

# HTML

HTML stands for Hyper Text Markup Language, which is the most widely used language on Web to develop web pages. HTML was created by Berners-Lee in late 1991 but "HTML 2.0" was the first standard HTML specification which was published in 1995. HTML 4.01 was a major version of HTML and it was published in late 1999. Though HTML 4.01 version is widely used but currently we are having HTML-5 version which is an extension to HTML 4.01, and this version was published in 2012.

## What is HTML?

HTML is the standard markup language for creating Web pages.

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

## HTML Tags

HTML tags are element names surrounded by angle brackets:

**<tagname>content here...</tagname>**

- HTML tags normally come in pairs like <p> and </p>
- The first tag in a pair is the start tag, the second tag is the end tag
- The end tag is written like the start tag, but with a forward slash inserted before the tag name

## The <!DOCTYPE> Declaration

The <!DOCTYPE> declaration represents the document type, and helps browsers to display web pages correctly. It must only appear once, at the top of the page (before any HTML tags).


The <!DOCTYPE> declaration is not case sensitive.

The <!DOCTYPE> declaration for HTML5 is: **<!DOCTYPE html>**

## HTML Tags Chart

To use any of the following HTML tags, simply select the HTML code you'd like and copy and paste it into your web page.

| Tag      | Name                          | Code Example  | Browser View                                     |
|----------|-------------------------------|---|--|
| <!--     | <b>comment</b>                | <!--This can be viewed in the <b>HTML</b> part of a document-->   | Nothing will show                                |
| <a -     | <b>anchor</b>                 | <a href="http://www.google.co">Visit Our Site</a>   | <a href="#">Visit google</a>                     |
| <b>      | <b>bold</b>                   | <b>Example</b>  | <b>Example</b>                                   |
| <big>    | <b>big (text)</b>             | <big>Example</big>  | Example  |
| <body>   | <b>body of HTML document</b>  | <body>The content of your <b>HTML</b> page</body>   | Contents of your web page                        |
| <br>     | <b>line break</b>             | Keonics<br>Hubli  | Keonics<br>Hubli                                 |
| <center> | <b>center</b>                 | <center>This will center your contents</center>   | This will center your contents                   |
| <dd>     | <b>definition description</b> | <dl><dt>Definition Term</dt><dd> <b>Definition of the term</b> </dd><dt>Definition Term</dt><dd> <b>Definition of the term</b> </dd></dl> | Definition Term<br><b>Definition of the term</b> |
| <dl>     | <b>definition list</b>        |   | Definition Term<br><b>Definition of the term</b> |

|  |  |  |  |
|--|--|--|--|
| <dt>   | <b>definition term</b>   |  |  |
| <em>   | <b>emphasis</b>  | This is an <em>Example</em> of using the emphasis tag  | This is an <i>Example</i> of using the emphasis tag  |
| <font>                                       | <b>font</b>  | <font face="Times New Roman" size="3" color="#ff0000">Example</font>   | Example  |
| <h1><br><h2><br><h3><br><h4><br><h5><br><h6> | <b>heading 1</b><br><b>heading 2</b><br><b>heading 3</b><br><b>heading 4</b><br><b>heading 5</b><br><b>heading 6</b> | <h1>Heading 1 Example</h1><br><h2>Heading 2 Example</h2><br><h3>Heading 3 Example</h3><br><h4>Heading 4 Example</h4><br><h5>Heading 5 Example</h5><br><h6>Heading 6 Example</h6> | <h1>Heading 1</h1><br><h2>Heading 2</h2><br><h3>Heading 3</h3><br><h4>Heading 4</h4><br><h5>Heading 5</h5><br><h6>Heading 6</h6> |
| <hr>   | <b>horizontal rule</b>   | <hr />   | Contents of your web page<br><hr/> Contents of your web page   |
| <hr>   | <b>horizontal rule</b>   | <hr width="50%" size="3" />  | Contents of your web page<br><hr/> Contents of your web page   |
| <hr>   | <b>horizontal rule</b>   | <hr width="50%" size="3" noshade />  | Contents of your web page<br><hr/> Contents of your web page   |
| <hr><br>(Internet Explorer)                  | <b>horizontal rule</b>   | <hr width="75%" color="#ff0000" size="4" />  | Contents of your web page<br><hr/> Contents of your web page   |
| <html>                                       | <b>hypertext markup language</b>   | <html><br><head><br><meta><br><title>Title of your web page</title><br></head><br><body> <b>HTML web page</b> contents<br></body><br></html>                                     | Contents of your web page  |
| <i>  | <b>italic</b>  | <i>Example</i>   | <i>Example</i>   |
| <img>  | <b>image</b>   |   |    |

```

1)<html>
  <head>
    <title>
      My First web page
    </title>
  </head>
  <body>
    Welcome to HTML!...
  </body>
</html>

```

```

2)
<html>
  <head>
    <title>
      back color
    </title>
  </head>
  <body bgcolor="wheat">
    Welcome to HTML!...
  </body>
</html>

```

```

3)
<html>
  <head>
    <title>
      text color
    </title>
  </head>
  <body text="red">
    Welcome to HTML!...
  </body>
</html>

```

```

4)
<html>
  <head>
    <title>
      heading tags
    </title>
  </head>
  <body>
    <h1>Heading 1</h1>
    <h2>Heading 2</h2>
    <h3>Heading 3</h3>
    <h4>Heading 4</h4>
    <h5>Heading 5</h5>
    <h6>Heading 6</h6>
  </body>
</html>

```

```

5)
<html>
  <head>
    <title>
      font size tags
    </title>
  </head>
  <body>
    <font size="1">Font size 1</font><br>
    <font size="2">Font size 2</font><br>
    <font size="3">Font size 3</font><br>
    <font size="4">Font size 4</font><br>
    <font size="5">Font size 5</font><br>
    <font size="6">Font size 6</font><br>
    <font size="7">Font size 7</font>
  </body>
</html>

```

```

6)
<html>
  <head>
    <title>
      iframe
    </title>
  </head>
  <body>
    <h1>KEONICS</h1>
    <iframe src="29.html" align="right"
height="500" width="180"></iframe>
    <iframe src="25.html" align="bottom"
align="left" height="200" width="300">
    </body>
</html>

```

```

7)
<html>
  <head>
    <title>
      iframe
    </title>
  </head>
  <body>
    <h1>KEONICS</h1>
    <iframe src="29.html" align="right"
height="500" width="180"></iframe>
    <iframe src="25.html" align="bottom"
align="left" height="200" width="300">
    </body>
</html>

```

8)

```
<html>
  <head>
    <title>
      Image
    </title>
  </head>
  <body>
    
  </body>
</html>
```

9)

```
<html>
  <head>
    <title>
      iframe
    </title>
  </head>
  <body>
    <h1>KEONICS</h1>
    <iframe src="29.html" align="right"
height="500" width="180"></iframe>
    <iframe src="25.html" align="bottom"
align="left" height="200" width="300">
  </body>
</html>
```

10)

```
<html>
<head>
<title>
  use of horizontal line
</title>
</head>
<body>
<h1>This is heading 1</h1>
<p>This is some text.</p>
<hr>
<h2>This is heading 2</h2>
<p>This is some other text.</p>
<hr>
<h2>This is heading 3</h2>
<p>This is some other text.</p>
</body>
</html>
```

11)

```
<html>
  <head>
    <title>
      Paragraph with alignment tags
    </title>
  </head>
  <body>
    <p align="right">Paragraph 1</p>
    <p align="center">Paragraph 2</p>
    <p align="left">Paragraph 3</p>
  </body>
</html>
```

12)

```
<html>
  <head>
    <title>
      order listing
    </title>
  </head>
  <body>
    <H2>ORDERED LIST</H2>
    <ol>
      <li>C</li>
      <li>C++</li>
      <li>Java</li>
    </ol>
    <ol type="I">
      <li>C</li>
      <li>C++</li>
      <li>Java</li>
    </ol>
    <ol type="i">
      <li>C</li>
      <li>C++</li>
      <li>Java</li>
    </ol>
    <ol type="A">
      <li>C</li>
      <li>C++</li>
      <li>Java</li>
    </ol>
    <ol type="a">
      <li>C</li>
      <li>C++</li>
      <li>Java</li>
    </ol>
  </body>
</html>
```

13)

```
<html>
  <head>
    <title>
      Unorder listing
    </title>
  </head>
<body>
  <H2>UNORDERED LIST</H2>
  <ul>
    <li>C</li>
    <li>C++</li>
    <li>Java</li>
  </ul>
  <ul type="square">
    <li>C</li>
    <li>C++</li>
    <li>Java</li>
  </ul>
  <ul type="circle">
    <li>C</li>
    <li>C++</li>
    <li>Java</li>
  </ul>
  <ul type="disc">
    <li>C</li>
    <li>C++</li>
    <li>Java</li>
  </ul>
</body>
</html>
```

14)

```
<html>
  <head>
    <title>definition list</title>
  </head>
<body>
  <h2>Definition List</h2>
  <dl>
    <dt>COMPUTER</dt>
    <dd>Computer is an electronic device
      accepts data from
        user and process the data ,gives you
      required result as      output
    </dd>
  </dt>
</dl>
  <dl>
    <dt>What is HTML?</dt>
    <dd>HTML is the standard markup
      language for creating    Web pages.
    </dd>
```

```
</dt>
```

```
</dl>
```

```
</body>
```

```
</html>
```

15)

```
<html>
  <head>
    <title>
      MOVING text
    </title>
  </head>
<body>
  <marquee direction="right" bgcolor="orange">
    <h1>KEONICS COMPUTER CENTRE</h1>
  </marquee>
</body>
</html>
```

16)

```
<html>
  <head>
    <title>
      Login page
    </title>
  </head>
<body>
  <table border="1" bgcolor="pink">
    <tr>
      <th colspan="2">Login Up </th>
    </tr>
    <tr>
      <th>User Name</th>
      <td><input type="text"></td>
    </tr>
    <tr>
      <th>Password</th>
      <td><input type="password"
        maxlength="8"></td>
    </tr>
    <tr>
      <td colspan="2" align="center">
        <input type="Submit"
          value="Login">
        <input type="Reset"
          value="Clear">
      </td>
    </tr>
  </table>
</body>
</html>
```

17)

```
<html>
  <head>
    <title>
      Sign up page
    </title>
  </head>
  <body>
    <table border="1" bgcolor="pink">
      <tr>
        <th colspan="2">Sign Up </th>
      </tr>
      <tr>
        <th>First Name</th>
        <td><input type="text"></td>
      </tr>
      <tr>
        <th>Middle Name</th>
        <td><input type="text"></td>
      </tr>
      <tr>
        <th>Last Name</th>
        <td><input type="text"></td>
      </tr>
      <tr>
        <th>Address</th>
        <td><TextArea>hubli</TextArea></td>
      </tr>
      <tr>
        <th>Gender</th>
        <td>
          <input type="radio"
name="gr1">Male
          <input type="radio"
name="gr1">Female
        </td>
      </tr>
      <tr>
        <th>Language Known</th>
        <td>
          <input type="Checkbox">Kannada
          <br>
          <input
type="Checkbox">Hindi<br>
          <input
type="Checkbox">English<br>
        </td>
      </tr>
      <tr>
        <th>Select Country</th>
        <td>
          <Select>
```

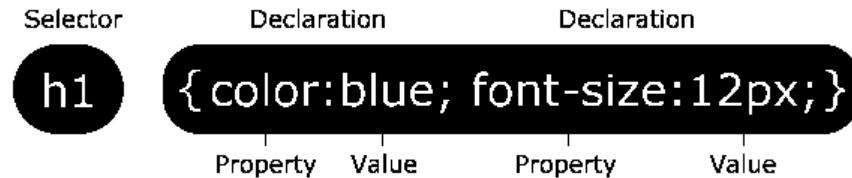
```
            <option>India</option>
            <option>Aus</option>
            <option>England</option>
            <option>USA</option>
          </Select>
        </td>
      </tr>
      <tr>
        <td colspan="2" align="center">
          <input type="Submit"
value="Save">
          <input type="Reset"
value="Clear">
        </td>
      </tr>
    </table>
  </body>
</html>
```

18)

```
<frameset cols="20%,20%,30%,30%">
  <frame src="29.html">
  <frame src="36.html">
  <frame src="25.html">
  <frame src="26.html">
</frameset>
```

# What is CSS?

- CSS stands for **Cascading Style Sheets**
- CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**
- CSS **saves a lot of work**. It can control the layout of multiple web pages all at once
- External style sheets are stored in **CSS files**
- CSS Syntax
- A CSS rule-set consists of a selector and a declaration block:



- The selector points to the HTML element you want to style.
  - The declaration block contains one or more declarations separated by semicolons.
  - Each declaration includes a CSS property name and a value, separated by a colon.
- 
- A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.
  - In the following example all <p> elements will be center-aligned, with a red text color:

## The element Selector

The element selector selects elements based on the element name.

