



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

SCHOOL OF SCIENCE AND HUMANITIES

DEPARTMENT OF MATHEMATICS

I YEAR M.SC PHYSICS

SPHA5203

NUMERICAL METHODS AND COMPUTER PROGRAMMING

NUMERICAL METHODS AND COMPUTER PROGRAMMING (SPH 5107)

UNIT 1

SYSTEM OF EQUATIONS

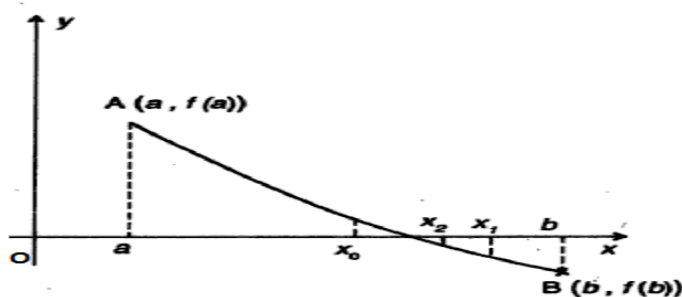
In the field of Science and Engineering, the solution of equations of the form $f(x) = 0$ occurs in many applications. If $f(x)$ is a polynomial of degree two or three or four, exact formulae are available. But, if $f(x)$ is a transcendental function like $a + be^x + c \sin x + d \log x$ etc., the solution is not exact and we do not have formulae to get the solutions. When the coefficients are numerical values, we can adopt various numerical approximate methods to solve such algebraic and transcendental equations. We will see below some methods of solving such numerical equations. From the theory of equations, we recall to our memory the following theorem:

If $f(x)$ is continuous in the interval (a, b) and if $f(a)$ and $f(b)$ are of opposite signs, then the equation $f(x) = 0$ will have atleast one real root between a and b .

The Bisection method (or BOLZANO's method) (or Interval halving method)

Suppose we have an equation of the form $f(x) = 0$ whose solution in the range (a, b) is to be searched. We also assume that $f(x)$ is continuous and it can be algebraic or transcendental. If $f(a)$ and $f(b)$ are of opposite signs, atleast one real root between a and b should exist. For convenience, let $f(a)$ be positive and $f(b)$ be negative. Then atleast one root exists between a and b . As a first approximation, we assume that root to be $x_0 = \frac{a+b}{2}$ (mid point of the ends of the range). Now, find the sign of $f(x_0)$. If $f(x_0)$ is negative, the root lies between a and x_0 . If $f(x_0)$ is positive, the root lies between x_0 and b . Any one of this is true. Suppose

$f(x_0)$ is positive as shown in the Figure, then the root lies between x_0 and



b and take the root as $x_1 = \frac{x_0 + b}{2}$. Now $f(x_1)$ is negative

Hence the root lies between x_0 and x_1 and let the root be (approximate) $x_2 = \frac{x_0 + x_1}{2}$. Now $f(x_2)$ is negative as in the Fig. then

the root lies between x_0 and x_2 and let $x_3 = \frac{x_0 + x_2}{2}$ and so on. In this way, taking the mid-point of the range as the approximate root, we form a sequence of approximate roots x_0, x_1, x_2, \dots whose limit of convergence is the exact root. However, depending on the precision required, we stop the process after some steps. Though simple, the convergence of this method is *slow but sure*.

Note. After n bisections, the length of the subinterval which contains x_n is $\frac{b-a}{2^n}$. If the error is to be made less than a small quantity ϵ , say,

$$\frac{b-a}{2^n} < \epsilon. \text{ That is, } 2^n > \frac{b-a}{\epsilon}$$

The number of iterations n should be greater than $\frac{\log\left(\frac{b-a}{\epsilon}\right)}{\log 2}$.

Example 1. Find the positive root of $x^3 - x = 1$ correct to four decimal places by bisection method.

Solution. Let $f(x) = x^3 - x - 1$

Here, $f(0) = -1 = -ve$ and $f(1) = -ve$

$f(2) = 5 = +ve$. Hence a root lies between 1 and 2. We can take the range as (1, 2) and proceed. We can still shorten the range.

$$f(1.5) = 0.8750 = +ve$$

and

$$f(1) = -1 = -ve$$

Hence, the root lies between 1 and 1.5

...(1)

Let $x_0 = \frac{1 + 1.5}{2} = 1.2500$

$$f(x_0) = f(1.25) = -0.29688$$

Hence the root lies between 1.25 and 1.5

...(2)

Now, $x_1 = \frac{1.25 + 1.5}{2} = 1.3750$

$$f(1.3750) = 0.22461 = +ve$$

The root lies between 1.2500 and 1.3750.

Now $x_2 = \frac{1.2500 + 1.3750}{2} = 1.3125$

$$f(1.3125) = -0.051514$$

Therefore, root lies between 1.3750 and 1.3125

Now $x_3 = \frac{1.3125 + 1.3750}{2} = 1.3438$

$$f(x_3) = f(1.3438) = 0.082832 = +ve$$

The root lies between 1.3125 and 1.3438

Hence $x_4 = \frac{1.3125 + 1.3438}{2} = 1.3282$

$$f(1.3282) = 0.014898$$

Therefore the root lies between 1.3125 and 1.3282

$$x_5 = \frac{1}{2} (1.3125 + 1.3282) = 1.3204$$

$$f(1.3204) = -0.018340$$

The root lies between 1.3204 and 1.3282

$$x_6 = \frac{1}{2} (1.3204 + 1.3282) = 1.3243$$

$$f(1.3243) = -ve$$

Hence, the root lies between 1.3243 and 1.3282

$$\therefore x_7 = \frac{1}{2} (1.3243 + 1.3282) = 1.3263$$

$$f(1.3263) = +ve$$

\therefore The root lies between 1.3243 and 1.3263

$$x_8 = \frac{1}{2} (1.3243 + 1.3263) = 1.3253$$

$$f(1.3253) = +ve$$

The root lies between 1.3243 and 1.3253

$$\therefore x_9 = \frac{1}{2} (1.3243 + 1.3253) = 1.3248$$

$$f(1.3248) = +ve$$

\therefore The root lies between 1.3243 and 1.3248

$$x_{10} = \frac{1}{2} (1.3243 + 1.3248) = 1.32455$$

$$f(1.32455) = -ve$$

The root lies between 1.3248 and 1.32455

$$\therefore x_{11} = \frac{1}{2} (1.3248 + 1.32455) = 1.3247$$

$$f(1.3247) = -ve$$

\therefore The root lies between 1.3247 and 1.3248

$$\text{Hence, } x_{12} = \frac{1}{2} (1.3247 + 1.3248) = 1.32475$$

Therefore, the approximate root is 1.32475

(This is not correct to 5 decimal places).

Example 2. Assuming that a root of $x^3 - 9x + 1 = 0$ lies in the interval (2, 4), find that root by bisection method.

Solution. Let $f(x) = x^3 - 9x + 1$

$$f(2) = -ve \text{ and } f(4) = +ve$$

Therefore, a root lies between 2 and 4

Let $x_0 = \frac{2+4}{2} = 3$

Now $f(3) = +ve$; hence the root lies between 2 and 3

$$x_1 = \frac{2+3}{2} = 2.5$$

$$f(x_1) = f(2.5) = -ve$$

The root lies between 2.5 and 3.

$$x_2 = \frac{2.5+3}{2} = 2.75$$

$$f(2.75) = -ve$$

The root lies between 2.75 and 3

$$x_3 = \frac{1}{2}(2.75 + 3) = 2.875$$

$$f(x_3) = f(2.875) = -ve$$

Therefore, the root lies between 2.875 and 3

$$x_4 = \frac{1}{2}(2.875 + 3) = 2.9375$$

$$f(2.9375) = -ve$$

∴ The root lies between 2.9375 and 3

$$x_5 = \frac{1}{2}(2.9375 + 3) = 2.9688$$

$$f(2.9688) = +ve$$

The root lies between 2.9688 and 2.9375

$$x_6 = \frac{1}{2}(2.9375 + 2.9688) = 2.9532$$

$$f(2.9532) = +ve$$

∴ The root lies between 2.9375 and 2.9532

$$x_7 = \frac{1}{2}(2.9375 + 2.9532) = 2.9454$$

$$f(2.9454) = +ve$$

The root lies between 2.9375 and 2.9454

$$x_8 = \frac{1}{2}(2.9375 + 2.9454) = 2.9415$$

$$f(2.9415) = -ve$$

The root lies between 2.9415 and 2.9454.

$$x_9 = \frac{1}{2}(2.9415 + 2.9454) = 2.9435$$

$$f(2.9435) = +ve$$

The root lies between 2.9415 and 2.9435.

$$x_{10} = 2.9425$$

$$f(2.9425) = -ve$$

The root lies between 2.9425 and 2.9435.

$$x_{11} = 2.9430$$

$$f(2.9430) = +ve$$

$$x_{12} = 2.94275$$

$$x_{13} = 2.942875$$

Approximate root is 2.9429.

Example 3. Find the positive root of $x - \cos x = 0$ by bisection method.

Solution. Let $f(x) = x - \cos x$

$$f(0) = -ve, f(0.5) = 0.5 - \cos(0.5) = -0.37758$$

$$f(1) = 1 - \cos 1 = 0.45970$$

Hence, the root lies between 0.5 and 1.

$$x_0 = \frac{0.5 + 1}{2} = 0.75$$

$$f(0.75) = 0.75 - \cos(0.75) = 0.018311 = +ve$$

\therefore The root lies between 0.5 and 0.75.

$$x_1 = \frac{0.5 + 0.75}{2} = 0.625$$

$$f(0.625) = 0.625 - \cos(0.625) = -0.18596$$

The root lies between 0.625 and 0.750.

$$x_2 = \frac{1}{2}(0.625 + 0.750) = 0.6875$$

$$f(0.6875) = -0.085335$$

\therefore The root lies between 0.6875 and 0.75.

$$x_3 = \frac{1}{2} (0.6875 + 0.75) = 0.71875$$

$$f(0.71875) = 0.71875 - \cos(0.71875) = -0.033879$$

The root lies between 0.71875 and 0.75

$$x_4 = \frac{1}{2} (0.71875 + 0.75) = 0.73438$$

$$f(0.73438) = -0.0078664 = -ve$$

\therefore The root lies between 0.73438 and 0.75

$$x_5 = 0.742190$$

$$f(0.74219) = 0.0051999 = +ve$$

$$x_6 = \frac{1}{2} (0.73438 + 0.742190) = 0.73829$$

$$f(0.73829) = -0.0013305$$

The root lies between 0.73829 and 0.74219

$$x_7 = \frac{1}{2} (0.73829 + 0.74219) = 0.7402$$

$$f(0.7402) = 0.7402 - \cos(0.7402) = 0.0018663$$

The root lies between 0.73829 and 0.7402

$$x_8 = 0.73925$$

$$f(0.73925) = 0.00027593$$

$$x_9 = 0.7388 \quad (\text{correct to 4 places})$$

The root is 0.7388.

Example 4. Find the positive of $x^4 - x^3 - 2x^2 - 6x - 4 = 0$ by bisection method.

Solution. Let $f(x) = x^4 - x^3 - 2x^2 - 6x - 4$

$f(2)$ and $f(3)$ are opposite in sign since $f(2) = -ve$ and $f(3) = +$

$f(2.5) = -ve$; hence root lies between 2.5 and 3

$$\therefore x_1 = \frac{2.5 + 3}{2} = 2.75$$

Therefore the root lies between 2 and 3

$$x_0 = \frac{2 + 3}{2} = 2.5$$

Proceeding in the same manner, the sequence of mid-points is 2.5, 2.75, 2.63, 2.69, 2.72, 2.735, 2.728, 2.7315, 2.7298, 2.7307, 2.7311, 2.7313, 2.7314, 2.7315,...

$$f(2.7315) = -0.0232$$

The root of the equation is 2.7315 approximately.

Example 5. Using bisection method, find the negative root of $x^3 - 4x + 9 = 0$ by bisection method.

Solution. Let $f(x) = x^3 - 4x + 9$... (1)

$$f(-x) = -x^3 + 4x + 9 \quad \dots (2)$$

The negative root of $f(x) = 0$ is the positive root of $f(-x) = 0$.

\therefore We will find the positive root of $f(-x) = 0$, firstly,

i.e., $\phi(x) = x^3 - 4x - 9 = 0$

$$\phi(2) = -ve \text{ and } \phi(3) = +ve$$

\therefore The root lies between 2 and 3.

Hence $x_0 = \frac{2+3}{2} = 2.5$

$$\phi(2.5) = (2.5)^3 - 4(2.5) - 9 = -ve$$

Therefore, the root lies between 2.5 and 3.

Hence, $x_1 = \frac{1}{2}(2.5 + 3) = 2.75$

$$\phi(2.75) = +ve$$

\therefore The root lies between 2.5 and 2.75

$$x_2 = \frac{1}{2}(2.5 + 2.75) = 2.625$$

$$\phi(2.625) = (2.625)^3 - 4(2.625) - 9 = -1.4121 = -ve$$

The root lies between 2.625 and 2.75.

$$x_3 = \frac{1}{2}(2.625 + 2.75) = 2.6875$$

$$\phi(2.6875) = -ve$$

∴ The root lies between 2.6875 and 2.75.

$$x_4 = \frac{1}{2}(2.6875 + 2.75) = 2.71875$$

$$\phi(2.71875) = +ve$$

∴ The root lies between 2.6875 and 2.71875.

$$\therefore x_5 = \frac{1}{2}(2.6875 + 2.71875) = 2.703125$$

$$\phi(2.703125) = (2.703125)^3 - 4(2.703125) - 9 = -ve$$

∴ The root lies between 2.703125 and 2.71875.

$$x_6 = \frac{1}{2}(2.703125 + 2.71875) = 2.710938$$

Proceeding in the same way,

$$x_7 = 2.707031, x_8 = 2.705078, x_9 = 2.706054,$$

$$x_{10} = 2.70654, x_{11} = 2.706297, x_{12} = 2.706418,$$

$$x_{13} = 2.70648, x_{14} = 2.70651 \text{ etc.}$$

We can conclude the root to be 2.7065 for $\phi(x) = 0$

Hence the negative root of the given equation is - 2.7065.

Regula Falsi method (or the method of false position)

Consider the equation $f(x) = 0$ and let $f(a)$ and $f(b)$ be of opposite signs. Also, let $a < b$. The curve $y = f(x)$ will meet the x -axis at some point between $A(a, f(a))$ and $B(b, f(b))$. The equation of the chord joining the two points $A(a, f(a))$ and $B(b, f(b))$ is $\frac{y - f(a)}{x - a} = \frac{f(a) - f(b)}{a - b}$. The x -coordinate of the point of intersection of this chord with the x -axis gives an approximate value for the root of $f(x) = 0$. Setting $y = 0$ in the chord equation, we get

$$\frac{-f(a)}{x - a} = \frac{f(a) - f(b)}{a - b}$$

$$x[f(a) - f(b)] - af(a) + af(b) = -af(a) + bf(a)$$

$$x[f(a) - f(b)] = bf(a) - af(b)$$

$$x_1 = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

This value of x_1 gives an approximate value of the root of $f(x) = 0$. ($a < x_1 < b$)

Now $f(x_1)$ and $f(a)$ are of opposite signs or $f(x_1)$ and $f(b)$ are of opposite signs.

If $f(x_1)f(a) < 0$, then x_2 lies between x_1 and a .

$$\text{Hence } x_2 = \frac{af(x_1) - x_1f(a)}{f(x_1) - f(a)}$$

In the same way, we get x_3, x_4, \dots

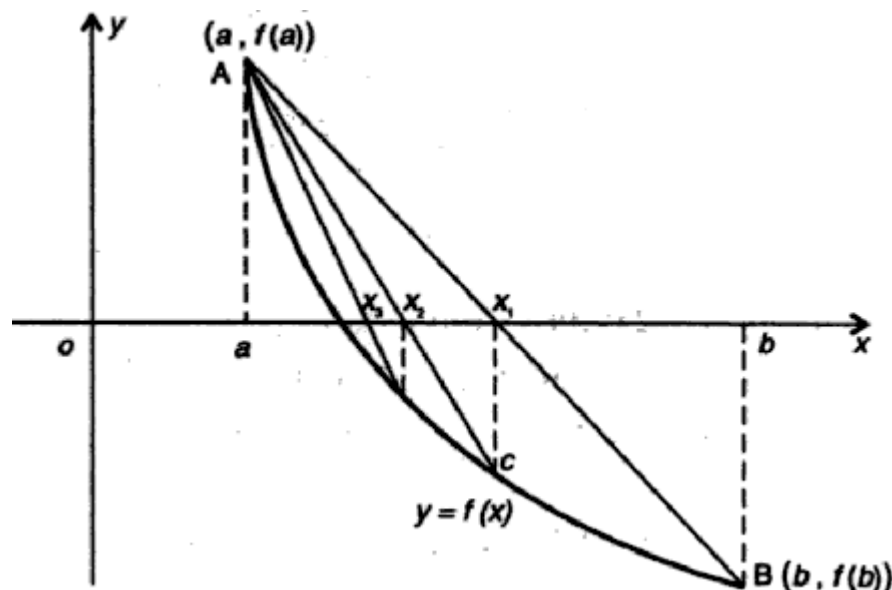
This sequence x_1, x_2, x_3, \dots will converge to the required root. In practice, we get x_i and x_{i+1} such that $|x_i - x_{i+1}| < \epsilon$, the required accuracy.

Geometrical interpretation

If $A(a, f(a))$ and $B(b, f(b))$ are two points on $y = f(x)$ such that $f(a)$ and $f(b)$ are opposite in sign, then the chord AB meets x -axis at $x = x_1$. This x_1 is the approximate root of $f(x) = 0$. Now $C(x_1, f(x_1))$ is on the curve.

If $f(a).f(x_1) < 0$, join the chord AC which cuts x -axis at $x = x_2$. Then x_2 is the second approximate root of $f(x) = 0$. This process is continued until we get the root to the desired accuracy.

The order of convergence of Regula Falsi method is 1.618. (This may be assumed.)



Example 1. Solve for a positive root of $x^3 - 4x + 1 = 0$ by Regula Falsi method.

Solution. Let $f(x) = x^3 - 4x + 1 = 0$

$$f(1) = -2 = -ve; \quad f(2) = 1 = +ve, \quad f(0) = 1 = +ve$$

\therefore a root lies between 0 and 1

Another root lies between 1 and 2

We shall find the root that lies between 0 and 1

Here $a = 0, b = 1$

$$x_1 = \frac{af(b) - bf(a)}{f(b) - f(a)} = \frac{0 \times f(1) - 1 \times f(0)}{f(1) - f(0)} = \frac{-1}{-2 - 1} = 0.333333$$

$$f(x_1) = f\left(\frac{1}{3}\right) = \frac{1}{27} - \frac{4}{3} + 1 = -0.2963$$

Now $f(0)$ and $f\left(\frac{1}{3}\right)$ are opposite in sign.

Hence the root lies between 0 and $1/3$.

$$\text{Hence } x_2 = \frac{0f\left(\frac{1}{3}\right) - \frac{1}{3}f(0)}{f\left(\frac{1}{3}\right) - f(0)}$$

$$x_2 = \frac{-\frac{1}{3}}{-1.2963} = 0.25714$$

Now $f(x_2) = f(0.25714) = -0.011558 = -ve$

\therefore The root lies between 0 and 0.25714

$$\begin{aligned}x_3 &= \frac{0 \times f(0.25714) - 0.25714 f(0)}{f(0.25714) - f(0)} \\&= \frac{-0.25714}{-1.011558} = 0.25420\end{aligned}$$

$f(x_3) = f(0.25420) = -0.0003742$

\therefore The root lies between 0 and 0.25420

$$\begin{aligned}\therefore x_4 &= \frac{0 \times f(0.25420) - 0.25420 \times f(0)}{f(0.25420) - f(0)} \\&= \frac{-0.25420}{-1.0003742} = 0.25410\end{aligned}$$

$f(x_4) = f(0.25410) = -0.000012936$

The root lies between 0 and 0.25410

$$\begin{aligned}x_5 &= \frac{0 \times f(0.25410) - 0.25410 \times f(0)}{f(0.25410) - f(0)} \\&= \frac{-0.25410}{-1.000012936} = 0.25410\end{aligned}$$

Hence the root is 0.25410.

Example 2. Find an approximate root of $x \log_{10} x - 1.2 = 0$ by False position method.

Solution. Let $f(x) = x \log_{10} x - 1.2$

$$f(1) = -1.2 = -ve; f(2) = 2 \times 0.30103 - 1.2 = -0.59794$$

$$f(3) = 3 \times 0.47712 - 1.2 = 0.23136 = +ve$$

Hence a root lies between 2 and 3.

$$\therefore x_1 = \frac{2f(3) - 3f(2)}{f(3) - f(2)} = \frac{2 \times 0.23136 - 3 \times (-0.59794)}{0.23136 + 0.59794} = 2.721014$$

$$f(x_1) = f(2.7210) = -0.017104$$

The root lies between x_1 and 3.

$$\begin{aligned}x_2 &= \frac{x_1 \times f(3) - 3 \times f(x_1)}{f(3) - f(x_1)} \\&= \frac{2.721014 \times 0.231364 - 3 \times (-0.017104)}{0.23136 + 0.017104} \\&= \frac{0.68084}{0.24846} = 2.740211\end{aligned}$$

$$\begin{aligned}f(x_2) &= f(2.7402) = 2.7402 \times \log(2.7402) - 1.2 \\&= -0.00038905\end{aligned}$$

\therefore The root lies between 2.740211 and 3

$$\begin{aligned}\therefore x_3 &= \frac{2.7402 \times f(3) - 3 \times f(2.7402)}{f(3) - f(2.7402)} \\&= \frac{2.7402 \times 0.23136 + 3 \times 0.00038905}{0.23136 + 0.00038905}\end{aligned}$$

$$= \frac{0.63514}{0.23175} = 2.740627$$

$$f(2.7406) = 0.00011998$$

\therefore The root lies between 2.740211 and 2.740627

$$\begin{aligned}x_4 &= \frac{2.7402 \times f(2.7406) - 2.7406 \times f(2.7402)}{f(2.7406) - f(2.7402)} \\&= \frac{2.7402 \times 0.00011998 + 2.7406 \times 0.00038905}{0.00011998 + 0.00038905} \\&= \frac{0.0013950}{0.00050903} = 2.7405\end{aligned}$$

Hence the root is 2.7405.

Example 3. Find the positive root of $x^3 = 2x + 5$ by False Position method.

Solution. Let $f(x) = x^3 - 2x - 5 = 0$

There is only one positive root by Descarte's rule of signs.

$$f(2) = 8 - 9 = -1 = -ve ; f(3) = 16 = +ve$$

∴ the positive root lies between 2 and 3. It is closer to 2 also.

$$\begin{aligned}x_1 &= \frac{af(b) - bf(a)}{f(b) - f(a)} = \frac{2 \times f(3) - 3 \times f(2)}{f(3) - f(2)} \\&= \frac{32 + 3}{17} = 2.058824\end{aligned}$$

$$f(x_1) = f(2.058824) = -0.390795$$

∴ The root lies between 2.058824 and 3

$$\begin{aligned} x_2 &= \frac{2.058824 \times f(3) - 3 \times f(2.058824)}{f(3) - f(2.058824)} \\ &= \frac{34.113569}{16.390795} = 2.081264 \end{aligned}$$

$$f(x_2) = f(2.081264) = -0.147200$$

∴ The root lies between 2.081264 and 3

$$x_3 = \frac{2.081264 \times 16 - 3 \times (-0.147200)}{16 + 0.147200} = 2.089639$$

$$f(x_3) = f(2.089639) = -0.054679$$

The root lies between 2.089639 and 3

$$\therefore x_4 = \frac{2.089639 \times f(3) - 3 \times f(2.089639)}{f(3) - f(2.089639)} = 2.092740$$

$$f(x_4) = f(2.09274) = -0.020198$$

∴ The root lies between 2.09274 and 3

$$x_5 = \frac{2.09274 \times 16 + 3 \times (0.020198)}{16.020198} = 2.093884$$

$$f(x_5) = f(2.093884) = -0.007447$$

The root lies between 2.093884 and 3

$$x_6 = \frac{2.093884 \times 16 + 3 \times 0.007447}{16.007447} = 2.094306$$

$$f(x_6) = f(2.094306) = -0.002740$$

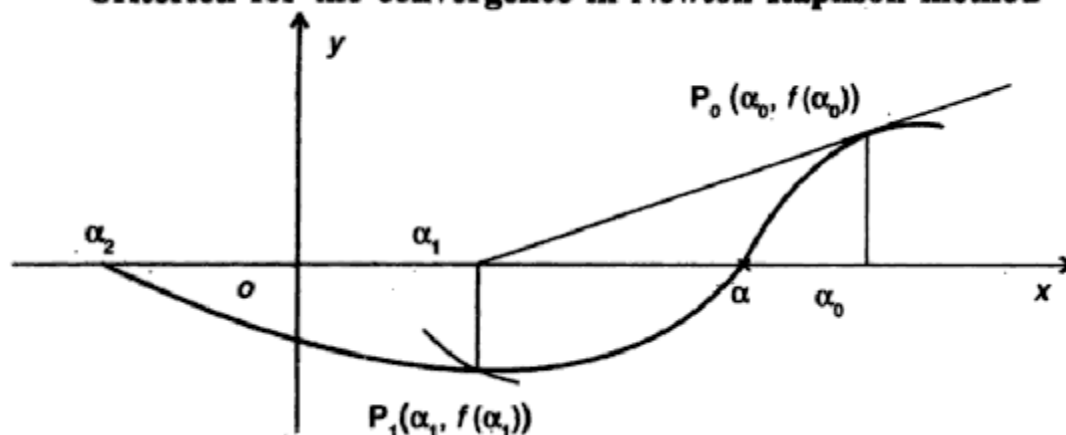
∴ The root lies between 2.094306 and 3

$$x_7 = \frac{2.094306 \times 16 - 3 \times (-0.002740)}{16.002740} = 2.094461$$

Similarly, $x_8 = 2.0945$ correct to 4 decimal places.

NEWTON –RAPHSON METHOD:

Criterion for the convergence in Newton-Raphson method



Here, in Newton's method,

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

This is really an iteration method where

$$x_{i+1} = \phi(x_i) \text{ and } \phi(x_i) = x_i - \frac{f(x_i)}{f'(x_i)}$$

Hence the equation is

$$x = \phi(x) \text{ where } \phi(x) = x - \frac{f(x)}{f'(x)}$$

The sequence x_1, x_2, x_3, \dots converges to the exact value if $|\phi'(x)| < 1$

$$\text{i.e., if } \left| 1 - \frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} \right| < 1$$

$$\text{i.e., if } \left| \frac{f(x)f''(x)}{[f'(x)]^2} \right| < 1$$

$$\text{if } |f(x)f''(x)| < [f'(x)]^2$$

Order of convergence of Newton's method

convergence is quadratic and is of order 2.

Example 1. Find the positive root of $f(x) = 2x^3 - 3x - 6 = 0$ by Newton-Raphson method correct to five decimal places.

Solution. Let $f(x) = 2x^3 - 3x - 6$; $f'(x) = 6x^2 - 3$

$$f(1) = 2 - 3 - 6 = -7 = -ve \text{ and } f(2) = 16 - 6 - 6 = 4 = +ve$$

\therefore a root lies between 1 and 2

By Descarte's rule of sign, we can prove that there is only one positive root.

