normal flow affect the layout of the objects around them. This is the behavior you've come to expect in web pages—elements don't overlap or bunch up, they make room for one another.

# FLOATING

- The float property moves an element as far as possible to the left or right, allowing the following content to wrap around it.
- Floats are one of the primary tools of modern CSS-based web design, used to create multicolumn layouts, navigation toolbars from lists, and table-like alignment without tables.

**float**

Values: left | right | none | inherit Default: none

Applies to: all elements Inherits: no

**Refer This Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/CSS/float.css>



# Key behaviors of floated elements

**A floated element is like an island in a stream.**

- Floated element influence the surrounding content.
- One popular analogy compares floats to islands in a stream—they are not in the flow, but the stream has to flow around them. This behavior is unique to floated elements.

**Floats stay in the content area of the containing element.**

- It is also important to note that the floated image is placed within the content area (the inner edges) of the paragraph that contains it. It does not extend into the padding area of the paragraph.

**Margins are maintained.**

- In addition, the margin is held on all sides of the floated image, as indicated in previous Figure by the blue, dotted line. In other words, the entire element box, from outer edge to outer edge, is floated.



# Floating block element

**You must provide a width for floated block elements.**

- If you do not provide a width value, the width of the floated block will be set to auto, which fills the available width of the browser window or other containing element.

**Elements do not float higher than their reference in the source.**

- A floated block will float to the left or right relative to where it occurs in the source allowing the following elements in the flow to wrap around it. It will stay below any block elements that precede it in the flow (in effect, it is “blocked” by them). That means you can’t float an element up to the top corner of a page, even if its nearest ancestor is the body element.
- If you want a floated element to start at the top of the page, it must appear first in the document source.



# Floating block element

Refer This Example:

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



# Clearing floated elements

- Applying the clear property to an element prevents it from appearing next to a floated element, and forces it to start against the **next available “clear” space below the float/**

clear

Values: left | right | both | none | inherit

Default: none

Applies to: block-level elements only Inherits: no

- **Refer This Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/CSS/floatClear.html>

# Positioning

- Elements on the page can be positioned relative to where they would normally appear in the flow, or removed from the flow altogether and placed at a particular spot on the page.
- You can also position an element relative to the browser window (technically known as the viewport in the CSS Recommendations) and it will stay put while the rest of the page scrolls.

## **position**

Values: static | relative | absolute | fixed | inherit Default: static

Applies to: all elements Inherits: no



# Positioning Types

## Static:

- This is the normal positioning scheme in which elements are positioned as they occur in the normal document flow.

## relative

- Relative positioning moves the box relative to its original position in the flow. The distinctive behavior of relative positioning is that the space the element would have occupied in the normal flow is preserved.

## absolute

- Absolutely positioned elements are removed from the document flow entirely and positioned relative to a containing element (we'll talk more about this later).
- Unlike relatively positioned elements, the space they would have occupied is closed up. In fact, they have no influence at all on the layout of surrounding elements.



## **fixed**

- The distinguishing characteristic of fixed positioning is that the element stays in one position in the window even when the document scrolls.
- Fixed elements are removed from the document flow and positioned relative to the browser window (or other viewport). rather than another element in the document.

## **Refer This Example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/CSS/position2.html>



# Stacking order

- Absolutely positioned elements overlap other elements, so it follows that multiple positioned elements have the potential to stack up on one another.
- By default, elements stack up in the order in which they appear in the document, but you can change the stacking order with the z-index property.

## z-index

Values: (number) | auto | inherit

Default: auto

Applies to: positioned elements Inherits: no



- The value of the z-index property is a number (positive or negative). The higher the number, the higher the element will appear in the stack. Lower numbers and negative values move the element lower in the stack.

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

# CSS Box Model

Explanation of the different parts:

**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.



# CSS Box Model

**Refer This Example:**



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

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



# CSS Background

CSS background properties are used to define the background effects of an element.

CSS properties used for background effects:

- background-color
- background-image
- background-repeat
- background-attachment
- background-position



# Refer This Example

---

**Background Color:**

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

**Background Image:**

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

**Background Repeat:**

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

**Background Attachment:**

<https://github.com/TopsCode/Software-Engineering/blob/master/CSS/background/backgroundAttachment.htm>





*Welcome In My Home Town*

[Home](#)

[About Us](#)

[Contact Us](#)

[Home](#) || [About us](#) || [Contact Us](#)

*Created By Koyal Pragapan*

# CSS Pseudo Classes

## Styles for Special Cases :

Although primarily intended to add styles to particular elements created using HTML tags, there are several cases where we can use CSS to style content on the page that is not specifically set off by HTML tags or to create a dynamic style in reaction to something that your Web site visitor is doing on the screen. These are known as pseudo-elements and pseudo-classes:

- 1. Link pseudo-classes:** Used to style hypertext links. Although primarily associated with color, you can actually use any CSS property to set off links and provide user feedback during interaction.
- 2. Dynamic pseudo-classes:** Used to style any element on the screen depending on how the user is interacting with it.
- 3. Pseudo-elements:** Used to style the first letter or first line in a block of text.



# CSS Link Pseudo Classes

- **Styles for Link Actions**

Although the link tag can be styled just like any other HTML tag, it is a special case, because people visiting your site can interact with it. To that end, CSS includes four different pseudo-classes for each of the four interaction states:

link visited hover active

- They need to be in the above order—link, visited, hover, active—to function properly.

**Refer This Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/CSS/pseudoClass/link.html>

# Dynamic Pseudo Class

- **Styles for Dynamic Actions**

The hover and active states are not just for links. You can actually place your cursor over and click on any element on the screen and style elements for those actions. The third action state is when the user selects an element on the screen (usually a form field) and that element is in focus and it is ready for user input.

The default text color for the class `formField` in an input box is gray.

```
input.formField { color: gray; }
```

When the user hovers over an input field with the `formField` class, its text color is green.

```
input.formField:hover { color: green; }
```

When the user clicks an input field with the `formField` class, its text color is red.

```
input.formField:active { color: red; }
```

# CSS Pseudo Classes

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/CSS/pseudoClass/example2.html>



# Module 3 Topics:

- Programming Language
- High level Programming language
- POP
- C Programming
  - Overview, Identifiers, Constants
  - Variable, Storage classes, Symbolic Constants
  - Operators, Expressions
  - Type Conversion
  - Decision Making and branching statements
  - Array, String, Functions
  - Structure , Unions , Pointer
  - String Function, Dynamic Memory Allocation
  - Linked list , Preprocessor
  - File Management

# Programming Language

- A programming language is a set of commands, instructions, and other syntax used to create software.
- Example: For writing, a meaningful story in English language we use English alphabets, words and grammar.
- Similarly, Software are created using a Programming language that has its own syntax and rules.
- Example of programming Language: Python,C, C++, Java.



# High Level Language

- High level languages are written in a form that is close to our human language, enabling the programmer to just focus on the problem being solved
- No particular knowledge of the hardware is needed as high level languages create programs that are portable and not tied to a particular computer or microchip.
- These programmer friendly languages are called ‘high level’ as they are far removed from the machine code instructions understood by the computer.
- Examples include: C,C++, Java, Pascal, Python, Visual Basic.

## Advantages

- Easier to modify as it uses English like statements
- Easier/faster to write code as it uses English like statements
- Easier to debug during development due to English like statements
- Portable code – not designed to run on just one type of machine



# Low Level Language

- Low level languages are used to write programs that relate to the specific architecture and hardware of a particular type of computer.
- They are closer to the native language of a computer (binary), making them harder for programmers to understand.
- Examples of low level language: Assembly Language, Machine Code



# Procedural Programming Language (POP)

- POP follows a step-by-step approach to break down a task into a sequence of instructions.
- Each step is carried out in order in a systematic manner so that a computer can understand what to do. The program is divided into small parts called functions and then it follows a series of computational steps to be carried out in order.
- It follows a top-down approach to actually solve a problem, hence the name.
- Procedures correspond to functions and each function has its own purpose.
- Dividing the program into functions is the key to procedural programming. So a number of different functions are written in order to accomplish the tasks
- E.g.: c, basic, FORTRAN.



# Why C?

- C is small (only 32 keywords).
- C is common (lots of C code about).
- C is stable (the language doesn't change much).
- C is quick running.
- C is the basis for many other languages (Java, C++, awk, Perl,C#).
- It may not feel like it but C is one of the easiest languages to learn.



# Advantages of C language

- C is a general purpose programming language, meaning that it is not limited to any one specific kind of programming. You can write all sorts of software using C.
- C is not a very high-level language. C allows you to directly access memory addresses, create bit fields and structures and map them to memory, perform bitwise operations and so on. C facilitates hardware programming.
- Not being high-level also means there is little overhead; it is highly efficient and provides fast execution speed.
- There are C language compilers and development tools available for many different platforms from small embedded systems to large mainframes and supercomputers..

# Structure of C programming

```
#include <stdio.h> // Library Files  
void main()  
{  
    printf("Hello World"); // printing statement  
}
```

Hello World

**Refer this Example:**

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

(printing hello world)



# Structure of C programming

#include<stdio.h>

- stdio.h, which stands for "standard input/output header", is the header in the C standard library that contains macro definitions, constants, and declarations of functions and types used for various standard input and output operations.
- This is called preprocessor in C.

void main()

- The second line main() tell the compiler that it is the starting point of the program, every program should essentially have the main function only once in the program.
- The opening and closing braces indicates the beginning and ending of the program. All the statements between these two braces form the function body. These statements are actually the C code which tells the computer to do something



## printf()

- Printf functions (which stands for "print formatted") are a class of functions typically associated with some types of programming languages. They accept a string parameter called the format string
- The `/* .... */` is a comment and will not be executed, the compiler simply ignores this statement. These are essential since it enhances the readability and understandability. of the program



# Keywords and Identifiers

## Keyword:

- Keywords are predefined reserved identifiers that have special meanings
- They cannot be used as identifiers in your program.

```
-----  
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  
-----
```

## Identifier:

- Identifiers refers to the name of user-defined variables, array and functions.

# Rules for Identifiers

**The identifiers must conform to the following rules:**

1. First character must be an alphabet (or underscore)
2. Identifier names must consists of only letters, digits and underscore.
3. A identifier name should have less than 31 characters.
4. Any standard C language keyword cannot be used as a variable name.
5. A identifier should not contain a space.



# Variables in C

- It is a data name which is used to store data and may change during program execution.
- Variable name is a name given to memory cells location of a computer where data is stored.

## Rules for variables:

- First character should be letter or alphabet.
- Keywords are not allowed to use as a variable name.
- White space is not allowed.
- C is case sensitive i.e. UPPER and lower case are significant.
- Only underscore, special symbol is allowed between two characters
- The length of identifier may be upto 31 characters but only the first 8 characters are significant by compiler.
- Some compilers allow variable names whose length may be upto 247 characters. But, it is recommended to use maximum 31 characters in variable name. Large variable name leads to occur errors.

# Constants in C

- A constant is an entity that doesn't change during the execution of a program.

Followings are the different types of constants :

## 1. Real Constant :

- It must have at least one digit.
- It must have a decimal point which may be positive or negative.
- Use of blank space and comma is not allowed between real constants.

Example:

+194.143, -416.41



## **2. Integer Constant :**

- It must have at least one digit.
- It should not contain a decimal place.
- It can be positive or negative.
- Use of blank space and comma is not allowed between real constants.
- Example: 1990, 194, -394

## **3. Character Constant :**

- It is a single alphabet or a digit or a special symbol enclosed in a single quote.
- Maximum length of a character constant is 1.
- Example: 'T', '9', '\$'

## **4. String Constant :**

- It is collection of characters enclosed in double quotes.
- It may contain letters, digits, special characters and blank space.
- Example: "Technowell Web Solutions, Sangli"



# Data Types in C

- “Data type can be defined as the type of data of variable or constant store.”
- When we use a variable in a program then we have to mention the type of data. This can be handled using data type in C.
- Followings are the most commonly used data types in C.

<u>Keyword</u>	<u>Format Specifier</u>	<u>Size</u>	<u>Data Range</u>
<b>char</b>	<b>%c</b>	<b>1 Byte</b>	<b>-128 to +127</b>
<b>unsigned char</b>	<b>&lt;--- -&gt;</b>	<b>8 Bytes</b>	<b>0 to 255</b>
<b>int</b>	<b>%d</b>	<b>2 Bytes</b>	<b>-32768 to +32767</b>
<b>long int</b>	<b>%ld</b>	<b>4 Bytes</b>	<b>-2<sup>31</sup> to +2<sup>31</sup></b>
<b>unsigned int</b>	<b>%u</b>	<b>2 Bytes</b>	<b>0 to 65535</b>
<b>float</b>	<b>%f</b>	<b>4 Bytes</b>	<b>-3.4e<sup>38</sup> to +3.4e<sup>38</sup></b>
<b>double</b>	<b>%lf</b>	<b>8 Bytes</b>	<b>-1.7e<sup>308</sup> to +1.7e<sup>308</sup></b>
<b>long double</b>	<b>%Lf</b>	<b>12-16 Bytes</b>	<b>-1.7e<sup>308</sup> to +1.7e<sup>308</sup></b>

## Qualifier :

- When qualifier is applied to the data type then it changes its size.
- Size qualifiers : short, long
- Sign qualifiers : signed, unsigned

## Integer Types

- Integers are used to store whole numbers.

Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

## Float Types

Float types are used to store real numbers.

Type	Size(bytes)	Range
Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

## Character Type :

Character types are used to store characters value

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

## **Void Type :**

- The void type basically means "nothing". A void type cannot hold any values. You can also declare a function's return type as void to indicate that the function does not return any value.

## **Refer This Example:**

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



## **Enum Data Type :**

This is an user defined data type having finite set of enumeration constants. The keyword 'enum' is used to create enumerated data type.

Syntax:

```
enum [data_type] {const1, const2, ...., const n};
```

Example:

```
enum mca(software, web, seo);
```

## **Typedef :**

It is used to create new data type. But it is commonly used to change existing data type with another name.

Syntax:

```
typedef [data_type] synonym; OR
```

```
typedef [data_type] new_data_type;
```

Example:

```
typedef int integer; integer rno;
```



# Managing Input and Output Operations

- To display output to the user, we can use `printf()` function
- To take input from user , use `scanf()` function.

**Format specifier:**

Format specifier	Type of value
%d	Integer
%f	Float
%lf	Double
%c	Single character
%s	String
%u	Unsigned int
%ld	Long int
%lf	Long double

## Refer This Example:

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



# Single character input output:

The basic operation done in input output is to read a characters from the standard input device such as the keyboard and to output or writing it to the output unit usually the screen. The getchar function can be used to read a character from the standard input device. The scanf can also be used to achieve the function. The getchar has the following form.

```
character _variable = getchar();
```

- Variable name is a valid ‘C’ variable, that has been declared already and that possess the type char.

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



The putchar function which is analogous to getchar function can be used for writing characters one at a time to the output terminal. The general form is

**putchar (variable name);**

Where variable is a valid C type variable that has already been declared Ex:-

**putchar (ch);**

Displays the value stored in variable ch to the standard screen.

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



# Declaration of Storage Classes

- A storage class defines the scope (visibility) and life time of variables and/or functions within a C Program.
- There are following storage classes which can be used in a C Program
  - auto
  - register
  - static
  - Extern

## auto - Storage Class

- auto is the default storage class for all local variables.
- Keyword : auto
- Storage Location : Main memory
- Initial Value : Garbage Value
- Life : Control remains in a block where it is defined.
- Scope : Local to the block in which variable is declared.
- Refer the Example:
- <https://github.com/TopsCode/Software-Engineering/blob/master/C/autoExample.c>



## **Register Storage Class :**

Keyword : register

Storage Location : CPU Register

Initial Value : Garbage

Life : Local to the block in which variable is declared. Scope : Local to the block.

Syntax : register [data\_type] [variable\_name];

## **Static Storage Class :**

Keyword : static

Storage Location : Main memory

Initial Value : Zero and can be initialize once only.

Life : depends on function calls and the whole application or program.

Scope : Local to the block.

Syntax : static [data\_type] [variable\_name]; Example : static int a;

## **Refer this example:**

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



# Types of Static Variable:

There are two types of static variables as :

- a) Local Static Variable
- b) Global Static Variable

Static storage class can be used only if we want the value of a variable to persist between different function calls.

**Refer this example:** <https://github.com/TopsCode/Software-Engineering/blob/master/C/staticVariable.c>



# External Storage Class :

Keyword : extern

Storage Location : Main memory Initial

Value : Zero

Life : Until the program ends. Scope : Global  
to the program.

Syntax : `extern [data_type] [variable_name];`

Example : `extern int a;`

**Refer this example:**

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



# Defining Symbolic Constants

A symbolic constant value can be defined as a preprocessor statement and used in the program as any other constant value. The general form of a symbolic constant is

#define symbolic\_name value\_of\_constant Valid examples of constant definitions are :

```
# define MARKS 100  
# define TOTAL 50 # define PI 3.14159
```

These values may appear anywhere in the program, but must come before it is referenced in the program.

It is a standard practice to place them at the beginning of the program.

# Volatile Variable

- A volatile variable is the one whose values may be changed at any time by some external sources.
- Example:

`volatile int num;`

- The value of data may be altered by some external factor, even if it does not appear on the left hand side of the assignment statement.
- When we declare a variable as volatile the compiler will examine the value of the variable each time it is encountered to see if an external factor has changed the value



# Operators

- "Operator is a symbol that is used to perform mathematical operations."
- When we use a variable in a program then we have to mention the type of data. This can be handled using data type in C.
- Followings are the most commonly used data types in C.

Operator Name	Operators
Assignment	=
Arithmetic	+ , - , * , / , %
Logical	&& ,    , !
Relational	< , > , <= , >= , == , !=
Shorthand	+ = , - = , * = , / = , % =
Unary	++ , --
Conditional	( exp1 )? exp2: exp3 ;

**Assignment Operator:** It is used to assign a value to variable.

**Arithmetic Operator:** It is also called as 'Binary operators'. It is used to perform arithmetical operations. These operators operate on two operands.

**Logical Operator:** They compare or evaluate logical and relational expressions.

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT



- **Logical AND (&&):**

<b>a</b>	<b>b</b>	<b>a &amp;&amp; b</b>
true	true	true
true	false	false
false	true	false
false	false	false

- **Logical OR(||):**

<b>a</b>	<b>b</b>	<b>a    b</b>
true	true	true
true	false	true
false	true	true
false	false	false

- **Logical Not**

<b>Boolean NOT</b>	
0	1
1	0

# Refer this Example:

- **Assignment Operator**

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

- **Arithmetric 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>



# Relational operator

- Relational operator is used when there is relation between 2 or more variables.

<u>Operator</u>	<u>Meaning</u>
<	<b>Less than</b>
<=	<b>Less than or equal to</b>
>	<b>greater than</b>
>=	<b>greater than or equal to</b>
!=	<b>Not equal to</b>
==	<b>Is equal to</b>

```
int age=90; if(age>18)
{
printf("\n u are eligible for watting..");
}
```



## Shorthand Operator

- It is used to perform mathematical operations at which the result or output can affect on operands.

## Unary Operator

- It operates on a single operand. Therefore, this operator is called as 'unary operator.' It is used to increase or decrease the value of variable by 1.

## Conditional Operator

- Conditional operator is also called as 'ternary operator.'
- It is widely used to execute condition in true part or in false part. It operates on three operands.

The logical or relational operator can be used to check conditions.