You can select all <p> elements on a page like this (in this case, all <p> elements will be center-aligned, with a red text color):

Example

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

## The id Selector

The id selector uses the id attribute of an HTML element to select a specific element. The id of an element should be unique within a page, so the id selector is used to select one unique element! To select an element with a specific id, write a hash (#) character, followed by the id of the element.

The style rule below will be applied to the HTML element with id="para1":

Example

```
#para1 {
  text-align: center;
  color: red;
}
```

## The class Selector

The class selector selects elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the name of the class.

In the example below, all HTML elements with class="center" will be red and center-aligned:

Example

```
.center {
  text-align: center;
  color: red;
}
```

## Grouping Selectors

If you have elements with the same style definitions, like this:

```
h1 { text-align: center;
    color: red; }
h2 { text-align: center;
    color: red; }
p { text-align: center;
    color: red; }
```

It will be better to group the selectors, to minimize the code.

To group selectors, separate each selector with a comma.

In the example below we have grouped the selectors from the code above:

### Example

```
h1, h2, p {  
    text-align: center;  
    color: red; }
```

### CSS Comments

Comments are used to explain the code, and may help when you edit the source code at a later date. Comments are ignored by browsers.

A CSS comment starts with `/*` and ends with `*/`. Comments can also span multiple lines:

### Example

```
p {  
    color: red;  
    /* This is a single-line comment */  
    text-align: center; }  
/* This is a multi-line  
comment */
```

### Three Ways to Insert CSS

There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet
- Inline style

### External Style Sheet

With an external style sheet, you can change the look of an entire website by changing just one file!

Each page must include a reference to the external style sheet file inside the `<link>` element. The `<link>` element goes inside the `<head>` section:

### Example

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

An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet file must be saved with a **.css extension**.

Here is how the **"mystyle.css"** looks:

```
body {  
    background-color: lightblue;  
}  
h1 {  
    color: navy;  
    margin-left: 20px;  
}
```

### Internal Style Sheet

An internal style sheet may be used if one single page has a unique style.

Internal styles are defined within the `<style>` element, inside the `<head>` section of an HTML page:

### Example

```
<head>  
<style>  
body {  
    background-color: linen;  
}  
  
h1 {  
    color: maroon;  
    margin-left: 40px;  
}  
</style>  
</head>
```

### Inline Styles

An inline style may be used to apply a unique style for a single element.



To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

The example below shows how to change the color and the left margin of a <h1> element:

Example

```
<h1 style="color:blue;margin-left:30px;">This is a heading</h1>
```

### **CSS Comments**

Comments are used to explain your code, and may help you when you edit the source code at a later date. Comments are ignored by browsers.

A CSS comment starts with /\* and ends with \*/. Comments can also span multiple lines:

Example

```
p {  
  color: red;  
  /* This is a single-line comment */  
  text-align: center;  
}
```

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

### **CSS Programs**

1)

```
<html>  
<body>  
<h2 style="background-color:red">  
Red background-color  
</h2>  
<h2 style="background-color:green">  
Green background-color  
</h2>  
<h2 style="background-color:blue;color:FFFFFF">  
Blue background-color and white text color  
</h2>  
<h2 style="background-color:orange">  
Orange background-color  
</h2>  
<h2 style="background-color:yellow">  
Yellow background-color  
</h2>  
<h2 style="background-color:CYAN">  
Cyan background-color  
</h2>  
<h2 style="background-color:black;color:white">  
Black background-color and white text color  
</h2>  
</body>  
</html>
```

2)

```
<html>
<head>
<style>
body {
  background-image: url("1.jpg");
  background-repeat: repeat;
  background-position: left top;
  background-attachment: fixed;
}
</style>
</head>
<body>
<h1>Hello World!</h1>
<p>The background-image is fixed. Try to scroll
down the page.</p>
<p>copy paragraph 20 times for scrolling</p>
</body>
</html>
```

3)

```
<html>
<head>
<style>
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
p.solid {border-style: solid;}
p.double {border-style: double;}
p.groove {border-style: groove;}
p.ridge {border-style: ridge;}
p.inset {border-style: inset;}
p.outset {border-style: outset;}
p.none {border-style: none;}
p.hidden {border-style: hidden;}
p.mix {border-style: dotted dashed solid double;}
</style>
</head>
<body>
<h2>The border-style Property</h2>
<p>This property specifies what kind of border to
display:</p>
```

```
<p class="dotted">A dotted border.</p>
<p class="dashed">A dashed border.</p>
<p class="solid">A solid border.</p>
<p class="double">A double border.</p>
<p class="groove">A groove border.</p>
<p class="ridge">A ridge border.</p>
<p class="inset">An inset border.</p>
<p class="outset">An outset border.</p>
<p class="none">No border.</p>
<p class="hidden">A hidden border.</p>
<p class="mix">A mixed border.</p>
</body>
</html>
```

4)

```
<html>
<head>
<style>
```

```
div {
  background-color: lightgrey;
  width: 300px;
  border: 25px solid green;
  padding: 25px;
  margin: 50px;
}
</style>
</head>
<body>
<h2>Demonstrating the Box Model</h2>
<p>The CSS box model is essentially a box that
wraps around every HTML element. It consists of:
borders, padding, margins, and the actual
content.</p>
<div>This text is the actual content of the box. We
have added a 25px padding, 25px margin and a 25px
green border. </div>
</body>
</html>
```

5)

```
<html>
<body>

<h3 style="color:Tomato;">Hello World</h3>

<p style="color:Blue;">You can set the color of
text</p>

<p style="color:Green;">You can set the color of
text</p>

</body>
</html>
```

6)

```
<html>
<head>
<style>
p.normal {
  font-weight: normal;
  font-family: IMPACT;
  font-size: 250%;
  font-style: italic;
}
p.light {
  font-weight: lighter;
  font-size: 300%;
  font-style: oblique;
}
p.thick {
  font-weight: bold;
  font-size: 40px;
  font-variant: small-caps;
}
p.thicker {
  font-weight: 900;
}
</style>
```

```

</head>
<body>
<p class="normal">This is a paragraph. normal</p>
<p class="light">This is a paragraph. light</p>
<p class="thick">This is a paragraph . thick</p>
<p class="thicker">This is a paragraph. thicker</p>
</body>
</html>

```

## 7)

```

<html>
<head>
<style>
/* unvisited link */
a:link {
    color: red;
}

/* visited link */
a:visited {
    color: green;
}

/* mouse over link */
a:hover {
    color: hotpink;
}

/* selected link */
a:active {
    color: blue;
}
</style>
</head>
<body>

```

```

<p><b><a href="f1.html" >This is a
link</a></b></p>
<p><b>Note:</b> a:hover MUST come after a:link
and a:visited in the CSS definition in order to be
effective.</p>
<p><b>Note:</b> a:active MUST come after
a:hover in the CSS definition in order to be
effective.</p>

```

```

</body>
</html>

```

## 8)

```

<html>
<head>
<style>
h1
{
    color:Crimson ; /* any color value*/
    letter-spacing:3px; /* any pixel value*/
    word-spacing:5px; /* any pixel value */
    text-align:center; /* left center right justify */
    text-decoration:underline; /* overline line-
through */

```

```

text-transform:capitalize; /* uppercase lowercase
*/
text-shadow: 3px 3px Gold ;
/* horizontal_shadow vertical_shadow color */
}

```

## h2

```

{
    text-indent: 25px;
    direction:ltr; /* rtl */
}
</style>
</head>
<body>
    <h1>KEONICS    COMPUTER    TRAINING
CENTRE</h1>
<h2> IT PARK HUBLI</h2>
</body>
</html>

```

## 9)

```

<html>
<body>
<h1 style="border: 2px solid Tomato;">Hello
World</h1>
<h1 style="border: 2px solid DodgerBlue;">Hello
World</h1>
<h1 style="border: 2px solid Violet;">Hello
World</h1>
</body>
</html>

```

## 10)

```

<html>
<head>
<style>
p { display: block; }
h1 { display:inline; }
</style>
</head>
<body>
<p>Hello Javatpoint</p>
<p>Java </p>
<h1>SQL </h1>
<h1>HTML </h1>
<h1>CSS </h1>
</body>
</html>

```

## JavaScript - Overview

### What is JavaScript ?

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

### Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

### Advantages of JavaScript

The merits of using JavaScript are –

- **Less server interaction** – You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** – They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** – You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** – You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

## JavaScript - Syntax

JavaScript can be implemented using JavaScript statements that are placed within the `<script>...</script>` HTML tags in a web page.

You can place the `<script>` tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the `<head>` tags.

The `<script>` tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
  JavaScript code
</script>
```

The script tag takes two important attributes –

- **Language** – This attribute specifies what scripting language you are using.
- **Type** – This attribute is what is now recommended to indicate the scripting language in use and its value should be set to **"text/javascript"**.

```
<script language = "javascript" type = "text/javascript">
  JavaScript code
</script>
```

### First JavaScript Code

```
<html>
<body>
  <script language = "javascript" type = "text/javascript">
    document.write("Hello World!")
  </script>
</body>
</html>
```

### Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus –

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /\* and \*/ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.
- JavaScript Variables
- Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.
- Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
<script type = "text/javascript">
    var name;
    var address;
</script>
```

**Note** – Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

#### JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables** – A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables** – A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

#### JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **\_123test** is a valid one.
- JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

### JavaScript – Operators

JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

### Java script Programs

```

1) <HTML>
    <body>
        <p> SUM OF TWO NUMBERS</P>
        <script type="text/javascript">
            var a,b;
            a=10;
            b=20;
            var sum=a+b;
            document.write("a+b="+sum);
        </script>
    </body>
</HTML>

```

```

2)
<HTML>
<body>
<p> ARITHMATIC OPERATION</P>
<script type="text/javascript">
    var a,b;
    a=20;
    b=3;
    var sum=a+b;
    var d=a-b;
    var p=a*b;
    var q=a/b;
    var r=a%b;
    document.write("</h3>a+b="+sum+"</br>");
    document.write("a-b="+d+"</br>");
    document.write("a*b="+p+"</br>");
    document.write("a/b="+q+"</br>");
    document.write("a%b="+r+"</br></h3>");
</script>
</body>
</HTML>

```

```

3)
<!--program to illustrate do while -->
<html>
<body>
    <script type = "text/javascript">
        var count = 0;
        document.write("Starting Loop" + "<br
/>");
        do {
            document.write("Current Count : " + count
+ "<br />");
            count++;
        }
        while (count < 5);
        document.write ("Loop stopped!");

    </script>
    <p>Set the variable to different value and then
try...</p>
</body>
</html>

```

```

4)
<html>
<body>
<script type = "text/javascript">
    var n=-5;
    if(n>0)
    {
        document.write("<br> Given no "+n+" is +ve
no")
    }
    if(n<0)
    {
        document.write("<br> Given no "+n+" is -ve
no")
    }
</script>
</body>
</html>

```

```

5)
<html>

<body>
<script type = "text/javascript">

    var n1=3,n2=4;
    if(n1>n2)
    {
        document.write("<br>"+n1+" is big")
    }
    else
    {
        document.write("<br>"+n2+" is big")
    }
</script>
</body>
</html>

```

```

6)
<html>
<body>
<script type = "text/javascript">
    var area,l=2,b=2,r=2,pi=3.142,h=2;
    var ch=2;
    document.write("<br>1-->area of a traingle");
    document.write("<br>2-->area of a rectangle");
    document.write("<br>3-->area of a circle") ;
    document.write("<br>Your choice is 2:");
    switch(ch)
    {
        case 1:
            area=0.5*b*h;
            document.write("<br>Area of a
traingle="+area);
            break;
        case 2:
            area=l*b;

```

```

        document.write("<br>Area of a
rectangle="+area);
        break;
    case 3:
        area=pi*r*r;
        document.write("<br>Area of a
circle="+area);
        break;
    default:
        document.write("<br>Invalid Choice");
    }
</script>
</body>
</html>

```

```

7)
<html>
<body>
<script type = "text/javascript">
var i=1;
while(i<=20)
{
    document.write("<br>" +i);
    i=i+2;
}
</script>
</body>
</html>

```

```

8)
<html>
<body>
<script type = "text/javascript">
var i;
for(i=1;i<=5;i++)
{
    document.write("<br>" +i);
}
</script>
</body>
</html>

```

```

9)
<html>
<head>
    <script type = "text/javascript">
        function sayHello(name, age) {
            document.write (name + " is " + age + "
years old.");
        }
    </script>
</head>

<body>
    <p>Click the following button to call the
function</p>
    <form>
        <input type = "button" onclick =
"sayHello('John', 7)" value = "Say Hello">

```

```

    </form>
    <p>Use different parameters inside the function
and then try...</p>
</body>
</html>

```

```

10) <html>
<body>
<p id="demo">JavaScript can change HTML
content.</p>
<button type="button"
onclick='document.getElementById("demo").innerH
TML = "Hello JavaScript!">
Click Me!</button>
</body>
</html>

```

# C Programming

## Introduction to C Programming

C is a general-purpose, high-level language that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. C was originally first implemented on the DEC PDP-11 computer in 1972.