Take $\alpha_0 = 2$

$$\therefore \alpha_1 = \alpha_0 - \frac{f(\alpha_0)}{f'(\alpha_0)} = \alpha_0 - \frac{2\alpha_0^3 - 3\alpha_0 - 6}{6\alpha_0^2 - 3} = \frac{4\alpha_0^3 + 6}{6\alpha_0^2 - 3}$$

$$\alpha_{i+1} = \frac{4\alpha_i^3 + 6}{6\alpha_i^2 - 3}$$

$$\alpha_1 = \frac{4(2)^2 + 6}{6(2)^2 - 3} = \frac{38}{21} = 1.809524$$

$$\alpha_2 = \frac{4(1.809524)^3 + 6}{6(1.809524)^2 - 3} = \frac{29.700256}{16.646263} = 1.784200$$

$$\alpha_3 = \frac{4(1.784200)^3 + 6}{6(1.784200)^2 - 3} = \frac{28.719072}{16.100218} = 1.783769$$

SOLUTION OF SIMULTANEOUS LINEAR ALGEBRAIC EQUATIONS:

4.2 Gauss-Elimination Method (Direct method). This is a *direct* method based on the elimination of the unknowns by combining equations such that the n equations in n unknowns are reduced to an equivalent upper triangular system which could be solved by back substitution.

Consider the n linear equations in n unknowns, viz.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\ &\dots\dots\dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n &= b_n \end{aligned} \quad \dots(1)$$

where a_{ij} and b_i are known constants and x_i 's are unknowns.

The system (1) is equivalent to

$$AX = B \quad \dots(2)$$

where

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \text{ and } B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Now our aim is to reduce the augmented matrix (A, B) to upper triangular matrix.

$$(A, B) = \left(\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array} \right) \quad \dots(3)$$

Now, multiply the first row of (3) (if $a_{11} \neq 0$) by $-\frac{a_{i1}}{a_{11}}$ and add to the i th row of (A, B) , where $i = 2, 3, \dots, n$. By this, all elements in the first column of (A, B) except a_{11} are made to zero. Now (3) is of the form

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & b_{22} & \dots & b_{2n} & c_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & b_{n2} & \dots & b_{nn} & c_n \end{array} \right) \quad \dots(4)$$

Now take the pivot b_{22} . Now, considering b_{22} as the pivot, we will make all elements below b_{22} in the second column of (4) as zeros. That

is, multiply second row of (4) by $-\frac{a_{12}}{b_{22}}$ and add to the corresponding elements of the i th row ($i = 3, 4, \dots, n$). Now all elements below b_{22} are reduced to zero. Now (4) reduces to

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} \dots a_{1n} & b_1 \\ 0 & b_{22} & b_{23} \dots b_{2n} & c_2 \\ 0 & 0 & c_{33} \dots c_{3n} & d_3 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & c_{n3} \dots c_{nn} & d_n \end{array} \right) \quad \dots(5)$$

Now taking c_{33} as the pivot, using elementary operations, we make all elements below c_{33} as zeros. Continuing the process, all elements below the leading diagonal elements of A are made to zero.

Hence, we get (A, B) after all these operations as

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & \dots & a_{1n} & b_1 \\ 0 & b_{22} & b_{23} & \dots & b_{2n} & c_2 \\ 0 & 0 & c_{33} & c_{34} \dots & c_{3n} & d_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \alpha_{nn} & K_n \end{array} \right) \quad \dots(6)$$

From, (6), the given system of linear equations is equivalent to

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1$$

$$b_{22}x_2 + b_{23}x_3 + \dots + b_{2n}x_n = c_2$$

$$c_{33}x_3 + \dots + c_{3n}x_n = d_3$$

$$\dots \dots \dots$$

$$\alpha_{nn}x_n = K_n$$

Going from the bottom of these equation, we solve for $x_n = \frac{K_n}{\alpha_{nn}}$.

Using this in the penultimate equation, we get x_{n-1} and so. By this back substitution method, we solve for

$$x_n, x_{n-1}, x_{n-2}, \dots, x_2, x_1.$$

Note. This method of making the matrix A as upper triangular matrix had been taught in lower classes while finding the rank of the matrix A .

4.2.1 Gauss-Jordan elimination method (Direct method)

This method is a modification of the above Gauss elimination method. In this method, the coefficient matrix A of the system $AX=B$ is brought to a diagonal matrix or unit matrix by making the matrix A not only upper triangular but also lower triangular by making all elements above the leading diagonal of A also as zeros. By this way, the system $AX=B$ will reduce to the form.

$$\left(\begin{array}{ccccc|c} a_{11} & 0 & 0 & 0 & 0 & b_1 \\ 0 & b_{22} & 0 & 0 & 0 & c_2 \\ \cdot & \cdot & \cdot & \cdot & \cdot & d_3 \\ 0 & 0 & 0 & 0 & \alpha_{nn} & K_n \end{array} \right) \quad \dots(7)$$

From (7)

$$x_n = \frac{K_n}{\alpha_{nn}}, \dots, x_2 = \frac{c_2}{b_{22}}, x_1 = \frac{b_1}{a_{11}}$$

Note. By this method, the values of x_1, x_2, \dots, x_n are got immediately without using the process of back substitution.

Example 1. Solve the system of equations by (i) Gauss elimination method (ii) Gauss-Jordan method.

$$x + 2y + z = 3, \quad 2x + 3y + 3z = 10; \quad 3x - y + 2z = 13. \quad [MKU 1981]$$

Solution. (By Gauss method)

The given system is equivalent to

$$\begin{array}{ccc} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 3 & 3 \\ 3 & -1 & 2 \end{pmatrix} & \begin{pmatrix} x \\ y \\ z \end{pmatrix} & = \begin{pmatrix} 3 \\ 10 \\ 13 \end{pmatrix} \\ A & X & = B \end{array}$$
$$(A, B) = \left(\begin{array}{ccc|c} 1 & 2 & 1 & 3 \\ 2 & 3 & 3 & 10 \\ 3 & -1 & 2 & 13 \end{array} \right) \quad \dots(1)$$

Now, we will make the matrix A upper triangular.

$$(A, B) = \left(\begin{array}{ccc|c} 1 & 2 & 1 & 3 \\ 2 & 3 & 3 & 10 \\ 3 & -1 & 2 & 13 \end{array} \right)$$

$$\sim \left(\begin{array}{ccc|c} 1 & 2 & 1 & 3 \\ 0 & -1 & 1 & 4 \\ 0 & -7 & -1 & 4 \end{array} \right) \quad \begin{array}{l} R_2 + (-2)R_1 \text{ i.e., } R_{21}(-2) \\ R_3 + (-3)R_1 \text{ i.e., } R_{31}(-3) \end{array}$$

Now take $b_{22} = -1$ as the pivot and make b_{32} as zero.

$$(A, B) \sim \left(\begin{array}{ccc|c} 1 & 2 & 1 & 3 \\ 0 & -1 & 1 & 4 \\ 0 & 0 & -8 & -24 \end{array} \right) R_{32}(-7) \quad \dots(2)$$

From this, we get

$$\begin{aligned} x + 2y + z &= 3 \\ -y + z &= 4 \\ -8z &= -24 \end{aligned}$$

$\therefore z = 3, y = -1, x = 2$ by back substitution.

Solution. (Gauss-Jordan method)

In stage 2, make the element, in the position (1, 2), also zero.

$$(A, B) \sim \left(\begin{array}{ccc|c} 1 & 2 & 1 & 3 \\ 0 & -1 & 1 & 4 \\ 0 & 0 & -8 & -24 \end{array} \right)$$

$$\sim \left(\begin{array}{ccc|c} 1 & 0 & 3 & 11 \\ 0 & -1 & 1 & 4 \\ 0 & 0 & -8 & -24 \end{array} \right) R_{12}(2)$$

$$\sim \left(\begin{array}{ccc|c} 1 & 0 & 3 & 11 \\ 0 & -1 & 1 & 4 \\ 0 & 0 & -1 & -3 \end{array} \right) R_3\left(\frac{1}{8}\right)$$

$$\sim \left(\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & -3 \end{array} \right) R_{13}(3), R_{23}(1)$$

$$x = 2, -y = 1, -z = -3$$

$$x = 2, y = -1, z = 3$$

Example 2. Solve the system by Gauss-Elimination method

$$2x + 3y - z = 5; \quad 4x + 4y - 3z = 3 \text{ and } 2x - 3y + 2z = 2. \quad [\text{MKU 1980}]$$

Solution. The system is equivalent to

$$\begin{pmatrix} 2 & 3 & -1 \\ 4 & 4 & -3 \\ 2 & -3 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \\ 2 \end{pmatrix}$$

$$A \quad X = B$$

$$\therefore (A, B) = \left(\begin{array}{ccc|c} 2 & 3 & -1 & 5 \\ 4 & 4 & -3 & 3 \\ 2 & -3 & 2 & 2 \end{array} \right)$$

Step 1. Taking $a_{11} = 2$ as the pivot, reduce all elements below that to zero.

$$(A, B) \sim \left(\begin{array}{ccc|c} 2 & 3 & -1 & 5 \\ 0 & -2 & -1 & -7 \\ 0 & -6 & 3 & -3 \end{array} \right) \quad R_{21}(-2), R_{31}(-1)$$

Step 2. Taking the element -2 in the position $(2, 2)$ as pivot, reduce all elements below that to zero.

$$(A, B) \sim \left(\begin{array}{ccc|c} 2 & 3 & -1 & 5 \\ 0 & -2 & -1 & -7 \\ 0 & 0 & 6 & 18 \end{array} \right) \quad R_{32}(-3)$$

$$\text{Hence} \quad 2x + 3y - z = 5$$

$$-2y - z = -7$$

$$6z = 18$$

$$\therefore z = 3, y = 2, x = 1. \text{ by back substitution.}$$

Example 3. Solve the following system by Gauss-Jordan method:

$$5x_1 + x_2 + x_3 + x_4 = 4; \quad x_1 + 7x_2 + x_3 + x_4 = 12$$

$$x_1 + x_2 + 6x_3 + x_4 = -5; \quad x_1 + x_2 + x_3 + 4x_4 = -6$$

Solution. Interchange the first and the last equation, so that the coefficient of x_1 in the first equation is 1. Then we have

$$(A, B) = \left(\begin{array}{cccc|c} 1 & 1 & 1 & 4 & -6 \\ 1 & 7 & 1 & 1 & 12 \\ 1 & 1 & 6 & 1 & -5 \\ 5 & 1 & 1 & 1 & 4 \end{array} \right)$$

$$\sim \left(\begin{array}{cccc|c} 1 & 1 & 1 & 4 & -6 \\ 0 & 6 & 0 & -3 & 18 \\ 0 & 0 & 5 & -3 & 1 \\ 0 & -4 & -4 & -19 & 34 \end{array} \right) \quad R_{21}(-1), R_{31}(-1), R_{41}(-5)$$

$$\therefore l_{32} = \frac{1-15}{-9} = \frac{14}{9}; \quad u_{33} = 4-3-\frac{14}{9} = -\frac{5}{9}$$

$$LUX = B \text{ implies } LY = B \text{ where } UX = Y$$

$$LY = B \text{ gives,}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & \frac{14}{9} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 14 \\ 13 \\ 17 \end{pmatrix}$$

$$y_1 = 14, \quad 2y_1 + y_2 = 13, \quad 3y_1 + \frac{14}{9}y_2 + y_3 = 17$$

$$\therefore y_1 = 14, \quad y_2 = -15, \quad y_3 = -\frac{5}{3}$$

$$UX = Y \text{ implies,}$$

$$\begin{pmatrix} 1 & 5 & 1 \\ 0 & -9 & 1 \\ 0 & 0 & -\frac{5}{9} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 14 \\ -15 \\ -\frac{5}{3} \end{pmatrix}$$

$$x + 5y + z = 14$$

$$-9y + z = -15$$

$$-\frac{5}{9}z = -\frac{5}{3}$$

$$z = 3, \quad y = 2, \quad x = 1.$$

Example 3. Solve the following system by triangularization method:

$$x + y + z = 1, \quad 4x + 3y - z = 6, \quad 3x + 5y + 3z = 4.$$

Solution. Here $A = \begin{pmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{pmatrix}$, $X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$, $B = \begin{pmatrix} 1 \\ 6 \\ 4 \end{pmatrix}$

$$\therefore LU = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{pmatrix}$$

$$\therefore u_{11} = u_{12} = u_{13} = 1.$$

$$l_{21} u_{11} = 4, \quad l_{21} u_{12} + u_{22} = 3, \quad l_{21} u_{13} + u_{23} = -1$$

$$\therefore l_{21} = 4, \quad u_{22} = -1, \quad u_{23} = -5$$

$$l_{31} = 3, \quad l_{31} + l_{32} u_{22} = 5, \quad l_{31} + l_{32} u_{23} + u_{33} = 3$$

$$l_{32} = -2, \quad u_{33} = -10$$

Now, $LUX = B$ implies $LY = B$ where $UX = Y$

$$\begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 6 \\ 4 \end{pmatrix}$$

$$y_1 = 1; \quad 4y_1 + y_2 = 6, \quad 3y_1 - 2y_2 + y_3 = 4$$

$$\therefore y_1 = 1, \quad y_2 = 2, \quad y_3 = 5$$

$UX = Y$ gives,

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix}$$

$$x + y + z = 1$$

$$-y - 5z = 2$$

$$-10z = 5$$

Hence, $z = -\frac{1}{2}$, $y = \frac{1}{2}$, $x = 1$.

Gauss Elimination Method

This is direct method based on number of unknowns, by eliminating the same by combining the equations to a triangular form. To illustrate the method consider the following system equations

Steps to solve the system of three equation with three unknowns :

Let us consider the system of equation

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

1. To eliminate x_1 from the second equation, multiply the first row of the equation matrix by $-a_{21}/a_{11}$ and add it to second equation. Similarly eliminate x_1 from the third equation and subsequently all other equations. We get new equation of the form

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$+b_{22}x_2 + b_{23}x_3 = c_2$$

$$+b_{32}x_2 + b_{33}x_3 = c_3$$

$$\text{Where } b_{22} = a_{22} - (a_{21}/a_{11}) \times a_{12}$$

$$b_{23} = a_{23} - (a_{21}/a_{11}) \times a_{13}$$

$$c_2 = b_2 - (a_{21}/a_{11}) \times b_1$$

$$\begin{aligned}
 b_{32} &= a_{32} - (a_{31} / a_{11}) \times a_{12} \\
 b_{33} &= a_{33} - (a_{31} / a_{11}) \times a_{13} \\
 c_3 &= b_3 - (a_{31} / a_{11}) \times b_1
 \end{aligned}$$

2. To eliminate x_2 from the third equation, multiply the second row of the equation matrix by $-b_{32}/b_{22}$ and add it to third equation. Similarly eliminate x_2 from the third equation and subsequently all other equations.

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\
 \phantom{a_{11}x_1} + b_{22}x_2 + b_{23}x_3 &= c_2 \\
 \phantom{a_{11}x_1} \phantom{+ b_{22}x_2} + b_{33}x_3 &= d_3
 \end{aligned}$$

$$\text{where } c_{33} = b_{33} - (b_{32} / b_{22}) \times b_{23}$$

$$d_3 = c_3 - (b_{32} / b_{22}) \times c_2$$

3. From the above reduced system of equation substitute the values x_3 , x_2 and x_1 by backward substitution we get the solution of the given equations.

Illustration 1 :

Solve the system of equation by Gauss elimination method

$$\begin{aligned}
 x + 2y + z &= 3 \\
 2x + 3y + 3z &= 10 \\
 3x - y + 2z &= 13
 \end{aligned}$$

Solution :

The given system of equation is equivalent to

$$\begin{array}{cccc}
 - & 1 & & 2 & & 1 & & 3 & - \\
 - & 2 & & 3 & & 3 & & 10 & - \\
 - & 3 & & -1 & & 2 & & 13 & -
 \end{array}$$

Now, we have to make the above matrix as upper triangular

By using the following modifications

$$R_2' = R_2 + (-2) R_1 ; \quad R_3' = R_3 + (-3) R_1$$

$$\begin{array}{cccc}
 -1 & 2 & 1 & 3 \\
 -0 & -1 & 1 & 4 \\
 -0 & -7 & -8 & 4
 \end{array}$$

Now we have to take $b_{22} = -1$ as the key element and reduce b_{32} as 0

By using the following modifications

$$R_3' = R_3 + (+7) R_2$$

$$\begin{array}{cccc}
 -1 & 2 & 1 & 3 \\
 -0 & -1 & 1 & 4 \\
 -0 & 0 & -8 & -24
 \end{array}$$

From the above matrix

$$\begin{array}{l}
 x + 2y + z = 3 \\
 -y + z = 4 \\
 -8z = -24
 \end{array}$$

Therefore $z = 3, y = -1, x = 2$ by back substitution.

Solve the system of equation by Gauss elimination method

$$2x + y + 4z = 12$$

$$8x - 3y + 2z = 20$$

$$4x + 11y - z = 33$$

Solution :

The given system of equation is equivalent to

$$\begin{array}{cccc}
 -2 & 1 & 4 & 12 \\
 -8 & -3 & 2 & 20 \\
 -4 & 11 & -1 & 33
 \end{array}$$

Now, we have to eliminate x from the second and third equation

By using the following modifications

$$R_2' = R_2 + (-4) R_1 ; R_3' = R_3 + (-2) R_1$$

$$\begin{array}{cccc}
 -2 & 1 & 4 & 12 \\
 -0 & -7 & -14 & -28 \\
 -0 & 9 & -9 & 9
 \end{array}$$

Second step we eliminate y from the third equation. Taking ($b_{23} = 9/7$) as the key element multiply the second equation by key element and add it to the third equation

By using the following modifications

$$R_3' = R_3 + (9/7) R_2$$

- 1	2	1	3	-
- 0	-7	-14	-28	-
- 0	0	-27	-27	-

From the above matrix

$$\begin{aligned} 2x + y + 4z &= 12 \\ -7y - 14z &= -28 \\ -27z &= -27 \end{aligned}$$

By back substitution, we get the solution of the equation

$$z = 1, y = 2, x = 3$$

Therefore $z = 3, y = -1, x = 2$ by back substitution.

Check Your Progress

Solve the system of equation by Gauss elimination method

$$\begin{aligned} 20x + y + 4z &= 25 \\ 8x + 13y + 2z &= 23 \\ 4x - 11y + 21z &= 14 \end{aligned}$$

(Ans: $x = y = z = 1$)

Gauss Jordan Method

This method is a slightly modification of the above Gauss Elimination method . Here elimination is performed not only in the lower triangular but also upper triangular . This leads to unit matrix and hence solution is obtained . This is

Jordan's modification of the Gauss elimination and hence the name is Gauss-Jordan Method.

The above system of equation is written in the matrix form as

$$AX = B$$

Now our aim is to reduce the given matrix (A,) to unit matrix.

The system of equation can be solved simply thus :

$$\begin{array}{ccccccc} a_{11} & 0 & 0 & \dots & 0 & b_1 & - \\ -0 & b_{22} & 0 & \dots & 0 & c_2 & - \\ -0 & 0 & c_{33} & \dots & 0 & d_3 & - \\ \dots & \dots & \dots & \dots & \dots & \dots & - \\ -0 & 0 & 0 & \dots & n_{nn} & m_n & - \end{array}$$

The above system of linear equations is equivalent to

$$\begin{array}{ccccccc} a_{11}x_1 + 0 & + & \dots & +0 & = & b_1 \\ b_{22}x_2 + & \dots & +0 & = & c_2 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ n_{nn}x_n & = & m_n \end{array}$$

he above the equation we get solution directly..

Solve the system of equation by Gauss-Jordon method

$$10x + y + z = 12$$

$$2x + 10y + z = 10$$

$$x + y + 5z = 13$$

Solution :

The given system of equation is rearranged for computation convenience,
Interchange the first and last equation, since coefficient of the x in the last equation is
unity (1) :

$$\begin{array}{cccc} -1 & 1 & 5 & 7 \\ -2 & 10 & 1 & 13 \\ -10 & 1 & 1 & 12 \end{array}$$

Now, we have to make the above matrix as upper triangular

By using the following modifications

$$R_2' = R_2 + (-2) R_1 ; \quad R_3' = R_3 + (-10) R_1$$

$$\begin{array}{cccc} -1 & 1 & 5 & 7 \\ -0 & 8 & -9 & -1 \\ -0 & -9 & -49 & -58 \end{array}$$

Now we have to take $b_{22} = 1/8$ as the key element and reduce b_{32} as 0
By using the following modifications

$$R_2' = R_2 / 8 ; \quad R_3' = R_3 + (+9/8) R_2$$

$$\begin{array}{cccc} -1 & 1 & 5 & 7 \\ -0 & 1 & -9/8 & -1/8 \\ -0 & 0 & -473/8 & -473/8 \end{array}$$

Now we have to make $b_{33} = 1$ as the key element and reduce b_{32} as 0
By using the following modifications

$$R_3' = R_3 \times -8/473 ; \quad R_1' = R_1 + (-1) R_2$$

$$\begin{array}{cccc} -1 & 0 & 49/8 & 57/8 \\ -0 & 1 & -9/8 & -1/8 \\ -0 & 0 & 1 & 1 \end{array}$$

Now we have to make $b_{23} = 0$ and $b_{13} = 0$.

By using the following modifications

$$R_2' = R_2 + (-9/8) R_3 ; \quad R_1' = R_1 + (-49/8) R_3$$

$$\begin{array}{cccc} -1 & 0 & 0 & 1 \\ -0 & 1 & 0 & 1 \\ -0 & 0 & 1 & 1 \end{array}$$

Therefore $x = 1, y = 1, z = 1$.

Solve the system of equation by Gauss-Jordon method

$$2x + y + 4z = 12$$

$$8x - 3y + 2z = 20$$

$$4x + 11y - z = 33$$

Solution :

The given system of equation is equivalent to

$$\begin{array}{cccc} - & 2 & 1 & 4 & 12 & - \\ - & 8 & -3 & 2 & 20 & - \\ - & 4 & 11 & -1 & 33 & - \end{array}$$

Now, we have to eliminate x from the second and third equation

By using the following modifications

$$R_2' = R_2 + (-4) R_1 ; \quad R_3' = R_3 + (-2) R_1$$

$$\begin{array}{cccc} - & 2 & 1 & 4 & 12 & - \\ - & 0 & -7 & -14 & -28 & - \\ - & 0 & 9 & -9 & 9 & - \end{array}$$

Second step we eliminate y from the third equation. Taking $(b_{23} = 9/7)$ as the key element multiply the second equation by key element and add it to the third equation

By using the following modifications

$$R_3' = R_3 + (9/7) R_2$$

$$\begin{array}{cccc} -2 & 1 & 4 & 12 \\ -0 & -7 & -14 & -28 \\ -0 & 0 & -27 & -27 \end{array}$$

At this stage, we eliminate y from the first equation. Z from the first and second equation. By using following modifications ;

$$R_1' = R_1 / 2 ; R_2' = R_2 / -7 ; R_3' = R_3 / (-27) R_1$$

$$\begin{array}{cccc} -1 & 1/2 & 2 & 6 \\ -0 & 1 & 2 & 4 \\ -0 & 0 & 1 & 1 \end{array}$$

$$R_2' = R_2 + (-2) R_3 ; R_1' = R_1 + (-1/2) R_2 ;$$

$$\begin{array}{cccc} -1 & 0 & 1 & 4 \\ -0 & 1 & 0 & 2 \\ -0 & 0 & 1 & 1 \end{array}$$

$$R_1' = R_1 + (-1) R_1 ;$$

$$\begin{array}{cccc} -1 & 0 & 0 & 3 \\ -0 & 1 & 0 & 2 \\ -0 & 0 & 1 & 1 \end{array}$$

Therefore $x = 3, y = 2, z = 1$.

Check Your Progress

Solve the system of equation by Gauss-Jordon method

$$10x + y + z = 13$$

$$2x + 10y + z = 14$$

$$x + y + 15z = 32$$

(Ans: $x=y=1, z=2$)

2.4 Lesson End Activities

1. Solve the system of equation by Gauss Elimination method

$$3.15x - 1.96y + 3.85z = 12.95$$

$$\begin{aligned}2.13x - 5.12y - 2.892z &= -8.61 \\5.92x + 3.051y + 2.15z &= 6.88\end{aligned}$$

2. Solve the system of equation by Gauss elimination method

$$3x + 4y + 6z = 18$$

$$2x - y + 8z = 13$$

$$5x - 2y + 7z = 20$$

3. Solve the system of equation by Gauss-Jordan method

$$2x + y + 4z = 9$$

$$8x - 3y + z = 12$$

$$4x + 11y - z = 18$$

4. Solve the system of equation by Gauss-Jordan method

$$2x - y + 4z = 5$$

$$8x - 3y + z = 6$$

$$x + 11y - z = 11$$

Answer For Lesson End Activities

1. (Ans : $x = 1.7089$, $y = -1.8005$, $z = 1.0488$)

2. (Ans: $x = 3$, $y = 1$, $z = 1$)

3. (Ans: $x = 2$, $y = 1$, $z = 1$)

4. (Ans: $x = 1$, $y = 1$, $z = 1$)

Gauss-Jacobi Method

Let us consider this method in the case of three equations in three unknowns.

Consider the 3 linear equations in 3 unknowns,

$$a_1x + b_1y + c_1z = d_1$$

$$a_2x + b_2y + c_2z = d_2$$

$$a_3x + b_3y + c_3z = d_3$$

This method is applied only when diagonal elements are exceeding all other elements in the respective equations i.e.,

$$|a_1| > |b_1| + |c_1| = d_1$$

$$|a_2| > |b_2| + |c_2| = d_2$$

$$|a_3| > |b_3| + |c_3| = d_3$$

Let the above condition is true we apply this method or we have to rearrange the equations in the above form to fulfill the above condition.

We start with initial values of x, y and z as zero. Solve x, y, z in terms of other variables

$$x^{(r+1)} = \frac{1}{a_1} (d_1 - b_1 y^{(r)} - c_1 z^{(r)})$$

$$y^{(r+1)} = \frac{1}{b_2} (d_2 - a_2 x^{(r)} - c_2 z^{(r)})$$

$$z^{(r+1)} = \frac{1}{c_3} (d_3 - a_3 x^{(r)} - b_3 y^{(r)})$$

The above iteration is continued until any two successive values are equal.