# Refer this Example:

## **Shorthand Operator**

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

## **Unary Operator:**

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

## **Conditional Operator:**

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



# Special Operator:

- C supports some special operators of interest such as comma operator, size of operator, pointer operators (& and \*) and member selection operators (. and ->).

## The Comma Operator :

- The comma operator can be used to link related expressions together. A comma-linked list of expressions are evaluated left to right and value of right most expression is the value of the combined expression.

## Some Example of comma operator:

- `(x = 10, y = 5, x + y);`
- `for (n=1, m=10, n <=m; n++,m++)`
- `t = x, x = y, y = t;`



## The size of Operator :

- The operator size of gives the size of the data type or variable in terms of bytes occupied in the memory.
- The operand may be a variable, a constant or a data type qualifier.

### Example

```
m = sizeof (sum);  
n = sizeof (long int); k = sizeof (235L);
```

- Refer This Example:
- <https://github.com/TopsCode/Software-Engineering/blob/master/C/sizeOf.c>



# C Programming – Expressions

## ❖ Arithmetic Expressions

- An expression is a combination of variables constants and operators written according to the syntax of C language.

- In C every expression evaluates to a value .
- Algebraic Expression - C Expression

a x b – c -> a \* b – c

(m + n) (x + y) -> (m + n) \* (x + y) (ab / c) -> a \* b / c

3x<sup>2</sup> +2x + 1 -> 3\*x\*x+2\*x+1 (x / y) + c -> x / y + c



## Evaluation of Expressions

Expressions are evaluated using an assignment statement of the form

Variable = expression;

**Example of evaluation statements are**

`x = a * b - c` `y = b / c * a`

`z = a - b / c + d;`

**Refer this example:** <https://github.com/TopsCode/Software-Engineering/blob/master/C/expression1.c>



## Precedence in Arithmetic Operators

- An arithmetic expression without parenthesis will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C.
- **High priority \* / % ; Low priority + -**

### Rules for evaluation of expression

1. First parenthesized sub expression left to right are evaluated.
2. If parenthesis are nested, the evaluation begins with the innermost sub expression.
3. The precedence rule is applied in determining the order of application of operators in evaluating sub expressions.
4. The associability rule is applied when two or more operators of the same precedence level appear in the sub expression.
5. Arithmetic expressions are evaluated from left to right using the rules of precedence.
6. When Parenthesis are used, the expressions within parenthesis assume highest priority.

# Refer This Example

---

## Operator Precedence:

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

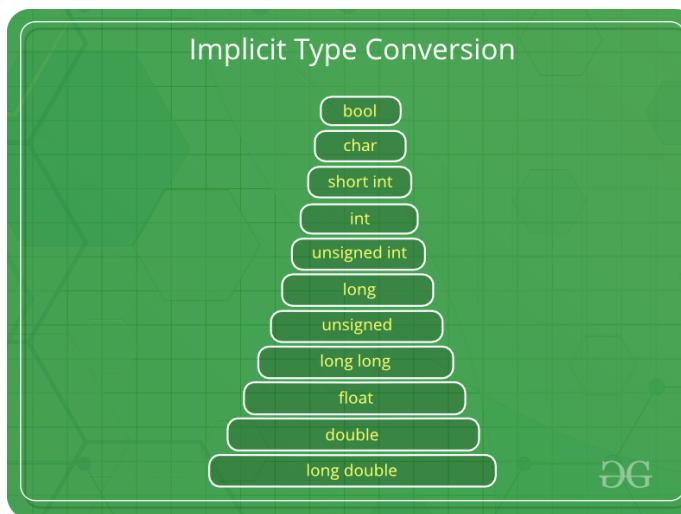


# Type conversions in expressions

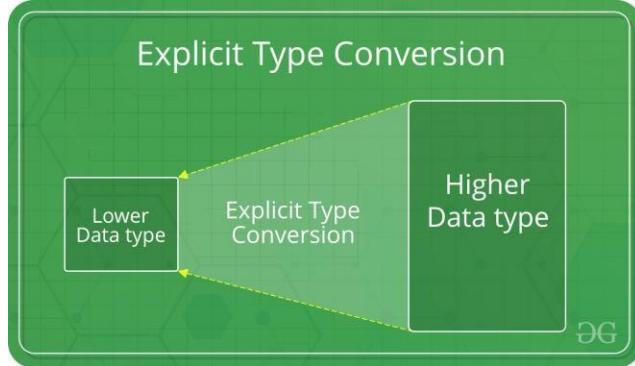
- Typecasting is also called as type conversion
- It means converting one data type into another.

## ❖ Implicit type conversion

- If the operands are of different types the lower type is automatically converted to the higher type before the operation proceeds. The result is of higher type.
- This automatic type conversion is known as implicit type conversion



# Explicit Conversion



- Forced Casting done by programmer.
- Syntax:  
**(type) expression**
- Example:  
`int x=7, y=5; float z;  
z = (float)x/(float)y;`

# Refer This Example

---

## Implicit Conversion

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

## Explicit Conversion:

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



# Decision Making – Branching

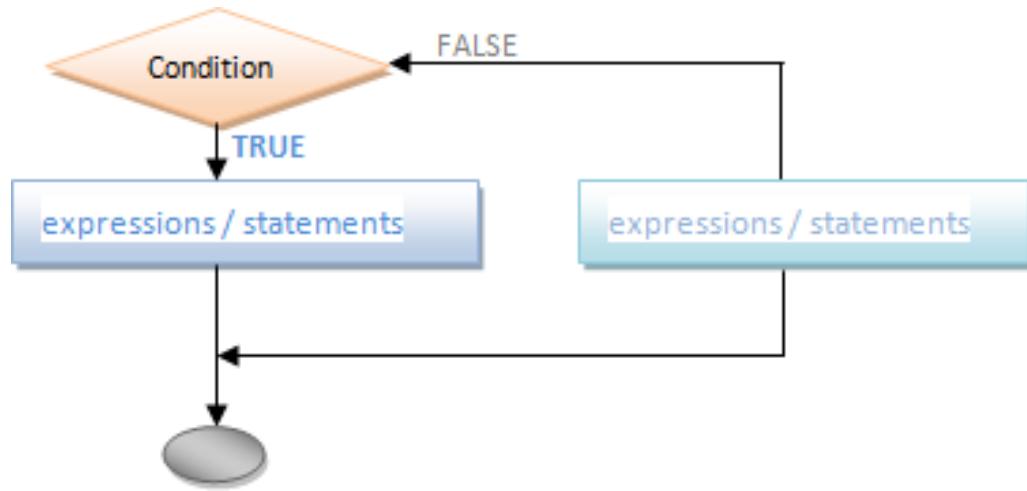
## Branching :

- The C language programs presented until now follows a sequential form of execution of statements.
- Many times it is required to alter the flow of the sequence of instructions. C language provides statements that can alter the flow of a sequence of instructions.
- These statements are called control statements. These statements help to jump from one part of the program to another. The control transfer may be conditional or unconditional.



# Control statements :

- 1) if statement.
- 2) if else statement.
- 3) Nested if statement.
- 4) Else if ladder statement.
- 5) Switch statement.



## 1. The if Statement:

- To conditionally execute statements, you can use the if or the if...else statement.
- The general form of the if statement is `if(expr) {`

```
s1;  
s2;  
....  
}
```

## 2. The If else construct:

- The if else is just an extension of the general format of if statement. If the result of the condition is true, then program statement 1 is executed, otherwise program statement 2 will be executed.

```
if (condition)  
simple or compound statement  
else  
simple or compound statement.
```



### 3. Nested if Statement:

- The if statement may itself contain another if statement is known as nested if statement.

### Compound Relational tests:

- C language provides the mechanisms necessary to perform compound relational tests.
- A compound relational test is simple one or more simple relational tests joined together by either the logical AND(&&) or the logical OR( || ) operators.

### Syntax :

```
a> if (condition1 && condition2 && condition3) b> if (condition1 || condition2  
|| condition3)
```



## 4. The ELSE If Ladder

- When a series of many conditions have to be checked we may use the ladder else if statement which takes the following general form.

```
if (condition1) statement – 1;  
else if (condition2) statement2;  
else if (condition3) statement3;  
else if (condition)  
statement n;  
else  
default statement;  
  
statement-x;
```

- This construct is known as if else construct or ladder. The conditions are evaluated from the top of the ladder to downwards.



# 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>



# The Switch Statement:

- Switch case statements are a substitute for long if statements that compare a variable to several integral values
- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
- Switch is a control statement that allows a value to change control of execution.

```
switch(switch_expr)
{
case constant expr1 : S1;
S2;
break;
case constant expr1 : S3;
S4;
break;
.....
default      : S5;
S6;
break
}
k;
```

## Refer This Example:

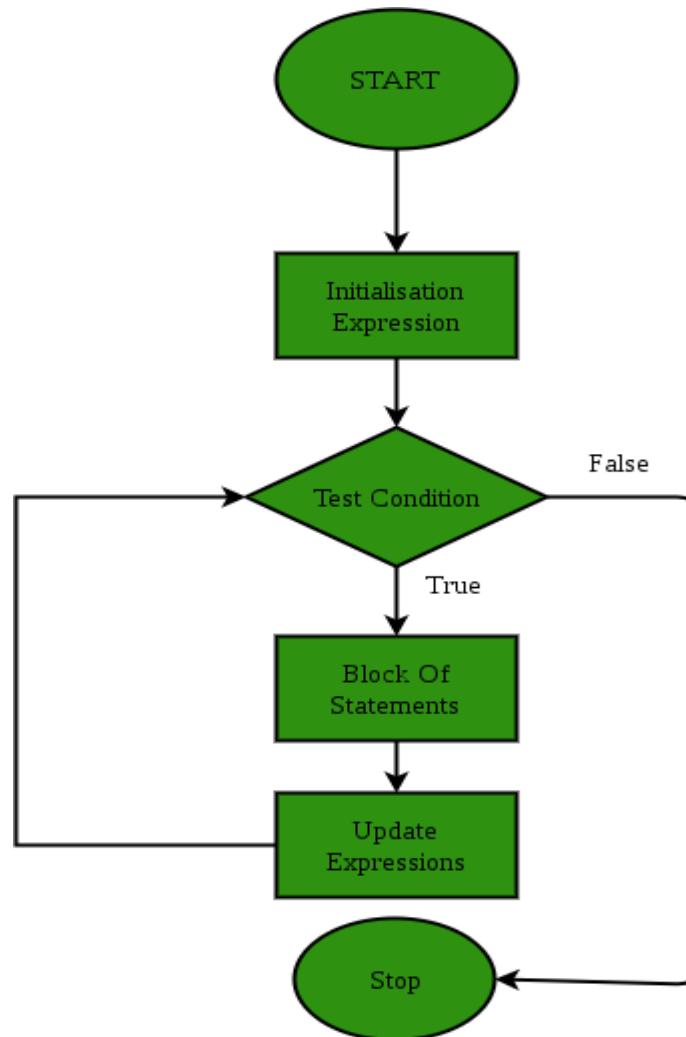
<https://github.com/TopsCode/Software-Engineering/blob/master/C/switchCase.c>

## Decision Making – Looping

- During looping a set of statements are executed until some conditions for termination of the loop is encountered. A program loop therefore consists of two segments one known as body of the loop and other is the control statement. The control statement tests certain conditions and then directs the repeated execution of
- the statements contained in the body of the loop.
- In looping process in general would include the following four steps
  - 1. Setting and initialization of a counter
  - 2. Exertion of the statements in the loop
  - 3. Test for a specified conditions for the execution of the loop
  - 4. Incrementing the counter
- The test may be either to determine whether the loop has repeated the specified number of times or to determine whether the particular condition has been met.

# Types of loops

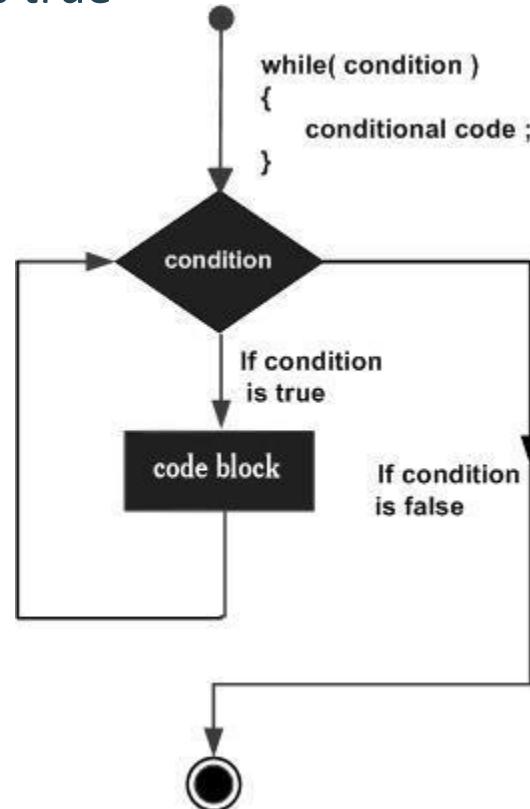
- 1) While loop,
- 2) Do.. While loop,
- 3) For loop,



## The While Statement:

- A while loop in C programming repeatedly executes a target statement as long as a given condition is true

```
• while (test condition)
{
//body of the loop updation;
}
```



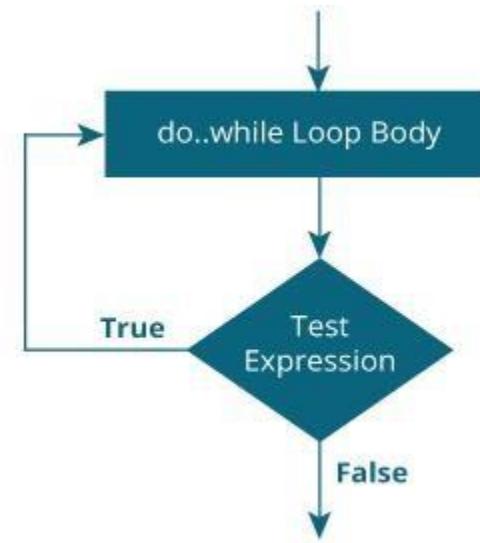
- Refer this Example:

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

# The Do while statement:

- The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

```
do  
{  
statement; updation;  
}  
while(condition);
```



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

# For Loop:

- The for loop provides a more concise loop control structure.

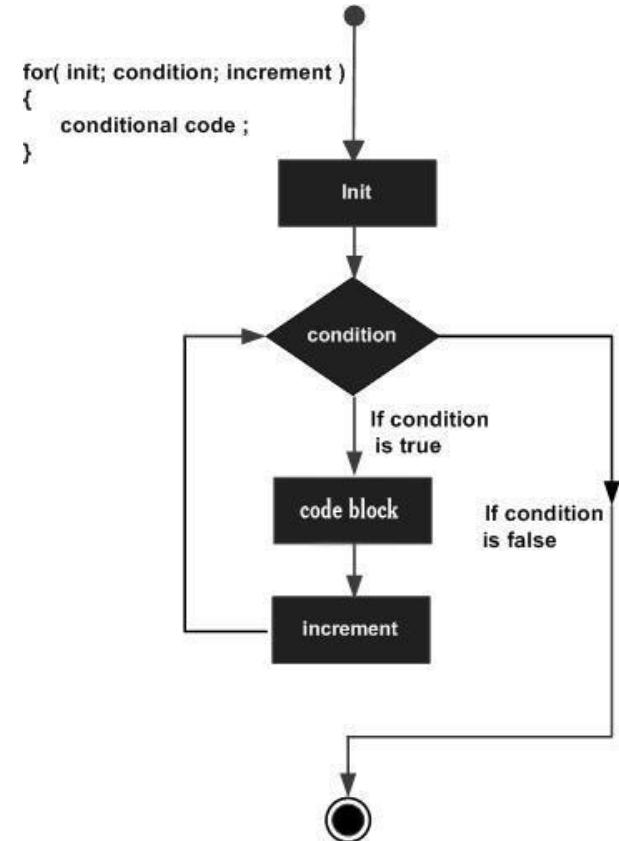
```
for (expr1;expr2;expr3){ s1;  
s2 ;  
}
```

The for loop is executed as follows:

- It first evaluates the initialization code.
- Then it checks the condition expression.
- If it is true, it executes the for-loop body.
- Then it evaluate the increment/decrement condition and again follows from step 2.
  - When the condition expression becomes false, it exits the loop.

- Refer this example:

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



# Jumping Statements:

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



## 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.

```
goto label;  
-----  
-----  
label:  
-----  
-----
```

## Refer this Example:

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

# The Break Statement:

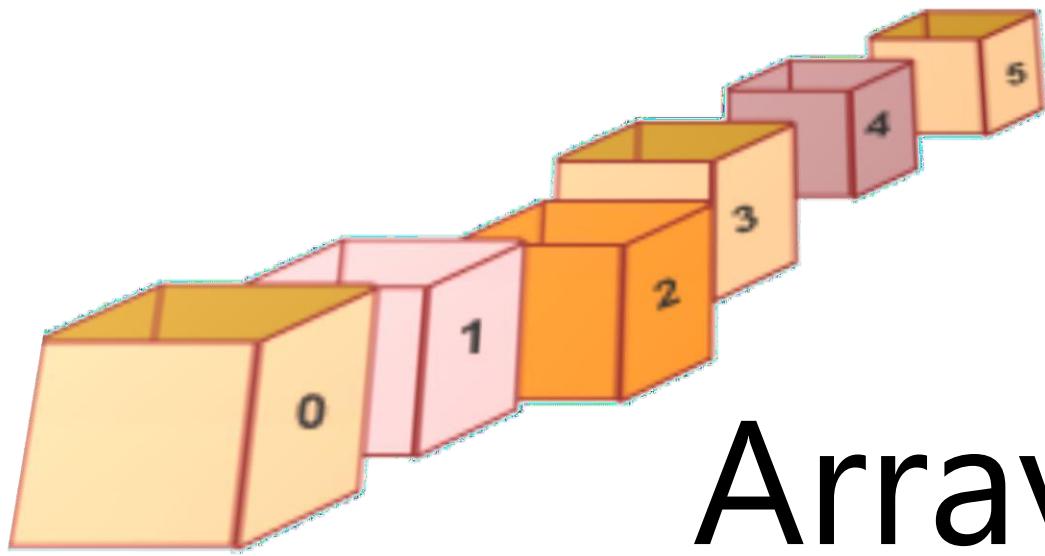
- The break statement is used inside loop or switch statement.
- When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.
- **Syntax:** break ;
- **Refer this Example:**  
<https://github.com/TopsCode/Software-Engineering/blob/master/C/Break.c>



## 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>





# Array



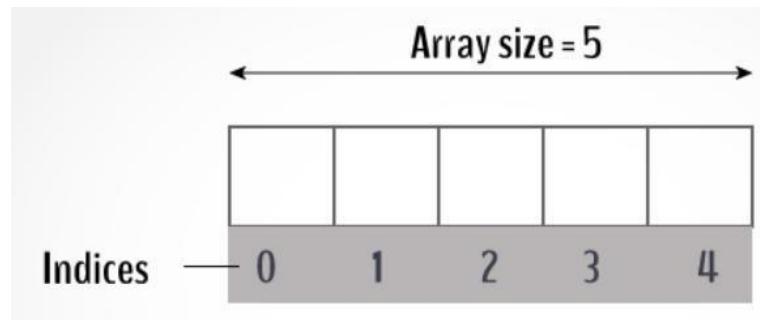
# ARRAY

- An array is a variable that can store multiple values of similar data type
- Declaration of arrays:

```
datatype array-name[array_size];
```

- Example:  

```
float mark[5];
```



## Access Array Elements:

- Array's element can be accessed by indices.
- Arrays have 0 as the first index, not 1



## Initialization of arrays:

- Initialize during declaration:

- Syntax:

type array\_name[size]={list of values};

- Example:

int mark[5] = {19, 10, 8, 17, 9};

Or

int mark[] = {19, 10, 8, 17, 9};

## Refer this Example:

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

• <https://github.com/TopsCode/Software-Engineering/blob/master/C/Array/Example2.c>



# Multi dimensional Arrays:

- In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays
- C allows arrays of three or more dimensions. The compiler determines the maximum number of dimension.
- The general form of a multidimensional array declaration is:

`data_type array_name*s1+*s2+*s3+.....*sn];`

`int survey[3][5][12]; //3-D array to hold 180 elements float table[5][4][5][3];//4-D array containing 300 elements`

- The declaration of **two dimension arrays** is as follows:

`data_type array_name[row_size][column_size]; float x[3][4];`

- The declaration of **three dimension array**: `float y[2][4][3];`

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]



## Initialization of multidimensional arrays:

- Like one dimension arrays, 2 dimension arrays may be initialized by following their declaration with list of initial values enclosed in braces

Example:

```
int table[2][3]={0,0,0,1,1,1};
```

- Initializes the elements of first row to zero and second row to 1. The initialization is done row by row. The above statement can be equivalently written as

```
int table[2][3]={{0,0,0},{1,1,1}}
```

Refer this Example: <https://github.com/TopsCode/Software-Engineering/blob/master/C/MultiDimensionalArray.c>



# String

# What is String?

In C programming, a string is a sequence of characters terminated with a null character \0.