In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.

The UNIX operating system, the C compiler, and essentially all UNIX application programs have been written in C. C has now become a widely used professional language for various reasons –

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platforms

## Facts about C

- C was invented to write an operating system called UNIX.
- C is a successor of B language which was introduced around the early 1970s.
- The language was formalized in 1988 by the American National Standard Institute (ANSI).
- The UNIX OS was totally written in C.
- Today C is the most widely used and popular System Programming Language.
- Most of the state-of-the-art software have been implemented using C.
- Today's most popular Linux OS and RDBMS MySQL have been written in C.

## Why use C?

C was initially used for system development work, particularly the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as the code written in assembly language.

Some examples of the use of C might be –

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Databases
- Language Interpreters
- Utilities

## Example:/\*Program to Display a message\*/

```
#include<stdio.h>
main()
{   clrscr();
    printf("Hello,World");
    return(0);
}
```

Let us take a look at the various parts of the above program –

- The first line of the program `#include <stdio.h>` is a preprocessor command, which tells a C compiler to include `stdio.h` file before going to actual compilation.
- The next line `int main()` is the main function where the program execution begins.
- The next line `/*...*/` will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
- The next line `printf(...)` is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
- The next line `return 0;` terminates the `main()` function and returns the value



## C - Basic Syntax

You have seen the basic structure of a C program, so it will be easy to understand other basic building blocks of the C programming language.

### Tokens in C

A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following C statement consists of five tokens –

```
printf("Hello, World! \n");
```

The individual tokens are –

```
printf  
(  
"Hello, World! \n"  
);
```

### Semicolons

In a C program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

Given below are two different statements –

```
printf("Hello, World! \n");  
return 0;
```

### Comments

Comments are like helping text in your C program and they are ignored by the compiler. They start with /\* and terminate with the characters \*/ as shown below –

```
/* my first program in C */
```

You cannot have comments within comments and they do not occur within a string or character literals.

### Identifiers

A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z, a to z, or an underscore '\_' followed by zero or more letters, underscores, and digits (0 to 9).

C does not allow punctuation characters such as @, \$, and % within identifiers. C is a **case-sensitive** programming language. Thus, *Manpower* and *manpower* are two different identifiers in C. Here are some examples of acceptable identifiers –

```
mohd    zara   abc   move_name  a_123  
myname50 _temp  j    a23b9   retVal
```

### Keywords

The following list shows the reserved words in C. These reserved words may not be used as constants or variables or any other identifier names.

|          |        |          |          |
|----------|--------|----------|----------|
| Auto     | else   | Long     | switch   |
| Break    | enum   | register | typedef  |
| Case     | extern | return   | union    |
| Char     | float  | short    | unsigned |
| Const    | for    | signed   | void     |
| Continue | goto   | sizeof   | volatile |
| Default  | If     | static   | while    |
| Do       | int    | struct   | _Packed  |
| Double   |        |          |          |

### Whitespace in C

A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it.

Whitespace is the term used in C to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the following statement –

```
int age;
```

there must be at least one whitespace character (usually a space) between `int` and `age` for the compiler to be able to distinguish them. On the other hand, in the following statement –

```
fruit = apples + oranges; // get the total fruit
```

no whitespace characters are necessary between `fruit` and `=`, or between `=` and `apples`, although you are free to include some if you wish to increase readability.

## C - Data Types

Data types in C refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

### C - Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive. Based on the basic types explained in the previous chapter, there will be the following basic variable types –

| Type   | Description  |
|--------|--|
| char   | Typically a single octet(one byte). This is an integer type. |
| int    | The most natural size of integer for the machine.            |
| float  | A single-precision floating point value.                     |
| double | A double-precision floating point value.                     |
| void   | Represents the absence of type.                              |

C programming language also allows to define various other types of variables, which we will cover in subsequent chapters like Enumeration, Pointer, Array, Structure, Union, etc. For this chapter, let us study only basic variable types.

### Variable Definition in C

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows –

```
type variable_list;
```

Here, **type** must be a valid C data type including `char`, `w_char`, `int`, `float`, `double`, `bool`, or any user-defined object; and **variable\_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here –

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

### C - Constants & Literals

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

### C - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators
- Relational Operators
- Logical Operators

- Bitwise Operators
- Assignment Operators

### Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then –

| Operator | Description  | Example     |
|----------|--|-------------|
| +        | Adds two operands.   | A + B = 30  |
| -        | Subtracts second operand from the first.                     | A - B = 10  |
| *        | Multiplies both operands.                                    | A * B = 200 |
| /        | Divides numerator by de-numerator.                           | B / A = 2   |
| %        | Modulus Operator and remainder of after an integer division. | B % A = 0   |
| ++       | Increment operator increases the integer value by one.       | A++ = 11    |
| --       | Decrement operator decreases the integer value by one.       | A-- = 9     |

### Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then –

| Operator | Description  | Example               |
|----------|--|-----------------------|
| ==       | Checks if the values of two operands are equal or not. If yes, then the condition becomes true.                                      | (A == B) is not true. |
| !=       | Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.                 | (A != B) is true.     |
| >        | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.             | (A > B) is not true.  |
| <        | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.                | (A < B) is true.      |
| >=       | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. | (A >= B) is not true. |
| <=       | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.    | (A <= B) is true.     |

### Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then –

| Operator | Description  | Example            |
|----------|--|--------------------|
| &&       | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.   | (A && B) is false. |
|          | Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.   | (A    B) is true.  |
| !        | Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. | !(A && B) is true. |

**Assignment Operators** The following table lists the assignment operators supported by the C language –

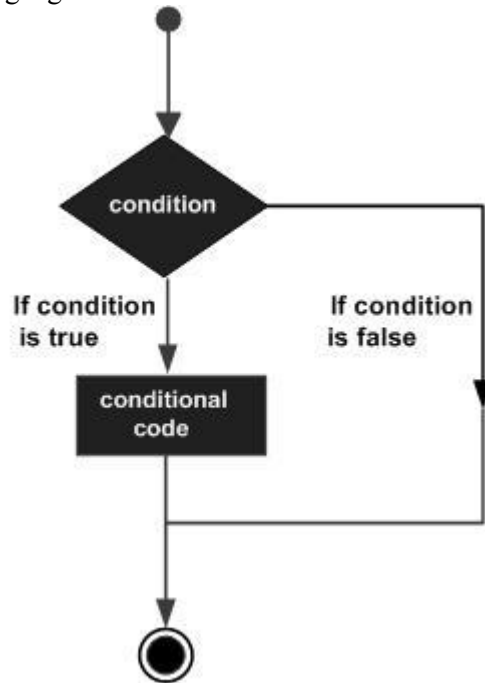
| Operator | Description  | Example                                       |
|----------|--|---|
| =        | Simple assignment operator. Assigns values from right side operands to left side operand   | C = A + B will assign the value of A + B to C |
| +=       | Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.              | C += A is equivalent to C = C + A             |
| -=       | Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand. | C -= A is equivalent to C = C - A             |

|                 |   |   |
|-----------------|---|---|
| <code>*=</code> | Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand. | <code>C *= A</code> is equivalent to <code>C = C * A</code> |
|-----------------|---|---|

## C - Decision Making

Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Show below is the general form of a typical decision making structure found in most of the programming languages –



C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

C programming language provides the following types of decision making statements.

- Simple if statements
- If. Else statements
- Ladder if statements
- Nested if statements
- Switch statements

## The ?: Operator

We have covered **conditional operator ? :** in the previous chapter which can be used to replace **if...else** statements. It has the following general form –

```
Exp1 ? Exp2 : Exp3;
```

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

The value of a ? expression is determined like this –

- Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression.
- If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

## C - Loops

You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

C programming language provides the following types of Looping statements

- While loop

- Do while loop
- For loop

## C - Functions

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

The C standard library provides numerous built-in functions that your program can call. For example, **strcat()** to concatenate two strings, **memcpy()** to copy one memory location to another location, and many more functions.

A function can also be referred as a method or a sub-routine or a procedure, etc.

### Defining a Function

The general form of a function definition in C programming language is as follows –

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

A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function –

- **Return Type** – A function may return a value. The **return\_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return\_type is the keyword **void**.
- **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **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. The parameter list refers to the type, order, and number of the parameters of a function. 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.

### Example

Given below is the source code for a function called **max()**. This function takes two parameters num1 and num2 and returns the maximum value between the two –

```
/* function returning the max between two numbers */
int max(int num1, int num2) {
    /* local variable declaration */
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

### Function Declarations

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts –

```
return_type function_name( parameter list );
```

For the above defined function max(), the function declaration is as follows –

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

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration –

```
int max(int, int);
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

### Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example –

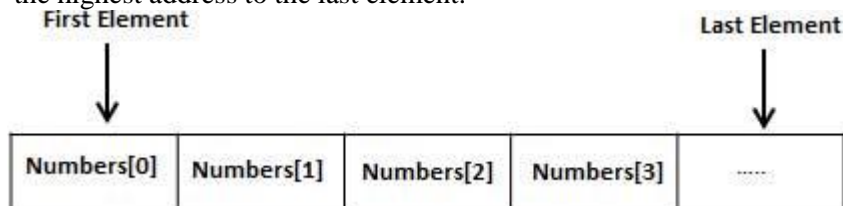
```
#include <stdio.h>
/* function declaration */
int max(int num1, int num2);
int main () {
    /* local variable definition */
    int a = 100;
    int b = 200;
    int ret;
    /* calling a function to get max value */
    ret = max(a, b);
    printf( "Max value is : %d\n", ret );
    return 0; }
/* function returning the max between two numbers */
int max(int num1, int num2) { /* local variable declaration */
    int result;
    if (num1 > num2)    result = num1;
    else    result = num2;    return result; }
```

### C - Arrays

Arrays are a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



### Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows –

```
type arrayName [ arraySize ];
```

This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement –

```
double balance[10];
```

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

### Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows –

```
double balance[5] = { 1000.0, 2.0, 3.4, 7.0, 50.0};
```

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write –

```
double balance[] = { 1000.0, 2.0, 3.4, 7.0, 50.0};
```

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array –

```
balance[4] = 50.0;
```

The above statement assigns the 5<sup>th</sup> element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above –

|         | 0      | 1   | 2   | 3   | 4    |
|---------|--------|-----|-----|-----|------|
| balance | 1000.0 | 2.0 | 3.4 | 7.0 | 50.0 |

### Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example –

```
double salary = balance[9];
```

The above statement will take the 10<sup>th</sup> element from the array and assign the value to salary variable. The following example Shows how to use all the three above mentioned concepts viz. declaration, assignment, and accessing arrays –

```
#include <stdio.h>
int main () {
    int n[ 10 ]; /* n is an array of 10 integers */
    int i,j;
    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ ) {
        n[ i ] = i + 100; /* set element at location i to i + 100 */
    }
    /* output each array element's value */
    for ( j = 0; j < 10; j++ ) {
        printf("Element[%d] = %d\n", j, n[j] );
    }
    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
```

```
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

## C - Pointers

Pointers in C are easy and fun to learn. Some C programming tasks are performed more easily with pointers, and other tasks, such as dynamic memory allocation, cannot be performed without using pointers. So it becomes necessary to learn pointers to become a perfect C programmer. Let's start learning them in simple and easy steps.

As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory. Consider the following example, which prints the address of the variables defined –