1 . Solve the system of equation by Gauss-Jacobi method

$$\begin{aligned} 27x + 6y - z &= 85 \\ 6x + 15y + 2z &= 72 \\ x + 6y + 54z &= 110 \end{aligned}$$

Solution:

To apply this method , first we have to check the diagonal elements are dominant.

i.e., $27 > 6 + 1$; $15 > 6 + 2$; $54 > 1 + 1$. So iteration method can be applied

$$\begin{aligned} x &= 1/27 (85 - 6y - z) \\ y &= 1/15 (72 - 6x - 2z) \\ z &= 1/54 (110 - x - y) \end{aligned}$$

First iteration : From the above equations, we start with $x = y = z = 0$

$$\begin{aligned} x^{(1)} &= 85/27 = 3.14815 && \dots\dots\dots(1) \\ y^{(1)} &= 72/15 = 4.8 && \dots\dots\dots(2) \\ z^{(1)} &= 110/54 = 2.03704 && \dots\dots\dots(3) \end{aligned}$$

Second iteration : Consider the new values of $y^{(1)} = 4.8$ and $z^{(1)} = 2.03704$ in the first equation

$$x^{(2)} = 1/27(85 | 6 \times 4.8 + 2.03704) = 2.15693$$

$$y^{(2)} = 1/15 (72 - 6 \times 3.14815 - 2 \times 2.03704) = 3.26913$$

$$z^{(2)} = 1/54 (110 | 3.14815 | 4.8) = -0.515$$

Fourth iteration : Consider the new values of $x^{(2)} = 2.15693$, $y^{(2)} = 3.26913$ and $z^{(2)} = -0.515$ in the first equation

$$x^{(3)} = 1/27(85 | 6 \times 3.26913 + -0.515) = 2.49167$$

$$y^{(3)} = 1/15 (72 - 6 \times 2.15693 - 2 \times 2.15693) = 3.68525$$

$$z^{(3)} = 1/54 (110 | 2.15693 | 3.26913) = 1.93655$$

Thus, we continue the iteration and result is noted below

Iteration No.	x	y	z
4	2.40093	3.54513	1.92265
5	2.43155	3.58327	1.92692
6	2.42323	3.57046	1.92565
7	2.42603	3.57395	1.92604
8	2.42527	3.57278	1.92593
9	2.42552	3.57310	1.92596
10	2.42546	3.57300	1.92595

From the above table 9th and 10th iterations are equal by considering the four decimal places. Hence the solution of the equation is

$$x = 2.4255 \qquad y = 3.5730 \qquad z = 1.9260.$$

Illustration 2 . Solve the system of equation by Gauss-Jacobi method

$$10x - 5y - 2z = 3$$

$$4x - 10y + 3z = -3$$

$$x + 6y + 10z = 3$$

Solution:

To apply this method , first we have to check the diagonal elements are dominant.

i.e., $10 > 5 + 2$; $10 > 4 + 3$; $10 > 1 + 6$. So iteration method can be applied

$$x = 1/10 (3 + 5y + 2z)$$

$$y = 1/10 (3 + 4x + 3z)$$

$$z = 1/10 (-3 - x - 6y)$$

First iteration : From the above equations, we start with $x = y = z = 0$

$$x^{(1)} = 3/10 = 0.3 \quad \dots\dots\dots(1)$$

$$y^{(1)} = 3/10 = 0.3 \quad \dots\dots\dots(2)$$

$$z^{(1)} = -3/10 = -0.3 \quad \dots\dots\dots(3)$$

Second iteration : Consider the new values of $y^{(1)} = 0.3$ and $z^{(1)} = -0.3$ in the first equation

$$x^{(2)} = 1/10 (3 + 5 \times 0.3 + (-0.3)) = 0.39$$

$$y^{(2)} = 1/10 (3 + 4 \times 0.3 + 3 \times (-0.3)) = 0.33$$

$$z^{(2)} = 1/10 [-3 - (0.3) - 6(0.3)] = -0.51$$

Third iteration : Consider the new values of $x^{(2)} = 0.39$, $y^{(2)} = 0.33$ and $z^{(2)} = -0.51$ in the first equation

$$x^{(3)} = 1/10 [3 + 5 \times 0.33 + (-0.51)] = 0.363$$

$$y^{(3)} = 1/10 (3 + 4 \times 0.39 + 3 \times (-0.51)) = 0.303$$

$$z^{(3)} = 1/10 [-3 + 0.39 + 6 \times (0.33)] = -0.537$$

Thus, we continue the iteration and result is noted below

Iteration No.	X	y	z
4	0.3441	0.2841	-0.5181
5	0.33843	0.2822	-0.50487
6	0.340126	0.283911	0.503163
7	0.3413229	0.2851015	-0.5043592
8	0.34167891	0.2852214	-0.50519319
9	0.341572062	0.285113607	-0.505300731

From the above table 8th and 9th iterations are equal by considering the 3 decimal places. Hence the solution of the equation is

$$x = 0.342, y = 0.285, z = -0.505.$$

Solve the following system of equations by using Gauss-Jacobi Method

1. $8x - 3y + 2z = 20$; $4x + 11y - z = 33$; $6x + 3y + 12z = 35$

2. $28x + 4y - z = 32$; $x + 3y + 10z = 24$; $2x + 3y + 10z = 24$

3. $5x - 2y + z = -4$; $x + 6y - 2z = -1$; $3x + y + 5z = 13$

4. $8x + y + z = 8$; $2x + 4y + z = 4$; $x + 3y + 3z = 5$

Model Answer For Lesson End Activities

1. (Ans: 3.017, 1.986, 0.912)

2. (Ans: 0.994, 1.507, 1.849)

3. (Ans: -1.0, .999, 3)

4. (Ans: 0.83, .32, 1.07)

Gauss-Seidel Method

This method is only an enhancement of Gauss-Jacobi Method.

Consider the 3 linear equations in 3 unknowns,

$$a_1x + b_1y + c_1z = d_1$$

$$a_2x + b_2y + c_2z = d_2$$

$$a_3x + b_3y + c_3z = d_3$$

This method is applied only when diagonal elements are exceeding all other elements in the respective equations i.e.,

$$|a_1| > |b_1| + |c_1| = d_1$$

$$|a_2| > |b_2| + |c_2| = d_2$$

$$|a_3| > |b_3| + |c_3| = d_3$$

We start with initial values of x,y and z as zero.

$$x^{(r+1)} = \frac{1}{a_1} (d_1 - b_1 y^{(r)} - c_1 z^{(r)})$$

$$y^{(r+1)} = \frac{1}{b_2} (d_2 - a_2 x^{(r+1)} - c_2 z^{(r)})$$

$$z^{(r+1)} = \frac{1}{c_3} (d_3 - a_3 x^{(r+1)} - b_3 y^{(r+1)})$$

- Note :*
1. For all systems of equation, this method will not work
 2. Iteration method is self correcting method. Any error made in computation is corrected automatically in subsequent iterations
 3. Iteration is stopped when any two successive iteration values are equal

Illustration : 1 . Solve the system of equation by Gauss-Seidel method

$$\begin{aligned} 10x - 5y - 2z &= 3 \\ 4x - 10y + 3z &= -3 \\ x + 6y + 10z &= 3 \end{aligned}$$

Solution:

To apply this method , first we have to check the diagonal elements are dominant.

ie., $10 > 5 + 2$; $10 > 4 + 3$; $10 > 1 + 6$. So iteration method can be applied

$$\begin{aligned} x &= 1/10 (3 + 5y + 2z) \\ y &= 1/10 (3 + 4x + 3z) \\ z &= 1/10 (-3 - x - 6y) \end{aligned}$$

First iteration :

From the above equations, we start with $x = y = z = 0$

$$x^{(1)} = 3/10 = 0.3 \quad \dots\dots\dots(1)$$

New value of x is used for further calculation ie., $x = 0.3$

$$y^{(1)} = 1/10 (3 + 4x \cdot 0.3 + 3(0)) = 0.42 \quad \dots\dots\dots(2)$$

New values of x and y is used for further calculation ie., $x = 0.3$ and $y = 0.42$

$$z^{(1)} = 1/10 (-3 - 0.3 - 6(0.42)) = -0.582 \quad \dots\dots\dots(3)$$

Second iteration :

Consider the new values of $y^{(1)} = 0.42$ and $z^{(1)} = -0.582$ in the first equation

$$x^{(2)} = 1/10 (3 + 5 \times 0.42 + (-0.582)) = 0.3936$$

$$y^{(2)} = 1/10 (3 + 4 \times 0.3936 + 3 \times (-0.582)) = 0.28284$$

$$z^{(2)} = 1/10 [-3 + (0.3936) + 6(0.28284)] = -0.509064$$

Third iteration : Consider the new values of $x^{(2)} = 0.3936$, $y^{(2)} = 0.28284$ and $z^{(2)} = -0.509064$ in the first equation

$$x^{(3)} = 1/10 [3 + 5 \times 0.28284 + (-0.509064)] = 0.3396072$$

$$y^{(3)} = 1/10 (3 + 4 \times 0.3396072 + 3 \times (-0.509064)) = 0.28312368$$

$$z^{(3)} = 1/10 [-3 + 0.3396072 + 6 \times (0.28312368)] = -0.503834928$$

Thus, we continue the iteration and result is noted below

Iteration No.	X	Y	Z
4	0.34079485	0.28516746	-0.50517996
5	0.3415547	0.28506792	-0.505196229
6	0.3414947	0.2850390	-0.5051728
7	0.3414849	0.28504212	-0.5051737

The values correct to 3 decimal places are

$$x = 0.342, \quad y = 0.285, \quad z = -0.505$$

1. Solve the system of equation by Gauss-Seidel method

$$28x + 4y - z = 32$$

$$4x + 3y + 10z = 24$$

$$2x + 17y + 4z = 35$$

To apply this method, first we have to rewrite the equation in such way that to fulfill diagonal elements are dominant.

$$28x + 4y - z = 32$$

$$2x + 17y + 4z = 35$$

$$4x + 3y + 10z = 24$$

ie., $28 > 4 + 1$; $17 > 2 + 4$; $10 > 4 + 3$. So iteration method can be applied

$$x = 1/28 (32 - 4y + z)$$

$$y = 1/17 (35 - 2x - 4z)$$

$$z = 1/10 (24 - x - 3y)$$

First iteration :

From the above equations, we start with $y = z = 0$, we get

$$x^{(1)} = 32/28 = 1.1429$$

New value of x is used for further calculation ie., $x = 1.1429$

$$y^{(1)} = 1/17 (35 + 1.1429 + 3(0)) = 1.9244$$

New values of x and y is used for further calculation ie., $x = 1.1429$
and $y = 1.9244$

$$z^{(1)} = 1/10 [24 - 1.1429 - 3(1.9244)] = 1.8084$$

Second iteration :

Consider the new values of $y^{(1)} = 1.9244$ and $z^{(1)} = 1.8084$

$$x^{(2)} = 1/28 [32 - 4(1.9244) + (1.8084)] = 0.9325$$

$$y^{(2)} = 1/17 [35 - 2(0.9325) - 4(1.8084)] = 1.5236$$

$$z^{(2)} = 1/10 [24 + (0.9325) + 3(1.5236)] = 1.8497$$

Third iteration :

Consider the new values of $x^{(2)} = 0.9325$, $y^{(2)} = 1.5236$ and $z^{(2)} = 1.8497$

$$x^{(3)} = 1/28 [32 - 4(1.5236) + (1.8497)] = 0.9913$$

$$y^{(3)} = 1/17 [35 - 2(0.9913) - 4(1.8497)] = 1.5070$$

$$z^{(3)} = 1/10 [24 + (0.9913) + 3(1.5070)] = 1.8488$$

Thus, we continue the iteration and result is noted below

Iteration No.	X	y	Z
4	0.9936	1.5069	1.8486
5	0.9936	1.5069	1.8486

Therefore $x = 0.9936$, $y = 1.5069$, $z = 1.8486$

1. $8x - 6y + z = 13.67$; $3x + y - 2z = 17.59$; $2x - 6y + 9z = 29.29$
2. $30x - 2y + 3z = 75$; $2x + 2y + 18z = 30$; $x + 17y - 2z = 48$
3. $y - x + 10z = 35.61$; $x + z + 10y = 20.08$; $y - z + 10x = 11.19$
4. $10x - 2y + z = 12$; $x + 9y - z = 10$; $2x - y + 11z = 20$
5. $8x - y + z = 18$; $2x + 5y - 2z = 3$; $x + y - 3z = -16$
6. $2x + y + z = 4$; $x + 2y - z = 4$; $x + y + 2z = 4$

1. 0.83, 0.32, 1.07
2. 2.5796, 2.7976, 1.0693
3. 1.321, 1.522, 3.541
4. 1.2624, 1.1591, 1.694
5. 2, 0.9998, 2.9999
6. 1, 1, 1

3.2 LU decomposition

The process of Gaussian Elimination also results in the factoring of the matrix A to

$$A = LU,$$

where L is a lower triangular matrix and U is an upper triangular matrix. Using the same matrix A as in the last section, we show how this factorization is realized. We have

$$\begin{pmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{pmatrix} \rightarrow \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & -2 & 3 \end{pmatrix} = M_1 A,$$

where

$$M_1 A = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{pmatrix} = \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & -2 & 3 \end{pmatrix}.$$

Note that the matrix M_1 performs row elimination on the first column. Two times the first row is added to the second row and one times the first row is added to the third row. The entries of the column of M_1 come from $2 = -(6/-3)$ and $1 = -(3/-3)$ as required for row elimination. The number -3 is called the pivot.

The next step is

$$\begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & -2 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{pmatrix} = M_2(M_1 A),$$

where

$$M_2(M_1 A) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & -2 & 3 \end{pmatrix} = \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{pmatrix}.$$

Here, M_2 multiplies the second row by $-1 = -(-2/-2)$ and adds it to the third row. The pivot is -2 .

We now have

$$M_2 M_1 A = U$$

or

$$A = M_1^{-1} M_2^{-1} U.$$

The inverse matrices are easy to find. The matrix M_1 multiplies the first row by 2 and adds it to the second row, and multiplies the first row by 1 and adds it to the third row. To invert these operations, we need to multiply the first row by -2 and add it to the second row, and multiply the first row by -1 and add it to the third row. To check, with

$$\mathbf{M}_1 \mathbf{M}_1^{-1} = \mathbf{I},$$

we have

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Similarly,

$$\mathbf{M}_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Therefore,

$$\mathbf{L} = \mathbf{M}_1^{-1} \mathbf{M}_2^{-1}$$

is given by

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix},$$

which is lower triangular. The off-diagonal elements of \mathbf{M}_1^{-1} and \mathbf{M}_2^{-1} are simply combined to form \mathbf{L} . Our LU decomposition is therefore

$$\begin{pmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{pmatrix}.$$

Another nice feature of the LU decomposition is that it can be done by overwriting A , therefore saving memory if the matrix A is very large.

The LU decomposition is useful when one needs to solve $A\mathbf{x} = \mathbf{b}$ for \mathbf{x} when A is fixed and there are many different \mathbf{b} 's. First one determines L and U using Gaussian elimination. Then one writes

$$(LU)\mathbf{x} = L(U\mathbf{x}) = \mathbf{b}.$$

We let

$$\mathbf{y} = U\mathbf{x},$$

and first solve

$$L\mathbf{y} = \mathbf{b}$$

for \mathbf{y} by forward substitution. We then solve

$$U\mathbf{x} = \mathbf{y}$$

for \mathbf{x} by backward substitution. When we count operations, we will see that solving $(LU)\mathbf{x} = \mathbf{b}$ is significantly faster once L and U are in hand than solving $A\mathbf{x} = \mathbf{b}$ directly by Gaussian elimination.

We now illustrate the solution of $LU\mathbf{x} = \mathbf{b}$ using our previous example, where

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} -1 \\ -7 \\ -6 \end{pmatrix}.$$

With $\mathbf{y} = U\mathbf{x}$, we first solve $L\mathbf{y} = \mathbf{b}$, that is

$$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} -1 \\ -7 \\ -6 \end{pmatrix}.$$

Using forward substitution

$$\begin{aligned} y_1 &= -1, \\ y_2 &= -7 + 2y_1 = -9, \\ y_3 &= -6 + y_1 - y_2 = 2. \end{aligned}$$

We now solve $U\mathbf{x} = \mathbf{y}$, that is

$$\begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ -9 \\ 2 \end{pmatrix}.$$

Using backward substitution,

$$\begin{aligned} -2x_3 &= 2 \rightarrow x_3 = -1, \\ -2x_2 &= -9 - 5x_3 = -4 \rightarrow x_2 = 2, \\ -3x_1 &= -1 - 2x_2 + x_3 = -6 \rightarrow x_1 = 2, \end{aligned}$$

and we have once again determined

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix}.$$

POWER METHOD FOR APPROXIMATING EIGENVALUES

the eigenvalues of an $n \times n$ matrix A are obtained by solving its characteristic equation

$$\lambda^n + c_{n-1}\lambda^{n-1} + c_{n-2}\lambda^{n-2} + \cdots + c_0 = 0.$$

For large values of n , polynomial equations like this one are difficult and time-consuming to solve. Moreover, numerical techniques for approximating roots of polynomial equations of high degree are sensitive to rounding errors. In this section you will look at an alternative method for approximating eigenvalues. As presented here, the method can be used only to find the eigenvalue of A that is largest in absolute value—this eigenvalue is called the **dominant eigenvalue** of A . Although this restriction may seem severe, dominant eigenvalues are of primary interest in many physical applications.

Definition of Dominant Eigenvalue and Dominant Eigenvector	<p>Let $\lambda_1, \lambda_2, \dots$, and λ_n be the eigenvalues of an $n \times n$ matrix A. λ_1 is called the dominant eigenvalue of A if</p> $ \lambda_1 > \lambda_i , \quad i = 2, \dots, n.$ <p>The eigenvectors corresponding to λ_1 are called dominant eigenvectors of A.</p>
---	---

Not every matrix has a dominant eigenvalue. For instance, the matrix

$$A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

(with eigenvalues of $\lambda_1 = 1$ and $\lambda_2 = -1$) has no dominant eigenvalue. Similarly, the matrix

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(with eigenvalues of $\lambda_1 = 2$, $\lambda_2 = 2$, and $\lambda_3 = 1$) has no dominant eigenvalue.

Finding a Dominant Eigenvalue

Find the dominant eigenvalue and corresponding eigenvectors of the matrix

$$A = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix}.$$

From Example 4 of Section 7.1 you know that the characteristic polynomial of A is $\lambda^2 + 3\lambda + 2 = (\lambda + 1)(\lambda + 2)$. So the eigenvalues of A are $\lambda_1 = -1$ and $\lambda_2 = -2$, of which the dominant one is $\lambda_2 = -2$. From the same example you know that the dominant eigenvectors of A (those corresponding to $\lambda_2 = -2$) are of the form

$$\mathbf{x} = t \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \quad t \neq 0.$$

The Power Method

Like the Jacobi and Gauss-Seidel methods, the power method for approximating eigenvalues is iterative. First assume that the matrix A has a dominant eigenvalue with corresponding dominant eigenvectors. Then choose an initial approximation \mathbf{x}_0 of one of the dominant eigenvectors of A . This initial approximation must be a *nonzero* vector in R^n . Finally, form the sequence given by

$$\begin{aligned}\mathbf{x}_1 &= A\mathbf{x}_0 \\ \mathbf{x}_2 &= A\mathbf{x}_1 = A(A\mathbf{x}_0) = A^2\mathbf{x}_0 \\ \mathbf{x}_3 &= A\mathbf{x}_2 = A(A^2\mathbf{x}_0) = A^3\mathbf{x}_0 \\ &\vdots \\ \mathbf{x}_k &= A\mathbf{x}_{k-1} = A(A^{k-1}\mathbf{x}_0) = A^k\mathbf{x}_0.\end{aligned}$$

For large powers of k , and by properly scaling this sequence, you will see that you obtain a good approximation of the dominant eigenvector of A . This procedure is illustrated in

Approximating a Dominant Eigenvector by the Power Method

Complete six iterations of the power method to approximate a dominant eigenvector of

$$A = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix}.$$

Begin with an initial nonzero approximation of

$$\mathbf{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Then obtain the following approximations.

Iteration		“Scaled” Approximation
$\mathbf{x}_1 = A\mathbf{x}_0 = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -10 \\ -4 \end{bmatrix}$	→	$-4 \begin{bmatrix} 2.50 \\ 1.00 \end{bmatrix}$
$\mathbf{x}_2 = A\mathbf{x}_1 = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix} \begin{bmatrix} -10 \\ -4 \end{bmatrix} = \begin{bmatrix} 28 \\ 10 \end{bmatrix}$	→	$10 \begin{bmatrix} 2.80 \\ 1.00 \end{bmatrix}$
$\mathbf{x}_3 = A\mathbf{x}_2 = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix} \begin{bmatrix} 28 \\ 10 \end{bmatrix} = \begin{bmatrix} -64 \\ -22 \end{bmatrix}$	→	$-22 \begin{bmatrix} 2.91 \\ 1.00 \end{bmatrix}$
$\mathbf{x}_4 = A\mathbf{x}_3 = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix} \begin{bmatrix} -64 \\ -22 \end{bmatrix} = \begin{bmatrix} 136 \\ 46 \end{bmatrix}$	→	$46 \begin{bmatrix} 2.96 \\ 1.00 \end{bmatrix}$
$\mathbf{x}_5 = A\mathbf{x}_4 = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix} \begin{bmatrix} 136 \\ 46 \end{bmatrix} = \begin{bmatrix} -280 \\ -94 \end{bmatrix}$	→	$-94 \begin{bmatrix} 2.98 \\ 1.00 \end{bmatrix}$
$\mathbf{x}_6 = A\mathbf{x}_5 = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix} \begin{bmatrix} -280 \\ -94 \end{bmatrix} = \begin{bmatrix} 568 \\ 190 \end{bmatrix}$	→	$190 \begin{bmatrix} 2.99 \\ 1.00 \end{bmatrix}$

Note that the approximations in Example 2 appear to be approaching scalar multiples of

$$\begin{bmatrix} 3 \\ 1 \end{bmatrix},$$

Theorem 10.2

If \mathbf{x} is an eigenvector of a matrix A , then its corresponding eigenvalue is given by

Determining an Eigenvalue
from an Eigenvector

$$\lambda = \frac{A\mathbf{x} \cdot \mathbf{x}}{\mathbf{x} \cdot \mathbf{x}}.$$

This quotient is called the **Rayleigh quotient**.

Because \mathbf{x} is an eigenvector of A , you know that $A\mathbf{x} = \lambda\mathbf{x}$ and can write

$$\frac{A\mathbf{x} \cdot \mathbf{x}}{\mathbf{x} \cdot \mathbf{x}} = \frac{\lambda\mathbf{x} \cdot \mathbf{x}}{\mathbf{x} \cdot \mathbf{x}} = \frac{\lambda(\mathbf{x} \cdot \mathbf{x})}{\mathbf{x} \cdot \mathbf{x}} = \lambda.$$

Approximating a Dominant Eigenvalue

Use the result of Example 2 to approximate the dominant eigenvalue of the matrix

$$A = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix}.$$

After the sixth iteration of the power method in Example 2, obtained

$$\mathbf{x}_6 = \begin{bmatrix} 568 \\ 190 \end{bmatrix} \approx 190 \begin{bmatrix} 2.99 \\ 1.00 \end{bmatrix}.$$

With $\mathbf{x} = (2.99, 1)$ as the approximation of a dominant eigenvector of A , use the Rayleigh quotient to obtain an approximation of the dominant eigenvalue of A . First compute the product $A\mathbf{x}$.

$$A\mathbf{x} = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix} \begin{bmatrix} 2.99 \\ 1.00 \end{bmatrix} = \begin{bmatrix} -6.02 \\ -2.01 \end{bmatrix}$$

Then, because

$$A\mathbf{x} \cdot \mathbf{x} = (-6.02)(2.99) + (-2.01)(1) \approx -20.0$$

and

$$\mathbf{x} \cdot \mathbf{x} = (2.99)(2.99) + (1)(1) \approx 9.94,$$

you can compute the Rayleigh quotient to be

$$\lambda = \frac{A\mathbf{x} \cdot \mathbf{x}}{\mathbf{x} \cdot \mathbf{x}} \approx \frac{-20.0}{9.94} \approx -2.01$$

which is a good approximation of the dominant eigenvalue $\lambda = -2$.

The Power Method with Scaling

Calculate seven iterations of the power method with *scaling* to approximate a dominant eigenvector of the matrix

$$A = \begin{bmatrix} 1 & 2 & 0 \\ -2 & 1 & 2 \\ 1 & 3 & 1 \end{bmatrix}.$$

Use $\mathbf{x}_0 = (1, 1, 1)$ as the initial approximation.

One iteration of the power method produces

$$A\mathbf{x}_0 = \begin{bmatrix} 1 & 2 & 0 \\ -2 & 1 & 2 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 5 \end{bmatrix},$$

and by scaling you obtain the approximation

$$\mathbf{x}_1 = \frac{1}{3} \begin{bmatrix} 3 \\ 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 0.60 \\ 0.20 \\ 1.00 \end{bmatrix}.$$

A second iteration yields

$$A\mathbf{x}_1 = \begin{bmatrix} 1 & 2 & 0 \\ -2 & 1 & 2 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0.60 \\ 0.20 \\ 1.00 \end{bmatrix} = \begin{bmatrix} 1.00 \\ 1.00 \\ 2.20 \end{bmatrix}$$

and

$$\mathbf{x}_2 = \frac{1}{2.20} \begin{bmatrix} 1.00 \\ 1.00 \\ 2.20 \end{bmatrix} = \begin{bmatrix} 0.45 \\ 0.45 \\ 1.00 \end{bmatrix}.$$

Continuing this process, you obtain the sequence of approximations shown in Table 10.6.

TABLE 10.6

\mathbf{x}_0	\mathbf{x}_1	\mathbf{x}_2	\mathbf{x}_3	\mathbf{x}_4	\mathbf{x}_5	\mathbf{x}_6	\mathbf{x}_7
$\begin{bmatrix} 1.00 \\ 1.00 \\ 1.00 \end{bmatrix}$	$\begin{bmatrix} 0.60 \\ 0.20 \\ 1.00 \end{bmatrix}$	$\begin{bmatrix} 0.45 \\ 0.45 \\ 1.00 \end{bmatrix}$	$\begin{bmatrix} 0.48 \\ 0.55 \\ 1.00 \end{bmatrix}$	$\begin{bmatrix} 0.51 \\ 0.51 \\ 1.00 \end{bmatrix}$	$\begin{bmatrix} 0.50 \\ 0.49 \\ 1.00 \end{bmatrix}$	$\begin{bmatrix} 0.50 \\ 0.50 \\ 1.00 \end{bmatrix}$	$\begin{bmatrix} 0.50 \\ 0.50 \\ 1.00 \end{bmatrix}$

From Table 10.6 you can approximate a dominant eigenvector of A to be

$$\mathbf{x} = \begin{bmatrix} 0.50 \\ 0.50 \\ 1.00 \end{bmatrix}.$$

Using the Rayleigh quotient, you can approximate the dominant eigenvalue of A to be $\lambda = 3$. (For this example you can check that the approximations of \mathbf{x} and λ are exact.)

Example 3. Use the power method to calculate an approximation to the dominant eigenpair for

$$A = \begin{pmatrix} -7 & 2 \\ 8 & -1 \end{pmatrix}$$

Let

$$X_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\begin{aligned} A(X_0) &= \begin{pmatrix} -7 \\ 8 \end{pmatrix} \\ A^2(X_0) &= \begin{pmatrix} 65 \\ -64 \end{pmatrix} \\ A^3(X_0) &= \begin{pmatrix} -583 \\ 584 \end{pmatrix} \end{aligned}$$

We stop here because the vectors $A^m X_0$ already appear to be approaching a multiple of

$$\begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Now

$$\lambda_1 \doteq \frac{A^3 X_0 \cdot A^2 X_0}{A^2 X_0 \cdot A^2 X_0} = \frac{-583 \cdot 65 - 584 \cdot 64}{65 \cdot 65 + 64 \cdot 64} = -9.05$$

So a dominant eigenpair is (approximately)

$$\left(-9.05, \frac{1}{583} \begin{pmatrix} -583 \\ 584 \end{pmatrix} \right) = \left(-9.05, \begin{pmatrix} -1 \\ 1.002 \end{pmatrix} \right)$$

Since the actual dominant eigenpair is

$$\left(-9, \begin{pmatrix} -1 \\ 1 \end{pmatrix} \right)$$

the method has worked well in this example.

Although the power method has worked well in these examples, we must say something about cases in which the power method may fail. There are basically three such cases:

1. Using the power method when A is not diagonalizable. Recall that A has n linearly independent eigenvectors **if and only if** A is diagonalizable. Of course, it is not easy to tell by just looking at A whether it is diagonalizable.

- Using the power method when A does not have a dominant eigenvalue, or when the dominant eigenvalue is such that

$$|\lambda_1| > |\lambda_2| \quad \text{but} \quad |\lambda_1| \doteq |\lambda_2|$$

(Then $|\lambda_1/\lambda_2|$ is barely less than 1, and high powers of $|\lambda_1/\lambda_2|$ do not tend to zero quickly.) Again, it is not easy to determine whether A has this defect by just looking at A .