For example:

```
int c[] = "c string";
```



**Declare a string:**

**Syntax:**

```
char str_name[size];
```

**Example:**

```
char s[5];
```



**Initialize a string:**

```
char c[] = "abcd"; char c[50] = "abcd";
```

```
char c[] = {'a', 'b', 'c', 'd', '\0'};
```

```
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```



# Reading Strings from the terminal:

- The function `scanf` with `%s` format specification is needed to read the character string from the terminal. Example is as follow.

```
char address[15]; scanf("%s",address);
```

- The function `getchar` can be used repeatedly to read a sequence of successive single characters and store it in the array.

**Refer This Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/C/String/scanf.c>

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



# Multiline string:

- We can take input and give output of a string that consists of more than one word by using **gets** and **puts**, where gets is used to take input of a string from user and puts is used to display the string.

**Refer This Example:**

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

**Strings always end with the NULL character**

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

# String Function Library



# String operations (string.h)

- C library supports a large number of string handling functions that can be used to array out many o f the string manipulations such as:

`strlen()` :number of characters in the string. `strcat()` : adding two are more strings  
`strcmp()` two strings : compare two strings. `strcpy()` : copies one string over another `strlwr()`: convert string to lower case `strupr()`: convert string to upper case `strrev()`:reverse a string

- To do all the operations described here it is essential to include `string.h` library header file in the program.

**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>



```
#include<stdio.h>
#include<conio.h>
void add(int x,int y)
{
    int result;
    result = x+y;
    printf("Sum of %d and %d is %d\n",x,y,result);
}
void main()
{
    clrscr();
    add(10,15);
    add(55,64);
    add(168,325);
    getch();
}
```

Used Defined Function

Function Called

# Functions



# What is a Function?

- A function is a set of statements that take inputs, do some specific computation and produces output.
- The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call the function.

## Types of functions :

1. Built in Functions
2. User Defined Functions



# 1. Built in Functions :

- These functions are also called as 'library functions'. These functions are provided by system. These functions are stored in library files.

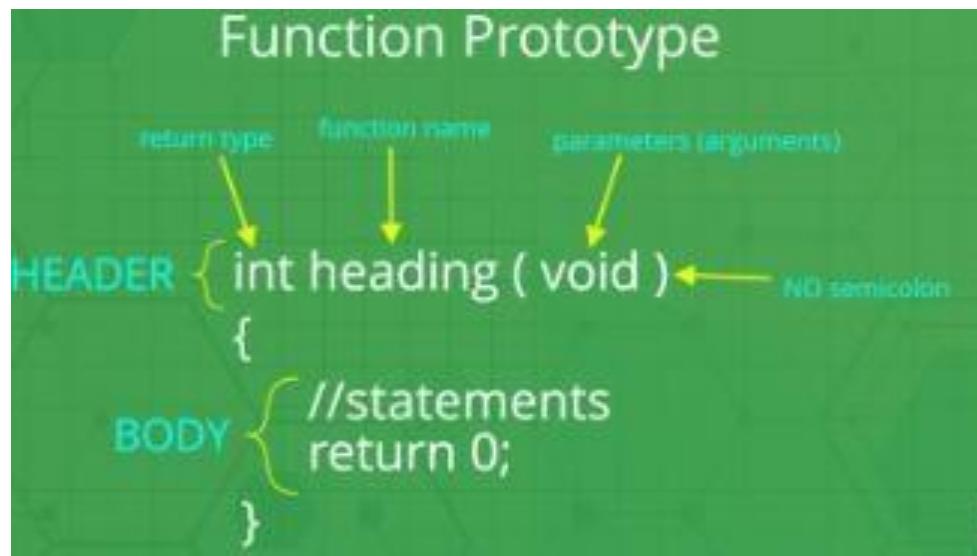
e.g.

scanf() printf() strcpy() strlwr() strcmp() strlen() strcat()

## 2. User Defined Functions :

- The functions which are created by user for program are known as 'User defined functions'.

Syntax:



# Categories of user define functions

- 1) Without return type without parameter(argument).
- 2) Without return type with parameter(argument).
- 3) With return type without parameter(argument)
- 4) With return type with parameter(argument)



# Refer this example

---

## **Without return type without argument**

<https://github.com/TopsCode/Software>

-  
[Engineering/blob/master/C/Function1.c](https://github.com/TopsCode/Software/blob/master/C/Function1.c)



# Function Call by Passing Value :

- When a function is called by passing value of variables then that function is known as 'function call by passing values.' Here syntax of without return type with parameter.
- Syntax:

```
// Declaration void  
return type function_name(var_nm); function_name(var_nm); //call  
// Definition void  
return type function_name(data_type var_nm)  
{  
<function_body>; -----;  
}
```

- Refer this Example: <https://github.com/TopsCode/Software-Engineering/blob/master/C/Function2.c>

# Function Call by Returning Value :

- When a function is called by passing value of variables then that function is known as 'function call by passing values.' Here syntax of with return type with parameter.
- Syntax:

// Declaration

```
return type function_name(var_nm);
```

```
variable name = function_name(var_nm); //call
```

// Definition with return datatype

```
return type function_name(data_type var_nm)
{
    function_body; -----
}
```

Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C%20%26%20C%2B%2B/FunctionCallByReturningValue.c>



# Function Call by Passing and Returning Value :

- When a function passes and returns value of variables then that function is known as 'function call by passing and returning values.'

Refer this example: <https://github.com/TopsCode/Software-Engineering/blob/master/C/function4.c>



# Recursion

- Recursive function is a function that contains a call to itself. C supports creating recursive function with ease and efficient.

Refer this example: <https://github.com/TopsCode/Software-Engineering/blob/master/C/recursion.c>



# Structures and Unions



# What is Structures?

- A structure is a user defined data type . A structure creates a data type that can be used to group items of possibly different types into a single type.

```
struct structure_name  
{  
datatype vari1, vari2...;  
datatype vari3,...;  
};
```

```
struct object  
{  
char id[20];  
int xpos;  
int ypos;  
};
```

- Structures can group data of different types as you can see in the example of a game object for a video game. The variables you declare inside the structure are called data members.

- **Initializing a Structure**

```
struct object player1 = ,“player1”, 0, 0};
```



- The above declaration will create a struct object called player1 with an id equal to “player1”, xpos equal to 0, and ypos equal to 0.
- To access the members of a structure, you use the “.” (scope resolution) operator.

Sample Code:

```
struct object player1; player1.id = “player1”; player1.xpos = 0;  
player1.ypos = 0;
```

Refer this example:

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

# Arrays of structure:

- It is possible to define a array of structures for example if we are maintaining information of all the students in the college and if 100 students are studying in the college. We need to use an array than single variables. We can define an array of structures as shown in the following example:

structure information

```
{  
int id_no;  
char name[20]; char address[20];  
char combination[3]; int age;  
} student[100];
```

- An array of structures can be assigned initial values just as any other array can. Remember that each element is a structure that must be assigned corresponding initial values as illustrated below.
- Refer this Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/C/arraysOfStructure.c>

## Structure within a structure:

- A structure may be defined as a member of another structure. In such structures the declaration of the embedded structure must appear before the declarations of other structures.

```
struct date
{
    int day; int month; int year;
};

struct student
{ int id_no;
    char name[20]; char address[20];
    char combination[3]; int age;
    structure date def; structure date doa;
}oldstudent, newstudent;
```

- Structure student contains another structure date as one of its members.



# Union:

- Like Structures, union is a user defined data type. In union, all members share the same memory location.

union item

```
{  
int m; float p; char c;  
} code;
```

- This declares a variable code of type union item. The union contains three members each with a different data type. However we can use only one of them at a time. This is because if only one location is allocated for union variable irrespective of size.

# DIFFERENCE BETWEEN STRUCTURE AND UNION

- All the members of the structure can be accessed at once, where as in an union only one member can be used at a time. Another important difference is in the size allocated to a structure and an union.

for eg:

```
struct example
```

```
{
```

```
int integer;
```

```
float floating_numbers;
```

```
};
```

- The size allocated here is `sizeof(int)+sizeof(float)`; where as in an union union example `{ int integer; float floating_numbers; }` size allocated is the size of the highest member. so size is `=sizeof(float)`;



# Pass Structure to a Function

- We can pass structures as arguments to functions. Unlike array names however, which always point to the start of the array, structure names are not pointers. As a result, when we change structure parameter inside a function, we don't effect its corresponding argument.

Refer this Example:

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

## Passing entire function to functions:

- In case of structures having to having numerous structure elements passing these individual elements would be a tedious task. In such cases we may pass whole structure to a function as shown below:

Refer this Example: <https://github.com/TopsCode/Software-Engineering/blob/master/C/functTofunct.c>

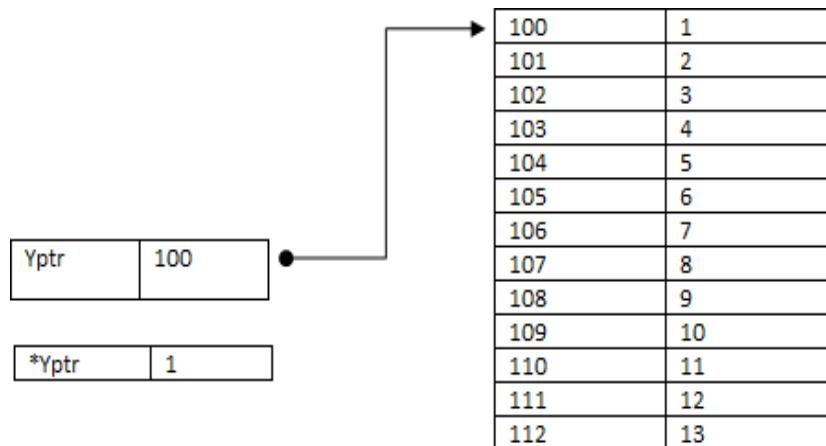


# Pointer In C



# Introduction

- Pointers store address of variables or a memory location.
- As you can see below; Yptr is pointing to memory address 100.



# Pointer and memory relationship

## Pointer Declaration

- Declaring pointers can be very confusing and difficult at times (working with structures and pointer to pointers).
- To declare pointer variable we need to use \* operator (indirection/dereferencing operator) before the variable identifier and after data type. Pointer can only point to variable of the same data type.

## Syntax

Datatype \*pointer\_name ;

**Refer this Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/C/characterPointer.c>



# Pointer Operators

---

Operator	Meaning
*	<p>Serves 2 purpose</p> <ol style="list-style-type: none"><li>1. Declaration of a pointer</li><li>2. Returns the value of the referenced variable</li></ol>
&	<p>Serves only 1 purpose</p> <ul style="list-style-type: none"><li>• Returns the address of a variable</li></ul>



## **Address operator**

- Address operator (&) is used to get the address of the operand. For example if variable x is stored at location 100 of memory; &x will return 100.
- This operator is used to assign value to the pointer variable. It is important to remember that you MUST NOT use this operator for arrays, no matter what data type the array is holding. This is because array name is pointer variable itself. When we call for ArrayA\*2+; ‘ArrayA’ returns the address of first item in that array so ArrayA[2] is the same as saying ArrayA+=2; and will return the third item in that array.

## **Pointer arithmetic**

- Pointers can be added and subtracted. However pointer arithmetic is quite meaningless unless performed on arrays. Addition and subtraction are mainly for moving forward and backward in an array.
- Note: you have to be very careful NOT to exceed the array elements when you use arithmetic; otherwise you will get horrible errors such as “access violation”.

Operator	Result
<code>++</code>	Goes to the next memory location that the pointer is pointing to.
<code>--</code>	Goes to the previous memory location that the pointer is pointing to.
<code>-= or -</code>	Subtracts value from pointer.
<code>+= or +</code>	Adding to the pointer

**Refer this Example:**

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

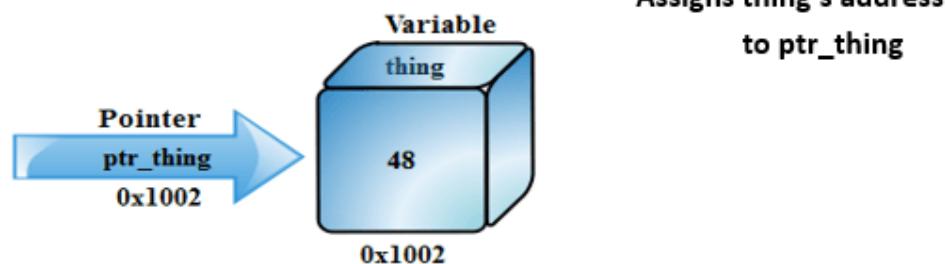
**Stack Using Pointer:**

**Refer this Example:**

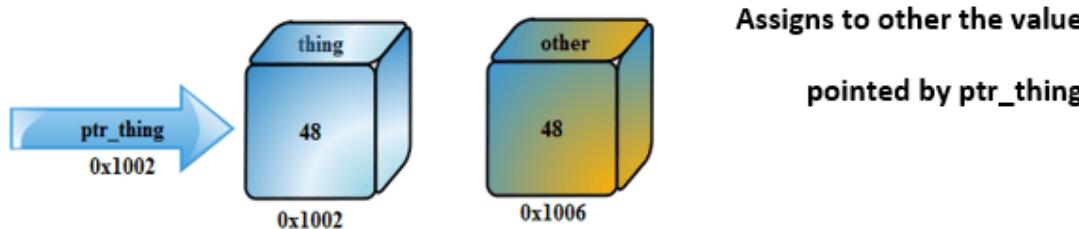
[C](https://github.com/TopsCode/Software-Engineering/blob/master/C/stackPointer.c)



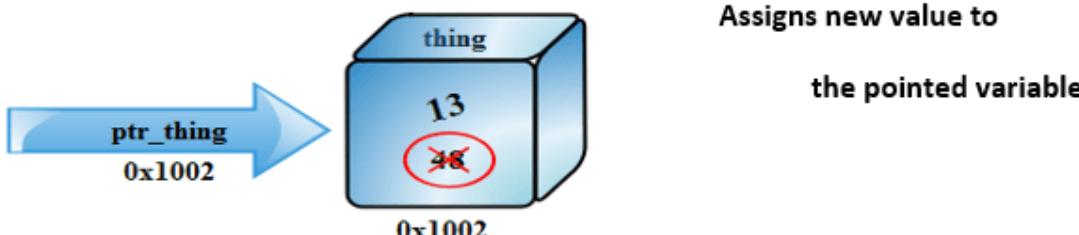
A) `ptr_thing = &thing;`



B) `other = *ptr_thing;`



C) `*ptr_thing=13;`



# Pointers and functions

- Pointers can be used with functions. The main use of pointers is ‘call by reference’ functions.
- Call by reference function is a type of function that has pointer/s (reference) as parameters to that function. All calculation of that function will be directly performed on referred variables.

**Refer this Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/C/pointerFunction.c>



- The arguments passed to function can be of two types namely
  1. Values passed
  2. Address passed
- The first type refers to call by value and the second type refers to call by reference.

### Refer this Example:

- 1) **Pass By Value:** [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)
- 2) **Pass By Reference:** [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)



# Pointer to arrays

- An array is actually very much like pointer. We can declare the arrays first element as `a[0]` or as `int *a` because `a[0]` is an address and `*a` is also an address the form of declaration is equivalent. The difference is pointer is a variable and can appear on the left of the assignment operator that is lvalue. The array name is constant and cannot appear as the left side of assignment operator.

Refer this Example: <https://github.com/TopsCode/Software-Engineering/blob/master/C/Pointers/pointerArray.c>

## Pointers and Structures

- However pointers and structures are great combinations. linked lists, stacks, queues and etc are all developed using pointers and structures in advanced systems.

Refer this Example: <https://github.com/TopsCode/Software-Engineering/blob/master/C/Pointers/pointerStruc.c>



# Dynamic Memory Allocation



# Dynamic memory allocation:

- Dynamic Memory Allocation can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.
- C provides some functions to achieve these tasks. There are 4 library functions provided by C defined under **<stdlib.h>** header file to facilitate dynamic memory allocation in C programming. They are:

**malloc** - Allocates memory requests size of bytes and returns a pointer to the 1st byte of allocated space

**calloc** - Allocates space for an array of elements initializes them to zero and returns a pointer to the memory

**free** - Frees previously allocated space

**realloc** - Modifies the size of previously allocated space



# Malloc:

- “malloc” or “memory allocation” method is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form.

```
ptr=(cast-type*)malloc(byte-size); Example:  
x=(int*)malloc(100*sizeof(int));
```

- Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory

## Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Dynamic%20MDynamic%20Memory%20Allocation/malloc.c>



## Calloc:

- “calloc” or “contiguous allocation” method is used to dynamically allocate the specified number of blocks of memory of the specified type. It initializes each block with a default value ‘0’.

```
ptr=(cast-type*) calloc(n,elem-size);
```

```
ptr = (float*) calloc(25, sizeof(float));
```

- This statement allocates contiguous space in memory for 25 elements each with the size of float.

## Free:

“free” method is used to dynamically de-allocate the memory.