```
#include <stdio.h>
int main () {
    int var1;
    char var2[10];
    printf("Address of var1 variable: %x\n", &var1 );
    printf("Address of var2 variable: %x\n", &var2 );
    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6
```

## What are Pointers?

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is –

```
type *var-name;
```

Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk \* used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations –

```
int  *ip;  /* pointer to an integer */
double *dp; /* pointer to a double */
float *fp; /* pointer to a float */
char  *ch  /* pointer to a character */
```

## C - Strings

Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization then you can write the above statement as follows –

```
char greeting[] = "Hello";
```

Following is the memory presentation of the above defined string in C/C++ –



|          |         |         |         |         |         |         |
|----------|---------|---------|---------|---------|---------|---------|
| Index    | 0       | 1       | 2       | 3       | 4       | 5       |
| Variable | H       | e       | l       | l       | o       | \0      |
| Address  | 0x23451 | 0x23452 | 0x23453 | 0x23454 | 0x23455 | 0x23456 |

Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print the above mentioned string –

```
#include <stdio.h>
int main () {
    char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
    printf("Greeting message: %s\n", greeting );
    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Greeting message: Hello

C supports a wide range of functions that manipulate null-terminated strings –

| S.N. | Function & Purpose  |
|------|---|
| 1    | <b>strcpy(s1, s2);</b><br>Copies string s2 into string s1.  |
| 2    | <b>strcat(s1, s2);</b><br>Concatenates string s2 onto the end of string s1.                                   |
| 3    | <b>strlen(s1);</b><br>Returns the length of string s1.  |
| 4    | <b>strcmp(s1, s2);</b><br>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><br>Returns a pointer to the first occurrence of character ch in string s1.             |
| 6    | <b>strstr(s1, s2);</b><br>Returns a pointer to the first occurrence of string s2 in string s1.                |

The following example uses some of the above-mentioned functions –

```
#include <stdio.h>
#include <string.h>
int main () {
    char str1[12] = "Hello";
    char str2[12] = "World";
    char str3[12];
    int len ;
    /* copy str1 into str3 */
    strcpy(str3, str1);
    printf("strcpy( str3, str1) : %s\n", str3 );
    /* concatenates str1 and str2 */
    strcat( str1, str2);
    printf("strcat( str1, str2): %s\n", str1 );
    /* total length of str1 after concatenation */
```

```
len = strlen(str1);
printf("strlen(str1) : %d\n", len );
return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```

## C - Structures

Arrays allow to define type of variables that can hold several data items of the same kind. Similarly **structure** is another user defined data type available in C that allows to combine data items of different kinds.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

- Title
- Author
- Subject
- Book ID

### Defining a Structure

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows –

```
struct [structure tag] {
    member definition;
    member definition;
    ...
    member definition;
} [one or more structure variables];
```

The **structure tag** is optional and each member definition is a normal variable definition, such as `int i;` or `float f;` or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure –

```
struct Books {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
} book;
```

### Accessing Structure Members

To access any member of a structure, we use the **member access operator** (`.`). The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword **struct** to define variables of structure type. The following example shows how to use a structure in a program –

```
#include <stdio.h>
#include <string.h>
struct Books {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};
```

```

int main() {
    struct Books Book1;    /* Declare Book1 of type Book */
    struct Books Book2;    /* Declare Book2 of type Book */
    /* book 1 specification */
    strcpy( Book1.title, "C Programming");
    strcpy( Book1.author, "Nuha Ali");
    strcpy( Book1.subject, "C Programming Tutorial");
    Book1.book_id = 6495407;

    /* book 2 specification */
    strcpy( Book2.title, "Telecom Billing");
    strcpy( Book2.author, "Zara Ali");
    strcpy( Book2.subject, "Telecom Billing Tutorial");
    Book2.book_id = 6495700;

    /* print Book1 info */
    printf( "Book 1 title : %s\n", Book1.title);
    printf( "Book 1 author : %s\n", Book1.author);
    printf( "Book 1 subject : %s\n", Book1.subject);
    printf( "Book 1 book_id : %d\n", Book1.book_id);

    /* print Book2 info */
    printf( "Book 2 title : %s\n", Book2.title);
    printf( "Book 2 author : %s\n", Book2.author);
    printf( "Book 2 subject : %s\n", Book2.subject);
    printf( "Book 2 book_id : %d\n", Book2.book_id);
    return 0;
}

```

When the above code is compiled and executed, it produces the following result –

```

Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700

```

1) // program to print the message.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
printf ("welcome to c program");
getch ();
}
```

2) // program to find sum and average of two numbers.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int a, b;
float sum, avg;
clrscr ();
printf ("enter the value of a and b :");
scanf ("%d%d", &a, &b);
sum=a+b;
avg=sum/2;
printf ("sum: =%f", sum);
printf ("\n average: =%f", avg);
getch ();
}
```

3) // program to find area and perimeter of rectangle.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int l, b;
float area, peri;
clrscr ();
printf ("enter the value of l and b :");
scanf ("%d%d", &l, &b);
area=l*b;
peri=2*(l+b);
printf ("area: =%f", area);
printf ("\n perimeter: =%f", peri);
getch ();
}
```

4) // program to find area and circumference of circle.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int r;
float area, cir;
clrscr ();
printf ("enter the value of r :");
scanf ("%d", &r);
area=3.142*r*r;
```

```
cir=2*3.142*r;
printf ("area: =%f", area);
printf ("\n circumference: =%f", cir);
getch ();
}
```

5) // program to find maximum of two no's using conditional operator.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int a, b, max;
clrscr ();
printf ("enter the value of a and b :");
scanf ("%d%d", &a, &b);
max=(a>b) ? a: b;
printf ("%d= maximum of two no's", max);
getch ();
}
```

6) // program to find even or odd no's.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int a;
clrscr ();
printf ("enter the value of a :");
scanf ("%d", &a);
if (a%2==0)
printf ("%d is even", a);
if (a%2!=0)
printf ("%d is odd", a);
getch ();
}
```

```
7) #include<stdio.h>
#include<conio.h>
void main ()
{
int a, b;
clrscr ();
printf ("enter the value of a and b :");
scanf ("%d%d", &a, &b);
if (a>b)
printf ("%d is big", a);
else
printf ("%d is big", b);
getch ();
}
```

8)

// program to find biggest of three no's.  
#include<stdio.h>

```
#include<conio.h>
void main ()
{
int a, b, c;
clrscr ();
printf ("enter the value of a, b and c :");
scanf ("%d%d%d", &a, &b, &c);
if (a>b)
{
if (a>b)
printf ("%d is big", a);
else
printf ("%d is big", c);
}
else
{
if (b>c)
printf ("%d is big", b);
else
printf ("%d is big", c);
}
getch ();
}
```

9)

```
// program to display n natural numbers using do while.
#include<stdio.h>
#include<conio.h>
void main ()
{
int i=1 , n;
clrscr ();
printf ("enter the value of n:");
scanf ("%d", &n);
do
{
printf ("%d\n", i);
i++;
} while (i<=10);
getch ();
}
```

10)

```
1) // program to display sum of n natural numbers using do while.
#include<stdio.h>
#include<conio.h>
void main ()
{
int i=1, n, sum=0;
clrscr ();
printf ("enter the value of n:");
scanf ("%d", &n);
```

```
do
{
printf ("%d\n", i);
sum=sum+i;
i++;
} while (i<=10);
printf ("sum=%d", sum);
getch ();
}
```

11)

// program to generate to print first n fibonacci number.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main ()
{
int fst=0, snd=1, n, count=2, next;
clrscr ();
printf ("enter the value of n :");
scanf ("%d", &n);
printf ("%d\n%d\n", fst, snd);
while (count<n)
{
next=fst+ snd;
printf ("%d\n", next);
count++;
fst=snd;
snd=next;
}
getch ();
}
```

12) // program to find sum & avg of array elements.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int a [100], i, n, sum=0, avg=0;
clrscr ();
printf ("enter the value of n :");
scanf ("%d", &n);
printf ("enter the elements of array :");
for (i=1; i<=n; i++)
scanf ("%d", &a[i]);
printf ("elements of array are\n");
for (i=1; i<=5; i++)
{
sum=sum + a [i];
}
avg= sum/n;
printf ("\n sum=%d", sum);
printf ("\n average=%d", avg);
getch ();}
```

# C++

## What is C++:

C++ is an object-oriented programming language. It is an extension to C programming. C++ programming language was developed in 1980 by **Bjarne Stroustrup** at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX..

**Bjarne Stroustrup** is known as the **founder of C++ language**.

## Object-Oriented Programming (OOPs)

C++ supports the object-oriented programming, the four major pillar of object oriented programming used in C++ are:

**1. Inheritance 2. Polymorphism 3. Encapsulation 4. Abstraction**

## Usage of C++

By the help of C++ programming language, we can develop different types of secured and robust applications:

- ✓ Window application
- ✓ Client-Server application
- ✓ Device drivers
- ✓ Embedded firmware etc.

## C vs C++

| No. | C  | C++  |
|-----|--|--|
| 1)  | C follows the <b>procedural style programming</b> .                              | C++ is multi-paradigm. It supports both <b>procedural and object oriented</b> .            |
| 2)  | Data is less secured in C.   | In C++, you can use modifiers for class members to make it inaccessible for outside users. |
| 3)  | C follows the <b>top-down approach</b> .   | C++ follows the <b>bottom-up approach</b> .  |
| 4)  | C does not support function overloading, Operator Overloading etc. OOP concepts. | C++ supports function overloading, Operator Overloading etc. OOP concepts..                |
| 5)  | In C, you can't use functions in structure.                                      | In C++, you can use functions in structure.  |
| 6)  | In C, <b>scanf()</b> and <b>printf()</b> are mainly used for input/output.       | C++ mainly uses stream <b>cin</b> and <b>cout</b> to perform input and output operations.  |

## C++ Features:

**C++ is object oriented programming language. It provides a lot of features that are given below.**

- The main focus remains on data rather than procedures.
- Object-oriented programs are segmented into parts called objects.
- Data structures are designed to categorize the objects.
- Data member and functions are tied together as a data structure.
- Data can be hidden and cannot be accessed by external functions using access specifier .
- Objects can communicate among themselves using functions.
- New data and functions can be easily added anywhere within a program whenever required.
- Since this is an object-oriented programming language, it follows a bottom up approach, i.e. the execution of codes starts from the main which resides at the lower section and then based on the member function call the working is done from the classes.

**C++ Program:** Basically, a C++ program involves the following section:

Documentation  
Pre-processor Statements  
Global Declarations  
The main () function{  
Local Declarations  
Program Statements & Expressions}  
User Defined Functions

**Example :** //program to display a line of text.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    cout<<"welcome to c++ programming language";
    getch();
}
```

**Standard output stream (cout):**The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The **cout** is used in conjunction with stream **insertion operator (<<)** to display the output on a console

**Standard input stream (cin):**The **cin** is a predefined object of **istream** class. It is connected with the standard input device, which is usually a keyboard. The **cin** is used in conjunction with stream **extraction operator (>>)** to read the input from a console.

**Standard end line (endl):**The **endl** is a predefined object of **ostream** class. It is used to insert a new line characters and flushes the stream.

**C++ Variable:** A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times. **Data\_type variable list;**

The example of declaring variable is given below: **1) int x; 2) float y; 3) char z;**

Here, x, y, z are variables and int, float, char are data types.

We can also provide values while declaring the variables as given below:

**1) int x=5,b=10; //declaring 2 variable of integer type 2)float f=30.8; 3)char c='A';**

- ✓ A variable can have alphabets, digits and underscore.
- ✓ A variable name can start with alphabet and underscore only. It can't start with digit.
- ✓ No white space is allowed within variable name.
- ✓ A variable name must not be any reserved word or keyword e.g. chars, float etc.