- If the entries of A contain significant error. Powers A^m of A will have significant roundoff error in their entries.

Here is a rule of thumb for using the power method:

- Try it, and if the numbers

$$\frac{A^{m+1}X_0 \cdot A^m X_0}{A^m X_0 \cdot A^m X_0}$$

approach a single number λ_1 , then stop and go to step 2.

- Check whether $(\lambda_1, A^m X_0)$ is an eigenpair by checking whether

$$A(A^m X_0) \doteq \lambda_1(A^m X_0)$$

- If step 2 checks, accept

$$(\lambda_1, A^m X_0)$$

as a dominant eigenpair.

Example 4. Check the answer to Example 3.

Solution The proposed eigenpair is

$$\left(-9.05, \begin{pmatrix} -1 \\ 1.005 \end{pmatrix}\right)$$

To check, we calculate

$$A \begin{pmatrix} -1 \\ 1.005 \end{pmatrix} \quad \text{and} \quad -9.05 \begin{pmatrix} -1 \\ 1.005 \end{pmatrix}$$

We have

$$A \begin{pmatrix} -1 \\ 1.005 \end{pmatrix} = \begin{pmatrix} -7 & 2 \\ 8 & -1 \end{pmatrix} \begin{pmatrix} -1 \\ 1.005 \end{pmatrix} = \begin{pmatrix} 9.01 \\ -9.005 \end{pmatrix}$$

and

$$-9.05 \begin{pmatrix} -1 \\ 1.005 \end{pmatrix} = \begin{pmatrix} 9.05 \\ -9.09525 \end{pmatrix}$$

Thus

$$A \begin{pmatrix} -1 \\ 1.005 \end{pmatrix} - (-9.05) \begin{pmatrix} -1 \\ 1.005 \end{pmatrix} = \begin{pmatrix} -0.04 \\ 0.09025 \end{pmatrix} \doteq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

so

$$A \begin{pmatrix} -1 \\ 1.005 \end{pmatrix} \doteq -9.05 \begin{pmatrix} -1 \\ 1.005 \end{pmatrix}$$

and the answer checks.

To illustrate possible failure of the power method, we show an example.

Example 5. Try the power method on

$$A = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}$$

with

$$X_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad X_0 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Explain the results.

Solution Let

$$X_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Then we have

$$AX_0 = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

$$A^2X_0 = A \begin{pmatrix} 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$A^3X_0 = A \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

$$A^4X_0 = A \begin{pmatrix} 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

\vdots

We see that

$$A^{2n}X_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad A^{2n+1}X_0 = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

so that $A^m X_0$ is not becoming parallel to any vector. Also

$$\frac{A^{2n+1}X_0 \cdot A^{2n}X_0}{A^{2n}X_0 \cdot A^{2n}X_0} = \frac{\begin{pmatrix} 2 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}}{\begin{pmatrix} 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}} = \frac{1}{2}$$

and

$$\frac{A^{2n}X_0 \cdot A^{2n-1}X_0}{A^{2n-1}X_0 \cdot A^{2n-1}X_0} = \frac{\begin{pmatrix} 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ -1 \end{pmatrix}}{\begin{pmatrix} 2 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ -1 \end{pmatrix}} = \frac{1}{5}$$

So we have no approximation to the eigenvalues. The power method has failed when

$$X_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

When

$$X_0 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

We find

$$\begin{aligned} AX_0 &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ A^2X_0 &= A \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \\ A^3X_0 &= A \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ A^4X_0 &= A \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \\ &\vdots \end{aligned}$$

and for the same reasons as before, the power method fails. Also in this case the approximations to λ_1 oscillate between $-\frac{1}{2}$ and -1 .

An explanation for the failure is that A has no dominant eigenvalue. In fact, the eigenvalues are 1 and -1 , which have the same magnitude.

When to Stop in the Power Method We would like to have a rule about when to stop in using the power method. Usually we would stop when $|\lambda_1^{\text{calc}} - \lambda_1^{\text{actual}}|$ is small. However, in most realistic problems we do not know $\lambda_1^{\text{actual}}$, so we can only estimate $|\lambda_1^{\text{calc}} - \lambda_1^{\text{actual}}|$. This can be done for symmetric A .

3.2. Iteration method (or Method of successive approximation)

Suppose we want the approximate roots of the equation

$$f(x) = 0 \quad \dots(1)$$

Now, write the equation (1) in the form

$$x = \phi(x) \quad \dots(2)$$

Solution. Let $f(x) = x^3 - 2x - 5 = 0$

$$f(2) = -1 = -ve ; \quad f(3) = 16 = +ve$$

The root lies between 2 and 3 and closer to 2. $f(x) = 0$ can be written as

$$x^3 = 2x + 5 : \quad i.e., \quad x = (2x + 5)^{\frac{1}{3}} = \phi(x)$$

$$\phi'(x) = \frac{2}{3} \cdot \frac{1}{(2x+5)^{2/3}}$$

$$|\phi'(x)| < 1 \text{ for all } x \text{ in } (2, 3).$$

Take $x_0 = 2.0$

$$x_1 = (2x_0 + 5)^{1/3} = 9^{1/3} = 2.0801$$

$$x_2 = (9.1602)^{1/3} = 2.0924; \quad x_3 = (9.1848)^{1/3} = 2.0942$$

$$x_4 = (9.1884)^{1/3} = 2.0945; \quad x_5 = (9.1890)^{1/3} = 2.0945$$

Therefore the root is 2.0945.

Example 6. Find a positive root of $3x - \sqrt{1 + \sin x} = 0$ by iteration method.

Solution. Writing the given equation as

$$x = \frac{1}{3} \sqrt{1 + \sin x} = \phi(x), \quad \phi'(x) = \frac{\cos x}{6\sqrt{1 + \sin x}}$$

The root of given equation lies in (0, 1)

since $f(0) = -ve$ and $f(1) = +ve$

In (0, 1), $|\phi'(x)| < 1$ for all x

So, we can use iteration method.

Taking $x_0 = 0.4$, $x_1 = \frac{1}{3} \sqrt{1 + \sin(0.4)} = 0.39291$

$$x_2 = \frac{1}{3} \sqrt{1 + \sin(0.39291)} = 0.39199$$

$$x_3 = \frac{1}{3} \sqrt{1 + \sin(0.39199)} = 0.39187$$

$$x_4 = 0.39185$$

$$x_5 = 0.39185$$

The root is 0.39185.

NUMERICAL METHODS AND COMPUTER PROGRAMMING (SPH5107)

UNIT – II

INTERPOLATION, CURVE FITTING AND STATISTICS

NEWTON FORWARD INTERPOLATION FORMULA

$$P_N(x) = y_0 + \frac{\Delta y_0}{h}(x - x_0) + \frac{\Delta^2 y_0}{2! h^2}(x - x_0)(x - x_1) + \cdots + \frac{\Delta^k y_0}{k! h^k}(x - x_0) \cdots (x - x_{k-1}) \\ + \frac{\Delta^N y_0}{N! h^N}(x - x_0) \cdots (x - x_{N-1}).$$

Let $u = \frac{x - x_0}{h}$, then

$$x - x_1 = hu + x_0 - (x_0 + h) = h(u - 1), x - x_2 = h(u - 2), \dots, x - x_k = h(u - k), \text{ etc..}$$

With this transformation the above forward interpolation formula is simplified to the following form:

$$P_N(u) = y_0 + \frac{\Delta y_0}{h}(hu) + \frac{\Delta^2 y_0}{2! h^2} \{ (hu)(h(u - 1)) \} + \cdots + \frac{\Delta^k y_0 h^k}{k! h^k} [u(u - 1) \cdots (u - k + 1)] \\ + \cdots + \frac{\Delta^N y_0}{N! h^N} \left[(hu)(h(u - 1)) \cdots (h(u - N + 1)) \right] \\ = y_0 + \Delta y_0(u) + \frac{\Delta^2 y_0}{2!} (u(u - 1)) + \cdots + \frac{\Delta^k y_0}{k!} \left[u(u - 1) \cdots (u - k + 1) \right] \\ + \cdots + \frac{\Delta^N y_0}{N!} \left[u(u - 1) \cdots (u - N + 1) \right].$$

If $N = 1$, we have a linear interpolation given by

$$f(u) \approx y_0 + \Delta y_0(u).$$

For $N = 2$, we get a quadratic interpolating polynomial:

$$f(u) \approx y_0 + \Delta y_0(u) + \frac{\Delta^2 y_0}{2!} [u(u - 1)]$$

NEWTON BACKWARD INTERPOLATION FORMULA

$$P_N(x) = b_0 + b_1(x - x_N) + b_2(x - x_N)(x - x_{N-1}) + \cdots + b_N(x - x_N)(x - x_{N-1}) \cdots (x - x_1),$$

then using the fact that $P_N(x_i) = y_i$, we have

$$b_0 = y_N$$

$$b_1 = \frac{1}{h}(y_N - y_{N-1}) = \frac{1}{h}\nabla y_N$$

$$b_2 = \frac{y_N - 2y_{N-1} + y_{N-2}}{2h^2} = \frac{1}{2h^2}(\nabla^2 y_N)$$

$$\vdots$$

$$b_k = \frac{1}{k! h^k} \nabla^k y_N.$$

Thus, using backward differences and the transformation $x = x_N + hu$, we obtain the Newton's backward formula as follows:

$$P_N(u) = y_N + u\nabla y_N + \frac{u(u+1)}{2!}\nabla^2 y_N + \cdots + \frac{u(u+1)\cdots(u+N-1)}{N!}\nabla^N y_N.$$

PROBLEM 1

Obtain the Newton's forward interpolating polynomial, $P_5(x)$ for the following tabular data and interpolate the value of the function at $x = 0.0045$.

x	0	0.001	0.002	0.003	0.004	0.005
y	1.121	1.123	1.1255	1.127	1.128	1.1285

Solution: For this data, we have the Forward difference table

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$
0	1.121	0.002	0.0005	-0.0015	0.002	-.0025
.001	1.123	0.0025	-0.0010	0.0005	-0.0005	
.002	1.1255	0.0015	-0.0005	0.0		
.003	1.127	0.001	-0.0005			
.004	1.128	0.0005				
.005	1.1285					

Thus, for $x = x_0 + hu$, where $x_0 = 0$, $h = 0.001$ and $u = \frac{x - x_0}{h}$, we get

$$\begin{aligned}
 P_5(x) = & 1.121 + u \times .002 + \frac{u(u-1)}{2}(.0005) + \frac{u(u-1)(u-2)}{3!} \times (-.0015) \\
 & + \frac{u(u-1)(u-2)(u-3)}{4!}(.002) + \frac{u(u-1)(u-2)(u-3)(u-4)}{5!} \times (-.0025).
 \end{aligned}$$

$$\begin{aligned}
 P_5(0.0045) &= P_5(0 + 0.001 \times 4.5) \\
 &= 1.121 + 0.002 \times 4.5 + \frac{0.0005}{2} \times 4.5 \times 3.5 - \frac{0.0015}{6} \times 4.5 \times 3.5 \times 2.5 \\
 &\quad + \frac{0.002}{24} \times 4.5 \times 3.5 \times 2.5 \times 1.5 - \frac{0.0025}{120} \times 4.5 \times 3.5 \times 2.5 \times 1.5 \times 0.5 \\
 &= 1.12840045.
 \end{aligned}$$

PROBLEM 2

Using the following table for $\tan x$, approximate its value at 0.71. Also, find an error estimate (Note $\tan(0.71) = 0.85953$).

x_i	0.70	0.72	0.74	0.76	0.78	
$\tan x_i$	0.84229	0.87707	0.91309	0.95045	0.98926	

Solution: As the point $x = 0.71$ lies towards the initial tabular values, we shall use Newton's Forward formula. The forward difference table is:

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$		
0.70	0.84229	0.03478	0.00124	0.0001	0.00001		
0.72	0.87707	0.03602	0.00134	0.00011			
0.74	0.91309	0.03736	0.00145				
0.76	0.95045	0.03881					
0.78	0.98926						

In the above table, we note that $\Delta^3 y$ is almost constant, so we shall attempt 3rd degree polynomial interpolation.

Note that $x_0 = 0.70$, $h = 0.02$ gives $u = \frac{0.71 - 0.70}{0.02} = 0.5$. Thus, using forward interpolating polynomial of degree 3, we get

$$P_3(u) = 0.84229 + 0.03478u + \frac{0.00124}{2!}u(u-1) + \frac{0.0001}{3!}u(u-1)(u-2).$$

$$\begin{aligned} \text{Thus, } \tan(0.71) &\approx 0.84229 + 0.03478(0.5) + \frac{0.00124}{2!} \times 0.5 \times (-0.5) \\ &\quad + \frac{0.0001}{3!} \times 0.5 \times (-0.5) \times (-1.5) \\ &= 0.859535. \end{aligned}$$

An error estimate for the approximate value is

$$\frac{\Delta^4 y_0}{4!} u(u-1)(u-2)(u-3) \Big|_{u=0.5} = 0.00000039.$$

Note that exact value of $\tan(0.71)$ (upto 5 decimal place) is 0.85953. and the approximate value, obtained using the Newton's interpolating polynomial is very close to this value. This is also reflected by the error estimate given above.

PROBLEM 3

Apply third degree polynomial for the set of values given by to estimate the value of $f(10.3)$ by taking

$$(i) \ x_0 = 9.0, \quad (ii) \ x_0 = 10.0.$$

Also, find approximate value of $f(13.5)$.

Solution: Note that $x = 10.3$ is closer to the values lying in the beginning of tabular values, while $x = 13.5$ is towards the end of tabular values. Therefore, we shall use forward difference formula for $x = 10.3$ and the backward difference formula for $x = 13.5$. Recall that the interpolating polynomial of degree 3 is given by

$$f(x_0 + hu) = y_0 + \Delta y_0 u + \frac{\Delta^2 y_0}{2!} u(u-1) + \frac{\Delta^3 y_0}{3!} u(u-1)(u-2).$$

Therefore,

1. for $x_0 = 9.0$, $h = 1.0$ and $x = 10.3$, we have $u = \frac{10.3 - 9.0}{1} = 1.3$. This gives,

$$\begin{aligned} f(10.3) &\approx 5 + .4 \times 1.3 + \frac{.2}{2!} (1.3) \times .3 + \frac{.0}{3!} (1.3) \times .3 \times (-0.7) \\ &= 5.559. \end{aligned}$$

2. for $x_0 = 10.0$, $h = 1.0$ and $x = 10.3$, we have $u = \frac{10.3 - 10.0}{1} = .3$. This gives,

$$\begin{aligned} f(10.3) &\approx 5.4 + .6 \times .3 + \frac{.2}{2!} (.3) \times (-0.7) + \frac{-0.3}{3!} (.3) \times (-0.7) \times (-1.7) \\ &= 5.54115. \end{aligned}$$

Note: as $x = 10.3$ is closer to $x = 10.0$, we may expect estimate calculated using $x_0 = 10.0$ to be a

better approximation.

for $x_0 = 13.5$, we use the backward interpolating polynomial, which gives,

$$f(x_N + hu) \approx y_0 + \nabla y_N u + \frac{\nabla^2 y_N}{2!} u(u+1) + \frac{\Delta^3 y_N}{3!} u(u+1)(u+2).$$

Therefore, taking $x_N = 14$, $h = 1.0$ and $x = 13.5$, we have $u = \frac{13.5 - 14}{1} = -0.5$. This gives,

$$\begin{aligned} f(13.5) &\approx 8.1 + .6 \times (-0.5) + \frac{-0.1}{2!} (-0.5) \times 0.5 + \frac{0.0}{3!} (-0.5) \times 0.5 \times (1.5) \\ &= 7.8125. \end{aligned}$$

LAGRANGES INTERPOLATION FORMULA

$$\begin{aligned} f(x) &= \frac{(x - x_1)(x - x_2) \cdots (x - x_n)}{(x_0 - x_1) \cdots (x_0 - x_n)} f(x_0) + \frac{(x - x_0)(x - x_2) \cdots (x - x_n)}{(x_1 - x_0)(x_1 - x_2) \cdots (x_1 - x_n)} f(x_1) \\ &+ \cdots + \frac{(x - x_0)(x - x_1) \cdots (x - x_{n-1})}{(x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1})} f(x_n) \end{aligned}$$

PROBLEM 1

Using the following data, find by Lagrange's formula, the value of $f(x)$ at $x = 10$:

\bar{x}	0	1	2	3	4
x_i	9.3	9.6	10.2	10.4	10.8
$y_i = f(x_i)$	11.40	12.80	14.70	17.00	19.80

Also find the value of x where $f(x) = 16.00$.

SOLUTION:

$$\begin{aligned}
 f(10) &\approx -0.01792 \times \left[\frac{11.40}{0.7 \times 0.4455} + \frac{12.80}{0.4 \times (-0.1728)} + \frac{14.70}{(-0.2) \times 0.0648} \right. \\
 &\quad \left. + \frac{17.00}{(-0.4) \times (-0.0704)} + \frac{19.80}{(-0.8) \times 0.4320} \right] \\
 &= 13.197845.
 \end{aligned}$$

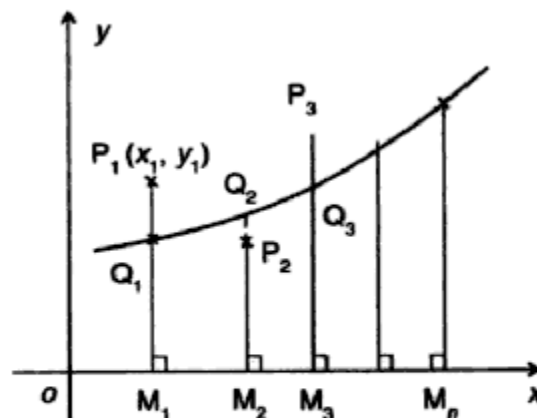
Now to find the value of x such that $f(x) = 16$, we interchange the roles of x and y and calculate the following products:

Thus, the required value of x is obtained as:

$$\begin{aligned}
 x &\approx 217.3248 \times \left[\frac{9.3}{4.6 \times 217.3248} + \frac{9.6}{3.2 \times (-78.204)} + \frac{10.2}{1.3 \times 73.5471} \right. \\
 &\quad \left. + \frac{10.40}{(-1.0) \times (-151.4688)} + \frac{10.80}{(-3.8) \times 839.664} \right] \\
 &\approx 10.39123.
 \end{aligned}$$

CURVE FITTING BY METHOD OF LEAST SQUARES

The principle of least squares the method of least squares which gives a unique set of values to the constants in the equation of the fitting curve.



Let $(x_i, y_i), i = 1, 2, \dots, n$ be the n sets of observations and let

$$y = f(x) \quad \dots(1)$$

be the relation suggested between x and y .

$$E = \sum_{i=1}^n d_i^2 = \sum_{i=1}^n [y_i - f(x_i)]^2 \text{ is the sum of the squares of the residuals.}$$

If $E = 0$, i.e., each $d_i = 0$, then all the n points P_i will lie on $y = f(x)$.

If not, we will choose $f(x)$ such that E is minimum. That is, the best fitting curve to the set of points is that for which E is minimum. This principle is known as the *principle of least squares* or the *least square criterion*.

(i) a straight line (ii) a second degree curve

Fitting a straight line by the method of least squares

Fitting a straight line

Let $y = a + bx$ be the equation of the line to be fitted. To estimate the values of a and b we have, the following normal equations.

$$\begin{aligned} \sum_{i=1}^n y_i &= na + b \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i y_i &= a \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i^2 \end{aligned}$$

Here n is the number of observations, and the quantities $\sum_{i=1}^n x_i$, $\sum_{i=1}^n y_i$, $\sum_{i=1}^n x_i y_i$ and

$\sum_{i=1}^n x_i^2$ can be obtained from the given set of points (x_i, y_i) ; $i = 1, 2, \dots, n$ and the above equations can be solved for a and b .

By the principle of least squares, E is minimum.

$$\therefore \frac{\partial E}{\partial a} = 0 \quad \text{and} \quad \frac{\partial E}{\partial b} = 0$$

$$\text{i.e.,} \quad 2 \sum [y_i - (ax_i + b)] (-x_i) = 0 \quad \text{and} \quad 2 \sum [y_i - (ax_i + b)] (-1) = 0$$

$$\text{i.e.,} \quad \sum_{i=1}^n (x_i y_i - ax_i^2 - bx_i) = 0 \quad \text{and} \quad \sum_{i=1}^n (y_i - ax_i - b) = 0$$

$$a \sum_{i=1}^n x_i^2 + b \sum_{i=1}^n x_i = \sum_{i=1}^n x_i y_i \quad \dots(1)$$

$$a \sum_{i=1}^n x_i + nb = \sum_{i=1}^n y_i \quad \dots(2)$$

Since, x_i, y_i are known, equations (1) and (2) give two equations in a and b . Solve for a and b from (1) and (2) and obtain the best fit $y = ax + b$.

Note 1. Equations (1) and (2) are called *normal equations*.

2. Dropping suffix i from (1) and (2), the normal equations are

$$a \sum x + nb = \sum y \quad \text{and} \quad a \sum x^2 + b \sum x = \sum xy$$

which are got by taking Σ on both sides of $y = ax + b$ and also taking Σ on both sides after multiplying by x both sides of $y = ax + b$.

3. Transformations like $X = \frac{x-a}{h}$, $Y = \frac{y-b}{k}$ reduce the linear

equation $y = \alpha x + \beta$ to the form $Y = AX + B$. Hence, a linear fit is another linear fit in both systems of coordinates.

PROBLEM 1. By the method of least squares find the best fitting straight line to the data given below

x :	5	10	15	20	25
y :	15	19	23	26	30

Solution. Let the straight line be $y = ax + b$

The normal equations are $a \sum x + 5b = \sum y$...(1)

$$a \sum x^2 + b \sum x = \sum xy \quad \dots(2)$$

To calculate $\sum x, \sum x^2, \sum y, \sum xy$ we form below the table.

x	y	x^2	xy
5	15	25	75
10	19	100	190
15	23	225	345
20	26	400	520
25	30	625	750
75	114	1375	1885

The normal equations are $75a + 5b = 114$... (1)

$$1375a + 75b = 1885 \quad \dots (2)$$

Eliminate b ; multiply (1) by 15

$$1125a + 75b = 1710 \quad \dots (3)$$

(2) - (3) gives, $250a = 175$ or $a = 0.7$

Hence $b = 12.3$

Hence, the best fitting line is $y = 0.7x + 12.3$

PROBLEM 2: *Fit a straight line to the data given below. Also estimate the value of y at $x = 2.5$.*

$x :$	0	1	2	3	4
$y :$	1	1.8	3.3	4.5	6.3

Solution. Let the best fit be $y = ax + b$... (1)

The normal equations are

$$a\sum x + 5b = \sum y \quad \dots (2)$$

$$a\sum x^2 + b\sum x = \sum xy \quad \dots (3)$$

We prepare the table for easy use.

	x	y	x^2	xy
	0	1.0	0	0
	1	1.8	1	1.8
	2	3.3	4	6.6
	3	4.5	9	13.5
	4	6.3	16	25.2
Total	10	16.9	30	47.1

Substituting in (2) and (3), we get,

$$10a + 5b = 16.9$$

$$30a + 10b = 47.1$$

Solving, we get, $a = 1.33$, $b = 0.72$

Hence, the equation is $y = 1.33x + 0.72$

$$y \text{ (at } x = 2.5) = 1.33 \times 2.5 + 0.72 = 4.045$$

Find the best-fit values of a and b so that $y = a + bx$ fits the data given in the table.

$x:$	0	1	2	3	4
$y:$	1	1.8	3.3	4.5	6.3

Sol. Let the straight line is $y = a + bx$... (1)

x	y	xy	x^2
0	1	0	0
1	1.8	1.8	1
2	3.3	6.6	4
3	4.5	13.5	9
4	6.3	25.2	16
$\sum x = 10$	$\sum y = 16.9$	$\sum xy = 47.1$	$\sum x^2 = 30$

Normal equations are, $\sum y = na + b \sum x$... (2)

$$\sum xy = a \sum x + b \sum x^2 \quad \dots (3)$$

Here $n = 5$, $\sum x = 10$, $\sum y = 16.9$, $\sum xy = 47.1$, $\sum x^2 = 30$

Putting these values in normal equations, we get

$$16.9 = 5a + 10b$$

$$47.1 = 10a + 30b$$

On solving these two equations, we get

$$a = 0.72, \quad b = 1.33.$$

So required line $y = 0.72 + 1.33x$. **Ans.**

Fitting a parabola

Let $y = a + bx + cx^2$ be the equation of the line to be fitted. To estimate the values of a and b and c , we have, the following normal equations.

$$\sum_{i=1}^n y_i = na + b \sum_{i=1}^n x_i + c \sum_{i=1}^n x_i^2$$

$$\sum_{i=1}^n x_i y_i = a \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i^2 + c \sum_{i=1}^n x_i^3$$

$$\sum_{i=1}^n x_i^2 y_i = a \sum_{i=1}^n x_i^2 + b \sum_{i=1}^n x_i^3 + c \sum_{i=1}^n x_i^4$$

Here n is the number of observations, and the quantities $\sum_{i=1}^n x_i$, $\sum_{i=1}^n y_i$, $\sum_{i=1}^n x_i y_i$, $\sum_{i=1}^n x_i^2$,

Find the least square polynomial approximation of degree two to the data.

x	0	1	2	3	4
y	-4	-1	4	11	20

Also compute the least error.

Sol. Let the equation of the polynomial be $y = a + bx + cx^2$... (1)

x	y	xy	x^2	x^2y	x^3	x^4
0	-4	0	0	0	0	0
1	-1	-1	1	-1	1	1
2	4	8	4	16	8	16
3	11	33	9	99	27	81
4	20	80	16	320	64	256
$\sum x = 10$	$\sum y = 30$	$\sum xy = 120$	$\sum x^2 = 30$	$\sum x^2y = 434$	$\sum x^3 = 100$	$\sum x^4 = 354$

The normal equations are,

$$\sum y = na + b \sum x + c \sum x^2 \quad \dots (2)$$

$$\sum xy = a \sum x + b \sum x^2 + c \sum x^3 \quad \dots (3)$$

$$\sum x^2y = a \sum x^2 + b \sum x^3 + c \sum x^4 \quad \dots (4)$$

Here $n = 5$, $\sum x = 10$, $\sum y = 30$, $\sum xy = 120$, $\sum x^2 = 30$, $\sum x^2y = 434$, $\sum x^3 = 100$, $\sum x^4 = 354$.