```
free(ptr)
```



# realloc():

- “realloc” or “re-allocation” method is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to dynamically re-allocate memory.
- The general statement of reallocation of memory is :  
`ptr=realloc(ptr,newsize);`

## Refer this Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/C/Dynamic%20Memory%20Allocation/reallocation.c>



# File management in C



- C supports a number of functions that have the ability to perform basic file operations, which include:
  1. Naming a file
  2. Opening a file
  3. Reading from a file
  4. Writing data into a file
  5. Closing a file
- Real life situations involve large volume of data and in such cases, the console oriented I/O operations pose two major problems .
- It becomes cumbersome and time consuming to handle large volumes of data through terminals.
- The entire data is lost when either the program is terminated or computer is turned off therefore it is necessary to have more flexible approach where data can be stored on the disks and read whenever necessary, without destroying the data.



# File operation functions in C:

`fopen()` - Creates a new file for use   `Opens a new existing file for use`

`fclose()` - Closes a file which has been opened for use

`getc()` - Reads a character from a file   `putc()` - Writes a character to a file

`fprintf()` - Writes a set of data values to a file   `fscanf()` - Reads a set of data values from a file

`getw()` - Reads a integer from a file

`putw()` - Writes an integer to the file

`fseek()` - Sets the position to a desired point in the file   `ftell()` - Gives the current position in the file

`rewind()` - Sets the position to the begining of the file



# Defining and opening a file:

- If we want to store data in a file into the secondary memory, we must specify certain things about the file to the operating system. They include the filename, data structure, purpose.
- The general format of the function used for opening a file is

```
FILE *fp; fp=fopen("filename","mode");
```



# File Mode

---

**r** – Opens a file in read mode and sets pointer to the first character in the file

**w** – Opens a file in write mode. It returns null if file could not be opened. If file exists, data are overwritten.

**a** – Opens a file in append mode. It returns null if file couldn't be opened.

**r+** – Opens a file for read and write mode and sets pointer to the first character in the file.

**w+** – opens a file for read and write mode and sets pointer to the first character in the file.

**a+** – Opens a file for read and write mode and sets pointer to the first character in the file. But, it can't modify existing contents.



# Closing a file:

- The input output library supports the function to close a file; it is in the following format.

`fclose(file_pointer);`

- A file must be closed as soon as all operations on it have been completed. This would close the file associated with the file pointer.
- Observe the following program.  
`FILE *p1 *p2;  
p1=fopen ("Input","w");  
p2=fopen ("Output","r");  
....  
... fclose(p1); fclose(p2)`
- The above program opens two files and closes them after all operations on them are completed, once a file is closed its file pointer can be reversed on other file.

- The **getc** and **putc** functions are analogous to getchar and putchar functions and handle one character at a time.
- The putc function writes the character contained in character variable c to the file associated with the pointer fp1. ex putc(c,fp1); similarly getc function is used to read a character from a file that has been open in read mode. **c=getc(fp2).**
- The program shown below displays use of a file operations. The data enter through the keyboard and the program writes it. Character by character, to the file input. The end of the data is indicated by entering an EOF character, which is control+z. the file input is closed at this signal.

**Refer This Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/C/File%20Management/file1.c>



# The getw and putw functions:

These are integer-oriented functions. They are similar to get c and putc functions and are used to read and write integer values. These functions would be useful when we deal with only integer data. The general forms of getw and putw are:

`putw(integer,fp); getw(fp);`

**Refer this Example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/File%20Management/getPut.c>



# The fprintf & fscanf functions:

- The fprintf and scanf functions are identical to printf and scanf functions except that they work on files. The first argument of these functions is a file pointer which specifies the file to be used. The general form of fprintf is `fprintf(fp,"control string", list);`
- Where fp is a file pointer associated with a file that has been opened for writing. The control string is file output specifications list may include variable, constant and string.

```
fprintf(f1,"%s%d%f",name,age,7.5);
```

- Here name is an array variable of type char and age is an int variable
- The general format of fscanf is `fscanf(fp,"controlstring",list);`

This statement would cause the reading of items in the control string.

```
fscanf(f2,"5s%d",item,&quantity");
```

- Like scanf, fscanf also returns the number of items that are successfully read.



# Refer This Example

---

**Program to handle Mixed Data Type:**

<https://github.com/TopsCode/Software-Engineering/blob/master/C/File%20Management/mixedDataType.c>



## Random access to files:

- Sometimes it is required to access only a particular part of the file and not the complete file. This can be accomplished by using the following function:

## Fseek() function:

- The general format of fseek function is as follows: `fseek(file pointer,offset, position);`
- This function is used to move the file position to a desired location within the file. Fileptr is a pointer to the file concerned. Offset is a number or variable of type long, and position is an integer number. Offset specifies the number of positions (bytes) to be moved from the location specified by the position. The position can take the 3 values.

Value	Meaning
0	Begnning of the File.
1	Current Position.
2	End of the File.

# The Preprocessor



# The Preprocessor

- A unique feature of c language is the preprocessor. A program can use the tools provided by preprocessor to make his program easy to read, modify, portable and more efficient.
- Preprocessor is a program that processes the code before it passes through the compiler. It operates under the control of preprocessor command lines and directives. Preprocessor directives are placed in the source program before the main line before the source code passes through the compiler it is examined by the preprocessor for any preprocessor directives. If there is any appropriate actions are taken then the source program is handed over to the compiler.
- Preprocessor directives follow the special syntax rules and begin with the symbol **#** and do not require any semicolon at the end.



# Preprocessor directives:

- There are following types of preprocessor directories are available.
- Directive – Function

<u>Directive</u>	<u>Function</u>
<b>#define</b>	Defines a macro substitution
<b>#undef</b>	Undefine a macro
<b>#include</b>	Specifies a file to be included
<b>#ifdef</b>	Tests for macro definition
<b>#endif</b>	Specifies the end of #if
<b>#ifndef</b>	Tests whether the macro is not def
<b>#if</b>	Tests a compile time condition
<b>#else</b>	Specifies alternatives when # if test fails

- The preprocessor directives can be divided into three categories.
  1. Macro substitution division
  2. File inclusion division
  3. Compiler control division

## Macros:

- Macro substitution is a process where an identifier in a program is replaced by a pre defined string composed of one or more tokens we can use the #define statement for the task.
- It has the following form  
`#define identifier_string`
- The preprocessor replaces every occurrence of the identifier int the source code by a string. The definition should start with the keyword `#define` and should follow on identifier and a string with at least one blank space between them. The string may be any text and identifier must be a valid c name.
- There are different forms of macro substitution. The most common form is...
  1. Simple macro substitution
  2. Argument macro substitution
  3. Nested macro substitution



## Simple macro substitution:

- Simple string replacement is commonly used to define constants example:

```
#define PI 3.1415926
```

- Writing macro definition in capitals is a convention not a rule a macro definition can include more than a simple constant value it can include expressions as well. Following are valid examples:

```
#define AREA 12.36
```



# Macros as arguments:

- The preprocessor permits us to define more complex and more useful form of replacements it takes the following form.

```
#define identifier(f1,f2,f3....fn) string
```

- There is no space between identifier and left parentheses and the identifier f1,f2,f3 .... Fn is analogous to formal arguments in a function definition.
- There is a basic difference between simple replacement discussed above and replacement of macro arguments is known as a macro call
- A simple example of a macro with arguments is...

```
#define CUBE (x) (x*x*x)
```

- If the following statements appears later in the program,

```
volume=CUBE(side);
```

- The preprocessor would expand the statement to

```
volume =(side*side*side)
```



# Nesting of macros:

- We can also use one macro in the definition of another macro. That is macro definitions may be nested. Consider the following macro definitions

```
# define SQUARE(x) ((x)*(x))
```

- **Undefining a macro:**
- A defined macro can be undefined using the statement  
`# undef identifier`
- This is useful when we want to restrict the definition only to a particular part of the program.



# File inclusion:

- The preprocessor directive "#include file name" can be used to include any file in to your program if the functions or macro definitions are present in an external file they can be included in your file
- In the directive the filename is the name of the file containing the required definitions or functions alternatively the this directive can take the form

#include< filename >

- Without double quotation marks. In this format searched in only standard directories.  
the file will be
- The c preprocessor also supports a more general form of test condition #if directive. This takes the following form

#if constant expression

```
{  
statement-1;  
statemet2'  
....  
....  
}  
#endif (Cont.)
```



- The constant expression can be a logical expression such as test < = 3 etc
- If the result of the constant expression is true then all the statements between the #if and #endif are included for processing otherwise they are skipped. The names TEST LEVEL etc., may be defined as macros.



# Module 4 Topics:

- **Programming OOPs**
  - OOPs Introduction and Features
    - Class and Object
    - Functions
    - Constructor and Destructor
    - Types of Inheritance
    - Memory Management and Command line Arguments
    - Template
    - Working with files.



# History

- C++ is an object-oriented extension of C
- C was designed by Dennis Ritchie at Bell Labs
- Used to write Unix, based on BCPL
- C++ designed by Bjarne Stroustrup at Bell Labs
- His original interest at Bell was research on simulation
- Early extensions to C are based primarily on Simula
- Called “C with classes” in early 1980’s
- Popularity increased in late 1980’s and early 1990’s
- Features were added incrementally
- Classes, templates, exceptions, multiple inheritance, type tests...



- Conventional programming, using high level language such as COBOL, Fortran and commonly known as **procedure-oriented programming (POP)**.
- In the procedure oriented approach, the problem is viewed as a sequence of things to be done such as reading, calculations and printing.
- OOP treats data as a critical element in the program development and does not allow free Around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions.
- OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.
- Object oriented programming as an approach that provides a way of modularizing programs by creating partitioned memroy area for both data and functions that can be used as templates for creating copies of such modules on demand.



# Characteristics of OOP

- Emphasis is on data rather than procedure
- Programs are divided into what are known as objects
- Data and Functions are tied together in the data structure
- Data is hidden and can not be accessed by external functions
- Objects may communicate with each other through functions
- New data and functions can be easily added whenever necessary
- Bottom-up approach



# Features of Object – Oriented Programming

- Objects
- Classes
- Data abstraction and Encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message Passing



# Structure of a C++ Program

```
// my first program in C++  
  
#include <iostream.h> using namespace std;  
  
int main ()  
{  
cout << "Hello World!"; return 0;  
}  
--- Output --- Hello World!
```



## // my first program in C++

- This is a comment line. All lines beginning with two slash signs (//) are considered comments

## #include <iostream>

- In this case the directive #include <iostream> tells the preprocessor to include the iostream standard file.
- This specific file (iostream) includes the declarations of the basic standard input-output library in C++,

## using namespace std;

- All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name std.

## int main ()

- The main function is the point by where all C++ programs start their execution, independently of its location within the source code.



```
cout << "Hello World!";
```

- cout represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the Hello World sequence of characters) into the standard output stream (which usually is the screen).

```
return 0;
```

- The return statement causes the main function to finish. return may be followed by a return code (in our example is followed by the return code 0).
- A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program.

The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. For example, instead of

```
int main ()  
{  
    cout << " Hello World!"; return 0;  
}
```

We could have written:

```
int main () { cout << "Hello World!"; return 0; }
```

- All in just one line and this would have had exactly the same meaning as the previous code.
- In C++, the separation between statements is specified with an ending semicolon (;) at the end of each one, so the separation in different code lines does not matter at all for this purpose.

# Comments

C++ supports two ways to insert comments:

```
// line comment  single line  
/* block comment */ multi line
```



# KEYWORDS

- Keywords are the reserved words in any language which have a special pre defined meaning and cannot be used for any other purpose. You cannot use keywords for naming variables or some other purpose.
- We saw the use of keywords main, include, return, int in our first C++ program.

# IDENTIFIERS

- Identifiers are the name of functions, variables, classes, arrays etc. which you create while writing your programs.
- Identifiers are just like names in any language. There are certain rules to name identifiers in C++.



# Identifiers Rules:

- Identifiers must contain combinations of digits, characters and underscore (\_).
- Identifier names cannot start with digit.
- Keyword cannot be used as an identifier name and upper case and lower case are distinct.
- Identifier names can contain any combination of characters as opposed to the restriction of 31 letters in C.

# CONSTANTS

- Constants are fixed values which cannot change. For example 123, 12.23, 'a' are constants.



# Declaration of variables

- Syntax:
  - Data\_type variable\_name
- For example:

```
int a;  
float mynumber;
```

```
int a, b, c;
```

**Refer this Example:**

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



# The scope of the Variable:

- Local Variables
- Global Variables

## Local Variables:

- The variables defined local to the block of the function would be accessible only within the block of the function and not outside the function. Such variables are called local variables.
- Scope of the local variables is limited to the function in which these variables are declared.

Refer this example:

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



# Global Variables:

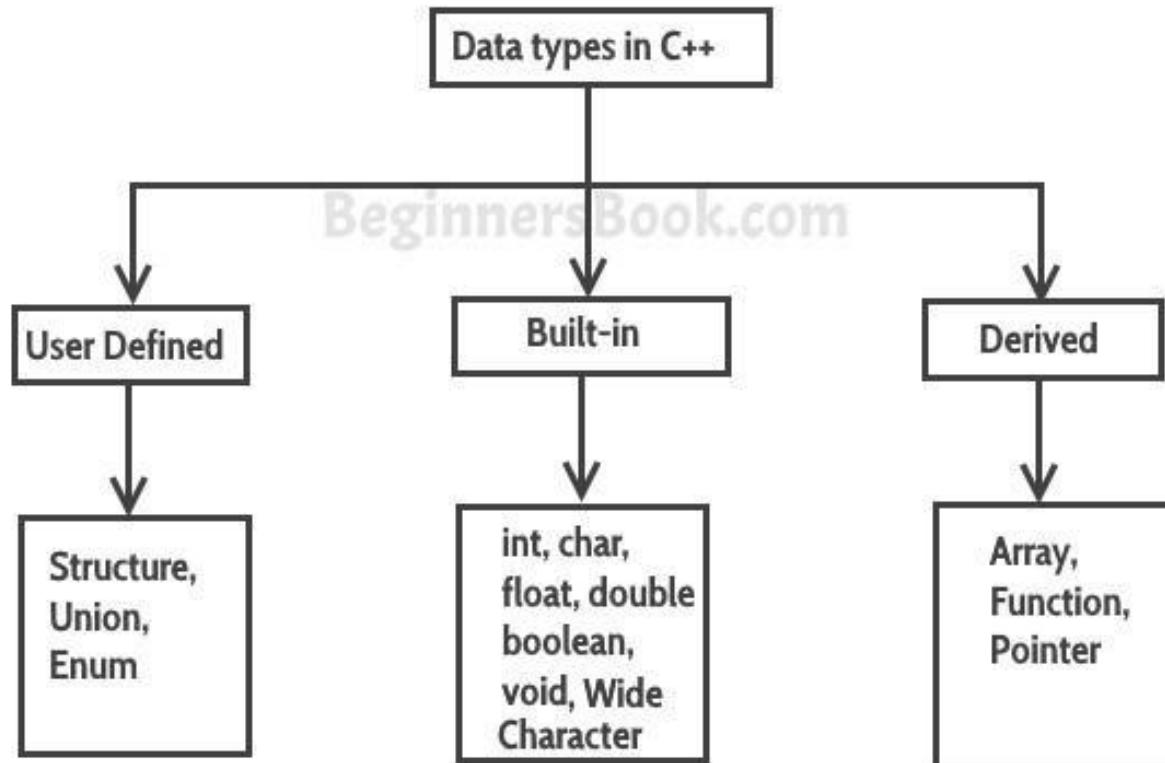
- Global variables are one which are visible in any part of the program code and can be used within all functions and outside all functions used in the program.
- The method of declaring global variables is to declare the variable outside the function or block.

## Refer this Example:

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



# Data Type



# Integer Data type

---

Integer Data Types

Type	Size(in bytes)	Range
int	2	-32768 to 32767
signed int	2	-32768 to 32767
unsigned int	2	0 to 65535
shortint	2	-32768 to 32767
signed short int	2	-32768 to 32767
unsigned short int	2	-32768 to 32767
longint	4	-2147483648 to 2147483647
signed longint	4	-2147483648 to 2147483647
unsigned longint	4	0 to 4294967295

# Float Data type:

Floating Point Data Types

Type	Size(in bytes)	Range	Digits of Precision
float	4	$3.4 \cdot 10^{-38}$ to $3.4 \cdot 10^{38}$	7
double	8	$1.7 \cdot 10^{-308}$ to $1.7 \cdot 10^{308}$	15
long double	10	$3.4 \cdot 10^{-4932}$ to $3.4 \cdot 10^{4932}$	18

# Character data type

Character Data Types

Type	Size (in bytes)	Range
char	1	- 128 to 127
Signed char	1	- 128 to 127
unsigned char	1	0 to 255

# Type Casting

The C-style casting takes the syntax as....

(type) expression

C++-style casting is as below namely: type (expression)

Refer this Example:

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



# C++ Storage Classes

**Automatic, External and Static.**

## What is Storage Class?

- Storage class defined for a variable determines the accessibility and longevity of the variable.
- The accessibility of the variable relates to the portion of the program that has access to the variable. The longevity of the variable refers to the length of time the variable exists within the program.

### Automatic:

- Variables defined within the function body are called automatic variables. **auto** is the keyword used to declare automatic variables.
  - By default and without the use of a Keyword, the variables defined inside a function are automatic variables.



# External

- External variables are also called global variables.
- External variables are defined outside any function, memory is set aside once it has been declared and remains until the end of the program.
- These variables are accessible by any function. This is mainly utilized when a programmer wants to make use of a variable and access the variable among different function calls.

# Static

- The static automatic variables, as with local variables, are accessible only within the function in which it is defined.
- Static automatic variables exist until the program ends in the same manner as external variables. In order to maintain value between function calls, the static variable takes its presence.



# C++ Character Functions

- The C++ char functions are extremely useful for testing and transforming characters.
- These functions are widely used and accepted. In order to use character functions header file <cctype> is included into the program.
- Some of the commonly used character functions are:

**isalnum()**- The function isalnum() returns nonzero value if its argument is either an alphabet or integer. If the character is not an integer or alphabet then it returns zero.

**isalpha()** - The function isalpha() returns nonzero if the character is an uppercase or lower case letter otherwise it returns zero.

**iscntrl()** - The function iscntrl() returns nonzero if the character is a control character otherwise it returns zero.

**isdigit()**- The function isdigit() returns nonzero if the character is a 0 to 9. It returns zero for non digit character.



**isgraph()**- The function isgraph() returns nonzero if the character is any printable character other than space otherwise it returns zero.

**islower()**- The function islower() returns nonzero for a lowercase letter otherwise it returns zero.

**isprint()**- The function isprint() returns nonzero for printable character including space otherwise it returns zero.

**isspace()**- The function isspace() returns nonzero for space, horizontal tab, newline character, vertical tab, formfeed, carriage return; otherwise it returns zero.

**ispunct()**- The function ispunct() returns nonzero for punctuation characters otherwise it returns zero. The punctuation character excludes alphabets, digits and space.



**isupper()**- The function `isupper()` returns nonzero for an uppercase letter otherwise it returns zero.

**tolower()**- The function `tolower()` changes the upper case letter to its equivalent lower case letter. The character other than upper case letter remains unchanged.

**toupper()**- The function `toupper()` changes the lower case letter to its equivalent upper case letter. The character other than lower case letter remains unchanged.

**isxdigit()**- The function `isxdigit()` returns nonzero for hexadecimal digit i.e. digit from 0 to 9, alphabet 'a' to 'f' or 'A' to 'F' otherwise it returns zero.



# C++ Functions



- A function is a block of code which only runs when it is called.
  - You can pass data, known as parameters, into a function.
  - Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.
- 
- **Refer This Example:**
  - <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Function/simpleFunction.cpp>

- A C++ function definition consists of a function header and a function body. Here are all the parts of a function:

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

- **Return Type:** The return\_type is the data type of the value the function returns. When function returns no value void is written
- **Function Name:** This is the actual name of the function.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. Parameters are optional; that is, a function may contain no parameters.
- **Function Body:**
- The function body contains a collection of statements that define what the function does.

# Inline Functions

## What is Inline Function?

- Inline functions are functions where the call is made to inline functions. The actual code then gets placed in the calling program.

**Refer this Example:** <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Function/inline.cpp>

# C++ function call by value

The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

**Refer this Example:**

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



# C++ function call by reference

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

**Refer this Example:**

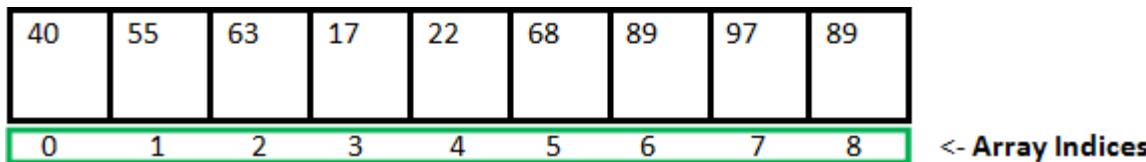
<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Function/callByReference.cpp>



# Array



- An array is a collection of several data items of the same data type stored in contiguous memory locations.



Array Length = 9

First Index = 0

Last Index = 8

- Array Declaration by specifying size:

Datatype array\_name[size];

int age [10];

- Array Declaration by initializing elements:

int arr[] = { 10, 20, 30, 40 }

- Array declaration by specifying size and initializing elements

int arr[5] = { 10, 20, 30, 40 }

# Refer This Example

---

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

# MULTIDIMENSIONAL ARRAY

The multidimensional arrays are arrays of arrays. The general form of a multidimensional array is: -

```
Type array_name[Size1][Size2]..[Size n];
```

The two dimensional array can be declared as ...