**C++ Data Types : There are 4 types of data types in C++ language.**

| Types                  | Data Types                    |
|------------------------|-------------------------------|
| Basic Data Type        | int, char, float, double, etc |
| Derived Data Type      | array, pointer, etc           |
| User Defined Data Type | structure                     |

### **C++ Keywords**

A keyword is a reserved word. You cannot use it as a variable name, constant name etc.

Example: if, else, switch, case, do, while, for, class etc

**C++ Operators:** Operators are special type of functions that takes one or more arguments and produces a new value.

- Arithmetic Operators [+ , - , \* , / , %]
- Relational Operators ]> , >= , < , <= , == , !=]
- Logical Operators [&& , || , ! ]
- Assignment Operator [=]
- Unary operator [++ , --]
- Ternary or Conditional Operator [ ? , : ]
- sizeof operator [sizeof(nt)]
- pointer to member operator [ .\* and ->\*]
- **scope resolution** operator [ :: ]
- **address** operator [ & ]
- **new** operator (memory allocation operator)
- **delete** operator (memory release operator)

### **C++ Control Statement**

#### **if-else**

In C++ programming, if statement is used to test the condition. There are various types of if statements in C++.

- **if statement**
- **if-else statement**
- **if-else-if ladder**

**IF Statement:** The C++ if statement tests the condition. It is executed if condition is true.

**Syntax:** **if(condition) {**  
**//code to be executed }**

Example: 

```
#include <iostream.h>
#include<conio.h>
void main ()
{
    int num = 10;
    if (num % 2 == 0) { cout<<"It is even number"; }
    getch();
}
```

**if-else Statement:** The C++ if-else statement also tests the condition. It executes if block if condition is true otherwise else block is executed.

Syntax:     if(condition) { //code if condition is true }  
              else { //code if condition is false }

Example:

```
#include <iostream.h>
#include <conio.h>
void main () {
    int num;
    cout<<"Enter a Number: ";
    cin>>num;
    if (num % 2 == 0) { cout<<"It is even number"<<endl; }
    else { cout<<"It is odd number"<<endl; }
    getch();
}
```

**if-else-if ladder** : statement executes one condition from multiple statements.

**Syntax:**

```
if(condition1){ //code to be executed if condition1 is true }
else if(condition2){ //code to be executed if condition2 is true }
else if(condition3){ //code to be executed if condition3 is true }
...
else{//code to be executed if all the conditions are false }
```

Example:

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int avg;
    clrscr();
    cout<<"Enter average marks obtained in 5 subjects :";
    cin>>avg;
    cout<<"Your Grade is ";
    if(avg>80)
    { cout<<"A";}
    else if(avg>60 && avg<=80)
    { cout<<"B";}
    else if(avg>40 && avg<=60)
    { cout<<"C";}
    else
    { cout<<"D";}
    getch();
}
```

**switch:** The C++ switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C++.

**Syntax:**

```
switch (expression)
{
    case label1: statements ;
                break;
    case label2: statements;
                break;
    .....
    default: staements;
            break;
}
```

**Example**

**// Program to build a simple calculator using switch Statement**

```
#include <iostream.h>
#include <conio.h>
void main()
{
    char op;
    float num1, num2;
    clrscr();
```



```

cout << "Enter an operator (+, -, *, /): ";
cin >> op;
cout << "Enter two operands: ";
cin >> num1 >> num2;
switch (op)
{
    case '+':
        cout << num1 << " + " << num2 << " = " << num1+num2;
        break;
    case '-':
        cout << num1 << " - " << num2 << " = " << num1-num2;
        break;
    case '*':
        cout << num1 << " * " << num2 << " = " << num1*num2;
        break;
    case '/':
        cout << num1 << " / " << num2 << " = " << num1/num2;
        break;
    default:
        // operator is doesn't match any case constant (+, -, *, /)
        cout << "Error! operator is not correct";
        break;
}
getch();
}

```

### **Looping:**

Loops are used in programming to repeat a specific block until some end condition is met. There are three types of loops in C++ programming:

- 1) for loop
- 2) while loop
- 3) do while loop

### **While loop:**

**Syntax:**     **while(condition){**  
                   **//code to be executed    }**

### **Example:**

```

// C++ Program to compute factorial of a number
// Factorial of n = 1*2*3...*n
#include <iostream.h>
#include<conio.h>
void main()
{
    int number, i = 1, factorial = 1;
    clrscr();
    cout << "Enter a positive integer: ";
    cin >> number;
    while ( i <= number)
    {
        factorial *= i;    //factorial = factorial * i;
        ++i;
    }
    cout<<"Factorial of "<< number <<" = "<< factorial;
    getch();
}

```

### **Do-While Loop :**

**Syntax:**                 **do**  
                               **{ //code to be executed**  
                               **}while(condition);**

**Example :**            //C++ Program to display natural numbers up to 10.  
                           #include <iostream>

```
#include<conio.h>
void main()
{
    int i = 1;
    do{
        cout<<i<<"\n";
        i++;
    } while (i <= 10) ;
    getch();
}
```

**Example:**

**Syntax:** for(initialization; condition; incr/decr) {  
                     //code to be executed }

**Example:** //program to display your name for specified number of time

```
#include <iostream.h>
#include<conio.h>
void main()
{
    int time,i;
    clrscr();
    cout << "how many times you want to display your name";
    cin >>time;
    for(i=0;i<time;i++)
    {
        0 Cout<<"\n KEONICS";
    }
    getch();
}
```

## C++ Functions

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is main().

### Advantages of using functions in a program

- You can divide your program in logical blocks. ...
- Use of function avoids typing same pieces of code multiple times. ...
- Individual functions can be easily tested.
- In case of any modification in the code you can modify only the function without changing the structure of the program.

### Types of Functions

There are two types of functions in C programming:

**1. Library Functions:** are the functions which are declared in the C++ header files such as ceil(x), floor(x), sqrt(x), sin(x) etc.

**2. User-defined functions:** are the functions which are created by the C++ programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.

### Declaration of a function:

#### Syntax:

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

#### Example:

```
#include<iostream.h>
#include<conio.h>
// function declaration
int max(int num1, int num2);//function Declaration
void main () {
    // local variable declaration:
    int a = 100;
    int b = 200;
    int ret;
    // calling a function to get max value.
    ret = max(a, b);
```

```

        cout << "Max value is : " << ret << endl;
        getch();
    }
    // function returning the max between two numbers
    int max(int num1, int num2) {
        // local variable declaration
        int result;
        if (num1 > num2)
            result = num1;
        else
            result = num2;
        return (result);
    }

```

### Inline functions

When a function is declared as inline, the compiler places a copy of the code of that specific function at each point where the function is called at compile time.

#### Syntax:

```

inline return_type function_name(arguments...)
{
    //function_code
    //return_value;
}

```

#### Example:

```

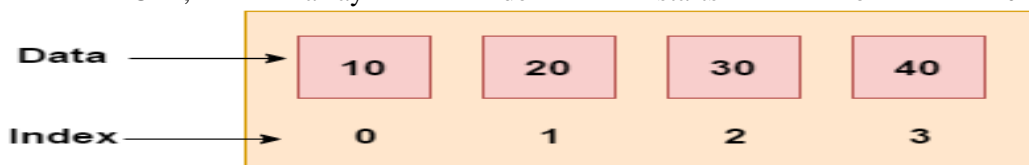
//Simple Inline Function Example Program in C++
#include<iostream.h>
#include<conio.h>
inline int square(int x);
void main() {
    int a = 100, b = 200;
    cout << "Simple Inline Function Example Program in C++\n";
    cout << "\nSquare value for " << a << " is      : " << square(a);
    cout << "\nSquare value for " << b << " is      : " << square(b);
    getch();
}
// Inline square function
inline int square(int x) {
    return (x * x);}

```

### C++ Arrays

Array is a group of similar types of elements that have contiguous memory location.

In C++, array index starts from 0 to (n-1)



#### Advantages of C++ Array

- Code Optimization (less code)
- Random Access
- Easy to traverse data
- Easy to manipulate data
- Easy to sort data etc.

#### Disadvantages of C++ Array -> Fixed size

#### Array Types :

There are 2 types of arrays in C++ programming:

1. Single Dimensional Array
2. Multidimensional Array

#### Single Dimensional Array:

A one-dimensional array is a group of elements having the same data type and same name.

**Declaration:**    `data_type array_name[array_size];`

**Initialization:** `data_type array_name[array_size]=comma_separated_element_list;`

**Example :** // C++ One Dimensional Array

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int arr[5] = { 1, 2, 3, 4, 5};
    int i;
    for(i=0; i<5; i++)
    { cout<<"arr["<<i<<" ] = "<<arr[i]<<"\n";}
    getch();
}
```

### C++ Passing Array to Function

**functionname(arrayname); //passing array to function**

**Example:**

```
#include <iostream.h>
#include<conio.h>
void printArray(int arr[5]);
void main()
{
    int arr1[5] = { 10, 20, 30, 40, 50 };
    int arr2[5] = { 5, 15, 25, 35, 45 };
    printArray(arr1); //passing array to function
    printArray(arr2);
}
void printArray(int arr[5])
{
    cout << "Printing array elements:"<< endl;
    for (int i = 0; i < 5; i++)
    {
        cout<<arr[i]<<"\n";
    }
    getch();
}
```

C++ Multidimensional Arrays:

**Declare Two Dimensional Array->**    `data_type array_name[row_size][column_size];`

**Initialization:**

`data_type array_name[row_size][column_size] = { comma_separated_value_list } ;`

**Example :**

// C++ Two Dimensional Array

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int arr[5][2] = { { 1, 2}, { 1, 3}, { 1, 4}, { 1, 5}, { 1, 6} };
    int i, j;
    for(i=0; i<5; i++)
    {
        for(j=0; j<2; j++)
        {
            cout<<"arr["<<i<<" ]["<<j<<" ] = "<<arr[i][j]<<"\n";
        }
    }
    getch();
}
```

## C++ OOPs Concepts

The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language. Object Oriented Programming is a paradigm that provides many concepts such as **inheritance, data binding, polymorphism etc.**

The programming paradigm where everything is represented as an object is known as truly object-oriented programming language.

### OOPs (Object Oriented Programming System):

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

#### Object

Any entity that has state and behaviour is known as an object.

For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

#### Class

**Collection of objects** is called class. It is a logical entity.

#### Inheritance

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

#### Polymorphism

**When one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc. In C++, we use Function overloading and Function overriding to achieve polymorphism.

#### Abstraction

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing. In C++, we use abstract class and interface to achieve abstraction.

#### Encapsulation

**Binding (or wrapping) code and data together into a single unit is known as encapsulation.**

For example: capsule, it is wrapped with different medicines.

#### *Advantage of OOPs over Procedure-oriented programming language*

- OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
- OOPs provide data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
- OOPs provide ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

## C++ Object and Class

Since C++ is an object-oriented language, program is designed using objects and classes in C++.

### C++ Object:

Object is a runtime entity, it is created at runtime. Object is an instance of a class. All the members of the class can be accessed through object.

### C++ Class

In C++, object is a group of similar objects.

**Syntax to create class:**

```
Class class_name
{
    Data Members;
    Methods;
}
```

Where class->keyword

Class Name->any valid identifies

**Example:**

```
class A
{
    public:
    double length; // Length of a box
    double breadth; // Breadth of a box
    double height; // Height of a box
}
```

✓ **Private, Protected, Public** is called visibility labels.

- ✓ The members that are declared private can be accessed only from within the class.
- ✓ Public members can be accessed from outside the class also.
- ✓ In C++, data can be hidden by making it private.

### **C++ Object and Class Example**

Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

```
#include <iostream.h>
#include <conio.h>
class Student {
public:
    int id;//data member (also instance variable)
    string name;//data member(also instance variable)
};
void main() {
    Student s1; //creating an object of Student
    s1.id = 201;
    s1.name = "Sonoo Jaiswal";
    cout<<s1.id<<endl;
    cout<<s1.name<<endl;
    getch();
}
```

### **C++ Class Example: Store and Display Employee Information**