Putting all these values in (2), (3) and (4), we get

$$30 = 5a + 10b + 30c \quad \dots (5)$$

$$120 = 10a + 30b + 100c \quad \dots (6)$$

$$434 = 30a + 100b + 354c \quad \dots (7)$$

On solving these equations, we get $a = -4$, $b = 2$, $c = 1$. Therefore required polynomial is $y = -4 + 2x + x^2$, errors = 0. **Ans.**

Example 5: Fit a second degree curve of regression of y on x to the following data:

x	1	2	3	4
y	6	11	18	27

Sol. We form the following table:

x	y	x^2	x^3	x^4	xy	x^2y
1	6	1	1	1	6	6
2	11	4	8	16	22	44
3	18	9	27	81	54	162
4	27	16	64	256	108	432
$\Sigma x = 10$	$\Sigma y = 62$	$\Sigma x^2 = 30$	$\Sigma x^3 = 100$	$\Sigma x^4 = 354$	$\Sigma xy = 190$	$\Sigma x^2y = 644$

The equation of second degree parabola is given by

$$y = a + bx + cx^2 \quad \dots(1)$$

And the normal equations are

$$\Sigma y = an + b\Sigma x + c\Sigma x^2 \quad \dots(2)$$

$$\Sigma xy = a\Sigma x + b\Sigma x^2 + c\Sigma x^3 \quad \dots(3)$$

$$\Sigma x^2y = a\Sigma x^2 + b\Sigma x^3 + c\Sigma x^4 \quad \dots(4)$$

$$\left. \begin{array}{l} 4a + 10b + 30c = 62 \\ 10a + 30b + 100c = 190 \\ 30a + 100b + 354c = 644 \end{array} \right\} \Rightarrow a = 3, b = 2, c = 1$$

$$y = 3 + 2x + x^2. \quad \mathbf{Ans.}$$

Problem:

Fit a second-degree parabola to the following data by least squares method.

x	1929	1930	1931	1932	1933	1934	1935	1936	1937
y	352	356	357	358	360	361	361	360	359

Sol. Taking $x_0 = 1933$, $y_0 = 357$ then $u = \frac{(x - x_0)}{h}$

Here $h = 1$

Taking $u = x - x_0$ and $v = y - y_0$, therefore, $u = x - 1933$ and $v = y - 357$

x	$u = x - 1933$	y	$v = y - 357$	uv	u^2	u^2v	u^3	u^4
1929	-4	352	-5	20	16	-80	-64	256
1930	-3	356	-1	3	9	-9	-27	81
1931	-2	357	0	0	4	0	-8	16
1932	-1	358	1	-1	1	1	-1	1
1933	0	360	3	0	0	0	0	0
1934	1	361	4	4	1	4	1	1
1935	2	361	4	8	4	16	8	16
1936	3	360	3	9	9	27	27	81
1937	4	359	2	8	16	32	64	256
Total	$\sum u = 0$		$\sum v = 11$	$\sum uv = 51$	$\sum u^2 = 60$	$\sum u^2v = -9$	$\sum u^3 = 0$	$\sum u^4 = 708$

Then the equation $y = a + bx + cx^2$ is transformed to $v = A + Bu + Cu^2$... (1)

Normal equations are:

$$\begin{aligned}\sum v &= 9A + B\sum u + C\sum u^2 \Rightarrow 11 = 9A + 60C \\ \sum uv &= A\sum u + B\sum u^2 + C\sum u^3 \Rightarrow B = 17/20 \\ \sum u^2v &= A\sum u^2 + B\sum u^3 + C\sum u^4 \Rightarrow -9 = 60A + 708C\end{aligned}$$

On solving these equations, we get $A = \frac{694}{231} = 3$, $B = \frac{17}{20} = 0.85$ and $C = -\frac{247}{924} = -0.27$

$$\therefore v = 3 + 0.85u - 0.27u^2$$

$$\Rightarrow y - 357 = 3 + 0.85(x - 1933) - 0.27(x - 1933)^2$$

$$\Rightarrow y = -1010135.08 + 1044.69x - 0.27x^2. \text{ Ans.}$$

CORRELATION AND REGRESSION

1. Karl Pearson Coefficient of Correlation

$$r(X, Y) = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}$$
$$r(X, Y) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\left[\sum_{i=1}^n (x_i - \bar{x})^2 \cdot \sum_{i=1}^n (y_i - \bar{y})^2 \right]^{\frac{1}{2}}}$$
$$r(X, Y) = \frac{n \sum_{i=1}^n x_i y_i - \left(\sum_{i=1}^n x_i \right) \left(\sum_{i=1}^n y_i \right)}{\sqrt{\left(n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2 \right) \left(n \sum_{i=1}^n y_i^2 - \left(\sum_{i=1}^n y_i \right)^2 \right)}}$$

The maximum value r can achieve is 1, and its minimum value is -1 .

Therefore, for any given set of observations, $-1 \leq r \leq 1$.

The values $r = 1$ and $r = -1$ occur when there is an exact linear relationship between x and y . As the relationship between x and y deviates from perfect linearity, r moves away from 1 or -1 and closer to 0.

If y tends to increase in magnitude as x increases, r is greater than 0 and x and y are said to be *positively correlated*; if y decreases as x increases, r is less than 0 and the two variables are *negatively correlated*.

If $r = 0$, there is no linear relationship between x and y and the variables are uncorrelated.

2. Spearman's Rank Correlation Coefficient

If X and Y are qualitative variables we use Spearman's rank correlation coefficient which is defined as follows:

$$\rho = 1 - \frac{6 \sum_{i=1}^n d_i^2}{n(n^2 - 1)} \quad \text{Where } d \text{ is the difference in ranks.}$$

Note: If there is more than one item with the same value or rank in the series then Spearman's formula for calculating the rank correlation coefficients is modified as follows. In this case, common ranks are given to the repeated ranks. This common rank is the average of the ranks which these items would have assumed if they are slightly different from each other and the next item will get the rank next the ranks already assumed. As a result of this, following adjustment is made in the formula: add the factor $\frac{m(m^2 - 1)}{12}$ to $\sum d^2$ where m is the number of items an item is repeated. This correction

factor is to be added for each repeated value.

$\sum_{i=1}^n x_i^3$, $\sum_{i=1}^n x_i^4$ and $\sum_{i=1}^n x_i^2 y_i$ can be obtained from the given set of points (x_i, y_i) ; $i = 1, 2, \dots, n$ and the above equations can be solved for a , b and c .

3. Regression equations

Prediction or estimation of most likely values of one variable for specified values of the other is done by using suitable equations involving the two variables. Such equations are known as *Regression Equations*

Regression equation of y on x:

$y - \bar{y} = b_{yx} (x - \bar{x})$ where y is the dependent variable and x is the independent variable and b_{yx} is given by

$$b_{yx} = \frac{\sum_{i=1}^n (x - \bar{x})(y - \bar{y})}{\sum_{i=1}^n (x - \bar{x})^2}$$

or

$$b_{yx} = r \frac{\sigma_y}{\sigma_x} = \frac{n \sum_{i=1}^n xy - \sum_{i=1}^n x \sum_{i=1}^n y}{n \sum_{i=1}^n x^2 - \left(\sum_{i=1}^n x \right)^2}$$

Regression equation of x on y:

$x - \bar{x} = b_{xy} (y - \bar{y})$ where y is the dependent variable and x is the independent variable and b_{yx} is given by

$$b_{xy} = \frac{\sum_{i=1}^n (x - \bar{x})(y - \bar{y})}{\sum_{i=1}^n (y - \bar{y})^2}$$

or

$$b_{xy} = r \frac{\sigma_x}{\sigma_y} = \frac{n \sum_{i=1}^n xy - \sum_{i=1}^n x \sum_{i=1}^n y}{n \sum_{i=1}^n y^2 - \left(\sum_{i=1}^n y \right)^2}$$

b_{yx} and b_{xy} are called as regression coefficients of y on x and x on y respectively.

Relation between correlation and regression coefficients:

$$b_{yx} = r \frac{\sigma_y}{\sigma_x} \quad \text{and} \quad b_{xy} = r \frac{\sigma_x}{\sigma_y}$$

$$b_{yx} \cdot b_{xy} = r \frac{\sigma_y}{\sigma_x} \cdot r \frac{\sigma_x}{\sigma_y} = r^2$$

Hence $r = \pm \sqrt{b_{yx} b_{xy}}$

Note: In the above expression the components inside the square root is valid only when b_{yx} and b_{xy} have the same sign. Therefore the regression coefficients will have the same sign.

The data below gives the marks obtained by 10 pupils taking Maths and Physics tests.

Pupil	A	B	C	D	E	F	G	H	I	J
Maths mark (out of 30) x	20	23	8	29	14	11	11	20	17	17
Physics mark (out of 40) y	30	35	21	33	33	26	22	31	33	36

Calculate the correlation co-efficient?

$$r = \frac{\frac{1}{n} \Sigma xy - \bar{x}\bar{y}}{s_x s_y}$$

where $s_x = \sqrt{\frac{1}{n} \Sigma x^2 - \bar{x}^2}$ and $s_y = \sqrt{\frac{1}{n} \Sigma y^2 - \bar{y}^2}$.

$$r = \frac{\frac{1}{10} \times 5313 - 17 \times 30}{s_x \times s_y}$$

$$\bar{x} = \frac{170}{10} = 17$$

$$\bar{y} = \frac{300}{10} = 30 \quad s_x = \sqrt{\frac{1}{10} \times 3250 - 17^2} = \sqrt{36} = 6$$

$$s_y = \sqrt{\frac{1}{10} \times 9250 - 30^2} = \sqrt{25} = 5$$

$$\Rightarrow r = \frac{531.3 - 510}{6 \times 5}$$

$$= 0.71$$

A group of twelve children participated in a psychological study designed to assess the relationship, if any, between age, x years, and average total sleep time (ATST), y minutes. To obtain a measure for ATST, recordings were taken on each child on five consecutive nights and then averaged. The results obtained are shown in the table.

Child	Age (x years)	ATST (y minutes)
A	4.4	586
B	6.7	565
C	10.5	515
D	9.6	532
E	12.4	478
F	5.5	560
G	11.1	493
H	8.6	533
I	14.0	575
J	10.1	490
K	7.2	530
L	7.9	515

$$\sum x = 108 \quad \sum y = 6372 \quad \sum x^2 = 1060.1 \quad \sum y^2 = 3396942 \quad \sum xy = 56825.4$$

Calculate the value of the product moment correlation coefficient between x and y . Assess the statistical significance of your value and interpret your results.

Solution

(a) Use the formula

$$s_{xy} = \frac{1}{n} \sum xy - \bar{x}\bar{y}$$

$$\text{when } \bar{x} = \frac{108}{12} = 9 \text{ and } \bar{y} = \frac{6372}{12} = 531.$$

$$\text{Thus } s_{xy} = \frac{1}{12} (56825.4) - 9 \times 531 = -43.55$$

$$\text{Also } s_x = \sqrt{\frac{1}{12} \times 1060.1 - 9^2} \approx 2.7096$$

$$s_y = \sqrt{\frac{1}{12} \times 3396942 - 531^2} \approx 33.4290$$

$$\text{Hence } r = \frac{-43.55}{2.7096 \times 33.4290} \approx -0.481$$

Rank correlation problem:

Two judges at a fete placed the ten entries for the 'best fruit cakes' competition in order as follows (1 denotes first, etc.)

Entry	A	B	C	D	E	F	G	H	I	J
Judge 1 (x)	2	9	1	3	10	4	6	8	5	7
Judge 2 (y)	6	9	2	1	8	4	3	10	7	5

Is there a linear relationship between the rankings produced by the two judges?

Solution:

$$r_s = 1 - \frac{6 \sum d^2}{n(n^2 - 1)}$$

$d = x - y$, is the difference in ranking.

Entry	A	B	C	D	E	F	G	H	I	J
Judge 1	2	9	1	3	10	4	6	8	5	7
Judge 2	6	9	2	1	8	4	3	10	7	5
$ d $	4	0	1	2	2	0	3	2	2	2
d^2	16	0	1	4	4	0	9	4	4	4

So $\Sigma d^2 = 16 + 0 + 1 + \dots + 4 = 46$

$$\begin{aligned}
 &\Rightarrow r_s = 1 - \frac{6 \times 46}{10(100 - 1)} \\
 &= 1 - \frac{6 \times 46}{10 \times 99} \\
 &= \frac{119}{165}
 \end{aligned}$$

≈ 0.721 to 3 decimal places.

Repeated rank problem:

Find the value of n for the following data

Ranks x	1	2 =	2 =	5	4	6	7	8
Ranks y	1	3	4	2	5	6 =	6 =	6 =

Solution

Those tied in the x rankings are given a value of $\frac{2+3}{2} = 2\frac{1}{2}$ and

those tied in y are allocated $\frac{6+7+8}{3} = 7$. (In general, each tie is

given the mean of the places that would have been occupied if a strict order had been produced.) The table, therefore, becomes

Ranks x	1	$2\frac{1}{2}$	$2\frac{1}{2}$	5	4	6	7	8
Ranks y	1	3	4	2	5	7	7	7
$ d $	0	$\frac{1}{2}$	$1\frac{1}{2}$	3	1	1	0	1
d^2	0	$\frac{1}{4}$	$2\frac{1}{4}$	9	1	1	0	1

Hence $\Sigma d^2 = 14.5$

and
$$r_s = 1 - \frac{6 \times 14.5}{8(64 - 1)}$$

$$= 0.827$$

Problems on Regression:

Find the lines of regression for the following data:

Mass g (x)	50	100	150	200	250	300	350	400
Length mm (y)	37	48	60	71	80	90	102	109

Solution:

$$\bar{x} = \frac{1800}{8} = 225, \quad \bar{y} = \frac{597}{8} = 74.625$$

$$s_{xy} = \frac{156150}{8} - 225 \times 74.625 = 2728.125$$

$$s_x^2 = \frac{510000}{8} - 225^2 = 13125$$

$$\Rightarrow y - 74.625 = \frac{2728.125}{13125}(x - 225)$$

$$y - \bar{y} = \frac{s_{xy}}{s_x^2}(x - \bar{x})$$

$$\Rightarrow y = 0.208x + 27.857$$

The values of 0.208 and 27.857 represent the gradient of the line and its intercept on the y -axis.

2.

In a decathlon held over two days the following performances were recorded in the high jump and long jump. All distances are in metres.

Competitor	A	B	C	D	E	F	G
High jump x	1.90	1.85	1.96	1.88	1.88	Abs	1.92
Long jump y	6.22	6.24	6.50	6.36	6.32	6.44	Abs

What performances might have been expected from F in the high jump and G in the long jump if they had competed?

Solution

To estimate G's performance in the long jump we use the y on x line.

$$\text{Now} \quad y - \bar{y} = \frac{s_{xy}}{s_x^2}(x - \bar{x})$$

and using competitors A to E,

$$\bar{y} = \frac{31.64}{5} = 6.328, \quad \bar{x} = \frac{9.47}{5} = 1.894$$

$$\text{Also} \quad s_{xy} = \frac{1}{5} \times 59.9404 - 6.328 \times 1.894 = 0.002848$$

$$s_x^2 = \frac{1}{5} \times 17.9429 - 1.894^2 = 0.001344$$

$$\Rightarrow \quad y - 6.328 = \frac{0.002848}{0.001344} (x - 1.894)$$

$$\Rightarrow \quad y = 2.119x + 2.315$$

$$\text{Thus} \quad x = 1.92 \text{ gives } \hat{y} = 2.119 \times 1.92 + 2.315 = 6.38\text{m}$$

Now to estimate F's high jump accurately we need a line for which the sum of the horizontal distances from it is a minimum.

This is the x on y line and its equation is

$$x - \bar{x} = \frac{s_{xy}}{s_y^2} (y - \bar{y})$$

$$s_y^2 = \frac{1}{5} \times 200.268 - 6.328^2 = 0.010016$$

$$\Rightarrow x - 1.894 = \frac{0.002848}{0.010016} (y - 6.328)$$

$$\Rightarrow x = 0.284y + 0.095.$$

$$y = 6.44 \Rightarrow \hat{x} = 0.284 \times 6.44 + 0.095 = 1.92 \text{ m}$$

$$x = 1.92 \Rightarrow \hat{y} = 6.38$$

$$y = 6.44 \Rightarrow \hat{x} = 1.92$$

THEORY OF SAMPLING TEST OF HYPOTHESIS

Population:

The group of individuals, under study is called is called population.

Sample:

A finite subset of statistical individuals in a population is called Sample

Sample size:

The number of individuals in a sample is called the Sample size.

Parameters and Statistics:

The statistical constants of the population are referred as Parameters and the statistical constants of the Sample are referred as Statistics.

Standard Error :

The standard deviation of sampling distribution of a statistic is known as its standard error and is denoted by (S.E)

Test of Significance :

It enable us to decide on the basis of the sample results if the deviation between the observed sample statistic and the hypothetical parameter value is significant or the deviation between two sample statistics is significant.

Null Hypothesis:

A definite statement about the population parameter which is usually a hypothesis of no-difference and is denoted by H_0 .

Alternative Hypothesis:

Any hypothesis which is complementary to the null hypothesis is called an Alternative Hypothesis and is denoted by H_1 .

Errors in Sampling:

Type I and Type II errors.

Type I error : Rejection of H_0 when it is true.

Type II error : Acceptance of H_0 when it is false.

Two types of errors occurs in practice when we decide to accept or reject a lot after examining a sample from it. They are Type 1 error occurs while rejecting H_0 when it is true. Type 2 error occurs while accepting H_0 when it is wrong.

Critical region:

A region corresponding to a statistic t in the sample space S which lead to the rejection of H_0 is called Critical region or Rejection region. Those regions which lead to the acceptance of H_0 are called Acceptance Region.

Level of Significance :

The probability α that a random value of the statistic “ t ” belongs to the critical region is known as the level of significance. In otherwords the level of significance is the size of the type I error. The levels of significance usually employed in testing of hypothesis are 5% and 1%.

A test of any statistical hypothesis where the alternate hypothesis is one tailed(right tailed/ left tailed) is called one tailed test.

For the null hypothesis H_0 if $\mu = \mu_0$ then.

$H_1 = \mu > \mu_0$ (Right tail)

$H_1 = \mu < \mu_0$ (Left tail)

$H_1 = \mu \neq \mu_0$ (Two tail test)

Types of samples :

Small sample and Large sample

Small sample ($n \leq 30$) : “Students t test, F test , Chi Square test

Large sample ($n > 30$) : Z test.

95 % confidence limit for the population mean μ in a small test.

Let \bar{x} be the sample mean and n be the sample size. Let s be the sample S.D.

Then $\bar{x} \pm t_{0.05} (s/\sqrt{n-1})$

Application of t – distribution

When the size of the sample is less than 30, ‘t’ test is used in (a) single mean and (b) difference of two means.

Distinguish between parameters and statistics.

Statistical constant of the population are usually referred to as parameters. Statistical measures computed from sample observations alone are usually referred to as statistic.

In practice, parameter values are not known and their estimates based

The critical or rejection region is the region which corresponds to a predetermined level of significance α . Whenever the sample statistic falls in the critical region we reject the null hypothesis as it will be considered to be probably false. The value that separates the rejection region from the acceptance region is called the critical value.

level of significance

The probability α that a random value of the statistic 't' belongs to the critical region is known as the level of significance. In other words level of significance is the size of type I error. The levels of significance usually employed in testing of hypothesis are 5% and 1%.

Type I Student t test for single mean

$$|t| = \frac{\bar{x} - \mu}{s / \sqrt{n-1}}$$

Where \bar{x} is the sample mean, μ is the population mean, s is the SD and n is the number of observations.

1. The mean weekly sales of soap bars in departmental stores were 146.3 bars per store. After an advertising campaign the mean weekly sales in 22 stores for a typical week increased to 153.7 and showed a SD of 17.2. Was the advertising campaign successful?

Solution:

Calculated t value = 1.97 and Tabulated Value = 1.72(at 5% level of significance with 21 degrees of freedom)

Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

2. A sample of 26 bulbs gives a mean life of 990 hours with SD of 20 hours. The manufacturer claims that the mean life of bulbs is 1000 hours. Is the sample not upto the standard.

Solution:

Calculated t value = 2.5

Tabulated Value = 1.708(at 5% level of significance with 25 degrees of freedom)

Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

3. The average breaking strength of steel rod is specified to be 18.5 thousand pounds. To test this a sample of 14 rods was tested. The mean and SD obtained were 17.85 and 1.955 respectively. Is the result of the experiment significant?

Solution:

Calculated t value = 1.199

Tabulated Value = 2.16(at 5% level of significance with 13 degrees of freedom)

Calculated value < Tabulated value, Accept H_0 (Null hypothesis)

A machinist is making engine parts with axle diameter of 0.700 inch. A random sample of 10 parts shows a mean diameter of 0.742 inch with a standard deviation of 0.040 inch. Compute the statistic you would use to test whether the work is meeting the specifications

Solution:

Here we are given

$\mu = 0.700$ inch $\bar{x} = 0.742$ inch $s = 0.040$ inch and $n = 10$

Null hypothesis $H_0: \mu = 0.700$ inch

i.e., the product is conforming to specifications

Alternative hypothesis: $H_1: \mu \neq 0.700$ inch

$$\text{The test statistic } t = \frac{\bar{x} - \mu}{\sqrt{s^2 / n - 1}} = \frac{0.742 - 0.700}{\sqrt{(0.040)^2 / (10 - 1)}} = 3.15$$

Here the test statistic follows the student's t distribution with $10-1 = 9$ degrees of freedom. At 5 % level of significance, the table value is 2.26

Hence the calculated value > table value

So the null hypothesis is rejected and we conclude that there is significant difference and the product is not conforming to specifications.

Type II Student t test when SD not given

$$|t| = (\bar{x} - \mu) / (s/\sqrt{n})$$

Where $\bar{x} = \Sigma(x)/n$ and $s^2 = 1/(n-1) \Sigma (x - \bar{x})^2$

Students t test where SD of the sample is not given directly)

1. A random sample of 10 boys had the following IQ's 70,120,110,101,88,83,95,98,107,100. Do these data support the assumption of a population mean IQ of 100? Find the reasonable range in which most of the mean IQ values of samples of 10 boys lie?

Solution:

Calculated t value = 0.62

Tabulated Value = 2.26 (at 5% level of significance with 9 degrees of freedom)

Calculated value < Tabulated value, Accept Ho (Null hypothesis)

95% confidence limits : (86.99,107.4)

2. The heights of 10 males of a given locality are found to be 70,67,62,68,61,68,70,64,64,66 inches. Is it reasonable to believe that the average height is greater than 64 inches Test at 5%.

Solution:

Calculated t value = 2

Tabulated Value = 1.833 (at 5% level of significance with 9 degrees of freedom)

Calculated value > Tabulated value, Reject Ho (Null hypothesis)

3. Certain pesticide is packed into bags by a machine. A random sample of 10 bags is drawn and their contents are found to be as follows: 50,49,52,44,45,48,46,45,49,45. Test if the average packing to be taken 50 grams

Solution:

Calculated t value = 3.19

Tabulated Value = 2.262 (at 5% level of significance with 9 degrees of freedom)

Calculated value > Tabulated value, Reject Ho (Null hypothesis)

Type III Student t test for difference of means of two samples

To test the significant difference between two mean \bar{x}_1 and \bar{x}_2 of sample sizes n_1 and n_2 use the statistic.

$$|t| = (\bar{x}_1 - \bar{x}_2) / s \sqrt{(1/n_1) + (1/n_2)}$$

where $s^2 = (n_1 s_1^2 + n_2 s_2^2) / (n_1 + n_2 - 2)$

s_1 and s_2 being the sample standard deviations degree of freedom being $n_1 + n_2 - 2$.

1. Samples of two types of electric light bulbs were tested for length of life and following data were obtained.

Type I	Type II
Sample size $n_1 = 8$	$n_2 = 7$
Sample means $\bar{x}_1 = 1234$ hrs	$\bar{x}_2 = 1036$ hrs
Sample S.D. $s_1 = 36$ hrs	$s_2 = 40$ hrs

Is the difference in the means sufficient to warrant that type I is superior to type II regarding length of life.

Solution:

Calculated t value = 9.39

Tabulated Value = 1.77 (at 5% level of significance with 13 degrees of freedom)

Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

2. Below are given the gain in weights (in N) of pigs fed on two diets A and B.

Diet A	25	32	30	34	24	14	32	24	30	31	35	25		
Diet B	44	34	22	10	47	31	40	32	35	18	21	35	29	22

Test if the two diets differ significantly as regards their effect on increase in weight.

Solution:

Calculated t value = 0.609

Tabulated Value = 2.06 (at 5% level of significance with 25 degrees of freedom)

Calculated value < Tabulated value, Accept Ho (Null hypothesis)

3. The nicotine content in milligrams of two samples of tobacco were found to be as follows:

Sample A	24	27	26	21	25	
Sample B	27	30	28	31	22	36

Can it be said that two samples come from normal populations having the same mean.

Solution:

Calculated t value = 1.92

Tabulated Value = 2.262 (at 5% level of significance with 9 degrees of freedom)

Calculated value < Tabulated value, Accept Ho (Null hypothesis)

CHI-SQUARE TEST FORMULAE

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

Where O is the observed frequency and E is the Expected frequency

Under the test of goodness of fit we try to find out how far observed values of a given phenomenon are significantly different from the expected values. The Chi square statistic can be used to judge the difference between the observed and expected frequencies.

χ^2 test can be applied only for small samples.

Degree of freedom = n – 1 where n is the no. of observations.

CHI-SQUARE TEST FOR INDEPENDENCE OF ATTRIBUTES

An attribute means a quality or characteristic. Eg. Drinking, smoking, blindness, honesty

2 X 2 CONTINGENCY TABLE

Consider any two attributes A and B. A and B are divided into two classes.

OBSERVED FREQUENCIES

A	a	b
B	c	d

EXPECTED FREQUENCIES

$E(a) = \frac{(a+c)(a+b)}{N}$	$E(b) = \frac{(b+d)(a+b)}{N}$	a+b
$E(c) = \frac{(a+c)(c+d)}{N}$	$E(d) = \frac{(b+d)(c+d)}{N}$	c+d
a+c	b+d	N(Total frequencies)

PROBLEMS

1. A die is thrown 264 times with the following results. Show that the die is biased

No appeared on the die	1	2	3	4	5	6
Frequency	40	32	28	58	54	60

Solution:

Calculated χ^2 value = 17.6362

Tabulated Value = 11.07 (at 5% level of significance with 5 degrees of freedom)

Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

2. 200 digits were chosen at random from a set of tables. The frequencies of the digits were

Digits	0	1	2	3	4	5	6	7	8	9
Frequency	18	19	23	21	16	25	22	20	21	15

Use the χ^2 test to assess the correctness of the hypothesis that the digits were distributed in the equal number in the tables from which these were chosen.

Solution:

Calculated χ^2 value = 4.3

Tabulated Value = 16.919 (at 5% level of significance with 9 degrees of freedom)

Calculated value < Tabulated value, Accept H_0 (Null hypothesis)

3. Two groups of 100 people each were taken for testing the use of a vaccine 15 persons contracted the disease out of the inoculated persons while 25 contracted the disease in the other group. Test the efficiency of the vaccine using chi square test.

Solution:

Calculated χ^2 value = 3.125

Tabulated Value = 3.84 (at 5% level of significance with 1 degrees of freedom)

Calculated value < Tabulated value, Accept H_0 (Null hypothesis)

Problem 4:

Given the following contingency table for hair colour and eye colour. Find the value of Chi-Square and is there any good association between the two

Hair colour Eye colour	Fair	Brown	Black
Grey	20	10	20
Brown	25	15	20
Black	15	5	20

Solution:

Calculated χ^2 value = 3.6458

Tabulated Value = 9.488 (at 5% level of significance with 4 degrees of freedom)

Calculated value < Tabulated value, Accept H_0 (Null hypothesis)

LARGE SAMPLES**TEST OF SIGNIFICANCE OF LARGE SAMPLES**

If the size of the sample $n > 30$ then that sample is called large sample.

Type 1. Test of significance for single proportion

Let p be the sample proportion and P be the population proportion, we use the statistic $Z = (p - P) / \sqrt{(PQ/n)}$

Limits for population proportion P are given by $p \pm 3\sqrt{(PQ/n)}$

Where $q = 1 - p$

1. A manufacture claimed that at least 95% of the equipment which he supplied to a factory conformed to specifications. An examination of a sample of 200 pieces of equipment revealed that 18 were faulty. Test his claim at 5% level of significance.

Solution:

Calculated Z value = 2.59

Tabulated Value = 1.96 (at 5% level of significance) Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

2. In a big city 325 men out of 600 men were found to be smokers. Does this information support the conclusion that the majority of men in this city are smokers.

Solution:

Calculated Z value = 2.04

Tabulated Value = 1.645 (at 5% level of significance) Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

3. A die is thrown 9000 times and of these 3220 yielded 3 or 4. Is this consistent with the hypothesis that the die was unbiased?