```
int age[2][5];
```

This array has two rows and 5 columns. The three dimensional array can be declared as

```
int age[2][3][5];
```

- When an element of the array of n dimensional is referenced it uses n index values.
- For example first element of a two dimensional array declared above is referred as age[0][0] as the index starts from zero and last element is referred as age[1][4]. Here is a program which calculates the average of the elements of the row of the two dimensional array.

## Example of 2D array

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



# Null Terminated String

- The most common use of one dimensional array is for character strings.
- The null terminated string is a character array with a null character at the end. The characters form the string with a null character indicating the termination of the string. A null terminated string can be declared as...

```
char name[10];
```

- It will hold 10 characters with a null at the end. The size of the array can be more than the length of the array. The array can be initialized as
- The first 5 elements are initialized and rest elements are null characters. Here is a program which calculates the length of the string.

```
char name[10]={'j','e','s','u','s'};
```

Refer this Example: <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/String/length.cpp>

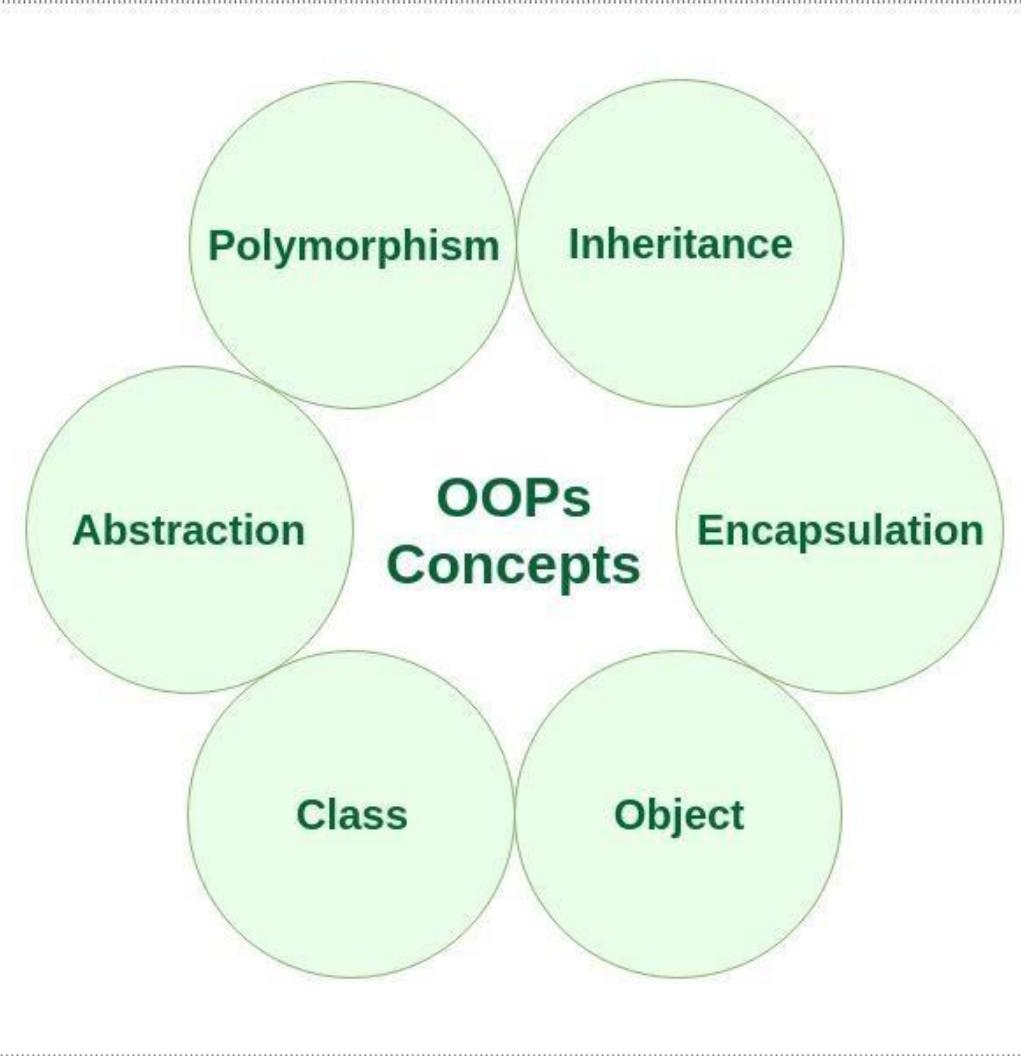
# OOPS



# Introduction

- Object-oriented programming is a paradigm in programming that represents real-life objects or entities in code,
- Some Language that supports OOPS:  
Java, J2EE, C++, C#, Visual Basic.NET, Python and JavaScript
- OOP is widely accepted as being far more flexible than other computer programming languages.
- The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

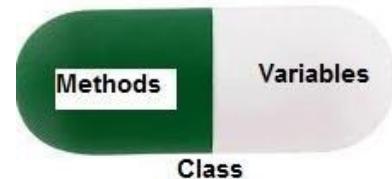




## **Encapsulation:**

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.

Encapsulation in C++



## **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

- 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 C++ 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);
```

# 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;
    }
};
```



# 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.

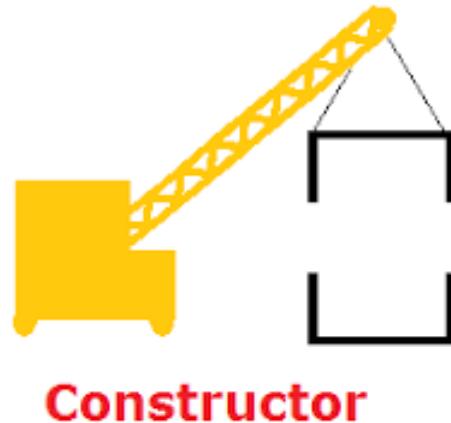
You must supply the arguments to the constructor when a new instance is created.

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



Bike b=new Bike();

# 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-](https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/Constructor/DynamicConstructor.cpp)

[Engineering/blob/master/c%2B%2B/OOPS/Constructor/DynamicConstructor.cpp](https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/Constructor/DynamicConstructor.cpp)



# Virtual Constructor

- C++ being static typed (the purpose of RTTI is different) language, it is meaningless to the C++ compiler to create an object polymorphically.
- The compiler must be aware of the class type to create the object. In other words, what type of object to be created is a compile time decision from C++ compiler perspective. If we make constructor virtual, compiler flags an error. In fact except inline, no other keyword is allowed in the declaration of constructor.
- In practical scenarios we would need to create a derived class object in a class hierarchy based on some input. Putting in other words, object creation and object type are tightly coupled which forces modifications to extended. The objective of virtual constructor is to decouple object creation from its type.

Refer This Example:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/Constructor/VirtualCons>



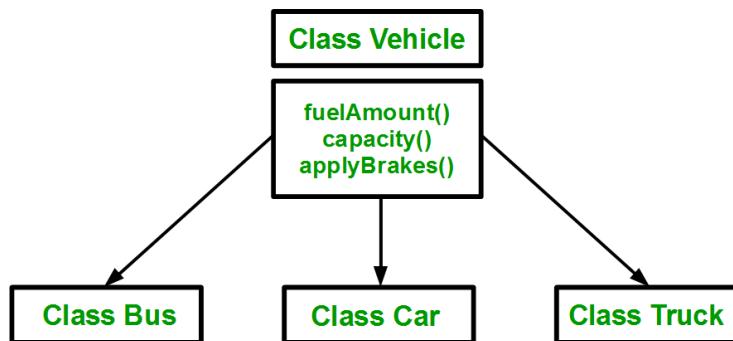
# What is a Destructor?

- 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:
  - (i) the same name as the class but is preceded by a tilde (~)
  - (ii) no arguments and return no values

# Inheritance

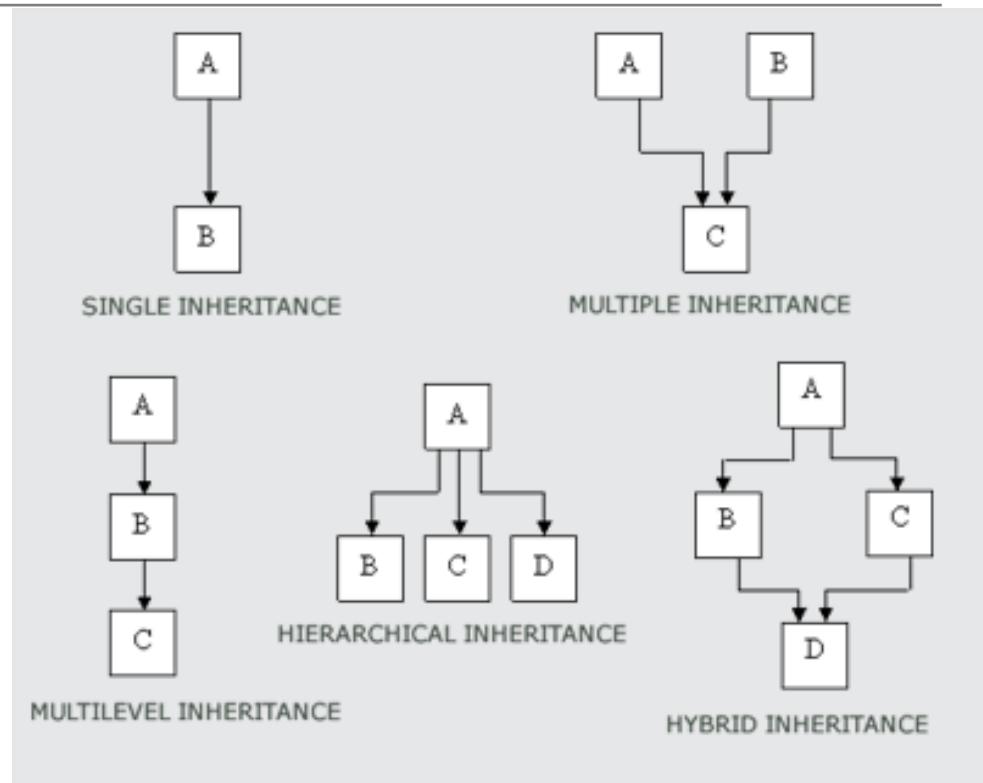
---

- 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 Inheritcance
- Hierarchical Inheritance
- Multilevel Inheritance
- Hybrid Inheritance



# Refer this Example

---

**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>

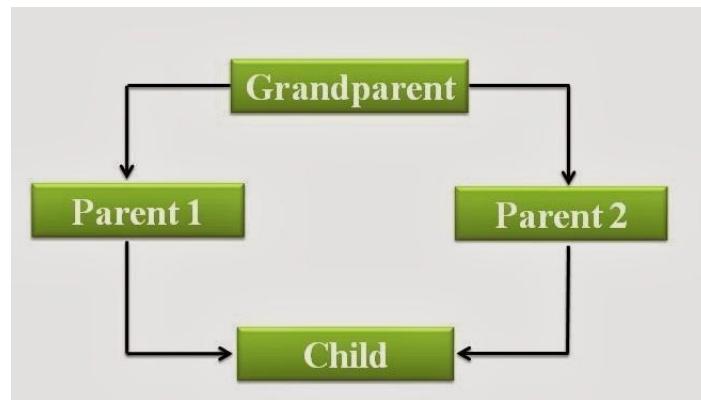
**Hybrid Inheritance**

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



# 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>

# The this pointer (C++ only)

---

- Every object in C++ has access to its own address through an important pointer called this pointer.
- The **this** pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.
- **Refer This Example:**
- <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/class/thisPointer.cpp>

---

# 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:
  - **Static** : compile time
  - **Dynamic** : run time

# Static Polymorphism

---

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%2BOOPS/FunctionOverloading/example1.cpp>

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



# Refer This Example

---

## Unary Operator Overloading:

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

## Binary Operator Overloading:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2BOOPS/OperatorOverloadiing/binar.y.c>



# Dynamic Polymorphism

---

- 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](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>

# Friend Function

A friend function is used for accessing the non-public members of a class.

A class can allow non-member functions and other classes to access its own private data, by making them friends. Thus, a friend function is an ordinary function or a member of another class.

## How to define and use Friend Function in C++:

- The friend function is written as any other normal function, except the function declaration of these functions is preceded with the keyword friend.
- The friend function must have the class to which it is declared as friend passed to it in argument.



## **Some important points to note while using friend functions in C++:**

- The keyword friend is placed only in the function declaration of the friend function and not in the function definition.

It is possible to declare a function as friend in any number of classes.

When a class is declared as a friend, the friend class has access to the private data of the class that made this a friend.

A friend function, even though it is not a member function, would have the rights to access the private members of the class.

It is possible to declare the friend function as either private or public.

The function can be invoked without the use of an object. **Refer This Example:**

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/OOPS/Function/FriendFunction.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 in C++ 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%2B%2B/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>



# File in C++

- C++ provides the following classes to perform output and input of characters to/from files:
- **ofstream**: Stream class to write on files
- **ifstream**: Stream class to read from files
- **fstream**: Stream class to both read and write from/to files.

## Operations in File Handling:

- Creating a file: `open()`
- Reading data: `read()`
- Writing new data: `write()`
- Closing a file: `close()`

## Opening a File:

•A file must be opened before you can read from it or write to it. Either the ifstream or fstream object may be used to open a file for writing and ifstream object is used to open a file for reading purpose only.

•Syntax:

```
void open(const char *filename, ios::openmode mode);
```

Sr.No	Mode Flag & Description
1	<b>ios::app</b> Append mode. All output to that file to be appended to the end.
2	<b>ios::ate</b> Open a file for output and move the read/write control to the end of the file.
3	<b>ios::in</b> Open a file for reading.
4	<b>ios::out</b> Open a file for writing.
5	<b>ios::trunc</b> If the file already exists, its contents will be truncated before opening the file.

## Writing to a File:

- For writing in a file we use **ios::out**.
- Syntax
  - `FilePointer << " Lines"`

## Reading from a File:

- To read from a file use **ios::in**.
- information from the output file is obtained with the help of following syntax
  - `FilePointer >>variable`

Refer This Example: [https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/File/Read\\_Write.cpp](https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/File/Read_Write.cpp)

# File Position Pointers:

- Both **istream** and **ostream** provide member functions for repositioning the file-position pointer.
- These member functions are **seekg** ("seek get") for istream and **seekp** ("seek put") for ostream.
- **seekg()** - for get pointer
- **seekp()** - for put pointer
- These functions take two arguments:
  - The first argument is the relative offset i.e. the number of bytes the file pointer has to be moved. (+ for forward and - for backward.).
  - The second argument is the position of the file pointer from where the offset is to be considered. The default argument for this is the beg (beginning of the file). It can take values `ios::beg` (beginning), `ios::end` (end of file), and `ios::cur` (current pointer position).

# Refer This Example

---

Seek:

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



# String in C++

C++ provides following two types of string representations:

- 1.The C-style character string.
- 2.The string class type introduced with Standard C++.

**Refer This Example:**

C-style String: <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/String/cString.cpp>



# C++ supports a wide range of functions that manipulate null-terminated strings:

S.N.	Function & Purpose
1	<b>strcpy(s1, s2);</b> Copies string s2 into string s1.
2	<b>strcat(s1, s2);</b> Concatenates string s2 onto the end of string s1.
3	<b>strlen(s1);</b> Returns the length of string s1.
4	<b>strcmp(s1, s2);</b> Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	<b>strchr(s1, ch);</b> Returns a pointer to the first occurrence of character ch in string s1.
6	<b>strstr(s1, s2);</b> Returns a pointer to the first occurrence of string s2 in string s1.

# Refer This Example

---

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



# The String Class in C++:

- The standard C++ library provides a string class type that supports all the operations mentioned above, additionally much more functionality. We will study this class in C++ Standard Library but for now let us check following example:
- At this point you may not understand this example because so far we have not discussed Classes and Objects. So can have a look and proceed until you have understanding on Object Oriented Concepts.
- **Refer This Example:**
- <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/String/stringClass.cpp>

# C++ Memory Management operators

---

## Need for Memory Management operators

- The concept of arrays has a block of memory reserved. The disadvantage with the concept of arrays is that the programmer must know, while programming, the size of memory to be allocated in addition to the array size remaining constant.

## What are memory management operators?

- There are two types of memory management operators in C++:
  1. new
  2. delete



## New operator:

- The new operator in C++ is used for dynamic storage allocation. This operator can be used to create object of any type.

**Syntax:** pointer variable = new datatype;

**Example:** int \*a=new int;

## Delete Operator:

The delete operator in C++ is used **for releasing memory space** when the object is no longer needed. Once a new operator is used, it is efficient to use the corresponding delete operator for release of memory

**Syntax:** delete pointer\_variable;

**Example:** delete a;

# Refer This Example

---

## New And Delete Operator:

<https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Memory%20Management%20Operator/Example1.cpp>



# Templates

---

- Templates are powerful features of C++ which allows you to write generic programs.
- In simple terms, you can create a single function or a class to work with different data types using templates.
- Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs.

The concept of templates can be used in two different ways:

1. Function Templates
2. Class Templates



# Function Template

---

Function Template can work with different data types at once.

A function template defines a family of functions.

## Syntax:

```
template <class type> ret-type func-name(parameter list) {  
    // body of function  
}
```

## Refer This Example:

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



# Class Template

---

Class templates are useful when a class defines something that is independent of the data type. Can be useful for classes like `LinkedList`, `BinaryTree`, `Stack`, `Queue`, `Array`, etc.

A class template defines a family of classes.

Syntax:

```
template <class type> class class-name {  
    .  
    .  
    .  
}
```



# Refer This Example

---

## Template Class:

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

## Function Template with Multiple Parameter:

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

# Command Line Argument

---

- In C & C++, it is possible to accept command line arguments.
- To use command line argument , main function accepts two arguments:
  - one argument is number of command line arguments,
  - and the other argument is a full list of all of the command line arguments.
- **Refer This Example:**
- <https://github.com/TopsCode/Software-Engineering/blob/master/c%2B%2B/Command%20Line%20Argument/cm d.cpp>



# Module 5 Topics:

- DBMS and RDBMS
- E-R Relational Schema
- Types of database
- Normalization
- Algebra
- Database Programming language SQL
- Keys: Primary Key, Unique Key, Foreign Key
- SQL Statement types: DML, DDL, TQL, TCL
- Joins
- Function
- Procedure
- Trigger
- Transaction Concept (Properties, Rollback, Commit, Save Point)
- Cursor
- Database Backup and Recovery

# 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.

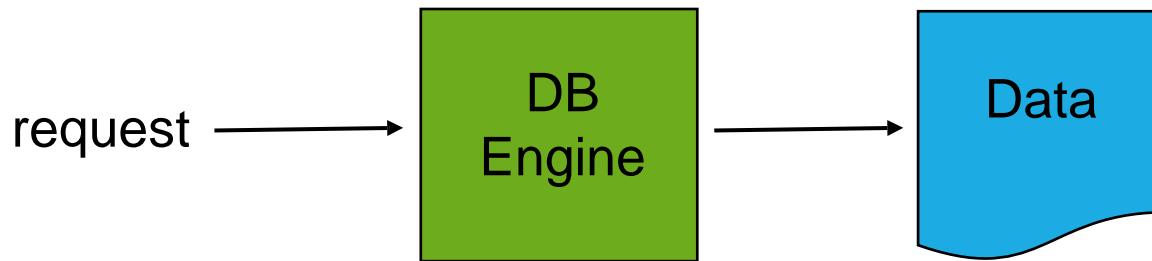
For Example, university database organizes the data about students, faculty, and admin staff etc. which helps in efficient retrieval, insertion and deletion of data from it.



# Database Management Systems

---

- A DBMS consists of 2 main pieces:
  - the data
  - the DB engine
  - the data is typically stored in one or more files

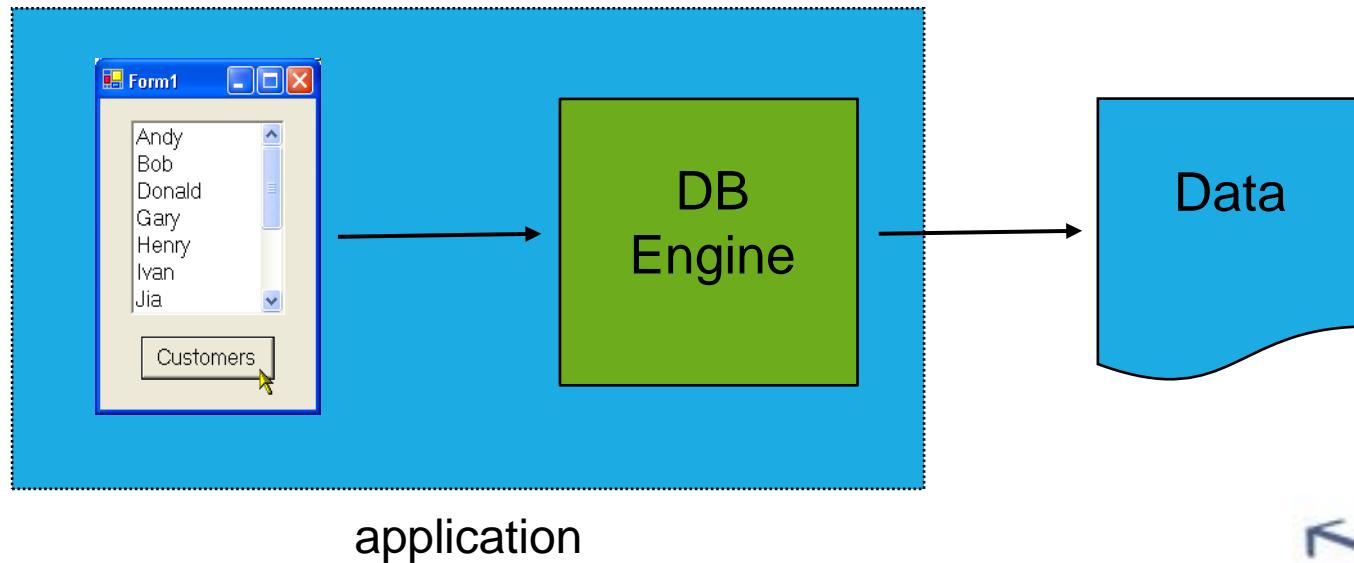


- Two most common types of DBMS are:
  - Local
  - Server

# Local DBMS

---

- A local DBMS is where DB engine runs as part of application
- Example?
  - MS Access
  - underlying DB engine is JET ("Joint Engine Technology")



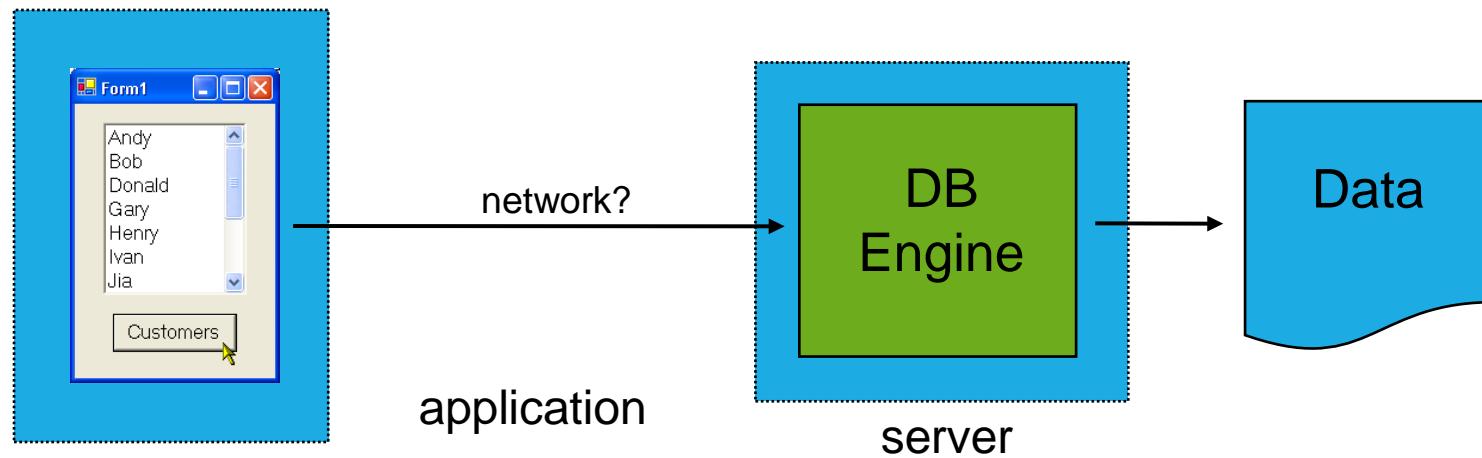
# Server DBMS

A server DBMS is where DB engine runs as a separate process

- typically on a different machine (i.e. server)

Examples?

- MS SQL Server, Oracle, DB2, MySQL



# Popular DBMS Software

---

Here, is the list of some popular DBMS system:

- MySQL
- Microsoft Access
- Oracle
- PostgreSQL
- dBASE
- FoxPro
- SQLite
- IBM DB2
- LibreOffice Base
- MariaDB
- Microsoft SQL Server etc.



# What is the 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.



# PURPOSE OF DBMS

---

The main purpose of database systems is to manage the data.

Consider a university that keeps the data of students, teachers, courses, books etc. To manage this data we need to store this data somewhere where we can add new data, delete unused data, update outdated data, retrieve data, to perform these operations on data we need a Database management system that allows us to store the data in such a way so that all these operations can be performed on the data efficiently.

# RDBMS

---

Stands for "Relational Database Management System."

An RDBMS is a type of DBMS designed specifically for relational databases.

A relational database refers to a database that stores data in a structured format, using rows and columns.

This makes it easy to locate and access specific values within the database.

It is "relational" because the values within each table are related to each other.

Tables may also be related to other tables. The relational structure makes it possible to run queries across multiple tables at once.

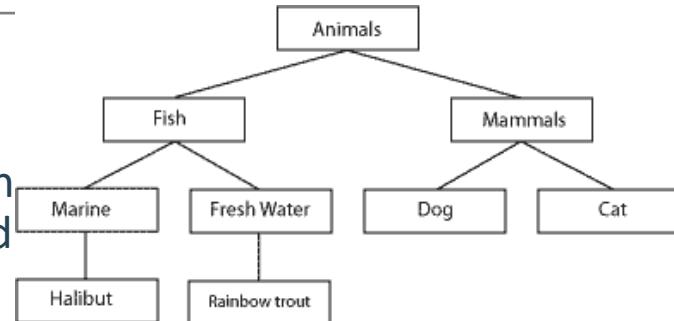
The RDBMS refers to the software that executes queries on the data, including adding, updating, and searching for values.

An RDBMS may also provide a visual representation of the data. For example, it may display data in a table like a spreadsheet, allowing you to view and even edit individual values in the table.

# Types of DBMS

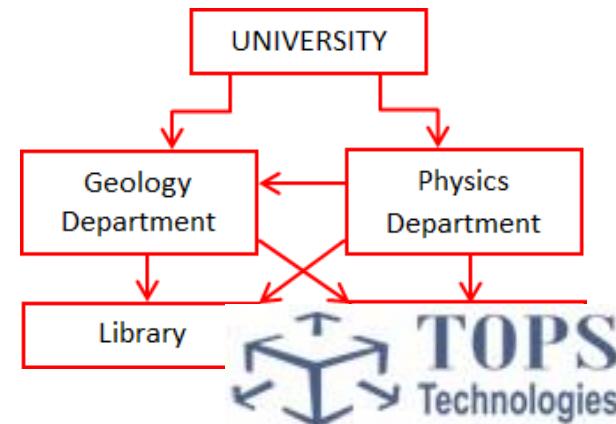
## 1. Hierarchical databases -

- It is very fast and simple.
- This kind of database model uses a tree-like structure which links a number of dissimilar elements to one primary record – the "owner" or "parent".
- Each record in a hierarchical database contains information about a group of parent child relationships.



## 2. Network databases –

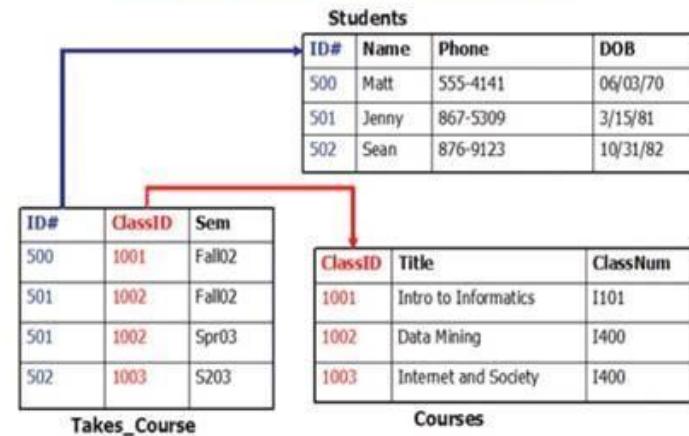
- The network database model can be viewed as a net-like form where a single element can point to multiple data elements and can itself be pointed to by multiple data elements.
- The network database model allows each record to have multiple parents as well as multiple child records, which can be visualized as a web-like structure of networked records.



### 3. Relational databases –

- A relational database is one in which data is stored in the form of tables, using rows and columns.
- This arrangement makes it easy to locate and access specific data within the database. It is “relational” because the data within each table are related to each other.

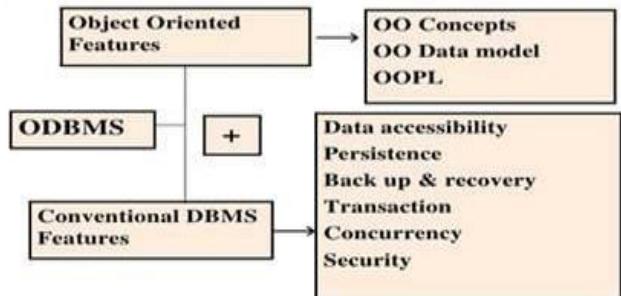
### Relational DBMS



### 4. Object-oriented databases -

- The recent development in database technology is the incorporation of the object concept that has become significant in programming languages.
- In object-oriented databases, all data are objects. Objects may be linked to each other by an “is-part-of” relationship to represent larger, composite objects
- Object DBMS's increase the semantics of the C++ and Java

### ODBMS



# Relational Databases

---

RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

Most of today's databases are relational:

- database contains 1 or more tables
- table contains 1 or more records
- record contains 1 or more fields
- fields contain the data

So why is it called "relational"?

- tables are related (joined) based on common fields



# E-R Model

---

The ER or (Entity Relational Model) is a high-level conceptual data model diagram.

Entity-Relation model is based on the notion of real-world entities and the relationship between them.

ER modeling helps you to analyze data requirements systematically to produce a well-designed database.

So, it is considered a best practice to complete ER modeling before implementing your database.

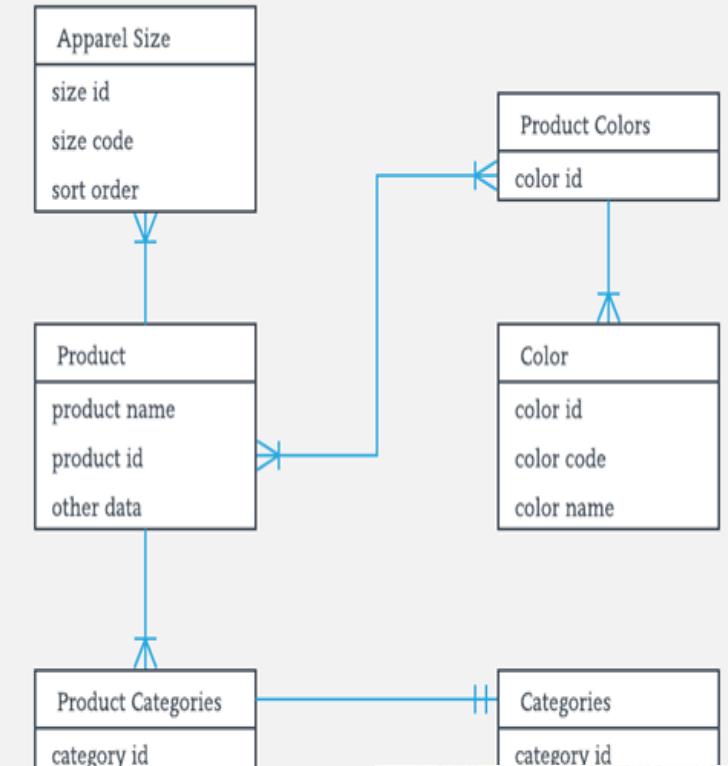


# ER Diagram

Entity relationship diagram displays the relationships of entity set stored in a database.

In other words, we can say that ER diagrams help you to explain the logical structure of databases.

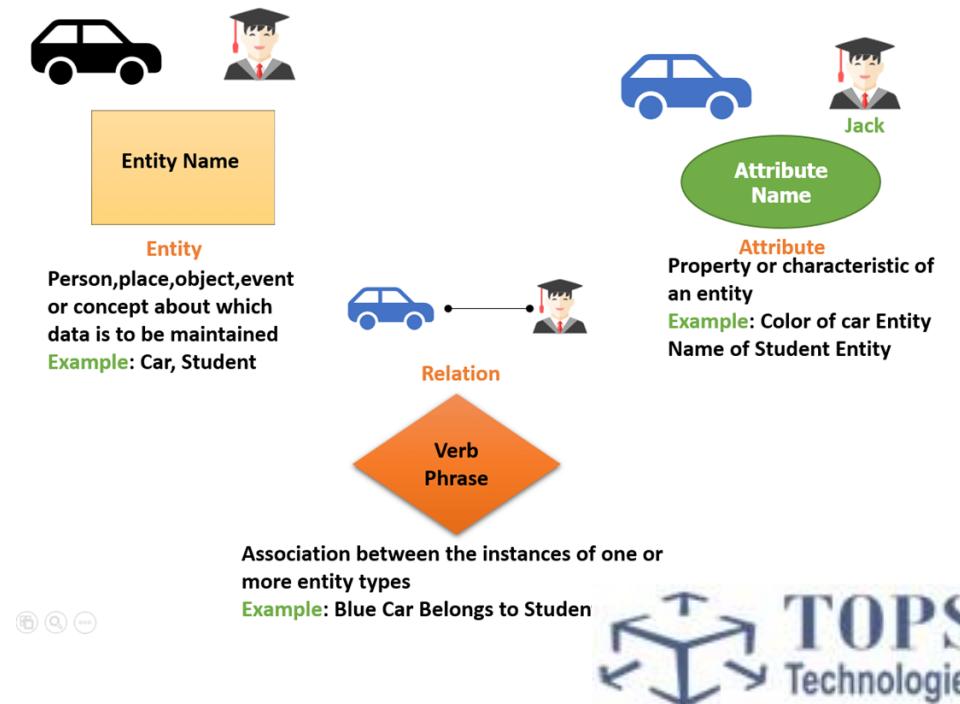
At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique.



# Components of ER Diagram

## Entities Attributes Relationships

For example, in a University database, we might have entities for Students, Courses, and Lecturers. Students entity can have attributes like Rollno, Name, and DeptID. They might have relationships with Courses and Lecturers.



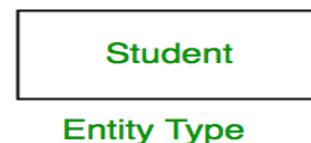
# Entity

---

A real-world thing either living or non-living that is easily recognizable and non recognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world.

An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attribute' which represent that entity.

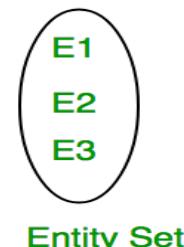
Set of all entity is called **entity set**



Examples of entities:

**Person:** Employee, Student, Patient

**Place:** Store, Building



# Attribute

---

Attributes are the properties which define the entity type. For example, Roll\_No, Name, DOB, Age, Address, Mobile\_No are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.

Attribute

## 1. Key Attribute:

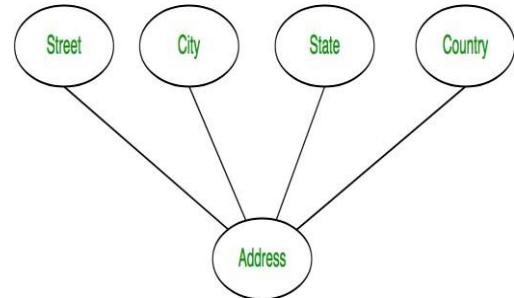
The attribute which uniquely identifies each entity in the entity set is called key attribute.

For example, Roll\_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

Roll\_No

## 2. Composite Attribute –

An attribute **composed of many other attribute** is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals



## 3. Multivalued Attribute –

An attribute consisting more than one value for a given entity. For example, Phone\_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.

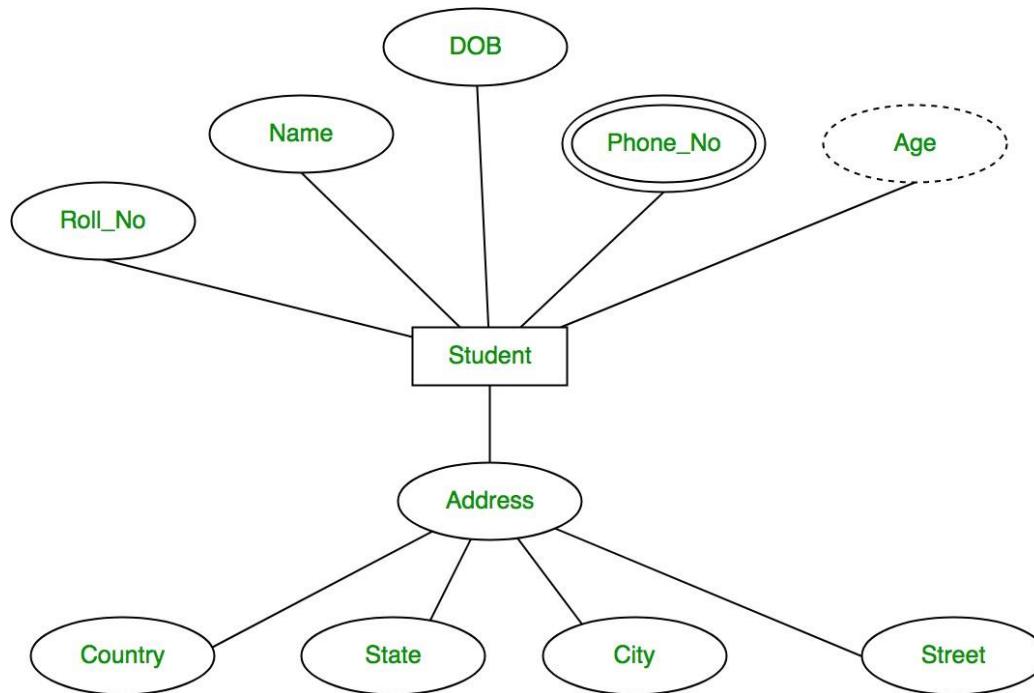


## 4. Derived Attribute –

An attribute which can be derived from other attributes of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

The complete entity type Student with its attributes can be represented as:

---

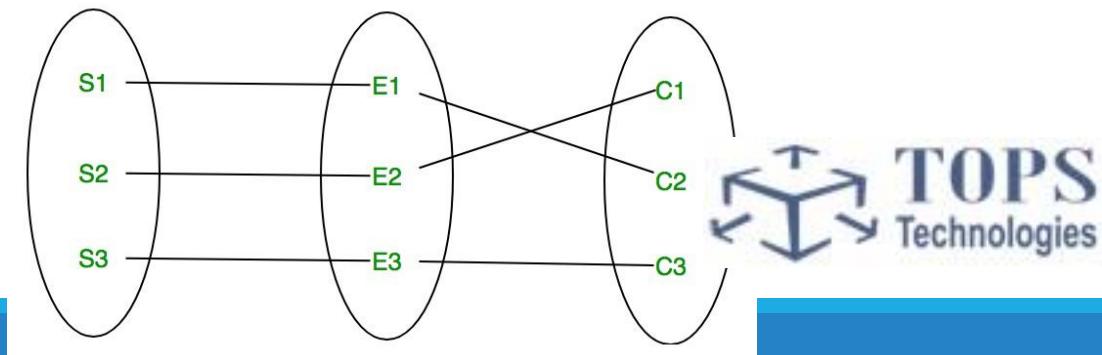


# Relationship Type and Relationship Set:

A relationship type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.



A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.

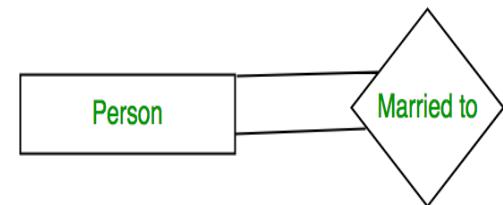


# Degree of a relationship set:

---

## 1. Unary Relationship –

When there is only ONE entity set participating in a relation, the relationship is called as unary relationship. For example, one person is married to only one person.



## 2. Binary Relationship –

When there are TWO entities set participating in a relation, the relationship is called as binary relationship. For example, Student is enrolled in Course.



## 3. n-ary Relationship –

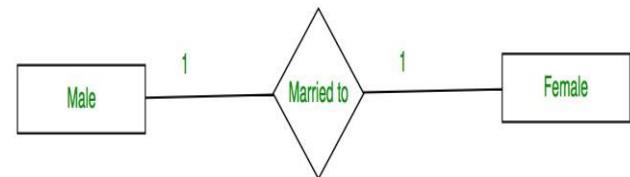
When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

# Cardinality:

---

The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

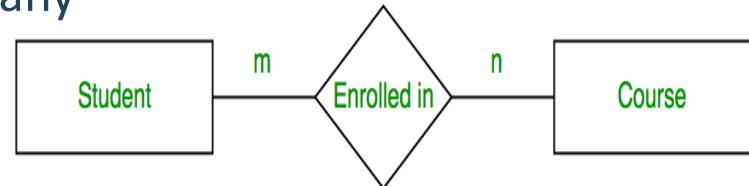
**One to one** – When each entity in each entity set can take part only once in the relationship, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.



**Many to one** – When entity set can take part only on entities in once in the relationship set and entit other entity set can take part more than once i relationship set, cardinality is many to one.



**Many to many** – When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.



### Participation Constraint:

Participation Constraint is applied to the relationship set.

**Total Participation** – Each entity must participate in the relationship.

If each student must enroll in a course, the participation of student will be total.

Total

participation is shown by double line in ER diagram.

**Partial Participation** – The entity in the entity set may or may NOT participate in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial. The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation

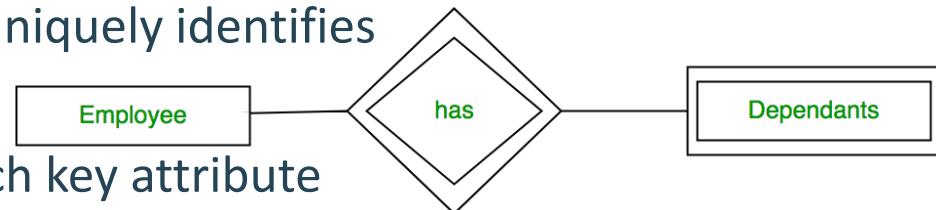
## Weak Entity Type and Identifying Relationship:

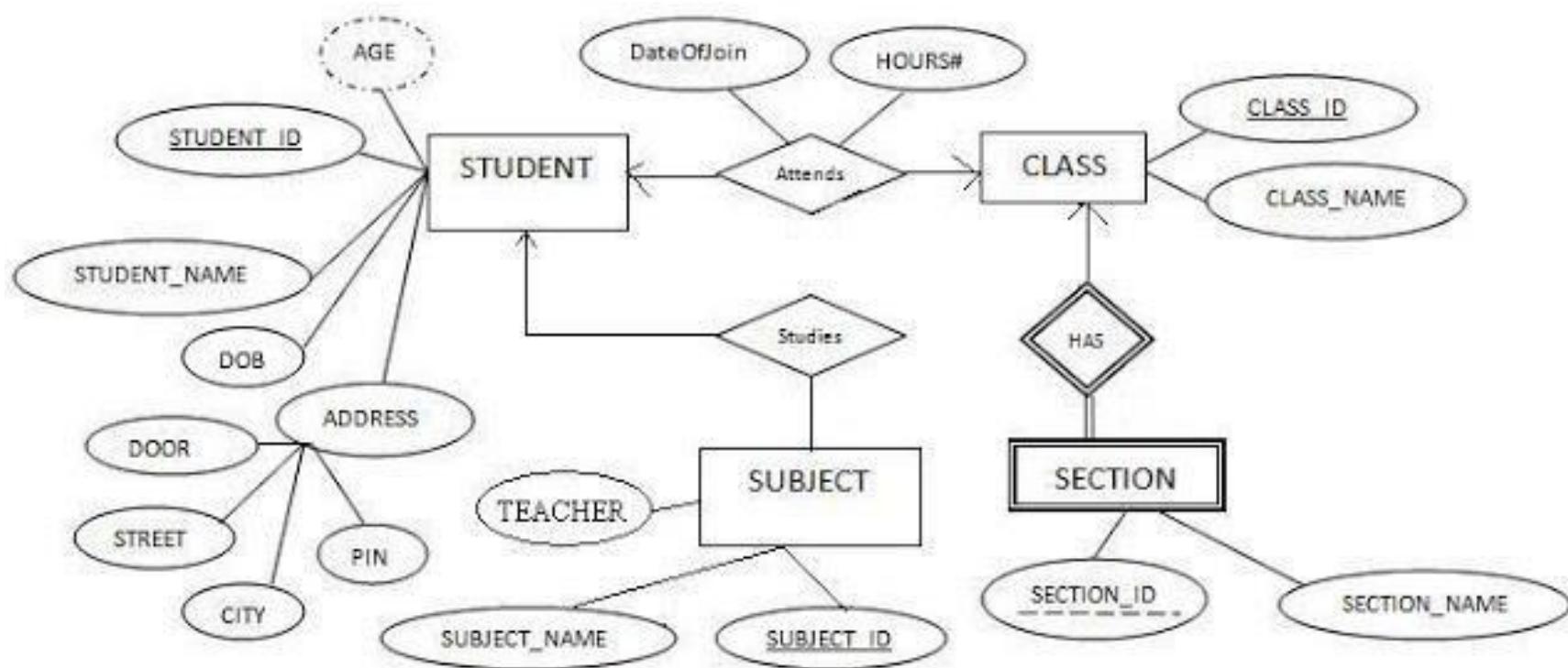
---

An entity type has a key attribute which uniquely identifies each entity in the entity set.

But there exists some entity type for which key attribute can't be defined. These are called Weak Entity type.

For example, A company may store the information of dependants (Parents, Children, Spouse) of an Employee. But the dependants don't have existence without the employee. So Dependant will be weak entity type and Employee will be Identifying Entity type for Dependant.





# Algebra

---

Relational Algebra is procedural query language, which takes Relation as input and generate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.

## Unary Relational Operations

SELECT (symbol:  $\sigma$ )

PROJECT (symbol:  $\pi$ ) RENAME (symbol:  $\lambda$ )

## Binary Relational Operations

JOIN DIVISION

## Relational Algebra Operations From Set Theory

UNION ( $\cup$ ) INTERSECTION ( $\cap$ ), DIFFERENCE ( $-$ )

CARTESIAN PRODUCT ( $\times$ )



# 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.

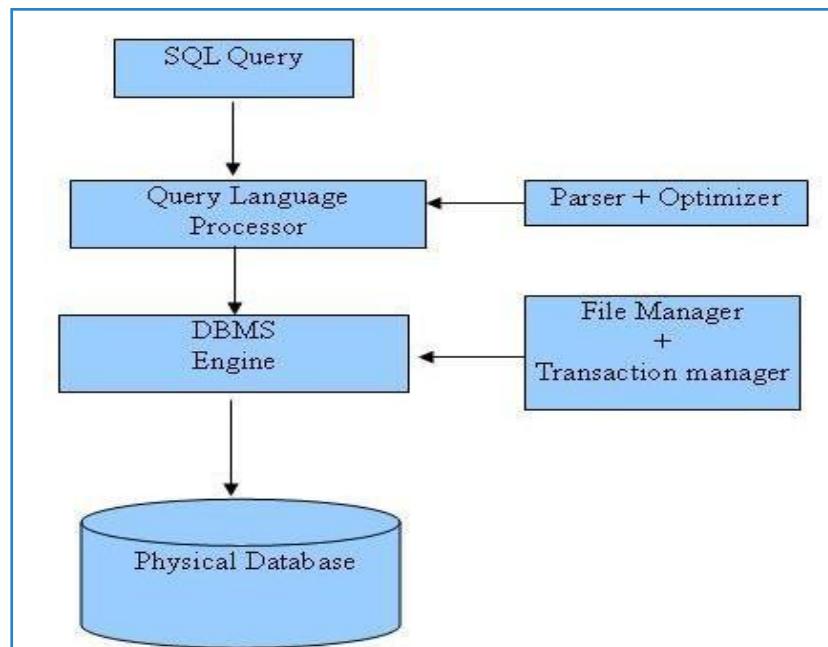
SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

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...”



# Why 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



# What is SQL? (Cont...)

---

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

# 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 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.

Table : Student

Roll_number	Name	Batch	Phone_number	Citizen_ID
Primary key				

# 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.
- It can be used when someone wants to enforce unique constraints on a column and a group of columns which is not a primary key.

Table : Student

Roll_number	Name	Batch	Phone_number	Citizen_ID

↑  
Unique key



# 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 "Employee" table. For example, salary hon", his "employee id" is stored with each salary record.

Table : Employee	
Employee_ID	Employee_Name
1	Jhon
2	Alex
3	James
4	Roy
5	Kay

Table : Salary				
Employee_ID	Ref	Year	Month	Salary
1	1	2012	April	30000
1	1	2012	May	31000
1	1	2012	June	32000
2	2	2012	April	40000
2	2	2012	May	41000
2	2	2012	June	42000

# Database Normalization

---

Normalization is the process of minimizing redundancy (duplicity) from a relation or set of relations.

Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations.

## **Most Commonly used normal forms:**

First normal form(1NF) Second normal form(2NF) Third normal form(3NF)

Boyce & Codd normal form (BCNF)



# First Normal Form

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute.

A relation is in first normal form if every attribute in that relation is singled valued attribute.

Example 1 – Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD\_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

# Second Normal Form

---

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency.

relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Partial Dependency – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

The table is not in 2nf form

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

- Note that, there are many courses having the same course fee.
- COURSE\_FEE cannot alone decide the value of COURSE\_NO or STUD\_NO;
- COURSE\_FEE together with STUD\_NO cannot decide the value of COURSE\_NO;
- COURSE\_FEE together with COURSE\_NO cannot decide the value of STUD\_NO;
- Hence, COURSE\_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD\_NO, COURSE\_NO} ;
- But, COURSE\_NO  $\rightarrow$  COURSE\_FEE , i.e., COURSE\_FEE is dependent on COURSE\_NO, which is a proper subset of the candidate key.
- Non-prime attribute COURSE\_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,  
we need to split the table into two tables such as : Table 1:  
STUD\_NO, COURSE\_NO  
Table 2: COURSE\_NO, COURSE\_FEE

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000



Table 1		Table 2	
STUD_NO	COURSE_NO	COURSE_NO	COURSE_FEE
1	C1	C1	1000
2	C2	C2	1500
1	C4	C3	1000
4	C3	C4	2000
4	C1	C5	2000

# Third Normal Form

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency  $X \rightarrow Y$

- $X$  is a super key.
- $Y$  is a prime attribute (each element of  $Y$  is part of some candidate key).

**Transitive dependency** – If  $A \rightarrow B$  and  $B \rightarrow C$  are two FDs then  $A \rightarrow C$  is called transitive dependency.

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

**Table 4**

In relation STUDENT given in Table 4, FD set: {STUD\_NO  $\rightarrow$  STUD\_NAME, STUD\_NO  $\rightarrow$  STUD\_STATE, STUD\_STATE  $\rightarrow$  STUD\_COUNTRY, STUD\_NO  $\rightarrow$  STUD\_AGE}

Candidate Key: {STUD\_NO}

For this relation in table 4, STUD\_NO  $\rightarrow$  STUD\_STATE and STUD\_STATE  $\rightarrow$  STUD\_COUNTRY are true. STUD\_COUNTRY is transitively dependent on STUD\_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY, STUD\_AGE) as:

STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_AGE)  
STATE\_COUNTRY (STATE, COUNTRY)

# Boyce Codd normal form (BCNF)

---

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF.

A table complies with BCNF if it is in 3NF and for every functional dependency  $X \rightarrow Y$ , X should be the super key of the table.

**Example:** Suppose there is a company wherein employees work in more than one department. They store the data like this:

**Functional dependencies :**

$\text{emp\_id} \rightarrow \text{emp\_nationality}$

$\text{emp\_dept} \rightarrow \{\text{dept\_type}, \text{dept\_no\_of\_emp}\}$

**Candidate key:** {emp\_id, emp\_dept}

The table is not in BCNF as neither emp\_id nor emp\_dept alone are keys.

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

To make the table comply with BCNF we can break the table in three tables like this:

**emp\_nationality table:**

emp_id	emp_nationality
1001	Austrian
1002	American

**emp\_dept table:**

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

**emp\_dept\_mapping table:**

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

# SQL Process

---

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process.

- These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc.
- Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

# 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.



# 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.

# DQL – 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

# DCL – Data Control Language

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

# Create, Drop, Use Database Syntax

---

## SQL CREATE DATABASE STATEMENT