```
#include <iostream.h>
#include <conio.h>
class Employee {
public:
    int id;//data member (also instance variable)
    float salary; //data member(also instance variable)
    void insert(int i, float s)
    {
        id = i;
        salary = s;
    }
    void display()
    {
        cout<<"id="<<id<<"\nsalary= "<<salary<<endl;
    }
};
void main() {
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    e1.insert(201,15000);
    e2.insert(202, 29000);
    e1.display();
    e2.display();
    getch(); }
```

### **C++ Constructor**

In C++, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C++ has the same name as class or structure.

**There can be two types of constructors in C++.**

1. Default constructor
2. Parameterized constructor

**C++ Default Constructor** :A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

Example:

```
#include <iostream.h>
#include <conio.h>
class Employee
```

```

        {    public:      Employee()      {
            cout<<"Default Constructor Invoked"<<endl;      }
        };
void main()
{
    Employee e1; //creating an object of Employee
    Employee e2;
    getch();    }
C++ Destructor

```

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a **tilde sign (~)**.

### **C++ friend function**

If a function is defined as a friend function in C++ then the protected and private data of a class can be accessed using the function. By using the keyword **friend** compiler knows the given function is a friend function. For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword **friend**.

### **Declaration of friend function in C++**

#### **Syntax:**

```

class class_name
{
    friend data_type function_name(arguments);
};

```

**Example:**        `#include<iostream.h>`  
`#include<conio.h>`  
class base {  
    int val1, val2;  
public:  
    void get() {  
        cout << "Enter two values:";  
        cin >> val1>>val2;    }  
    friend float mean(base ob);  
};  
float mean(base ob) {  
    return float(ob.val1 + ob.val2) / 2;}  
void main() {  
    clrscr();  
    base obj;  
    obj.get();  
    cout << "\n Mean value is : " << mean(obj);  
    getch(); }

### **C++ Inheritance**

In C++, inheritance is a process in which one object acquires all the properties and behaviours of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviours which are defined in other class.

In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

#### **Forms of inheritance in c++:**

|                          |   |
|--------------------------|---|
| Single inheritance       | When one class inherits another class, it is known as single level inheritance  |
| Multiple inheritance     | One derived class with several base classes is called multiple inheritance  |
| Hierarchical inheritance | One class inherits more than one derived class is known as hierarchical inheritance   |
| Multilevel inheritance   | When one class inherits another class which is further inherited by another class, it is known as multi-level inheritance in C++. Inheritance is transitive so the last derived class acquires all the members of all its base classes. |
| Hybrid inheritance       | Combination of all forms of inheritance is known as hybrid inheritance.   |

#### **Creation of derived class:**

**Syntax:** class derived-class: visibility-mode base-class

```
{
    Class body
}
```

**C++ Single Inheritance Example:**

```
#include<iostream.h>
#include<conio.h>
class emp {
    public:
        int eno;
        char name[20], des[20];
        void get() {
            cout << "Enter the employee number:";
            cin>>eno;
            cout << "Enter the employee name:";
            cin>>name;
            cout << "Enter the designation:";
            cin>>des;
        }
};

class salary : public emp
{
    float bp, hra, da, pf, np;
    public:
        void get1() {
            cout << "Enter the basic pay:";
            cin>>bp;
            cout << "Enter the HRA:";
            cin>>hra;
            cout << "Enter the DA :";
            cin>>da;
            cout << "Enter the PF";
            cin>>pf;
        }
        void calculate() {    np = bp + hra + da - pf;    }
        void display() {
            cout << eno << "\t" << name << "\t" << des << "\t" << bp << "\t" << hra << "\t" << da << "\t" <<
            pf << "\t" << np << "\n";
        }
};

void main()
{
    int i, n;
    char ch;
    salary s[10];
    clrscr();
    cout << "Enter the number of employee:";
    cin>>n;
    for (i = 0; i < n; i++)
    {
        s[i].get();
        s[i].get1();
        s[i].calculate();
    }
    cout << "\ne_no \t e_name\t des \t bp \t hra \t da \t pf \t np \n";
    for (i = 0; i < n; i++)
    {
        s[i].display();
    }
    getch();
}
```

**C++ Multiple Inheritance Example**



```

#include<iostream.h>
#include<conio.h>
class student {
    private:
        int rno, m1, m2;
    public:
        void get() {
            cout << "Enter the Roll no :";
            cin>>rno;
            cout << "Enter the two marks  :";
            cin >> m1>>m2;
        }
};

class sports
{
    private:
        int sm; // sm = Sports mark
    public:
        void gets() {
            cout << "\nEnter the sports mark :";
            cin>>sm;
        }
};

class statement : public student, public sports
{
    int tot, avg;
    public:
        void display() {
            tot = (m1 + m2 + sm);
            avg = tot / 3;
            cout << "\n\nRoll No   : " << rno << "\nTotal       : " << tot;
            cout << "\nAverage   : " << avg;
        }
};

void main() {
    clrscr();
    statement obj;
    obj.get();
    obj.gets();
    obj.display();
    getch();
}

```

### **C++ Hierarchical Inheritance**

**Syntax:**

```

Class A{
    public void methodA()
    { //Do Something }
}

Class B : public A {
    public void methodB()
    { //Do Something }
}

Class C : public A {
    public void methodC()
    { //Do Something }
}

```

### **C++ Multi Level Inheritance**

**Syntax:**

```

class A{ ... .. };
class B: public A{... .. };
class C: public B{... .. };

```

### **C++ Polymorphism**

The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms. It is a Greek word.

✂ **There are two types of polymorphism in C++:**

- ✓ **Compile time polymorphism:** It is achieved by function overloading and operator overloading which is also known as static binding or early binding.
- ✓ **Runtime polymorphism:** It is achieved by method overriding which is also known as dynamic binding or late binding.

**C++ Overloading (Function and Operator)**

If we create two or more members having same name but different in number or type of parameter, it is known as C++ overloading.

Types of overloading in C++ are:

- ✓ **Function overloading**
- ✓ **Operators overloading**

✂ **Function Overloading**

Having two or more function with same name but different in parameters, is known as function overloading in C++. The **advantage** of Function overloading is that it increases the readability of the program because you don't need to use different names for same action.

**Example:**

- ✓ `int test() { }`
- ✓ `int test(int a) { }`
- ✓ `float test(double a) { }`
- ✓ `int test(int a, double b) { }`

**Function Overloading Example**

```
#include <iostream>
class Cal {
public:
    int add(int a,int b)
    {
        return (a + b);
    }
    int add(int a, int b, int c)
    {
        return (a + b + c);
    }
};
void main(void)
{
    Cal C;
    cout<<C.add(10, 20)<<endl;
    cout<<C.add(12, 20, 23);
    getch();
}
```

✂ **Operators Overloading**

Operator overloading is used to overload or redefine most of the operators available in C++. It is used to perform operation on user define data type.

The advantage of Operators overloading is to perform different operations on the same operand.

**Following are list of operators, which cannot be overloaded:**

- :: Scope resolution operator**
- .\* Pointer to member operator**
- . Dot member5ship operator**
- ?: conditional operator**
- Sizeof() operator**

---

## Java

Java was developed by James Gosling and his team , from Sun Microsystems in 1995 as an object-oriented language for general-purpose business applications and for interactive, Web-based Internet applications

**Features of JAVA:**

- **Object Oriented** – In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** – Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

- **Simple** – Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** – With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** – Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** – Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** – Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded** – With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted** – Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** – With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** – Java is designed for the distributed environment of the internet.
- **Dynamic** – Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

#### **Tools Need:**

Computer with a minimum of 64 MB of RAM (128 MB of RAM recommended).

also need the following software –

- Linux 7.1 or Windows xp/7/8 operating system
- Java JDK 8
- Microsoft Notepad or any other text editor

| <b>Comparison Between c++ and Java</b>                       |  |
|--|--|
| <b><u>C++</u></b>  | <b><u>Java</u></b>   |
| C++ is platform-dependent.                                   | Java is platform-independent.  |
| C++ is mainly used for system programming.                   | Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.             |
| C++ supports multiple inheritance.                           | Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.                                       |
| C++ supports operator overloading.                           | Java doesn't support operator overloading.   |
| C++ supports pointers. You can write pointer program in C++. | Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java. |
| C++ uses compiler only.                                      | Java uses compiler and interpreter both.   |
| C++ supports both call by value and call by reference.       | Java supports call by value only. There is no call by reference in java.   |
| C++ supports structures and unions.                          | Java doesn't support structures and unions.  |

How to write the simple program of java. We can write a simple hello java program easily after installing the JDK.

To create a simple java program, you need to create a class that contains main method **in any text editor (notepad, notepad++, wordpad etc)**

#### **Creating hello java example**

```
class Simple{
    public static void main(String args[ ]){
        System.out.println("Hello Java");    } }
```

#### **Program Execution:**

Save the above Program as **Simple.java**, To compile and run this program, you need to open command prompt by **start menu -> All Programs -> Accessories -> command prompt**.

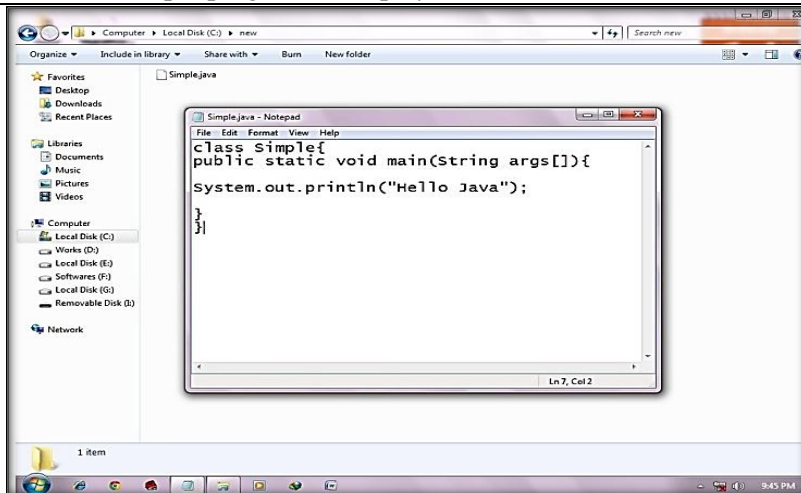
**To compile:** javac Simple.java

**To execute:** java Simple

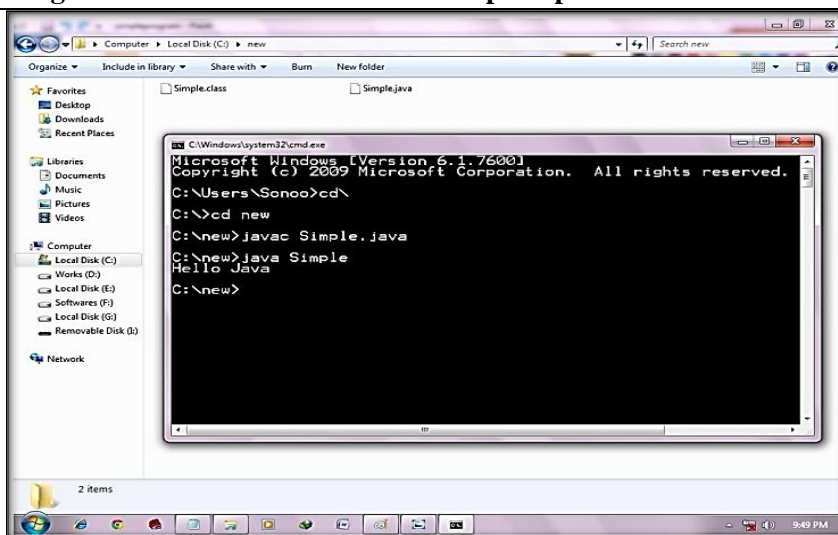
### Parameters used in first java program

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require creating object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[ ] args** is used for command line argument.
- **System.out.println()** is used print statement.

To write the simple program, open notepad by **start menu -> All Programs -> Accessories -> notepad** and write simple program as displayed below:



As displayed in the above diagram, write the simple program of java in notepad and saved it as Simple.java. To compile and run this program, you need to open command prompt by **start menu -> All Programs -> Accessories -> command prompt**.



### Basic Structure of JAVA Program:

A Java program involves the following sections:

- ✓ Documentation Section
- ✓ Package Statement
- ✓ Import Statements
- ✓ Interface Statement
- ✓ Class Definition
- ✓ Main Method Class
  - Main Method Definition

### How to set path in Java:

To set the temporary path of JDK, you need to follow following steps:

- Open command prompt
- Copy the path of jdk/bin directory
- Write in command prompt: set path=copied\_path

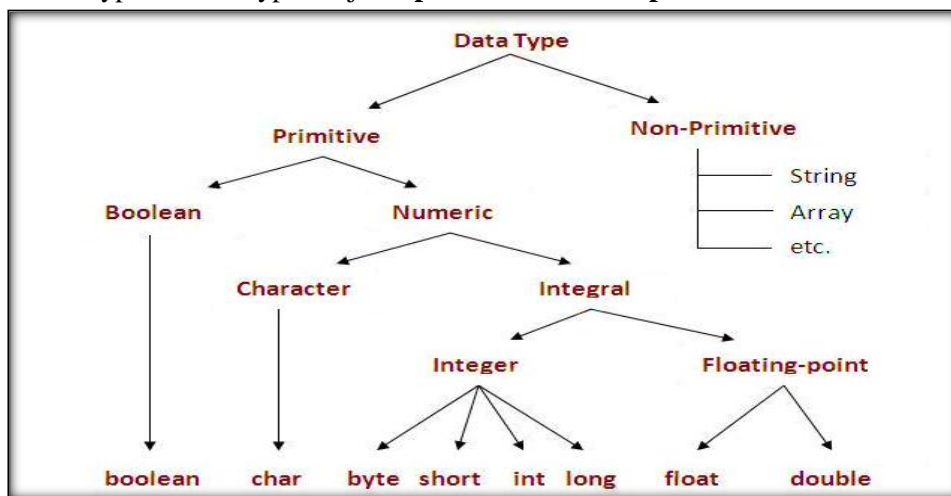
### For Example:

set path=C:\Program Files\Java\jdk1.6.0\_23\bin

### Variables and Data Types in Java

A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype.

There are two types of data types in java: **primitive and non-primitive.**



### Operators in java:

**Operator** in java is a symbol that is used to perform operations.

There are many types of operators in java which are given below:

- Unary Operator [ ++, -- ]
- Arithmetic Operator, [ +, -, \*, /, % ]
- Relational Operator [ >, >=, <, <=, =, != ]
- Logical Operator [ &&, ||, ! ]
- Ternary Operator [ ? : ]
- Assignment Operator [ = ]

### Java If-else Statement

The Java *if statement* is used to test the condition. It checks Boolean condition: *true* or *false*. There are various types of if statement in java.

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

**Java if Statement:** The Java if statement tests the condition. It executes the if block if condition is true.

**Syntax:**                    **if(condition) { //code to be executed }**

**Example:**