Solution:

Calculated Z value = 4.94 since $z > 3$

Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

4 A random sample of 500 apples were taken from the large consignment and 65 were found to be bad. Find the percentage of bad apples in the consignment.

Solution:

(0.175, 0.085) Hence percentage of bad apples in the consignment lies between 17.5% and 8.5%

Type II Test of significance for difference of proportions

Let n_1 and n_2 are the two sample sizes and sample proportions are p_1 and p_2

$$Z = \frac{(p_1 - p_2)}{\sqrt{pq(1/n_1 + 1/n_2)}} \text{ where } p = (n_1p_1 + n_2p_2)/n_1 + n_2 \text{ and } q = 1 - p$$

1. Before an increase in excise duty on tea, 800 persons out of a sample of 1000 persons were found to be tea drinkers. After an increase in duty 800 people were tea drinkers in the sample of 1200 people. Using standard error of proportions state whether there is a significant decrease in the consumption of tea after the increase in the excise duty.

Solution:

Calculated Z value = 6.972

Tabulated value at 5% (one tail) = 1.645

Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

2. In two large populations there are 30% and 25% respectively of fair haired people. Is this difference likely to be hidden in samples of 1200 and 900 respectively from the two populations.

Solution:

Calculated Z value = 2.55

Tabulated value at 5% = 1.96

Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

Type III Test of significance for single Mean

$z = \frac{\bar{x} - \mu}{(\sigma/\sqrt{n})}$ where \bar{x} is the sample mean
 μ is the population mean, σ is the population S.D.
 n is the sample size.

The values of $\bar{x} \pm 1.96 (\sigma/\sqrt{n})$ are called 95% confidence limits for the mean of the population corresponding to the given sample.

The values of $\bar{x} \pm 2.58 (\sigma/\sqrt{n})$ are called 99% confidence limits for the mean of the population corresponding to the given sample.

1. A sample of 900 members has a mean of 3.4 cms and SD 2.61 cms. Is the sample from a large population of mean is 3.25 cm and SD 2.61 cms. If the population is normal and its mean is unknown find the 95% confidence limits of true mean.

Solution:

Calculated Z value = 1.724

Tabulated value at 5% = 1.96

Calculated value < Tabulated value, Accept H_0 (Null hypothesis)

Limits (3.57, 3.2295)

2. An insurance agent has claimed that the average age of policy holders who issue through him is less than the average for all agents which is 30.5 years. A random sample of 100 policy holders who had issued through him gave the following age distribution.

Age	16-20	21-25	26-30	31-35	36-40
No of persons	12	22	20	30	16

Test the significant difference at 5% level of significance.

Solution:

Calculated Z value = 2.68

Tabulated value at 5% = 1.645

Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

3 Write down the test statistic for single mean for large samples.

$$Z = \frac{\bar{X} - \mu}{(\sigma/\sqrt{n})}$$
 where \bar{X} is the sample mean
 μ is the population mean, σ is the population S.D.
 n is the sample size.

4. The mean score of a random sample of 60 students is 145 with a SD of 40. Find the 95 % confidence limit for the population mean.

Solution
$$\bar{X} \pm Z (\sigma/\sqrt{n})$$

$$= 145 \pm (1.96) (40/\sqrt{60})$$

$$= 145 \pm 10.12$$

$$= 155.12 \text{ or } 134.88$$

\therefore The confidence limits are 155.12 and 134.88.

Type IV Test of significance for Difference of means

$$Z = \frac{(\bar{x}_1 - \bar{x}_2)}{\sqrt{(\sigma_1^2/n_1) + (\sigma_2^2/n_2)}}$$

PROBLEMS

1. The means of 2 large samples of 1000 and 2000 members are 67.5 inches and 68 inches respectively. Can the samples be regarded as drawn from the same population of SD 2.5 inches.

Solution:

Calculated Z value = 5.16

Tabulated value at 5% = 1.96

Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

2. The mean yield of wheat from a district A was 210 pounds with SD 10 pounds per acre from a sample of 100 plots. In another district the mean yield was 220 pounds with SD 12 pounds from a sample of 150 plots. Assuming that the SD of yield in the entire state was 11 pounds test whether there is any significant difference between the mean yield of crops in the two districts.

Solution:

Calculated Z value = 7.041

Tabulated value at 5% = 1.96

Calculated value > Tabulated value, Reject H_0 (Null hypothesis)

SPH5107 NUMERICAL METHODS AND COMPUTER PROGRAMMING

UNIT 3

NUMERICAL DIFFERENTIATION AND INTEGRATION

Newton's forward difference formula to get the derivative

We are given $(n + 1)$ ordered pairs (x_i, y_i) $i = 0, 1, \dots, n$. We want to

find the derivative of $y = f(x)$ passing through the $(n + 1)$ points, at a point nearer to the starting value $x = x_0$.

Newton's forward difference interpolation formula is

$$y(x_0 + uh) = y_u = y_0 + u \Delta y_0 + \frac{u(u-1)}{2!} \Delta^2 y_0 + \frac{u(u-1)(u-2)}{3!} \Delta^3 y_0 + \dots \quad \dots(1)$$

where $y(x)$ is a polynomial of degree n in x and $u = \frac{x - x_0}{h}$.

Differentiating $y(x)$ w.r.t. x ,

$$\begin{aligned} \frac{dy}{dx} &= \frac{dy}{du} \cdot \frac{du}{dx} = \frac{1}{h} \cdot \frac{dy}{du} \\ \frac{dy}{dx} &= \frac{1}{h} \left[\Delta y_0 + \frac{2u-1}{2} \Delta^2 y_0 + \frac{3u^2-6u+2}{6} \Delta^3 y_0 \right. \\ &\quad \left. + \frac{(4u^3-18u^2+22u-6)}{24} \Delta^4 y_0 + \dots \right] \quad \dots(2) \end{aligned}$$

Equation (2) gives the value of $\frac{dy}{dx}$ at general x which may be anywhere in the interval.

In special case like $x = x_0$, i.e., $u = 0$, (2) reduces to

$$\left(\frac{dy}{dx}\right)_{x=x_0} = \left(\frac{dy}{dx}\right)_{u=0} = \frac{1}{h} \left[\Delta y_0 - \frac{1}{2} \Delta^2 y_0 + \frac{1}{3} \Delta^3 y_0 - \frac{1}{4} \Delta^4 y_0 + \dots \right] \dots(3)$$

Differentiating (2) again w.r.t. x ,

$$\begin{aligned} \frac{d^2 y}{dx^2} &= \frac{d}{du} \left(\frac{dy}{dx} \right) \cdot \frac{du}{dx} = \frac{d}{du} \left(\frac{dy}{dx} \right) \cdot \frac{1}{h} \\ \frac{d^2 y}{dx^2} &= \frac{1}{h^2} \left[\Delta^2 y_0 + (u-1) \Delta^3 y_0 + \frac{(6u^2 - 18u + 11)}{12} \Delta^4 y_0 + \dots \right] \dots(4) \end{aligned}$$

$$\text{Hence, } \frac{d^3 y}{dx^3} = \frac{1}{h^3} \left[\Delta^3 y_0 + \frac{12u-18}{12} \Delta^4 y_0 + \dots \right] \dots(5)$$

Equations (4) and (5) give the second and third derivative value at $x = x_0$.

Setting $x = x_0$ i.e., $u = 0$ in (4) and (5)

$$\left(\frac{d^2 y}{dx^2}\right)_{x=x_0} = \frac{1}{h^2} \left[\Delta^2 y_0 - \Delta^3 y_0 + \frac{11}{12} \Delta^4 y_0 + \dots \right] \dots(6)$$

$$\left(\frac{d^3 y}{dx^3}\right)_{x=x_0} = \frac{1}{h^3} \left[\Delta^3 y_0 - \frac{3}{2} \Delta^4 y_0 + \dots \right] \dots(7)$$

Newton's backward difference formula to compute the derivative

Now, consider Newton's backward difference interpolation formula,

$$y(x) = y(x_n + vh) = y_n + v \nabla y_n + \frac{v(v+1)}{2!} \nabla^2 y_n + \frac{v(v+1)(v+2)}{3!} \nabla^3 y_n + \dots \dots(8)$$

where $v = \frac{x - x_n}{h}$.

Differentiate (8) w.r.t. x ,

$$\frac{dy}{dx} = \frac{dy}{dv} \cdot \frac{dv}{dx} = \frac{dy}{dv} \cdot \frac{1}{h}$$

$$\left(\frac{dy}{dx} \right) = \frac{1}{h} \left[\nabla y_n + \frac{2v+1}{2} \nabla^2 y_n + \frac{3v^2+6v+2}{6} \nabla^3 y_n + \frac{4v^3+18v^2+22v+6}{24} \nabla^4 y_n + \dots \right] \dots(9)$$

$$\therefore \frac{d^2y}{dx^2} = \frac{1}{h^2} \left[\nabla^2 y_n + (v+1) \nabla^3 y_n + \frac{6v^2+18v+11}{12} \nabla^4 y_n + \dots \right] \dots(10)$$

$$\therefore \frac{d^3y}{dx^3} = \frac{1}{h^3} \left[\nabla^3 y_n + \frac{12v+18}{12} \nabla^4 y_n + \dots \right] \dots(11)$$

Equations (9), (10), (11) give the first, second, and third derivative at any general x .

Setting $x = x_n$ or $v = 0$ in (9), (10), (11), we get

$$\left(\frac{dy}{dx} \right)_{x=x_n} = \frac{1}{h} \left[\nabla y_n + \frac{1}{2} \nabla^2 y_n + \frac{1}{3} \nabla^3 y_n + \frac{1}{4} \nabla^4 y_n + \dots \right] \dots(12)$$

$$\left(\frac{d^2y}{dx^2} \right)_{x=x_n} = \frac{1}{h^2} \left[\nabla^2 y_n + \nabla^3 y_n + \frac{11}{12} \nabla^4 y_n + \dots \right] \dots(13)$$

$$\left(\frac{d^3y}{dx^3} \right)_{x=x_n} = \frac{1}{h^3} \left[\nabla^3 y_n + \frac{3}{2} \nabla^4 y_n + \dots \right] \dots(14)$$

Example 1. Find the first two derivatives of $(x)^{1/3}$ at $x = 50$ and $x = 56$ given the table below:

x	:	50	51	52	53	54	55	56
$y = x^{1/3}$:	3.6840	3.7084	3.7325	3.7563	3.7798	3.8030	3.8259

Solution. Since we require $f'(x)$ at $x = 50$ we use Newton's forward formula and to get $f'(x)$ at $x = 56$ we use Newton's backward formula.

Difference Table

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
50	3.6840	0.0244		
51	3.7084	0.0241	- 0.0003	
52	3.7325	0.0238	- 0.0003	0
53	3.7563	0.0235	- 0.0003	0
54	3.7798	0.0232	- 0.0003	0
55	3.8030	0.0229	- 0.0003	0
56	3.8259			

By Newton's forward formula,

$$\begin{aligned}
 \left(\frac{dy}{dx} \right)_{x=x_0} &= \left(\frac{dy}{dx} \right)_{u=0} \\
 &= \frac{1}{h} \left[\Delta y_0 - \frac{1}{2} \Delta^2 y_0 + \frac{1}{3} \Delta^3 y_0 - \dots \right] \\
 &= \frac{1}{1} \left[0.0244 - \frac{1}{2} (-0.0003) + \frac{1}{3} (0) \right] \\
 &= \mathbf{0.02455}
 \end{aligned}$$

$$\begin{aligned}
 \left(\frac{d^2 y}{dx^2} \right)_{x=50} &= \frac{1}{h^2} \left[\Delta^2 y_0 - \Delta^3 y_0 + \dots \right] \\
 &= 1 [-0.0003] = \mathbf{-0.0003}.
 \end{aligned}$$

By Newton's backward difference formula,

$$\left(\frac{dy}{dx}\right)_{x=x_n} = \left(\frac{dy}{dx}\right)_{x=0} = \frac{1}{h} \left[\nabla y_n + \frac{1}{2} \nabla^2 y_n + \frac{1}{3} \nabla^3 y_n + \dots \right]$$

from equation (12)

$$\begin{aligned} \left(\frac{dy}{dx}\right)_{x=56} &= \frac{1}{1} \left[0.0229 + \frac{1}{2} (-0.0003) + 0 \right] \\ &= 0.02275 \end{aligned}$$

$$\left(\frac{d^2y}{dx^2}\right)_{x=56} = \frac{1}{h^2} [\nabla^2 y_n + \nabla^3 y_n + \dots] \quad \text{from equation (13)}$$

$$= \frac{1}{1} [-0.0003] = -0.0003$$

Example 2. The population of a certain town is given below. Find the rate of growth of the population in 1931, 1941, 1961 and 1971.

Year	x	1931	1941	1951	1961	1971
Population in thousands	y	40.62	60.80	79.95	103.56	132.65

Solution. We form the difference table.

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$
1931	40.62				
1941	60.80	20.18			
1951	79.95	19.15	-1.03		
1961	103.56	23.61	4.46	5.49	
1971	132.65	29.09	5.48	1.02	-4.47

We use the same table for backward and forward differences.

(i) To get $f'(1931)$ and $f'(1941)$ we use forward formula,

$$x_0 = 1931, \quad x_1 = 1941, \dots$$

$$u = \frac{x - x_0}{h}; \quad \therefore x_0 = 1931 \text{ corresponds } u = 0.$$

$$\begin{aligned} \left(\frac{dy}{dx} \right)_{x=1931} &= \left(\frac{dy}{dx} \right)_{u=0} = \frac{1}{h} \left[\Delta y_0 - \frac{1}{2} \Delta^2 y_0 + \frac{1}{3} \Delta^3 y_0 - \frac{1}{4} \Delta^4 y_0 + \dots \right] \\ &= \frac{1}{10} \left[20.18 - \frac{1}{2} (-1.03) + \frac{1}{3} (5.49) - \frac{1}{4} (-4.47) \right] \\ &= \frac{1}{10} [20.18 + 0.515 + 1.83 + 1.1175] \\ &= 2.36425. \end{aligned} \quad \dots(1)$$

$$(ii) \text{ If } x = 1941, u = \frac{x - x_0}{h} = \frac{1941 - 1931}{10} = 1$$

Putting $u = 1$, in

$$\begin{aligned} \frac{dy}{dx} &= \frac{1}{h} \left[\Delta y_0 + \frac{2u-1}{2} \Delta^2 y_0 + \frac{3u^2-6u+2}{6} \Delta^3 y_0 \right. \\ &\quad \left. + \frac{4u^3-18u^2+22u-6}{24} \Delta^4 y_0 - \dots \right] \end{aligned}$$

We get

$$\begin{aligned} \left(\frac{dy}{dx} \right)_{u=1} &= \frac{1}{10} \left[20.18 + \frac{1}{2} (-1.03) - \frac{1}{6} (5.49) + \frac{1}{12} (-4.47) \right] \\ &= \frac{1}{10} [20.18 - 0.515 - 0.915 - 0.3725] \\ &= 1.83775 \end{aligned}$$

EXAMPLE 3::

A rod is rotating in a plane. The following table gives the angle θ (in radians) through which the rod has turned for various values of time t (seconds). Calculate the angular velocity and angular acceleration of the rod at $t = 0.6$ seconds.

t	:	0	0.2	0.4	0.6	0.8	1.0
θ	:	0	0.12	0.49	1.12	2.02	3.20

Solution. We form the difference table below:

t	θ	$\nabla\theta$	$\nabla^2\theta$	$\nabla^3\theta$	$\nabla^4\theta$
0	0	0.12			
0.2	0.12	0.37	0.25		
0.4	0.49	0.63	0.26	0.01	0
0.6	1.12	0.90	0.27	0.01	0
0.8	2.02	1.18	0.28	$(\nabla^3\theta_n)$	
1.0	3.20	$(\nabla\theta_n)$	$(\nabla^2\theta_n)$		

Since $x = 0.6$ is towards the end, we will use backward difference formula. (We can also use central difference formula).

By Newton's backward difference formula,

$$\left(\frac{dy}{dx}\right)_{x=x} = \frac{1}{h} \left[\nabla y_n + \frac{2v+1}{2} \nabla^2 y_n + \frac{3v^2+6v+2}{6} \nabla^3 y_n + \frac{4v^3+18v^2+22v+6}{24} \nabla^4 y_n + \dots \right] \quad \dots(1)$$

Here $v = \frac{x - x_n}{h} = \frac{0.6 - 1.0}{0.2} = -2$

Using in (1),

$$\begin{aligned} \left(\frac{d\theta}{dt}\right)_{t=0.6} &= \frac{1}{0.2} \left[1.18 - \frac{3}{2} (0.28) + \frac{1}{3} (0.01) \right] \\ &= 5 [1.18 - 0.42 + 0.00333] \\ &= 3.81665 \text{ radians/sec.} \end{aligned}$$

Also, $\frac{d^2y}{dx^2} = \frac{1}{h^2} [\nabla^2 y_n + (v+1) \nabla^3 y_n + \dots]$

$$\begin{aligned} \left(\frac{d^2\theta}{dt^2}\right)_{t=0.6} &= \frac{1}{0.04} [0.28 - 0.01] \\ &= 6.75 \text{ radians/sec.}^2 \end{aligned}$$

TRAPEZOIDAL RULE FOR INTEGRATION:

$$\int_a^b f(x)dx = \frac{h}{2} [y_0 + y_1] + \frac{h}{2} [y_1 + y_2] + \cdots + \frac{h}{2} [y_k + y_{k+1}] + \cdots + \frac{h}{2} [y_{n-2} + y_{n-1}] + \frac{h}{2} [y_{n-1} + y_n]$$

i.e.

$$\int_a^b f(x)dx = \frac{h}{2} [y_0 + 2y_1 + 2y_2 + \cdots + 2y_k + \cdots + 2y_{n-1} + y_n]$$

$(h/2)$ [(sum of the first and last ordinates) +
(Sum of the remaining ordinates)]

This is called TRAPEZOIDAL RULE. It is a simple quadrature formula, but is not very accurate.

Remark An estimate for the error E_1 in numerical integration using the Trapezoidal rule is given by

$$E_1 = -\frac{b-a}{12} \overline{\Delta^2 y},$$

where $\overline{\Delta^2 y}$ is the average value of the second forward differences.

Recall that in the case of linear function, the second forward differences is zero, hence, the Trapezoidal rule gives exact value of the integral if the integrand is a linear function.

Example 1 Using Trapezoidal rule compute the integral $\int_0^1 e^{x^2} dx$, where the table for the values of $y =$

e^{x^2} is given below:

x	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y	1.00000	1.01005	1.04081	1.09417	1.17351	1.28402	1.43332	1.63231	1.89648	2.2479	2.71828

Solution: Here, $h = 0.1$, $n = 10$,

$$\frac{y_0 + y_{10}}{2} = \frac{1.0 + 2.71828}{2} = 1.85914,$$

and

$$\sum_{i=1}^9 y_i = 12.81257.$$

Thus,

$$\int_0^1 e^{x^2} dx = 0.1 \times [1.85914 + 12.81257] = 1.467171$$

Simpson's Rule

If we are given odd number of tabular points, i.e. n is even, then we can divide the given integral of integration in even number of sub-intervals $[x_{2k}, x_{2k+2}]$. Note that for each of these sub-intervals, we have the three tabular points $x_{2k}, x_{2k+1}, x_{2k+2}$ and so the integrand is replaced with a quadratic interpolating polynomial.

$$\begin{aligned}
 \int_a^b f(x)dx &= \frac{h}{3} [(y_0 + y_n) + 4 \times (y_1 + y_3 + \cdots + y_{2k+1} + \cdots + y_{n-1}) \\
 &\quad + 2 \times (y_2 + y_4 + \cdots + y_{2k} + \cdots + y_{n-2})] \\
 &= \frac{h}{3} \left[(y_0 + y_n) + 4 \times \left(\sum_{i=1, i-\text{odd}}^{n-1} y_i \right) + 2 \times \left(\sum_{i=2, i-\text{even}}^{n-2} y_i \right) \right].
 \end{aligned}$$

An estimate for the error E_2 in numerical integration using the Simpson's rule

$$E_2 = -\frac{b-a}{180} \Delta^4 y,$$

Simpson's one third Rule

Suppose the following table represents a set of values of x and y .

$x:$	x_0	x_1	x_2	x_3	x_n
$y:$	y_0	y_1	y_2	y_3	y_n

From the above values, we want to find the integration of $y = f(x)$ with the range x_0 and $x_0 + n^h$

$$\begin{aligned}
 \int_{x_0}^{x_n} f(x) dx &= (h/3) [(y_0 + y_n) + 2 (y_2 + y_4 + \cdots) + 4 (y_1 + y_3 + \cdots)] \\
 &= (h/3) [(\text{sum of the first and last ordinates}) + \\
 &\quad + 2 (\text{Sum of remaining even ordinates}) \\
 &\quad + 4 (\text{sum of remaining odd ordinates})]
 \end{aligned}$$

The above equation is called *Simpson's one third rule* and it is applicable only when **number of ordinates must be odd** (no. of pairs).

CALCULATE $\int_0^1 e^{x^2} dx$, by Simpson's rule.

Solution: Here, $h = 0.1$, $n = 10$, thus we have odd number of nodal points. Further,

$$y_0 + y_{10} = 1.0 + 2.71828 = 3.71828, \quad \sum_{i=1, i-\text{odd}}^9 y_i = y_1 + y_3 + y_5 + y_7 + y_9 = 7.26845,$$

and

$$\sum_{i=2, i-\text{even}}^8 y_i = y_2 + y_4 + y_6 + y_8 = 5.54412.$$

Thus,

$$\int_0^1 e^{x^2} dx = \frac{0.1}{3} \times [3.71828 + 4 \times 7.268361 + 2 \times 5.54412] = 1.46267733$$

1. Evaluate $\int_{-3}^3 x^4 dx$ by using Trapezoidal rule. Verify result by actual integration.

Step 1. We are given that $f(x) = x^4$. Interval length $(b-a) = (3 - (-3)) = 6$. So we divide 6 equal intervals with $h = 6/6 = 1.0$. And tabulate the values as below

x	:	-3	-2	-1	0	1	2	3
y	:	81	16	1	0	1	16	81

Step 2. Write down the trapezoidal rule and put the respective values in that rule

$$\begin{aligned}
 \int_{-3}^3 f(x) dx &= (h/2) [(y_0 + y_n) + 2 (y_1 + y_2 + y_3 + \dots + y_{n-1})] \\
 &= (h/2) [(\text{sum of the first and last ordinates}) + (\text{Sum of the remaining ordinates})] \\
 &= (1/2) [(81+81) + 2 (16+1+0+1+16)] \\
 &= 115
 \end{aligned}$$

By actual integration $\int_{-3}^3 f(x) dx = \int_{-3}^3 x^4 dx$

$$\begin{aligned}
 &= [(3^5/5) - (-3^5/5)] \\
 &= [(243/5) + (243/5)] \\
 &= 97.5
 \end{aligned}$$

Evaluate $\int_0^1 1/(1+x^2) dx$ by using Trapezoidal rule with $h = 0.2$

$$\begin{aligned}
 \int_0^1 f(x) dx &= (h/2) [(y_0 + y_n) + 2 (y_1 + y_2 + y_3 + \dots + y_{n-1})] \\
 &= (h/2) [(\text{sum of the first and last ordinates}) + (\text{Sum of the remaining ordinates})] \\
 &= (0.2/2) [(1+0.5) + 2 (0.96154+0.86207+0.73529+0.60976)] \\
 &= (0.1) [(1.05) + 6.33732] \\
 &= 0.783732
 \end{aligned}$$

Evaluate $\int_0^6 1/(1+x) dx$ by using Trapezoidal rule .

Step 1. We are given that $f(x) = 1/(1+x)$. Interval length $(b-a) = (6-0) = 6$. So we divide 6 equal intervals with $h=1$. And tabulate the values as below

x	:	0	1	2	3	4	5	6
$y=1/(1+x^2)$:		1	0.5	1/3	1/4	1/5	1/6	1/7

Step2. Write down the trapezoidal rule and put the respective values of y in that rule

$$\begin{aligned}
 \int_0^6 f(x) dx &= (h/2) [(y_0 + y_n) + 2 (y_1 + y_2 + y_3 + \dots + y_{n-1})] \\
 &= (h/2) [(\text{sum of the first and last ordinates}) + (\text{Sum of the remaining ordinates})] \\
 &= (1/2) [(1+1/7) + 2 (0.5+1/3 + 1/4 + 1/5 + 1/6)] \\
 &= (0.5) [(1.05) + 6.33732]
 \end{aligned}$$

Evaluate $\int_4^{5.2} \log_e x \, dx$ by using Trapezoidal rule .

Step 1. We are given that $f(x) = \log_e x$. Interval length $(b-a) = (5.2-4) = 1.2$. So we divide 6 equal intervals with $h=0.2$. And tabulate the values as below

x	:	4	4.2	4.4	4.6	4.8	5.0	5.2
y	:	1.39	1.44	1.48	1.53	1.57	1.61	1.65

Step2. Write down the trapezoidal rule and put the respective values of y in that rule

$$\begin{aligned}
 \int_4^{5.2} f(x) dx &= (h/2) [(y_0 + y_n) + 2 (y_1 + y_2 + y_3 + \dots + y_{n-1})] \\
 &= (h/2) [(\text{sum of the first and last ordinates}) + (\text{Sum of the remaining ordinates})] \\
 &= (0.2/2) [(1.39+1.65) + 2 (1.44 + 1.48 + 1.53 + 1.57 + 1.61)] \\
 &= (0.1) [3.04 + 2(7.63)] \\
 &= 1.83
 \end{aligned}$$

Evaluate $\int_0^\pi \sin x \, dx$ by using Trapezoidal rule, by dividing the range

into ten equal parts .

Solution :

Step 1. We are given that $f(x) = \sin x$ Interval length $(b - a) = (\pi - 0) = \pi$.

So we divide 10 equal intervals with $h = \pi/10$ (specified in the question itself), and tabulate the values as below

<i>x:</i>	<i>0</i>	<i>$\pi/10$</i>	<i>$2\pi/10$</i>	<i>$3\pi/10$</i>	<i>$4\pi/10$</i>	
<i>Y:</i>	<i>0.0</i>	<i>0.3090</i>	<i>0.5878</i>	<i>0.8090</i>	<i>0.9511</i>	
<i>x:</i>	<i>$5\pi/10$</i>	<i>$6\pi/10$</i>	<i>$7\pi/10$</i>	<i>$8\pi/10$</i>	<i>$9\pi/10$</i>	<i>π</i>
<i>Y:</i>	<i>1.0</i>	<i>0.9511</i>	<i>0.8090</i>	<i>0.5878</i>	<i>0.3090</i>	<i>0</i>

Step2. Write down the trapezoidal rule and put the respective values of y in that rule

$$\int_a^b f(x) dx = (h/2) [(y_0 + y_n) + 2 (y_1 + y_2 + y_3 + \dots + y_{n-1})]$$

$$= (h/2) [(\text{sum of the first and last ordinates}) + (\text{Sum of the remaining ordinates})]$$

$$= (\pi/20) [(0+0) + 2(0.3090+0.5878+0.8090+0.9511+1.0+0.9511+0.8090+0.5878+0.309)]$$

$$= 1.9843$$

2. Evaluate $\int_0^{1.2} 1/(1+x^2) dx$ by using Simpson's one third rule with $h = 0.2$

Solution:

Step 1. We are given that $f(x) = 1/(1+x^2)$. Interval length $(b-a) = (1.2-0) = 1.2$. So we divide 6 equal intervals with $h=0.2$ And tabulate the values as below

x	:	0	0.2	0.4	0.6	0.8	1.0	1.2
$y=1/(1+x^2)$:		1	0.9615	0.8621	0.7353	0.6098	0.5000	0.4098

Step2. Write down the Simpson's one third rule and put the respective values of y in that rule

$$\int_0^1 f(x) dx = (h/3) [(y_0 + y_6) + 2 (y_2 + y_4) + 4 (y_1 + y_3 + y_5)]$$

$$= (h/3) [(\text{sum of the first and last ordinates}) + \\ + 2 (\text{Sum of remaining even ordinates}) \\ + 4 (\text{sum of remaining odd ordinates})]$$

$$= (0.2/3) [(1+0.4098) + 2 (0.8621 + 0.6098) + 4 (0.9615+0.7353+0.5)]$$

$$= (0.0667) [(1.4098) + 2(1.4719) + 4 (2.1503)]$$

$$= (0.0667) [1.4098 + 2.9438 + 8.6012]$$

$$=0.8641$$

Evaluate $\int_{-3}^3 x^4 dx$ by using Simpson's one third rule. Verify result by actual integration.