```
CREATE DATABASE database_name;
```

## SQL DROP DATABASE Statement:

```
DROP DATABASE database_name;
```

## SQL USE STATEMENT

```
USE DATABASE database_name;
```



# Create, Drop, Alter Table Syntax

---

## SQL CREATE TABLE STATEMENT

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

## SQL DROP TABLE STATEMENT

```
DROP TABLE table_name;
```

## SQL TRUNCATE TABLE STATEMENT

```
TRUNCATE TABLE table_name;
```

## SQL ALTER TABLE STATEMENT

```
ALTER TABLE table_name{ADD|DROP|MODIFY}column_name{datatype};
```

## SQL ALTER TABLE STATEMENT (RENAME)

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



# Insert, Update, Delete Syntax

---

## SQL INSERT INTO STATEMENT

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

## SQL UPDATE STATEMENT

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

## SQL DELETE STATEMENT

```
DELETE FROM table_name WHERE {CONDITION};
```



# Select Statement Syntax

---

## SQL SELECT STATEMENT

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

## SQL DISTINCT CLAUSE

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

## 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;
```



# Select Statement Syntax

---

## SQL IN CLAUSE

```
SELECT column1, column2....columnN  
FROM (val-1, val-2,...val-N);
```

```
table_name WHERE column_name IN
```

## SQL BETWEEN CLAUSE

```
SELECT column1, column2....columnN  
FROM BETWEEN val-1 AND val-2;
```

```
table_name WHERE column_name
```

## SQL LIKE CLAUSE

```
SELECT column1, column2....columnN  
FROM PATTERN ;
```

```
table_name WHERE column_name LIKE {
```

## SQL ORDER BY CLAUSE

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

```
table_name WHERE CONDITION ORDER
```

# Select Statement Syntax

---

## 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 HAVING CLAUSE

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



# Create and Drop Index Syntax

---

## **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;
```



# Commit and Rollback Syntax

---

## **SQL COMMIT STATEMENT**

**COMMIT;**

## **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

# Inner Join Syntax

---

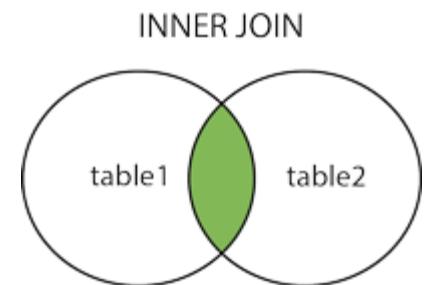
The most frequently used and important of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN.

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_filed = table2.common_field;
```



# Left Join Syntax

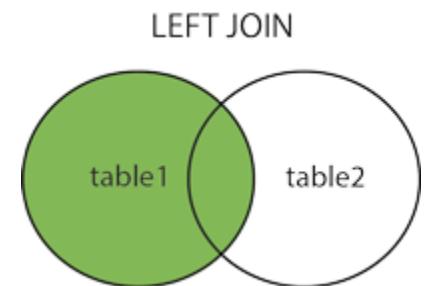
---

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;
```



# Right Join Syntax

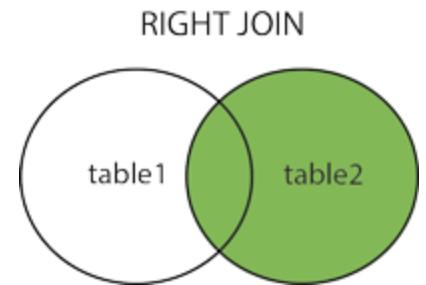
---

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;
```