```

public class IfExample
{
    public static void main(String[] args)
    {
        int age=20;
        if(age>18){    System.out.print("Eligible to VOTE");    }
    }
}
  
```

**Java if-else Statement:** The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

**Syntax:**                    **if(condition){ //code if condition is true }**

**Example:**

```

else{ //code if condition is false }
public class IfElseExample {
    public static void main(String[] args) {
        int number=13;
        if(number%2==0){    System.out.println("even number");    }
        else{    System.out.println("odd number");    }
    } }

```

**Java if-else-if ladder Statement:** The if-else-if ladder statement executes one condition from multiple statements.

**Syntax:**

```

if(condition1)
{ //code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}

```

**Example:**

```

public class Test {
    public static void main(String args[]) {
        int x = 30;
        if( x == 10 ) {
            System.out.print("Value of X is 10");
        }else if( x == 20 ) {
            System.out.print("Value of X is 20");
        }else if( x == 30 ) {
            System.out.print("Value of X is 30");
        }else {
            System.out.print("This is else statement");
        }
    }
}

```

### **Java Switch Statement**

**Syntax:**

```

switch(expression){
case value1:
    //code to be executed;
    break;
case value2:
    //code to be executed;
    break;
.....
default:
    code to be executed if all cases are not matched;
}

```

**Example:**

```

public class Test {
    public static void main(String args[]) {
        char grade = 'C';

        switch(grade) {
            case 'A' :
                System.out.println("Excellent!");
                break;
            case 'B' :
            case 'C' :
                System.out.println("Well done");
        }
    }
}

```

```

        break;
    case 'D' :
        System.out.println("You passed");
    case 'F' :
        System.out.println("Better try again");
        break;
    default :
        System.out.println("Invalid grade");
    }
    System.out.println("Your grade is " + grade);
}
}

```

## Loops in Java

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in java.

- for loop
- while loop
- do-while loop

### Java For Loop

**Syntax:**                    **for(initialization;condition;incr/decr){**  
                                  **//code to be executed**  
                                  **}**

**Example:**                public class ForExample {  
                               public static void main(String[] args) {  
                                   for(int i=1;i<=10;i++){  
                                      System.out.println(i);  
                                   }  
                               } } }

### Java While Loop

**Syntax:**                    **while(condition){**  
                                  **//code to be executed }**

**Example:**                public class WhileExample {  
                               public static void main(String[] args) {  
                                   int i=1;  
                                   while(i<=10){  
                                      System.out.println(i);  
                                   i++;  
                                   } } }

### Java Do While Loop

**Syntax:**                    **do{**  
                                  **//code to be executed**  
                                  **}while(condition);**

**Example:**                public class DoWhileExample {  
                               public static void main(String[] args) {  
                                   int i=1;  
                                   do{  
                                      System.out.println(i);  
                                   i++;  
                                   }while(i<=10);  
                               }  
                               }

## Java Comments

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

**Types of Java Comments: There are 3 types of comments in java.**

1. Single Line Comment    //This is single line comment

## 2. Multi Line Comment

```
/* This is  
multi line comment */
```

## 3. Documentation Comment

```
/**  
This  
is documentation  
comment  
*/
```

### **\*Example Programs:**

#### 1. // Fibonacci series program in java without using recursion

```
class FibonacciExample{  
    public static void main(String args[])  
    {   int n1=0,n2=1,n3,i,count=10;  
        System.out.print(n1+" "+n2);//printing 0 and 1  
        for(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed  
        {   n3=n1+n2;  
            System.out.print(" "+n3);  
            n1=n2;  
            n2=n3;  
        } } }
```

#### 2./\* prime number program in java. In this java program, we will take a number variable and check whether the number is prime or not.\*/

```
public class PrimeExample{  
    public static void main(String args[]){  
        int i,m=0,flag=0;  
        int n=3;//it is the number to be checked  
        m=n/2;  
        if(n==0||n==1){  
            System.out.println(n+" is not prime number");  
        }else{  
            for(i=2;i<=m;i++){  
                if(n%i==0){  
                    System.out.println(n+" is not prime number");  
                    flag=1;  
                    break;    }    }  
            if(flag==0) { System.out.println(n+" is prime number"); }  
        } //end of else  
    } }
```

#### 3.// factorial Program using loop in java.

```
class FactorialExample{  
    public static void main(String args[]){  
        int i,fact=1;  
        int number=5;//It is the number to calculate factorial  
        for(i=1;i<=number;i++){  
            fact=fact*i;  
        }  
        System.out.println("Factorial of "+number+" is: "+fact);  
    }  
}
```

### **OOPs (Object Oriented Programming System)**

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction



- Encapsulation

**Object:** Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

**Class:** Collection of objects is called class. It is a logical entity.

**Inheritance:** When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Polymorphism:** When one task is performed by different ways i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

**Abstraction:** Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

**Encapsulation:** Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

### Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

**Syntax to declare a class:**

```
class <class_name>{
    field;
    method;
}
```

**Example:**

```
1) class Student{
    int id =100;//field or data member or instance variable
    String name;

    public static void main(String args[]){
        Student s1=new Student();//creating an object of Student
        System.out.println(s1.id);//accessing member through reference variable
        System.out.println(s1.name); } }
```

### Object and Class

**Example: Rectangle**

```
class Rectangle{
    int length;
    int width;
    void insert(int l, int w){
        length=l;
        width=w;
    }
    void calculateArea(){System.out.println(length*width);}
}
class TestRectangle1{
    public static void main(String args[]){
        Rectangle r1=new Rectangle();
        Rectangle r2=new Rectangle();
        r1.insert(11,5);
        r2.insert(3,15);
        r1.calculateArea();
        r2.calculateArea(); } }
```

### Inheritance in Java

**Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parent object. It is an important part of OPPs(Object Oriented programming system).

**Terms used in Inheritance**

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in previous class.

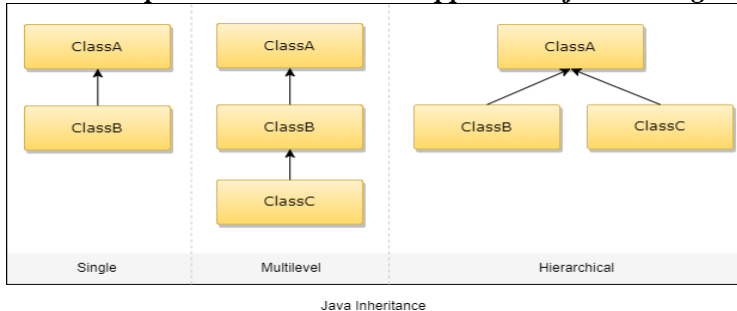
#### Syntax of Deriving a new class:

```
class Subclass-name extends Superclass-name
{
    //methods and fields
}
```

#### Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.[In java programming, multiple and hybrid inheritance is supported through interface only.]

**Note: Multiple inheritance is not supported in java through class.**



### 1. Single Inheritance

When a single class gets derived from its base class, then this type of inheritance is termed as single inheritance.

#### Example:

```
class Teacher {
    void teach() {
        System.out.println("Teaching subjects");
    }
}
class Students extends Teacher {
    void listen() {
        System.out.println("Listening to teacher");
    }
}
class CheckForInheritance {
    public static void main(String args[]) {
        Students s1 = new Students();
        s1.teach();
        s1.listen();
    }
}
```

**2. Multi-Level Inheritance:** In this type of inheritance, a derived class gets created from another derived class and can have any number of levels.

#### Example :

```
class Shape {
    public void display() {
        System.out.println("Inside display"); } }
class Rectangle extends Shape {
    public void area() {
        System.out.println("Inside area"); } }
class Cube extends Rectangle { public void volume() {
    System.out.println("Inside volume"); } }
public class Tester { public static void main(String[] arguments) {
    Cube cube = new Cube();
    cube.display();
    cube.area();
    cube.volume(); } }
```

**3. Hierarchical Inheritance:** In this type of inheritance, there are more than one derived classes which get created from one single base class.

**Example:**

```
class A
{ public void methodA()
{   System.out.println("method of Class A"); } }
class B extends A{ public void methodB()
{   System.out.println("method of Class B"); } }
class C extends A
{ public void methodC() {   System.out.println("method of Class C"); } }
class D extends A
{ public void methodD() {   System.out.println("method of Class D"); } }
class JavaExample
{ public static void main(String args[]) {
    B obj1 = new B();
    C obj2 = new C();
    D obj3 = new D();    //All classes can access the method of class A
    obj1.methodA();
    obj2.methodA();
    obj3.methodA(); } }
```

**Java Package**

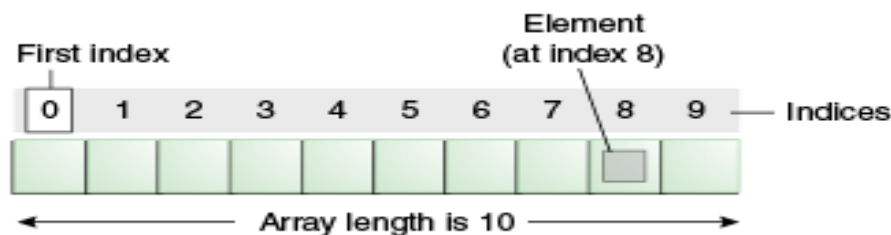
A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

**Simple example of java package**

The **package keyword** is used to create a package in java.