Step 1. We are given that $f(x) = x^4$. Interval length $(b-a) = (3 - (-3)) = 6$. So we divide 6 equal intervals with $h = 6/6 = 1.0$. And tabulate the values as below

x	:	-3	-2	-1	0	1	2	3
y	:	81	16	1	0	1	16	81

Step 2. Write down the Simpson's one third rule and put the respective values in that rule

$$\int_{-3}^3 f(x) dx = (h/3) [(y_0 + y_6) + 2(y_2 + y_4) + 4(y_1 + y_3 + y_5)]$$

$$= (h/3) [(\text{sum of the first and last ordinates}) + 2(\text{Sum of remaining even ordinates}) + 4(\text{sum of remaining odd ordinates})]$$

$$= (1/3) [(81+81) + 2(1+1) + 4(16+1+16)]$$

$$= 98$$

By actual integration $\int_{-3}^3 f(x) dx = \int_{-3}^3 x^4 dx$

$$= [(3^5/5) - (-3^5/5)]$$

$$= [(243/5) + (243/5)]$$

$$= 97.5$$

3. Evaluate $\int_0^6 \frac{1}{(1+x)} dx$ by using Simpson's one third rule .

Solution:

Step 1. We are given that $f(x) = 1/(1+x)$. Interval length $(b-a) = (6-0) = 6$. So we divide 6 equal intervals with $h=1$. And tabulate the values as below

x	:	0	1	2	3	4	5	6
$y=1/(1+x^2)$:		1	0.5	1/3	1/4	1/5	1/6	1/7

Step2. Write down the Simpson's one third rule and put the respective values of y in that rule

$$\begin{aligned}
 \int_0^6 f(x) dx &= (h/3) [(y_0 + y_6) + 2 (y_2 + y_4) + 4 (y_1 + y_3 + y_5)] \\
 &= (h/3) [(\text{sum of the first and last ordinates}) + \\
 &\quad + 2 (\text{Sum of remaining even ordinates}) \\
 &\quad + 4 (\text{sum of remaining odd ordinates})] \\
 &= (1/3) [(1+1/7) + 2 (1/3 + 1/5) + 4(0.5+1/4 +1/6)] \\
 &= \mathbf{1.9587}
 \end{aligned}$$

4. Evaluate $\int_4^{5.2} \log_e x dx$ by using Simpson's one third rule .

Solution:

Step 1. We are given that $f(x) = \log_e x$ Interval length $(b-a) = (5.2-4) = 1.2$. So we divide 6 equal intervals with $h=0.2$. And tabulate the values as below

x	:	4	4.2	4.4	4.6	4.8	5.0	5.2
y	:	1.39	1.44	1.48	1.53	1.57	1.61	1.65

Step2. Write down the Simpson's one third rule and put the respective values of y in that rule

$$\begin{aligned}
 \int_4^{5.2} f(x) dx &= (h/3) [(y_0 + y_6) + 2 (y_2 + y_4) + 4 (y_1 + y_3 + y_5)] \\
 &= (h/3) [(\text{sum of the first and last ordinates}) + \\
 &\quad + 2 (\text{Sum of remaining even ordinates}) \\
 &\quad + 4 (\text{sum of remaining odd ordinates})]
 \end{aligned}$$

$$\begin{aligned}
 &= (0.2/3) [(1.39+1.65) + 2 (1.48+ 1.57) + 4 (1.44+ 1.53++1.61)] \\
 &= (0.0667) [3.04 + 2(3.05)+ 4 (4.58)] \\
 &= 1.83
 \end{aligned}$$

5. Evaluate $\int_0^\pi \sin x \, dx$ by using Simpson's one third rule, by dividing the range into ten equal parts .

Solution :

Step 1. We are given that $f(x) = \sin x$ Interval length $(b - a) = (\pi - 0) = \pi$.

So we divide 10 equal intervals with $h = \pi/10$ (specified in the question itself), and tabulate the values as below

<i>x:</i>	<i>0</i>	<i>$\pi/10$</i>	<i>$2\pi/10$</i>	<i>$3\pi/10$</i>	<i>$4\pi/10$</i>	
<i>Y:</i>	<i>0.0</i>	<i>0.3090</i>	<i>0.5878</i>	<i>0.8090</i>	<i>0.9511</i>	
<i>x:</i>	<i>$5\pi/10$</i>	<i>$6\pi/10$</i>	<i>$7\pi/10$</i>	<i>$8\pi/10$</i>	<i>$9\pi/10$</i>	<i>π</i>
<i>Y:</i>	<i>1.0</i>	<i>0.9511</i>	<i>0.8090</i>	<i>0.5878</i>	<i>0.3090</i>	<i>0</i>

Step2. Write down the Simpson's one third rule and put the respective values of y in that rule

$$\begin{aligned}
 \int_0^\pi f(x) \, dx &= (h/3) [(y_0 + y_{10}) + 2 (y_2 + y_4 + y_6 + y_8) + 4 (y_1 + y_3 + y_5 + y_7 + y_9)] \\
 &= (h/3) [(\text{sum of the first and last ordinates}) + \\
 &\quad + 2 (\text{Sum of remaining even ordinates}) \\
 &\quad + 4 (\text{sum of remaining odd ordinates})] \\
 &= (\pi/20) [(0+0) + 2(0.5878+ 0.9511+0.9511+0.5878) + \\
 &\quad 4(0.3090+0.8090+ 1+ 0.8090+0.3090)] \\
 &= 2.0009
 \end{aligned}$$

Simpson's three-eighth Rule

Suppose the following table represents a set of values of x and y.

x:	x_0	x_1	x_2	x_3	x_n
y:	y_0	y_1	y_2	y_3	y_n

From the above values, we want to find the integration of $y = f(x)$ with the range x_0 and $x_0 + h$

$$\begin{aligned} \int_{x_0}^{x_n} f(x) dx &= (3h/8) [(y_0 + y_n) + 2 (y_3 + y_6 + y_9 + \dots) + 3 (y_1 + y_2 + y_4 + y_5 + \dots + y_{n-1})] \\ &= (3h/8) [(\text{sum of the first and last ordinates}) + \\ &\quad + 2 (\text{Sum of multiples of three ordinates}) \\ &\quad + 3 (\text{sum of remaining ordinates})] \end{aligned}$$

The above equation is called Simpson's three-eighths rule which is **applicable** only when **n is multiple of 3**.

1. Evaluate $\int_{-3}^3 x^4 dx$ by using Simpson's three-eighth rule. Verify result by actual integration.

Step 1. We are given that $f(x) = x^4$. Interval length $(b - a) = (3 - (-3)) = 6$. So we divide 6 equal intervals with $h = 6/6 = 1.0$. And tabulate the values as below

x	:	-3	-2	-1	0	1	2	3
y	:	81	16	1	0	1	16	81

Step2. Write down the Simpson's three-eighth rule and put the respective values in that rule

$$\begin{aligned} \int_{-3}^3 f(x) dx &= (3h/8) [(y_0 + y_6) + 2 (y_3) + 3 (y_1 + y_2 + y_4 + y_5)] \\ &= (3h/8) [(\text{sum of the first and last ordinates}) + \\ &\quad + 2 (\text{Sum of multiples of three, other than last ordinates}) \\ &\quad + 3 (\text{sum of remaining ordinates})] \\ &= (3/8) [(81 + 81) + 2 (0) + 3(16 + 1 + 1 + 16)] \\ &= 99 \end{aligned}$$

By actual integration $\int_{-3}^3 f(x) dx = \int_{-3}^3 x^4 dx$

$$= [(3^5/5) - (-3^5/5)]$$

$$= [(243/5) + (243/5)]$$

$$= 97.5$$

Evaluate $\int_0^{1.2} 1/(1+x^2) dx$ by using Simpson's three-eighth rule with $h = 0.2$

Step 1. We are given that $f(x) = 1/(1+x^2)$. Interval length $(b-a) = (1-0) = 1$. So we divide 6 equal intervals with $h=0.2$ and tabulate the values as below

x	:	0	0.2	0.4	0.6	0.8	1.0	1.2
$y=1/(1+x^2)$:		1	0.9615	0.8621	0.7353	0.6098	0.5000	0.4098

Step2. Write down the Simpson's three-eighth rule and put the respective values of y in that rule

$$\int_0^1 f(x) dx = (3h/8) [(y_0 + y_6) + 2(y_3) + 3(y_1 + y_2 + y_4 + y_5)]$$

$$= (3h/8) [(\text{sum of the first and last ordinates}) + \\ + 2 (\text{Sum of multiples of three, other than last ordinates}) \\ + 3 (\text{sum of remaining ordinates})]$$

$$= (3 \times 0.2 / 8) [(1 + 0.4098) + 2(0.7353) \\ + 3(0.9615 + 0.8621 + 0.6098 + 0.5)]$$

$$= (0.075) [1.4098 + 1.4706 + 3(2.9334)]$$

$$= (0.075) [1.4098 + 1.4706 + 8.8002]$$

$$= 0.8760$$

7. Evaluate $\int_0^6 1/(1+x) dx$ by using Simpson's three-eighth rule.

Solution:

Step 1. We are given that $f(x) = 1/(1+x)$. Interval length $(b-a) = (6-0) = 6$. So we divide 6 equal intervals with $h=1$. And tabulate the values as below

X	:	0	1	2	3	4	5	6
$y = 1/(1+x)$:		1	0.5	1/3	1/4	1/5	1/6	1/7

Step2. Write down the Simpson's three-eighth rule and put the respective values of y in that rule

$$\begin{aligned}
\int_3^5 f(x) dx &= (3h/8) [(y_0 + y_6) + 2(y_3) + 3(y_1 + y_2 + y_4 + y_5)] \\
&= (3h/8) [(\text{sum of the first and last ordinates}) + \\
&\quad + 2(\text{Sum of multiples of three, other than last ordinates}) \\
&\quad + 3(\text{sum of remaining ordinates})] \\
&= (3/8) [(1+1/7) + 2(1/4) + 3(0.5 + 1/3 + 1/5 + 1/6)] \\
&= \mathbf{1.9661}
\end{aligned}$$

8. Evaluate $\int_4^{5.2} \log_e x \, dx$ by using Simpson's three-eighth rule.

Solution:

Step 1. We are given that $f(x) = \log_e x$ Interval length $(b-a) = (5.2 - 4) = 1.2$. So we divide 6 equal intervals with $h = 0.2$. And tabulate the values as below

x	:	4	4.2	4.4	4.6	4.8	5.0	5.2
y	:	1.39	1.44	1.48	1.53	1.57	1.61	1.65

Step 2. Write down the Simpson's three-eighth rule and put the respective values of y in that rule

$$\begin{aligned}
\int_3^5 f(x) dx &= (3h/8) [(y_0 + y_6) + 2(y_3) + 3(y_1 + y_2 + y_4 + y_5)] \\
&= (3h/8) [(\text{sum of the first and last ordinates}) + \\
&\quad + 2(\text{Sum of multiples of three, other than last ordinates}) \\
&\quad + 3(\text{sum of remaining ordinates})] \\
&= (3 \times 0.2 / 8) [(1.39 + 1.65) + 2(1.53) + 3(1.44 + 1.48 + 1.57 + 1.61)] \\
&= (0.075) [3.04 + 3.06 + 3(6.1)] \\
&= \mathbf{1.83}
\end{aligned}$$

9. Evaluate $\int_0^9 x^2 \, dx$ by using Simpson's three-eighth rule, by dividing the range into nine equal parts and verify your answer with actual integration.

Solution :

Step 1. We are given that $f(x) = x^2$ Interval length $(b-a) = (9 - 0) = 9$.

So, we divide 9 equal intervals with $h=9/9 = 1$ (specified in the question itself), and tabulate the values as below

X	0	1	2	3	4
$Y = x^2$	0	1	4	9	16
x	5	6	7	8	9
Y	25	36	49	64	81

Step2. Write down the Simpson's three-eighth rule and put the respective values of y in that rule

$$\begin{aligned}
 \int_0^9 f(x) dx &= (3h/8) [(y_0 + y_9) + 2 (y_3 + y_6) + 3 (y_1 + y_2 + y_4 + y_5 + y_7 + y_8)] \\
 &= (3h/8) [(\text{sum of the first and last ordinates}) + \\
 &\quad + 2 (\text{Sum of multiples of three, other than last ordinates}) \\
 &\quad + 3 (\text{sum of remaining ordinates})] \\
 &= (3/8) [(0 + 81) + 2 (9 + 36) + 3 (1 + 4 + 16 + 25 + 49 + 64)] \\
 &= (.375) [81 + 90 + 477] \\
 &= 243
 \end{aligned}$$

By actual integration $\int_0^9 f(x) dx = \int_0^9 x^2 dx$

$$\begin{aligned}
 &= [(9^3/3) - (0^3/3)] \\
 &= [(729 / 3) + 0] \\
 &= 243
 \end{aligned}$$

NUMERICAL SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

Taylor Method

Suppose we want to find the numerical solution of the equation

$$\frac{dy}{dx} = f(x,y)$$

Given the initial condition $y(x_0) = y_0$

$y(x)$ can be expanded about the point $x = x_0$ in a Taylor's series as

Suppose the following table represents a set of values of x and y .

$x:$	x_0	x_1	x_2	x_3	x_n
$y:$	y_0	y_1	y_2	y_3	y_n

From the above values, we want to find the derivative of $y = f(x)$, passing through $(n+1)$ points, at a point closer to the starting value $x = x_0$

$$y(x) = y(x_0) + (x - x_0)^1 [y'(x)]_{x_0} / 1! + (x - x_0)^2 [y''(x)]_{x_0} / 2! + \dots$$

$$y(x) = y_0 + (x - x_0)^1 y'_0 / 1! + (x - x_0)^2 y''_0 / 2! + \dots$$

Putting $x = x_1 = x_0 + h$, we get

$$y_1 = y_0 + h y'_0 / 1! + h^2 y''_0 / 2! + h^3 y'''_0 / 3! + \dots$$

Now $y(x)$ can be expanded about the point $x = x_1$ in a Taylor's series as

$$y_2 = y_1 + h y'_1 / 1! + h^2 y''_1 / 2! + h^3 y'''_1 / 3! + \dots$$

Proceeding in the same way, we get

$$y_{n+1} = y_n + h y'_n / 1! + h^2 y''_n / 2! + h^3 y'''_n / 3! + \dots$$

1. Solve $dy/dx = x + y$, given $y(1) = 0$, and get $y(1.1)$, $y(1.2)$ by Taylor series method. Compare the result with the actual solution.

Solution :

We are given that $y(1) = 0 \Rightarrow x_0 = 1, y_0 = 0, h = 0.1$

$$\begin{array}{lcl} \text{Also } y' = x + y & & y'_0 = x_0 + y_0 = 1 + 0 = 1 \\ \Rightarrow y'' = 1 + y' & & y''_0 = y'_0 + 1 = 2 \\ \Rightarrow y''' = y'' & & y'''_0 = y''_0 + 2 = 2 \\ \Rightarrow y^{iv} = y''' & & y^{iv}_0 = 2. \end{array}$$

By Taylor series, we have

$$y_1 = y_0 + h y'_0 / 1! + h^2 y''_0 / 2! + h^3 y'''_0 / 3! + \dots$$

$$= y(1.1) = 0 + \frac{0.1}{1} (1) + \frac{(0.1)^2}{2} (2) + \frac{(0.1)^3}{6} (2) + \frac{(0.1)^4}{24} (2) + \frac{(0.1)^5}{120} (2) + \dots$$

$$= 0.1 + 0.01 + 0.00033 + 0.00000833 + 0.000000166$$

$$= 0.11033847$$

Consider that $x_0 = 1.1$, $h = 0.1$, and evaluate y_2

$$y_2 = y_1 + h y_1' / 1! + h^2 y_1'' / 2! + h^3 y_1''' / 3! + \dots$$

Calculate y_1' , y_1'' , y_1''' , y_1^{iv} , ... ; $x_1 = 1.1$, $y_1 = 0.11033847$

$$y_1' = x_1 + y_1 = 1.1 + 0.11033847 = 1.21033847$$

$$y_1'' = 1 + y_1' = 2.21033847$$

Using the above values :

$$y_2 = y(1.2) = 0.11033847 + 0.1(1.21033847) + (0.1)^2 (2.21033847) / 2 \\ + (0.1)^3 (2.21033847) / 6 + (0.1)^4 (2.21033847) / 24$$

$$= 0.11033847 + 0.121033847 + 2.21033847 (0.005) + \dots$$

$$= 0.2461077$$

The actual solution of $dy/dx = x + y$, is

$$y = -x - 1 + 2e^{x-1}$$

$$y(1.1) = -1.1 - 1 + 2e^{0.1} = 0.11034$$

$$y(1.2) = -1.2 - 1 + 2e^{0.2} = 0.2428$$

$$y(1.1) = 0.11033847$$

$$y(1.2) = 0.2461077$$

Actual values :

$$y(1.1) = 0.110341836$$

$$y(1.2) = 0.242805552.$$

2. Apply Taylor series method, find correct to four decimal places, the value of $y(0.1)$, given $dy/dx = x^2 + y^2$ and $y(0) = 1$.

Solution :

We are given that $y(0) = 1 \Rightarrow x_0 = 0, y_0 = 1, h = 0.1$

$x_1 = 0.1$, To find $y_1 = y(0.1)$ by using

following series :

$$\begin{array}{l|l} \text{Also } y' = x^2 + y^2 & y_0' = x_0^2 + y_0^2 = 0 + 1 = 1 \\ \Rightarrow y'' = 2x + 2y y' & y_0'' = 2x_0 + 2y_0 y_0' = 2 \\ \Rightarrow y''' = 2 + 2y y'' + 2(y')^2 & y_0''' = 2 + 2y_0 y_0'' + 2(y_0')^2 \\ \Rightarrow y^{iv} = 2y y''' + 2y' y'' + 4y' y'' & = 2 + 2(1)(2) + 2(1)^2 = 8 \\ & = 2yy''' + 6y'y'' \quad y_0^{iv} = 2(1)(8) + 6(1)(2) = 28 \end{array}$$

By Taylor series, we have

$$y_1 = y_0 + h y_0' / 1! + h^2 y_0'' / 2! + h^3 y_0''' / 3! + \dots$$

$$= y(0.1) = 1 + \frac{0.1}{1} (1) + \frac{(0.1)^2}{2} (2) + \frac{(0.1)^3}{6} (8) + \frac{(0.1)^4}{24} (28) + \dots$$

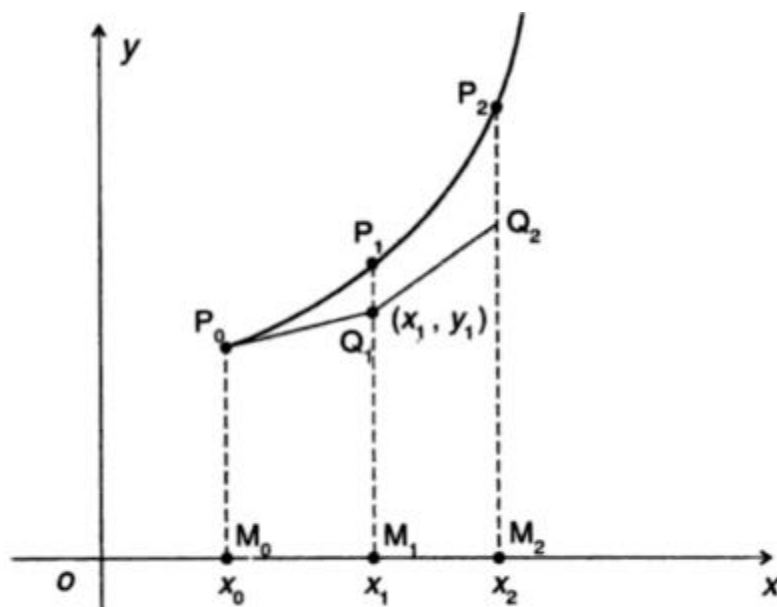
$$= 1 + 0.1 + 0.00133 + 0.00011$$

$$= 1.11144999$$

$$= 1.11145$$

Euler algorithm,

$$y_{n+1} = y_n + h y_n' = y_n + h f(x_n, y_n)$$



$$y(x+h) = y(x) + h f(x, y).$$

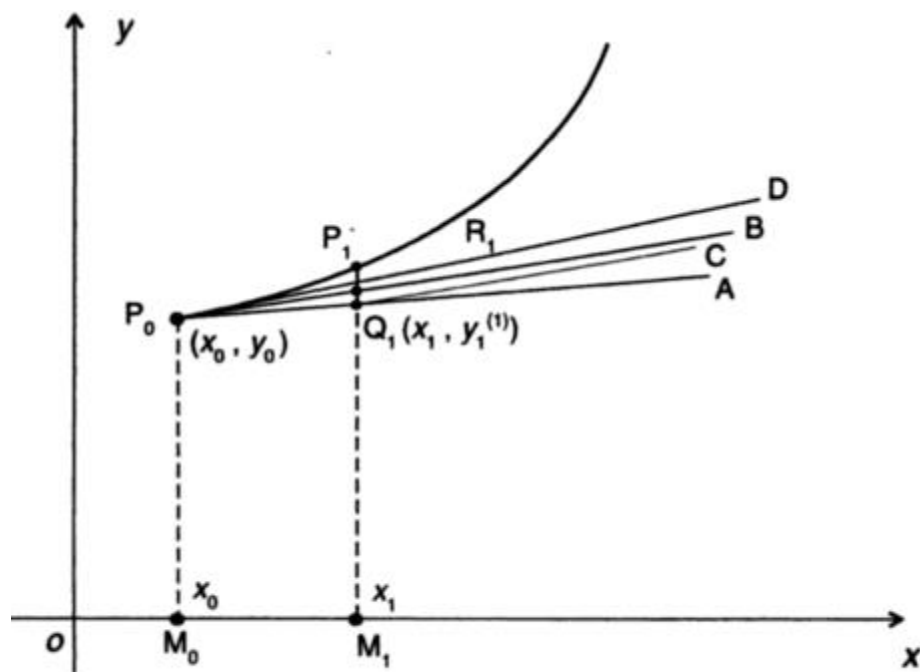
In this method, the actual curve is approximated by a sequence of short straight lines. As the intervals increase the straight line deviates much from the actual curve. Hence the accuracy cannot be obtained as the number of intervals increase.

$$Q_1 P_1 = \text{error at } x = x_1$$

$$= \frac{(x_1 - x_0)^2}{2!} y''(x_1, y_1) = \frac{h^2}{2} y''(x_1, y_1)$$

\therefore It is of order h^2 .

Improved Euler method



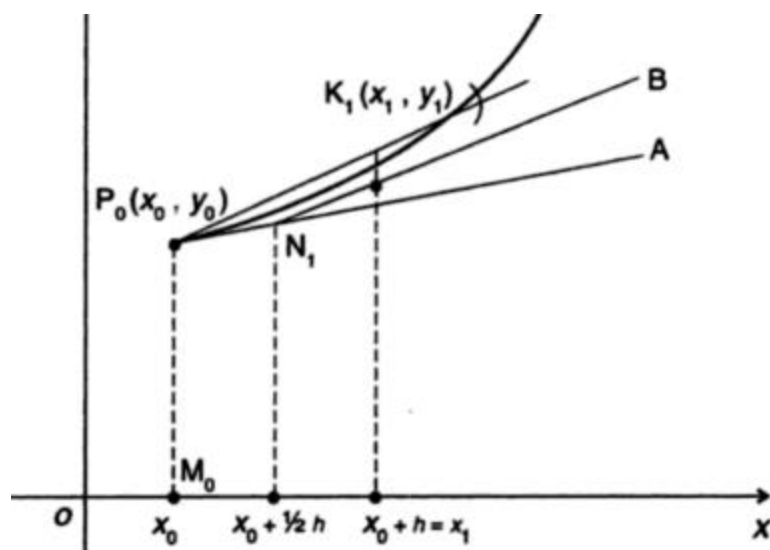
$$y_1 = y_0 + \frac{1}{2} h [f(x_0, y_0) + f(x_1, y_0 + h f(x_0, y_0))]$$

Writing generally,

$$y_{n+1} = y_n + \frac{1}{2} h [f(x_n, y_n) + f(x_n + h, y_n + h f(x_n, y_n))]$$

Note 1. The difference between Euler's method and improved Euler's method is that in the latter we take the average of the slopes at (x_0, y_0) and $(x_1, y_1^{(1)})$ instead of the slope at (x_0, y_0) in the former method.

Modified Euler method



$$\therefore y_1^{(1)} = y_0 + h \left[f \left(x_0 + \frac{1}{2} h, y_0 + \frac{1}{2} h f(x_0, y_0) \right) \right]$$

In general,

$$y_{n+1} = y_n + h \left[f \left(x_n + \frac{1}{2} h, y_n + \frac{1}{2} h f(x_n, y_n) \right) \right]$$

$$y(x+h) = y(x) + h \left[f \left(x + \frac{1}{2} h, y + \frac{1}{2} h f(x, y) \right) \right]$$

Example 1. Given $y' = -y$ and $y(0) = 1$, determine the values of y at $x = (0.01) (0.01) (0.04)$ by Euler method.

Solution. $y' = -y$ and $y(0) = 1$; $f(x, y) = -y$.

Here, $x_0 = 0$, $y_0 = 1$, $x_1 = 0.01$, $x_2 = 0.02$, $x_3 = 0.03$, $x_4 = 0.04$.

We have to find y_1, y_2, y_3, y_4 . Take $h = 0.01$.

By Euler algorithm,

$$y_{n+1} = y_n + h y_n' = y_n + h f(x_n, y_n) \quad \dots(1)$$

$$y_1 = y_0 + h f(x_0, y_0) = 1 + (0.01)(-1) = 1 - 0.01 = \mathbf{0.99}$$

$$\begin{aligned} y_2 &= y_1 + h y_1' = 0.99 + (0.01)(-y_1) \\ &= 0.99 + (0.01)(-0.99) \\ &= \mathbf{0.9801} \end{aligned}$$

$$\begin{aligned} y_3 &= y_2 + h f(x_2, y_2) = 0.9801 + (0.01)(-0.9801) \\ &= \mathbf{0.9703} \end{aligned}$$

$$\begin{aligned} y_4 &= y_3 + h f(x_3, y_3) = 0.9703 + (0.01)(-0.9703) \\ &= \mathbf{0.9606} \end{aligned}$$

Tabular values (step values) are:

x	0	0.01	0.02	0.03	0.04
y	1	0.9900	0.9801	0.9703	0.9606
Exact y	1	0.9900	0.9802	0.9704	0.9608

since, $y = e^{-x}$ is the exact solution.

Example 2. Using Euler's method, solve numerically the equation

$$y' = x + y, y(0) = 1, \text{ for } x = 0.0(0.2)(1.0)$$

check your answer with the exact solution.