# Full Join Syntax

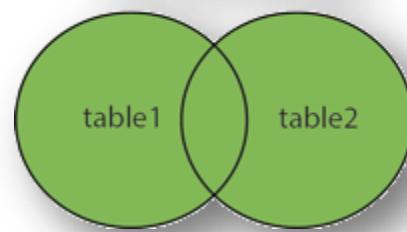
---

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;  
                          FULL OUTER JOIN
```



# Function

---

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

1. Aggregate Function
2. Scalar Function

# 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



# Student Table

## 1. Avg():

Syntax: SELECT AVG(column\_name) FROM table\_name;

Example:

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

Output: **AvgAge**

19.4

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

## 2. COUNT():

Syntax: SELECT COUNT(column\_name) FROM table\_name;

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

Output: **NumStudents**

5



# Student Table

## 3. First():

Syntax: SELECT FIRST(column\_name) FROM table\_name;

Example:

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

Output: `MarksFirst`

90

## 4. LAST():

Syntax: SELECT LAST(column\_name) FROM table\_name;

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

Output:

`MarksLast`

82

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18



# Student Table

## 5. MAX():

Syntax: SELECT MAX(column\_name) FROM table\_name;

Example:

SELECT MAX(MARKS) AS MaxMarks FROM Students;

Output:

MaxMarks

95

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

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 Students;

Output:

TotalMarks

400



# Scalar functions:

These functions are based on user input, these too returns single value.

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

# Student Table

## 1. UCASE()

Syntax: SELECT UCASE(column\_name) FROM table\_name;

Example:

SELECT UCASE(NAME) FROM Students;

Output:

NAME
HARSH
SURESH
PRATIK
DHANRAJ
RAM

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

2. LCASE(): Similar to UCASE() we can use LCASE() function.



### 3. MID()

Syntax:SELECT MID(column\_name,start,length) AS some\_name FROM table\_name;  
specifying length is optional here, and start signifies start position ( starting from 1 )

## Example:

**SELECT MID(NAME,1,4) FROM Students;**

## Output:

**NAME**  
HARS  
SURE  
PRAT  
DHAN  
RAM

#### 4. LEN():

Syntax: SELECT LENGTH(column\_name) FROM table\_name;

Example: SELECT LENGTH(NAME) FROM Students;

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18



# Student Table

## 5. ROUND():

Syntax: SELECT ROUND(column\_name,decimals) FROM table\_name;

decimals- number of decimals to be fetched. Example:

`SELECT ROUND(MARKS,0) FROM table_name;`

Output:

MARKS
90
50
80
95
85

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

## 6. NOW():

Syntax: `SELECT NOW() FROM table_name;`

Example: `SELECT NAME, NOW() AS DateTime FROM Students`

NAME	DateTime
HARSH	1/13/2017 1:30:11 PM
SURESH	1/13/2017 1:30:11 PM
PRATIK	1/13/2017 1:30:11 PM
DHANRAJ	1/13/2017 1:30:11 PM
RAM	1/13/2017 1:30:11 PM



# Student Table

## 7. FORMAT():

Syntax: SELECT FORMAT(column\_name) FROM table\_name;

Example:

```
SELECT NAME, FORMAT(Now(), 'YYYY-MM-DD') AS Date FROM  
Students;
```

Output:

NAME	Date
HARSH	2017-01-13
SURESH	2017-01-13
PRATIK	2017-01-13
DHANRAJ	2017-01-13
RAM	2017-01-13

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

# 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;
```



# Refer This Example

---

## 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>

# 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]

## **Explanation of syntax:**

**create trigger [trigger\_name]:** Creates or replaces an existing trigger with the trigger\_name.

**[before | after]:** This specifies when the trigger will be executed.

**{insert | update | delete}:** This specifies the DML operation.

**on [table\_name]:** This specifies the name of the table associated with the trigger.

**[for each row]:** This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.

**[trigger\_body]:** This provides the operation to be performed as trigger is fired

## **BEFORE and AFTER of Trigger:**

BEFORE triggers run the trigger action before the triggering statement is run.

AFTER triggers run the trigger action after the triggering statement is run.

# Refer This Example

---

## 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>

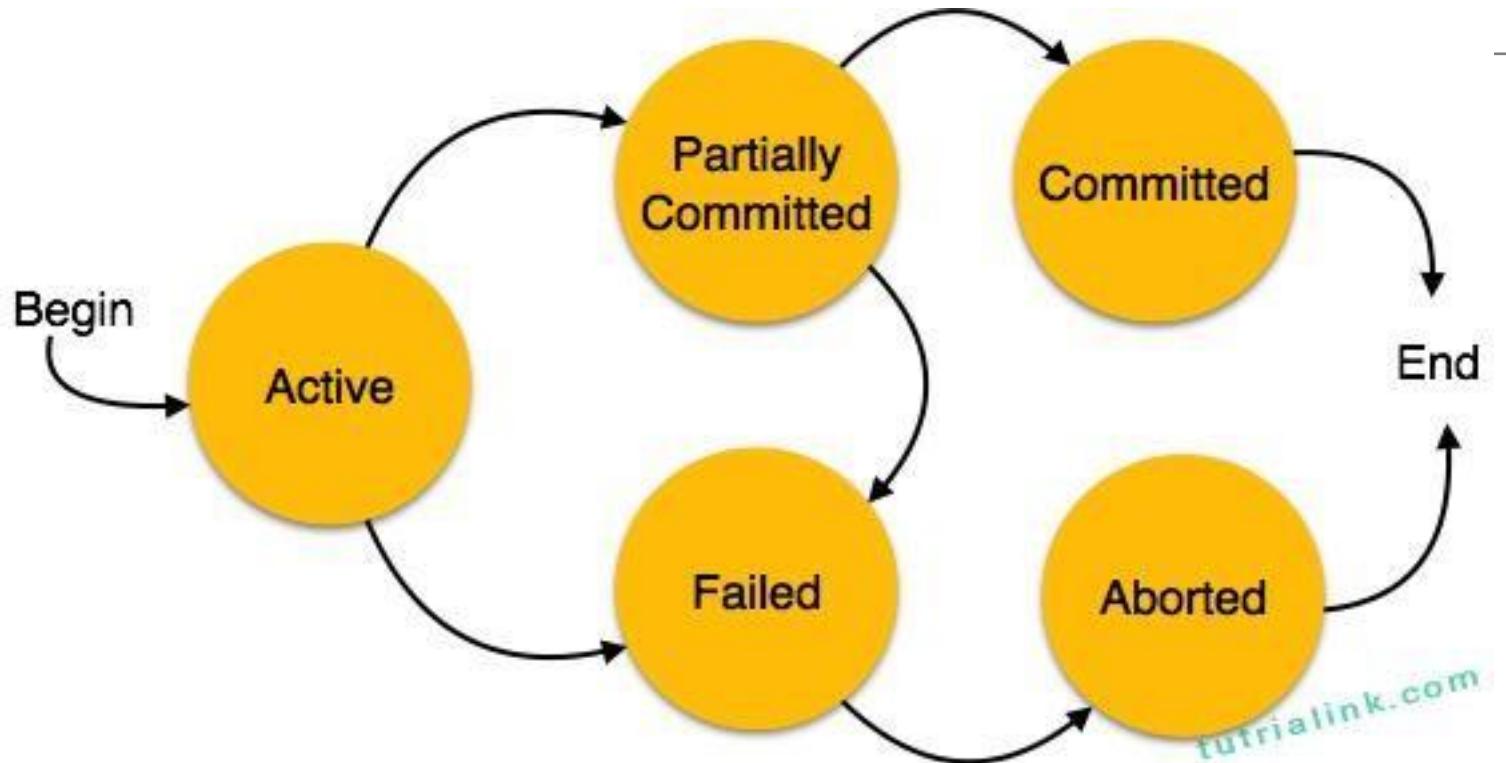


# Transaction

---

- A transaction is a logical unit of work of database processing that includes one or more database access operations.
- A transaction can be defined as an action or series of actions that is carried out by a single user or application program to perform operations for accessing the contents of the database.
- The operations can include retrieval, (Read), insertion (Write), deletion and modification.
- Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure.
- In order to maintain consistency in a database, before and after transaction, certain properties are followed. These are called **ACID** properties(**A**tomicity, **C**onsistency, **I**solation, **D**urability) .





# ACID PROPERTY

---

## 1. Atomicity:

This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none.

There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.



## **2. Consistency:**

- The database must remain in a consistent state after any transaction.
- No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well..
- For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

## **3. Isolation**

- In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.
- No transaction will affect the existence of any other transaction.
- For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

## 4. Durability

- The database should be durable enough to hold all its latest updates even if the system fails or restarts.
- If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data.
- If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action..
- For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.



# Transaction Control

---

The following commands are used to control transactions.

**COMMIT** – to save the changes.

**ROLLBACK** – to roll back the changes.

**SAVEPOINT** – creates points within the groups of transactions in which to ROLLBACK.

## 1. 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.

The syntax for the COMMIT command is as follows: **COMMIT**;



## 2. 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;**

## 3. 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.

The syntax for rolling back to a SAVEPOINT is as shown below.

**ROLLBACK TO SAVEPOINT\_NAME;**



# 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.



## Main components of Cursors

Each cursor contains the followings 5 parts,

**Declare Cursor:** In this part we declare variables and return a set of values.

`DECLARE cursor_name CURSOR FOR SELECT_statement;`

**Open:** This is the entering part of the cursor.

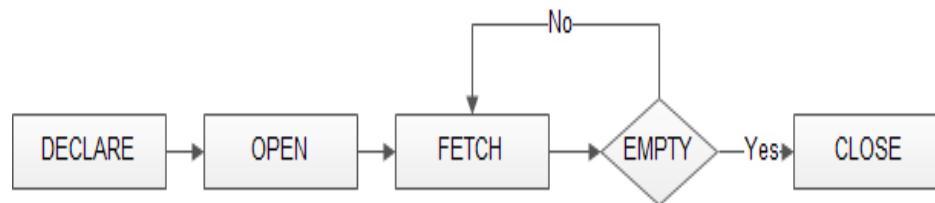
`OPEN cursor_name;`

**Fetch:** Used to retrieve the data row by row from a cursor.

`FETCH cursor_name INTO variables list;`

**Close:** This is an exit part of the cursor and used to close a cursor.

`1 CLOSE cursor_name;`



## Syntax:

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



# Database Backup and Recovery

---

- Database Backup is storage of data that means the copy of the data.
- It is a safeguard against unexpected data loss and application errors.
- It protects the database against data loss.
- If the original data is lost, then using the backup it can reconstructed.
- The backups are divided into two types,
  1. Physical Backup
  2. Logical Backup



## 1. Physical backups

- Physical Backups are the backups of the physical files used in storing and recovering your database, such as datafiles, control files and archived redo logs, log files.
- It is a copy of files storing database information to some other location, such as disk, some offline storage like magnetic tape.
- Physical backups are the foundation of the recovery mechanism in the database.
- Physical backup provides the minute details about the transaction and modification to the database.

## 2. Logical backup:

- Logical Backup contains logical data which is extracted from a database.
- It includes backup of logical data like views, procedures, functions, tables, etc.
- It is a useful supplement to physical backups in many circumstances but not a sufficient protection against data loss without physical backup. logical backup provides only structural information.

## Importance of Backups

- Planning and testing backup helps against failure of media, operating system, software and any other kind of failures that cause a serious data crash.
- It determines the speed and success of the recovery.
- Physical backup extracts data from physical storage (usually from disk to tape). Operating system is an example of physical backup.
- Logical backup extracts data using SQL from the database and store it in a binary file.
- Logical backup is used to restore the database objects into the database. So the logical backup utilities allow DBA (Database Administrator) to back up and recover selected objects within the database.



# Causes of Failure

---

## 1. System Crash

System crash occurs when there is a hardware or software failure or external factors like a power failure.

The data in the secondary memory is not affected when system crashes because the database has lots of integrity. Checkpoint prevents the loss of data from secondary memory.

## 2. Transaction Failure

The transaction failure is affected on only few tables or processes because of logical errors in the code.

This failure occurs when there are system errors like deadlock or unavailability of system resources to execute the transaction.



### **3. Network Failure**

A network failure occurs when a client – server configuration or distributed database system are connected by communication networks.

### **4. Disk Failure**

- Disk Failure occurs when there are issues with hard disks like formation of bad sectors, disk head crash, unavailability of disk etc.

### **5. Media Failure**

- Media failure is the most dangerous failure because, it takes more time to recover than any other kind of failures.
- A disk controller or disk head crash is a typical example of media failure.

# Recovery

---

Recovery is the process of restoring a database to the correct state in the event of a failure.

It ensures that the database is reliable and remains in consistent state in case of a failure.

Database recovery can be classified into two parts;

- 1. Rolling Forward** applies redo records to the corresponding data blocks.
- 2. Rolling Back** applies rollback segments to the datafiles. It is stored in transaction tables.



# Module-6 Linux Scripting

---

- What is Bash?
- shell scripting?
- Executing Script
- Variables
- Operators
- Statements
- loop

# What is Bash?

---

**Bash** is a "Unix shell": a command line interface for interacting with the operating system. It is widely available, being the default shell on many GNU/Linux distributions and on Mac OSX, with ports existing for many other systems.



# What is shell scripting?

---

A script might contain just a very simple list of commands — or even just a single command — or it might contain functions, loops, conditional constructs, and all the other hallmarks of imperative programming. In effect, a Bash shell script is a computer program written in the Bash programming language.

Shell scripting is the art of creating and maintaining such scripts.

Shell scripts can be called from the interactive command-line described above; or, they can be called from other parts of the system. One script might be set to run when the system boots up; another might be set to run every weekday at 2:30 AM; another might run whenever a user logs into the system.

# Creating and executing Bash shell Scripts

---

To create a shell script, open a new empty file in your editor. Any text editor will do: **vim**, **emacs**, **gedit**, **dtpad** et cetera are all valid. You might want to chose a more advanced editor like **vim** or **emacs**, however, because these can be configured to recognize shell and Bash syntax and can be a great help in preventing those errors that beginners frequently make, such as forgetting brackets and semi-colons.

first shell script, we'll just write a script which says "Hello World". We will then try to get more out of a Hello World program than any other tutorial you've ever read :-)

# Continue

---

Create a file (first.sh) as follows:

```
#!/bin/sh  
# This is a comment!  
echo Hello World # This is a comment, too!
```

The first line tells Unix that the file is to be executed by /bin/sh. This is the standard location of the Bourne shell on just about every Unix system. If you're using GNU/Linux, /bin/sh is normally a symbolic link to bash.

The second line begins with a special symbol: #. This marks the line as a comment, and it is ignored completely by the shell.

now run **chmod 755 first.sh** to make the text file executable, and run **./first.sh**.

Your screen should then look like this:

```
$ chmod 755 first.sh  
$ ./first.sh  
Hello World  
$
```



# Executing the script

---

The script should have execute permissions for the correct owners in order to be runnable. When setting permissions, check that you really obtained the permissions that you want. When this is done, the script can run like any other command.

**chmod u+x script.sh**

If you did not put the scripts directory in your PATH, and . (the current directory) is not in the PATH either, you can activate the script like this:

**./script\_name.sh**

A script can also explicitly be executed by a given shell, but generally we only do this if we want to obtain special behavior, such as checking if the script works with another shell or printing traces for debugging:

**rbash script\_name.sh sh script\_name.sh bash -x script\_name.sh**



# user input:

---

If we would like to ask the user for input then we use a command called **read**.

```
read var1  
#!/bin/bash  
# Ask the user for their name  
echo Hello, who am I talking to?  
read varname  
echo It's nice to meet you $varname  
#!/bin/bash  
# Demonstrate how read actually works
```

```
echo What cars do you like?  
read car1 car2 car3  
echo Your first car was: $car1  
echo Your second car was:  
$car2 echo Your third car was:  
$car3
```



# Quoting Special Characters:

The backslash (\) character is used to mark these special characters so that they are not interpreted by the shell, but passed on to the command being run (for example, **echo**).

So to output the string: (Assuming that the value of \$X is 5):

A quote is ", backslash is \, backtick is `.A few spaces are and dollar is \$. \$X is 5.

we would have to write:

```
$ echo "A quote is \", backslash is \\, backtick is \`."A quote is ",  
backslash is \, backtick is `.$ echo "A few spaces are   and dollar is \$.  
\$X is ${X}."
```

A few spaces are  and dollar is \$.  
\$X is 5.



# Variables:

---

## Assigning Values to variables:

**variablename=value**

[variables.sh](#)

```
#!/bin/sh MESSAGE="Hello  
World" echo $MESSAGE
```

# Arithmetc expansion

---

```
#!/bin/bash  
# Basic arithmetic using let  
a=5+4  
echo $a # 9  
"a = 5 + 4"  
echo $a # 9  
"a = 4 * 5"  
echo $a # 20  
"a = $1 + 30"  
echo $a # 30 + first command line argument  
a=${[2+2]}  
echo $a
```



# for Loop:

---

```
for var in <list> do  
<commands> done
```

```
#!/bin/bash
```

```
# Basic for loop
```

```
names='linux'
```

```
for name in $names
```

```
do
```

```
echo $name done
```

```
echo All done
```



# Ranges

---

```
#!/bin/bash
```

```
# Basic range in for loop
```

```
for value in {1..5} do
```

```
echo $value done
```

```
echo All done
```

```
#!/bin/bash
```

```
# Basic range with steps for loop
```

```
for value in {10..0..2} do
```

```
echo $value done
```

```
echo All done
```



# Bash Conditionals and Control Structures

---

```
if [ <some test> ] then  
<commands> fi
```

```
#!/bin/bash
```

```
# if statement if [ $1 -  
gt; 100 ] then  
echo Hey that's a large  
number.
```

```
pwd fi
```

```
date
```

```
#!/bin/bash
```

*# Nested if statements*

```
if [ $1 -gt 100 ] then  
echo Hey that's a large number.
```

```
if (( $1 % 2 == 0 ))  
then  
echo And is also an even number.  
fi  
Fi
```



# Continue

---

```
#!/bin/bash  
# else example
```

```
if [ $# -eq 1 ]  
then  
    nl $1  
else  
    nl /dev/stdin  
Fi
```

```
#!/bin/bash  
# elif statements
```

```
if [ $1 -ge 18 ] then  
    echo You may go to the party. elif [ $2 ==  
    'yes' ]  
    then  
        echo You may go to the party but be back  
        before midnight.  
    else  
        echo You may not go to the party. fi
```



# Operators

---

## Operator Description

**! EXPRESSION** The EXPRESSION is false.

**-n STRING** The length of STRING is greater than zero.

**-z STRING** The length of STRING is zero (ie it is empty).

**STRING1 = STRING2** STRING1 is equal to STRING2

**STRING1 != STRING2** STRING1 is not equal to STRING2

**INTEGER1 -eq INTEGER2** INTEGER1 is numerically equal to INTEGER2

**INTEGER1 -gt INTEGER2** INTEGER1 is numerically greater than INTEGER2

**INTEGER1 -lt INTEGER2** INTEGER1 is numerically less than INTEGER2

# Case Statements:

---

```
case <variable> in
<pattern 1>
<commands>
;;
<pattern 2>
<other commands>
;;
esac
```

```
#!/bin/bash
# case example
case $1 in start)
echo starting
;;
stop)
echo stoping
;;
restart)
echo restarting
;;
*)
echo don't know
;;
esac
```



---

# Thank You