```
//save as Simple.java
package mypack;
public class Simple{
    public static void main(String args[]){
        System.out.println("Welcome to package");    } }
```

**Access Modifiers in java**

**There are 4 types of java access modifiers:**

1. **private:** The private access modifier is accessible only within class.
2. **default:** If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package.
3. **protected:** The **protected access modifier** is accessible within package and outside the package but through inheritance only. The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
4. **public:** The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

**Java Array**

Java array is an object that contains elements of similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.

**Advantage of Java Array**

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
- **Random access:** We can get any data located at any index position.

## Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

## Single dimensional array in java

### Syntax to Declare an Array in java

```
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];
```

### Instantiation of an Array in java

```
arrayRefVar=new datatype[size];
```

### Example:

```
class Testarray{ public static void main(String args[]){
    int a[]=new int[5];//declaration and instantiation
    a[0]=10;//initialization
    a[1]=20;
    a[2]=70;
    a[3]=40;
    a[4]=50;
    //printing array
    for(int i=0;i<a.length;i++)//length is the property of array
    System.out.println(a[i]);
}}
```

### Declaration, Instantiation and Initialization of Java Array

```
class Testarray1{ public static void main(String args[]){
    int a[]={33,3,4,5};//declaration, instantiation and initialization
    //printing array
    for(int i=0;i<a.length;i++)//length is the property of array
    System.out.println(a[i]);
}}
```

## Multidimensional array in java

### Syntax to Declare Multidimensional Array in java

```
dataType[ ][ ] arrayRefVar; (or)
dataType [ ][ ]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
```

### Instantiation of an Multi Dimensional Array in java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

### Example:

```
class Testarray3{
    public static void main(String args[]){
        //declaring and initializing 2D array
        int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
        //printing 2D array
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                System.out.print(arr[i][j]+" ");
            } System.out.println();
        }
    }
}
```

## Java String

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. The java.lang.String class is used to create string object.

### Example:

1. char[] ch={'j','a','v','a','t','p','o','t','n','t'};
2. String s=new String(ch);

**Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

There are two ways to create String object:

1. By string literal→ String s="welcome";
2. By new keyword→ String s=new String("Welcome");

### Example:

```
public class StringExample{
```

```

public static void main(String args[ ]){
String s1="java";
char ch[ ]={'s','t','r','i','n','g','s'};
String s2=new String(ch);
String s3=new String("example");
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}

```

### Exception Handling in Java

The **exception handling** in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

**Exception Handling** is a mechanism to handle runtime errors such as ClassNotFoundException, IO etc.

### Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method. Java try block must be followed by either catch .

### Syntax of java try-catch

```

try{
//code that may throw exception
}catch(Exception_class_Name ref){ }

```

### Example:

```

public class Testtrycatch2{
public static void main(String args[]){
try{
int data=50/0;
}catch(ArithmeticException e){System.out.println(e);}
System.out.println("rest of the code...");
}}

```

**Compile by:** javac Testtrycatch2.java

**Run by:** java Testtrycatch2

java.lang.ArithmeticException: / by zero  
rest of the code...

# PHP

## What is PHP

- PHP stands for Hypertext Preprocessor.
- PHP is an interpreted language, i.e., there is no need for compilation.
- PHP is a server-side scripting language.
- PHP is faster than other scripting languages, for example, ASP and JSP.

## Web Development

PHP is widely used in web development nowadays. PHP can develop dynamic websites easily. But you must have the basic the knowledge of following technologies for web development as well.

- HTML
- CSS
- JavaScript

## Characteristics of PHP

Five important characterant characteristics make PHP's practical nature possible –

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

## PHP Features

### There are given many features of PHP.

- **Performance:** Script written in PHP executes much faster then those scripts written in other languages such as JSP & ASP.
- **Open Source Software:** PHP source code is free available on the web, you can developed all the version of PHP according to your requirement without paying any cost.
- **Platform Independent:** PHP are available for WINDOWS, MAC, LINUX & UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.
- **Compatibility:** PHP is compatible with almost all local servers used today like Apache, IIS etc.
- **Embedded:** PHP code can be easily embedded within HTML tags and script.

## Install PHP

To install PHP, we will suggest you to install AMP (Apache, MySQL, PHP) software stack. It is available for all operating systems. There are many AMP options available in the market that are given below:

- **WAMP** for Windows
- **LAMP** for Linux
- **MAMP** for Mac
- **SAMP** for Solaris
- **XAMPP** (Cross, Apache, MySQL, PHP, Perl) for Cross Platform: It includes some other components too such as FileZilla, OpenSSL, Webalizer, Mercury Mail etc.

## PHP Example

It is very easy to create a simple PHP example. To do so, create a file and write HTML tags + PHP code and save this file with .php extension.

All PHP code goes between php tag. A syntax of PHP tag is given below:

```
<?php
//your code here
?>
```

Example:

```
<html>
<body>
<?php
echo "<h2>Hello First PHP</h2>";
?>
```

```
</body>
</html>
```

PHP echo: printing string

```
<?php
echo "Hello by PHP echo";
?>
```

## **Run a PHP program in XAMPP Server**

PHP program can be run under various like WAMP, XAMPP etc.

- **WAMP Server:** this server is a web development platform which helps in creating dynamic web applications.
- **XAMPP Server:** It is a free open source cross-platform web server package.

I am using XamppServer to run my program, you can download it .  
After downloading, just follow the following step to start xampp server:

### **Step1**

Install XAMPP

### **Step2**

Assume you installed xampp in C Drive.

Go to: **C:\xampp\htdocs**

Create your own folder, name it for example as **tutorialspoint**.

### **Step3**

Now create your first php program in xampp and name it as “**add.php**”:

### **Step4**

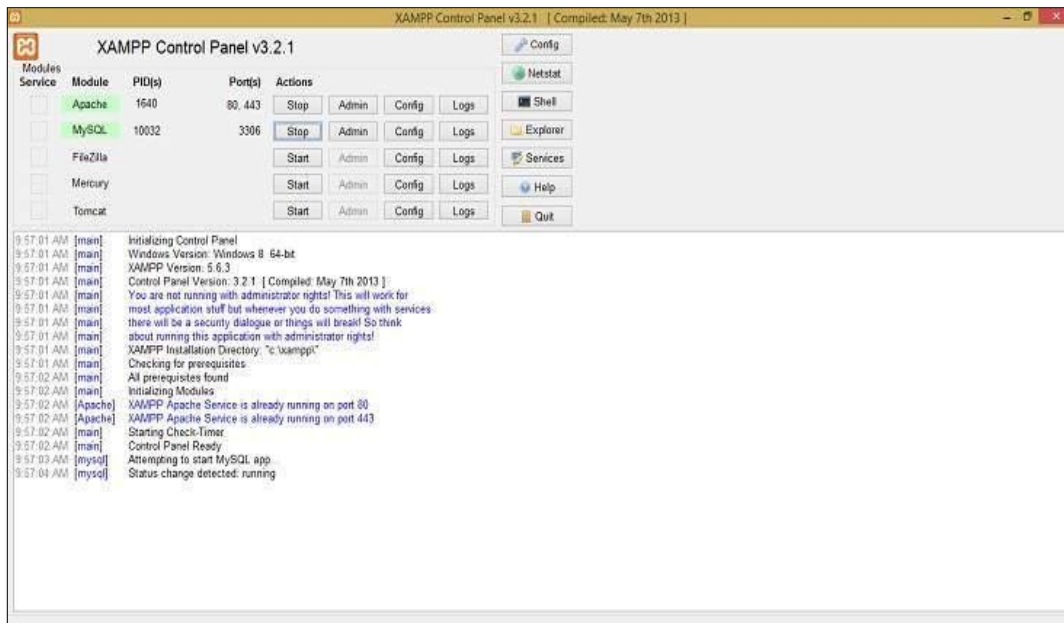
```
<html>
<head><title>Addition php</title></head>
<body>

<?php
    print "<h2>php program to add two numbers...</h2><br />";
    $val1 = 20;
    $val2 = 20;
    $sum = $val1 + $val2; /* Assignment operator */
    echo "Result(SUM): $sum";
?>
```

```
</body>
```

```
</html>
```

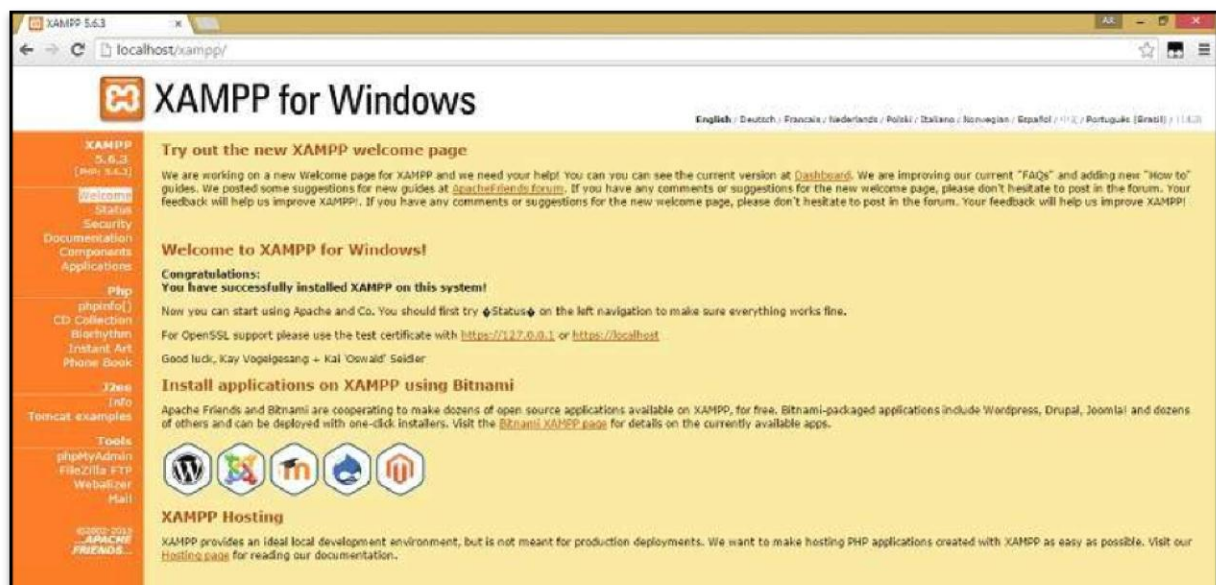
Now double click on “**XAAMP CONTROL PANEL**” on desktop and START “**Apache**”  
(icon also appears on the bottom)



## Step5

Type **localhost** on your browser and press enter:

It will show the following:



## Step6

Now type the following on browser:

**http://localhost/**

## Step7

Click on **"add.php"** and it will show the following:

The RESULT is 40 by adding both the values.

This way you can run your php program in XAMPP server...



### 1) table of a number

```
<?php
define('a', 7);
for($i=1; $i<=10; $i++)
{
    echo $i*a;
    echo '<br>';
}
?>
```

### 2) even odd number check

```
<?php
$number=1233456;
if($number%2==0)
{
    echo "$number is Even Number";
}
else {
    echo "$number is Odd Number"; }
?>
```

### 3) even odd program using form using php

```
<html>
<body>
<form method="post">
    Enter a number:
    <input type="number" name="number">

    <input type="submit" value="Submit">
</form>
</body>
</html>
<?php
if($_POST){
    $number = $_POST['number'];
    //divide entered number by 2
    //if the remainder is 0 then the number is
    even otherwise the number is odd
    if(($number % 2) == 0){
        echo "$number is an Even number";
    }else{
        echo "$number is Odd number";
    } } ?>
```

### 4) swap two numbers

```
<?php
$a = 45;
$b = 78;
// Swapping Logic
$third = $a;
$a = $b;
$b = $third;
echo "After swapping:<br><br>";
echo "a = ".$a." b=".$b;
?>
```

### 5) adding two numbers using form in php

```
<html>
<body>
<form method="post">
Enter First Number:
<input type="number" name="number1" />
<br><br>
Enter Second Number:
<input type="number" name="number2" />
<br><br>
<input type="submit" name="submit" value
="Add">
</form>
<?php
    if(isset($_POST['submit']))
    {
        $number1 = $_POST['number1'];
        $number2 = $_POST['number2'];
        $sum = $number1+$number2;
        echo "The sum of $number1 and $number2
        is: ".$sum;
    }
?>
</body>
</html>
```

### 6) Area of triangle

```
<?php
$base = 10;
$height = 15;
echo "area with base $base and height $hei
ght= " . ($base * $height) / 2;
?>
```

### 7) pattern printing

```
<?php
for($i=0;$i<=5;$i++){
    for($j=5-$i;$j>=1;$j--){
        echo "* ";
    }
    echo "<br>";
}
?>
```

### 8) pattern printing

```
<?php
for($i=0;$i<=5;$i++){
    for($j=1;$j<= $i;$j++){
        echo "* ";
    }
    echo "<br>";
}
?>
```