Solution. Here $h = 0.2$, $f(x, y) = x + y$, $x_0 = 0$, $y_0 = 1$

$$x_1 = 0.2, x_2 = 0.4, x_3 = 0.6, x_4 = 0.8, x_5 = 1.0$$

By Euler algorithm,

$$y_1 = y_0 + h f(x_0, y_0) = y_0 + h [x_0 + y_0]$$

$$= 1 + (0.2)(0 + 1) = 1.2$$

$$y_2 = y_1 + h [x_1 + y_1] = 1.2 + (0.2)(0.2 + 1.2) = 1.48$$

$$y_3 = y_2 + h [x_2 + y_2]$$

$$= 1.48 + (0.2)(0.4 + 1.48) = 1.856$$

$$y_4 = 1.856 + (0.2)(0.6 + 1.856) = 2.3472$$

$$y_5 = 2.3472 + (0.2)(0.8 + 2.3472) = 2.94664$$

Exact solution is $y = 2e^x - x - 1$. Hence the tabular values are:

x	0	0.2	0.4	0.6	0.8	1.0
Euler y	1	1.2	1.48	1.856	2.3472	2.94664
Exact y	1	1.2428	1.5836	2.0442	2.6511	3.4366

$$y_{n+1} = y_n + h f(x_n, y_n) = y_n + h y_n' ; n = 0, 1, 2, \dots$$

This formula is called **Euler's algorithm**.

Improved Euler method

Slight change may be included in the above mentioned algorithm ., *i.e.*, we approximate the curve by the tangent and we get improved Euler formula ;

$$y_{n+1} = y_n + (1/2) h [f(x_n, y_n) + f(x_n + h, y_n + h f(x_n, y_n))]$$

This equation is called **improved Euler's method**.

Modified Euler method

Slight change may be included in the above mentioned improved Euler's method., *i.e.*, we averaged the slopes, whereas in modified Euler method, we will average the points. We get the formula for Modified Euler method , given by

$$y_{n+1} = y_n + h [f(x_n + h/2, y_n + (h/2) f(x_n, y_n))]$$

or
$$y(x+h) = y(x) + h [f(x+h/2, y+(h/2) f(x,y))]$$

This equation is called **Modified Euler's method**.

1: Solve $y' = -y$, and $y(0) = 1$, determine the values of y at $x = (0.01)(0.01)(0.04)$ by Euler's method.

Solution.

Step 1.

Calculate various values of x_i 's and respective y_i 's

We are given that $y' = -y$ and $y(0) = 1$; $f(x,y) = -y$;

$$x = (0.01)(0.01)(0.04) \Rightarrow x_0 = 0, y_0 = 1$$

$$x_1 = 0.01, x_1 = 0.01, x_2 = 0.02, x_3 = 0.03, x_4 = 0.04$$

Step 2. To find y_1, y_2, y_3, y_4 . Take $h = 0.01$ (Specified in the problem itself)

Write down the Euler formula ,

$$y_{n+1} = y_n + h f(x_n, y_n) = y_n + h y_n' ; n = 0, 1, 2, \dots$$

$$y_1 = y_0 + h f(x_0, y_0) = 1 + (0.01) (-1) = 1 - 0.01 = 0.99$$

$$y_2 = y_1 + h f(x_1, y_1) = 0.99 + (0.01) (-y_1) = 0.99 + (0.01) (-0.99) = 0.9801$$

$$y_3 = y_2 + h f(x_2, y_2) = 0.9801 + (0.01) (-0.9801) = 0.9703$$

$$y_4 = y_3 + h f(x_3, y_3) = 0.9703 + (0.01) (-0.9703) = 0.9606$$

Step 3. Flash the values in tabular form

X	0	0.01	0.02	0.03	0.04
Y	1	0.9900	0.9801	0.9703	0.9606
Exact y	1	0.9900	0.9802	0.9704	0.9608

Since, $y = e^{-x}$ is the exact solution.

Illustration 2: Solve $y' = x + y$ and $y(0) = 1$, determine the values of y at $x = 0.0(0.2)(1.0)$ by Euler's method. Compare answer with actual answer.

Solution.

We are given that $h = 0.2$, $f(x, y) = x + y$

$$y(0) = 1 \Rightarrow x_0 = 0, y_0 = 1$$

$$x = (0.0)(0.2)(1.0) \Rightarrow x_1 = 0.0, x_1 = 0.2, x_2 = 0.4, x_3 = 0.6, x_4 = 0.8, x_5 = 1$$

$$y_{n+1} = y_n + hf(x_n, y_n) = y_n + h y_n' ; n = 0, 1, 2, \dots$$

$$y_1 = y_0 + hf(x_0, y_0) = 1 + (0.2)(0+1) = 1 + 0.2 = 1.2$$

$$y_2 = y_1 + hf(x_1, y_1) = 1.2 + (0.2)(1) = 1.2 + (0.2)(0.2 + 1.2) = 1.48$$

$$y_3 = y_2 + hf(x_2, y_2) = 1.48 + (0.2)(0.4 + 1.48) = 1.856$$

$$y_4 = y_3 + hf(x_3, y_3) = 1.856 + (0.2)(0.6 + 1.856) = 2.3472$$

$$y_5 = y_4 + hf(x_4, y_4) = 2.3472 + (0.2)(0.8 + 2.3472) = 2.94664$$

Exact solution is $y = 2e^x - x - 1$. Flash the values in tabular form

X	0	0.2	0.4	0.6	0.8	1.0
Euler y	1	1.2	1.48	1.856	2.3472	2.94664
Exact y	1	1.2428	1.5836	2.0442	2.6511	3.4366

The value of y deviates from the exact values as x increases. Hence we require to use either Modified Euler or Improved Euler method for the above problem.

Illustration 3 : Solve numerically $y' = y + e^x$, $y(0) = 0$, for $x = 0.2, 0.4$ by Improved Euler's method.

Solution :

Step 1.

We are given that $y' = y + e^x$, $y(0) = 0$; $f(x, y) = y + e^x$.

$$y(0) = 0 \Rightarrow x_0 = 0, y_0 = 0$$

$$x = 0.2, 0.4 \Rightarrow x_1 = 0.0, x_1 = 0.2, x_2 = 0.4$$

Step 2. Write down the formula for Improved Euler method

$$y_{n+1} = y_n + (1/2) h [f(x_n, y_n) + f(x_n+h, y_n+h f(x_n, y_n))]$$

$$\begin{aligned} y_1 &= y_0 + (1/2) h [f(x_0, y_0) + f(x_0+h, y_0+h f(x_0, y_0))] \\ &= 0 + (0.5) (0.2) [\overline{y_0 + e^{x_0}} + y_0 + h(y_0 + e^{x_0}) + e^{x_0+h}] \\ &= 0.1 [0 + 1 + 0 + 0.2(0+1) + e^{0.2}] \\ &= 0.1 (1 + 0.2 + 1.2214) \end{aligned}$$

$$\Rightarrow y(0.2) = \mathbf{0.24214}$$

$$y_2 = y_1 + (1/2) h [f(x_1, y_1) + f(x_1+h, y_1+h f(x_1, y_1))]$$

$$\begin{aligned} \text{where } f(x_1, y_1) &= y_1 + e^{x_1} = 0.24214 + e^{0.2} = 1.46354 \\ y_1+h f(x_1, y_1) &= 0.24214 + (0.2)(1.46354) = 0.53485 \\ f(x_1+h, y_1+h f(x_1, y_1)) &= f(0.4, 0.53485) = 0.53485 + e^{0.4} = 2.02667 \end{aligned}$$

Substituting the above values, we get

$$\begin{aligned} y(0.4) &= 0.24214 + (0.5)(0.2)[1.46354 + 2.02667] \\ &= \mathbf{0.59116} \end{aligned}$$

Tabulate the values and it given below:

x	0	0.2	0.4
y	0	0.24214	0.59116

Illustration 4. Compute y at $x = 0.25$ by Modified Euler method given $y' = 2xy$,

$$y(0) = 1.$$

Solution :

Step 1.

We are given that $f(x, y) = 2xy$;

$$y(0) = 1 \Rightarrow x_0 = 0, y_0 = 1$$

$$h = 0.25 \Rightarrow x_1 = 0 + 0.25 = 0.25$$

Step 2. Write down the Modified Euler formula

$$\begin{aligned} y_{n+1} &= y_n + h [f(x_n+h/2, y_n+(h/2) f(x_n, y_n))] \\ &\Rightarrow y_1 = y_0 + h [f(x_0+h/2, y_0+(h/2) f(x_0, y_0))] \\ &= 1 + (0.25) [f(0.125, 1)] \\ &= 1 + (0.25) [2 \times (0.125) \times 1] \\ &= \mathbf{1.0625}. \end{aligned}$$

Runge-Kutta Method

Suppose we want to find the numerical solution of the equation

$$\frac{dy}{dx} = f(x,y)$$

Given the initial condition $y(x_0) = y_0$ (1)

Calculate

$$\begin{aligned} k_1 &= h f(x_0, y_0) \\ k_2 &= h f(x_0 + (1/2)h, y_0 + (1/2)k_1) \\ \text{and } \Delta y &= k_2, \text{ where } h = \Delta x \end{aligned}$$

The above mentioned algorithm is **Second order Runge-Kutta Algorithm**

Calculate

$$\begin{aligned} k_1 &= h f(x_0, y_0) \\ k_2 &= h f(x_0 + (1/2)h, y_0 + (1/2)k_1) \\ k_3 &= h f(x_0 + (1/2)h, y_0 + (1/2)k_2) \\ \text{and } \Delta y &= (1/6)[k_1 + 4k_2 + k_3] \end{aligned}$$

The above mentioned algorithm is **Third order Runge-Kutta Algorithm**

Calculate

$$\begin{aligned} k_1 &= h f(x_0, y_0) \\ k_2 &= h f(x_0 + (1/2)h, y_0 + (1/2)k_1) \\ k_3 &= h f(x_0 + (1/2)h, y_0 + (1/2)k_2) \\ k_4 &= h f(x_0 + h, y_0 + k_3) \\ \text{and } \Delta y &= (1/6)[k_1 + 2k_2 + 2k_3 + k_4] \\ y(x+h) &= y(x) + \Delta y \end{aligned}$$

The above mentioned algorithm is **Fourth order Runge-Kutta Algorithm**

Where $\Delta x = h$.

Calculate

$$y_1 = y_0 + \Delta y$$

Now starting from (x_1, y_1) and repeating the above process, we get (x_2, y_2) etc.

Note 1: In second order Runge-Kutta method

$$\begin{aligned} \Delta y_0 &= k_2 = h f(x_0 + (1/2)h, y_0 + (1/2)k_1) \\ \Delta y_0 &= k_2 = h f(x_0 + (1/2)h, y_0 + (1/2)h f(x_0, y_0)) \end{aligned}$$

Therefore

$$y_1 = y_0 + h [f(x_0 + h/2, y_0 + (h/2) f(x_0, y_0))]$$

This is equivalent to modified Euler Method.

Hence, the **Runge-Kutta method of second order is nothing but the Modified Euler Method.**

Note 2: if $f(x,y) = f(x)$, i.e., $f(x,y)$ is only depending on a function x alone, then the fourth order Runge-Kutta method reduces to **Simpson's one third rule**

Note 3. In all the three methods the values of k_1, k_2, k_3 are same. Therefore, no need to calculate the constants while doing by all the three method.

Illustration 1. Apply the fourth order Runge-Kutta method to find $t(0.2)$ given that $y' = x+y$, $y(0) = 1$.

Solution:

Step 1. We are given that $y' = x+y$, $y(0) = 1 \Rightarrow f(x,y) = x + y$, $x_0 = 0$, $y_0 = 1$
Since h is not specified in the question, we take $h = 0.1$; $x_1 = 0.1$, $x_2 = 0.2$

Step 2. We have to find various constants in fourth order Runge-Kutta method

$$\begin{aligned} k_1 &= h f(x_0, y_0) = (0.1)(x_0 + y_0) = (0.1)(0+1) = 0.1 \\ k_2 &= h f(x_0 + (1/2)h, y_0 + (1/2)k_1) = (0.1)f(0.05, 1.05) \\ &= 0.1(0.05+1.05) = 0.11 \\ k_3 &= h f(x_0 + (1/2)h, y_0 + (1/2)k_2) = (0.1)f(0.05, 1.055) \\ &= 0.1(0.05+1.055) = 0.1105 \\ k_4 &= h f(x_0+h, y_0+k_3) = 0.1f(0.1, 1.1105) = 0.12105 \end{aligned}$$

$$\begin{aligned} \text{and } \Delta y &= (1/6)[k_1 + 2k_2 + 2k_3 + k_4] \\ &= (0.16666)(0.1+0.22+0.2210+0.12105) \\ &= 0.110342 \end{aligned}$$

$$\begin{aligned} y_1 &= y_0 + \Delta y \\ y(0.1) = y_1 &= y_0 + \Delta y = 1.110342 \end{aligned}$$

Step 3. Now starting from (x_1, y_1) and repeating the above process, we get (x_2, y_2) .
Again apply Runge-Kutta method replacing (x_0, y_0) by (x_1, y_1) .

$$\begin{aligned} k_1 &= h f(x_1, y_1) = (0.1)(x_1 + y_1) = (0.1)(0.1 + 1.110342) = 0.1 \\ k_2 &= h f(x_1 + (1/2)h, y_1 + (1/2)k_1) = (0.1)f(0.15, 1.170859) \\ &= 0.1(0.15+1.170859) = 0.1320859 \\ k_3 &= h f(x_1 + (1/2)h, y_1 + (1/2)k_2) = (0.1)f(0.15, 1.1763848) \\ &= 0.1(0.15+1.1763848) = 0.13263848 \\ k_4 &= h f(x_1+h, y_1+k_3) = 0.1f(0.2, 1.24298048) = 0.144298048 \\ \text{and } \Delta y &= (1/6)[k_1 + 2k_2 + 2k_3 + k_4] \\ &= (0.16666)(0.1+0.2641718+0.26527696+0.144298048) \end{aligned}$$

$$\begin{aligned} y(0.2) &= y_1 + \Delta y = 1.110342 + (0.166666)(0.7947810008) \\ y(0.2) &= 1.2428055 \Rightarrow \mathbf{1.2428} \text{ (Correct to four decimal places).} \end{aligned}$$

Illustration 2. Obtain the values of y at $x = 0.1, 0.2$ using R.K. method of (i) second order (ii) third order and (iii) fourth order for the differential equation $y' = -y$, given $y(0) = 1$.

Solution:

Step 1. We are given that $y' = -y$, $y(0) = 1 \Rightarrow f(x, y) = -y$, $x_0 = 0$, $y_0 = 1$
 Since h is clearly specified in the question, we take $h = 0.1$; $x_1 = 0.1$, $x_2 = 0.2$

Step 2. (i) We have to find various constants in **Second order** Runge-Kutta method

$$k_1 = h f(x_0, y_0) = (0.1)(-y_0) = (0.1)(-1) = -0.1$$

$$\begin{aligned} k_2 &= h f(x_0 + (1/2)h, y_0 + (1/2)k_1) = (0.1)f(0.05, .95) \\ &= 0.1(-0.95) = -0.095 = \Delta y \end{aligned}$$

$$y_1 = y_0 + \Delta y$$

$$y(0.1) = y_1 = y_0 + \Delta y = 1 - 0.095 = 0.905$$

Now starting from (x_1, y_1) i.e., $(.01, 0.905)$ and repeating the above process, we get (x_2, y_2) . Again apply Runge-Kutta method replacing (x_0, y_0) by (x_1, y_1) .

$$k_1 = h f(x_1, y_1) = (0.1)(-y_1) = (0.1)(-0.905) = -0.0905$$

$$\begin{aligned} k_2 &= h f(x_1 + (1/2)h, y_1 + (1/2)k_1) = (0.1)f(0.15, 0.85975) \\ &= 0.1(-0.85975) = -0.085975 = \Delta y \end{aligned}$$

$$y_1 = y_1 + \Delta y$$

$$y(0.2) = y_1 = y_1 + \Delta y = \mathbf{0.819025}$$

Step 3. (i) We have to find various constants in **Third order** Runge-Kutta method

$$k_1 = h f(x_0, y_0) = (0.1)(-y_0) = (0.1)(-1) = -0.1$$

$$\begin{aligned} k_2 &= h f(x_0 + (1/2)h, y_0 + (1/2)k_1) = (0.1)f(0.05, 0.95) \\ &= 0.1(-0.95) = -0.095 \end{aligned}$$

$$k_3 = h f(x_0 + (1/2)h, y_0 + (1/2)k_2) = (0.1)f(0.1, 0.9) = (-0.09)$$

$$\Delta y = (1/6)[k_1 + 4k_2 + k_3]$$

$$y_1 = y_0 + \Delta y$$

$$y(0.1) = y_1 = y_0 + \Delta y = 1 - 0.09 = 0.91$$

Now starting from (x_1, y_1) i.e., $(.01, 0.905)$ and repeating the above process, we get (x_2, y_2) . Again apply Runge-Kutta method replacing (x_0, y_0) by (x_1, y_1) .

$$k_1 = h f(x_1, y_1) = (0.1)(-y_1) = (0.1)(-0.91) = -0.091$$

$$\begin{aligned} k_2 &= h f(x_1 + (1/2)h, y_1 + (1/2)k_1) = (0.1)f(0.15, 0.865) \\ &= 0.1(-0.865) = -0.0865 \end{aligned}$$

$$k_3 = h f(x_0 + (1/2)h, y_0 + (1/2)k_2) = (0.1)f(0.2, 0.828) = -0.0828$$

$$\Delta y = (1/6)[k_1 + 4k_2 + k_3]$$

$$y_2 = y_1 + \Delta y$$

$$y(0.2) = y_2 = y_1 + \Delta y = 0.91 + (0.16666)(-0.091 - 0.346 - 0.0828)$$

$$= \mathbf{0.823366}$$

Step 4. (i) We have to find various constants in **fourth order** Runge-Kutta method

$$k_1 = hf(x_0, y_0) = (0.1)(-y_0) = (0.1)(-1) = -0.1$$

$$k_2 = hf(x_0 + (1/2)h, y_0 + (1/2)k_1) = (0.1)f(0.05, 0.95)$$

$$= 0.1(-0.95) = -0.095$$

$$k_3 = hf(x_0 + (1/2)h, y_0 + (1/2)k_2) = (0.1)f(0.1, 0.9525) = -0.09525$$

$$k_4 = hf(x_0 + h, y_0 + k_3) = 0.1f(0.1, 0.90475) = -0.090475$$

$$\text{and } \Delta y = (1/6)[k_1 + 2k_2 + 2k_3 + k_4]$$

$$= (0.16666)(-0.095 - 0.19 - 0.1905 - 0.090475)$$

$$= -0.0951625$$

$$y_1 = y_0 + \Delta y$$

$$y(0.1) = y_1 = y_0 + \Delta y = 1 + (0.0951625)$$

$$= \mathbf{0.9048375}$$

Now starting from (x_1, y_1) i.e., $(0.1, 0.9048375)$ and repeating the above process, we get (x_2, y_2) . Again apply Runge-Kutta method replacing (x_0, y_0) by (x_1, y_1) .

$$k_1 = hf(x_1, y_1) = (0.1)(-y_1) = (0.1)(-0.91) = -0.09048375$$

$$k_2 = hf(x_1 + (1/2)h, y_1 + (1/2)k_1) = (0.1)f(0.15, 0.8595956)$$

$$= 0.1(-0.8595956) = -0.08595956$$

$$k_3 = hf(x_1 + (1/2)h, y_1 + (1/2)k_2) = (0.1)f(0.15, 0.8618577) = -0.08618577$$

$$k_4 = hf(x_1 + h, y_1 + k_3) = 0.1f(0.2, 0.8186517) = -0.0818517$$

$$\text{and } \Delta y = (1/6)[k_1 + 2k_2 + 2k_3 + k_4]$$

$$= (0.16666)(-0.09048375 + 2(-0.08595956) + 2(-0.08618577) - 0.0818517)$$

$$= -0.086106607$$

$$y_2 = y_1 + \Delta y$$

$$y(0.2) = y_2 = y_1 + \Delta y = 0.9048375 + (-0.086106607)$$

$$= \mathbf{0.81873089}$$

Tabular values are

x	Second order	Third order	Fourth order	Exact Value
0.1	0.905	0.91	0.9048375	0.904837418
0.2	0.819025	0.823366	0.81873089	0.818730753

$$y_{n+1} = y_{n-3} + \frac{4}{3} h (2y'_n - y'_{n-1} + 2y'_{n-2}) + O(h^5)$$

$$y_{n+1} = y_{n-1} + \frac{1}{3} h (y'_{n-1} + 4y'_n + y'_{n+1}) + O(h^5).$$

Milne's Predictor-Corrector Method

Consider the differential equation $\frac{dy}{dx} = f(x, y); y(x_0) = y_0$

Milne's predictor and corrector formula is given by

$$y_{4,p} = y_0 + \frac{4h}{3} (2f_1 - f_2 + 2f_3) \text{ --- (1) } \rightarrow \text{Predictor formula}$$

$$y_{4,c}^{(r+1)} = y_2 + \frac{h}{3} (f_2 + 4f_3 + f_4^{(r)}) \text{ --- (2) } \rightarrow \text{Corrector formula}$$

where $f_1 = f(x_1, y_1)$, $f_2 = f(x_2, y_2)$, $f_3 = f(x_3, y_3)$, $f_4^{(r)} = f(x_4, y_4^{(r)})$

Note : $f_4^{(0)} = f(x_4, y_4^{(0)})$ where $y_4^{(0)} = y_{4,p}$ & $y_4^{(r)} = y_{4,c}^{(r)}$ for $r \neq 0$

Problem(1)

Find $y(2)$ if $y(x)$ is the solution of $\frac{dy}{dx} = \frac{x+y}{2}$, given that $y(0)=2$,

$y(0.5)=2.636$, $y(1)=3.595$, $y(1.5)=4.968$ using Milne's Predictor-Corrector

method correct to four decimal places.

Soln: Given data $f(x, y) = \frac{x+y}{2}$, $h=0.5$

$x_0 = 0$	$x_1 = 0.5$	$x_2 = 1.0$	$x_3 = 1.5$	$x_4 = 2.0$
$y_0 = 2$	$y_1 = 2.636$	$y_2 = 3.595$	$y_3 = 4.968$	$y_4 = ?$

Milne's Predictor formula is given by

$$y_{4,p} = y_0 + \frac{4h}{3} (2f_1 - f_2 + 2f_3) \text{ --- (1)}$$

x_i	y_i	$f_i = f(x_i, y_i) = \frac{x_i + y_i}{2}$
$x_1 = 0.5$	$y_1 = 2.636$	$f_1 = \frac{x_1 + y_1}{2} = \frac{0.5 + 2.636}{2} = 1.568$
$x_2 = 1$	$y_2 = 3.595$	$f_2 = \frac{x_2 + y_2}{2} = \frac{1 + 3.595}{2} = 2.2975$
$x_3 = 1.5$	$y_3 = 4.968$	$f_3 = \frac{x_3 + y_3}{2} = \frac{1.5 + 4.968}{2} = 3.234$

Substituting all the values in eqn(1) we get,

$$y_{4,p} = 2 + \frac{4(0.5)}{3} \{2(1.568) - 2.2975 + 2(3.234)\} = 6.871$$

Milne's Corrector formula is given by

$$y_{4,c}^{(r+1)} = y_2 + \frac{h}{3} (f_2 + 4f_3 + f_4^{(r)}) \text{----- (2) where } f_4^{(r)} = f(x_4, y_4^{(r)})$$

First improvement: Put $r=0$ in eqn(2)

$$y_{4,c}^{(1)} = y_2 + \frac{h}{3} (f_2 + 4f_3 + f_4^{(0)}), \text{ Where}$$

$$f_4^{(0)} = f(x_4, y_4^{(0)}) = f(x_4, y_{4,p}) = \frac{x_4 + y_{4,p}}{2} = \frac{2 + 6.871}{2} = 4.4355$$

$$\therefore y_{4,c}^{(1)} = 3.595 + \frac{0.5}{3} (2.2975 + 4(3.234) + 4.4355) = 6.8731$$

Second improvement: Put $r=1$ in eqn(2)

$$y_{4,c}^{(2)} = y_2 + \frac{h}{3}(f_2 + 4f_3 + f_4^{(1)})$$

where

$$f_4^{(1)} = f(x_4, y_4^{(1)}) = f(x_4, y_{4,c}^{(1)}) = \frac{x_4 + y_{4,c}^{(1)}}{2}$$

$$= \frac{2 + 6.8731}{2} = 4.4365$$

$$\therefore y_{4,c}^{(2)} = 3.595 + \frac{0.5}{3}(2.2975 + 4(3.234) + 4.4365)$$

Third improvement: Put $r=2$ in eqn(2)

$$y_{4,c}^{(3)} = y_2 + \frac{h}{3}(f_2 + 4f_3 + f_4^{(2)})$$

where

$$f_4^{(2)} = f(x_4, y_4^{(2)}) = f(x_4, y_{4,c}^{(2)}) = \frac{x_4 + y_{4,c}^{(2)}}{2} = \frac{2 + 6.8733}{2} = 4.4366$$

$$\therefore y_{4,c}^{(3)} = 3.595 + \frac{0.5}{3}(2.2975 + 4(3.234) + 4.4366) = 6.8733$$

Since $y_{4,c}^{(2)}$ & $y_{4,c}^{(3)}$ are the same up to four decimal places

$y(2)=6.8733$

Given $\frac{dy}{dx} = \frac{1}{2}(1+x^2)y^2$ and $y(0)=1$, $y(0.1)=1.06$, $y(0.2)=1.12$, $y(0.3)=1.21$. Evaluate $y(0.4)$ by Milne's Predictor-Corrector method.

Soln : Given data : $f(x, y) = \frac{1}{2}(1+x^2)y^2$, $h = 0.1$

$x_0 = 0$	$x_1 = 0.1$	$x_2 = 0.2$	$x_3 = 0.3$	$x_4 = 0.4$
$y_0 = 1$	$y_1 = 1.06$	$y_2 = 1.12$	$y_3 = 1.21$	$y_4 = ?$

Milne's Predictor formula is given by

$$y_{4,p} = y_0 + \frac{4h}{3}(2f_1 - f_2 + 2f_3) \text{ -----(1)}$$

x_i	y_i	$f_i = f(x_i, y_i) = \frac{1}{2}(1 + x_i^2)y_i^2$
$x_1 = 0.1$	$y_1 = 1.06$	$f_1 = \frac{1}{2}(1 + x_1^2)y_1^2 = 0.5674$
$x_2 = 0.2$	$y_2 = 1.12$	$f_2 = \frac{1}{2}(1 + x_2^2)y_2^2 = 0.6522$
$x_3 = 0.3$	$y_3 = 1.21$	$f_3 = \frac{1}{2}(1 + x_3^2)y_3^2 = 0.7979$

Substituting all the values in eqn(1) we get,

$$y_{4,p} = 1 + \frac{4(0.1)}{3} \{2(0.5674) - 0.6522 + 2(0.7979)\} = 1.2771$$

Milne's Corrector formula is given by

$$y_{4,c}^{(r+1)} = y_2 + \frac{h}{3} \left(f_2 + 4f_3 + f_4^{(r)} \right) \text{----- (2)}$$

$$\text{where } f_4^{(r)} = f(x_4, y_4^{(r)}), \quad f_4^{(0)} = y_{4,p} \quad \text{and} \quad f_4^{(r)} = y_{4,c}^{(r)}, r \neq 0$$

First improvement: Put $r=0$ in eqn(2)

$$y_{4,c}^{(1)} = y_2 + \frac{h}{3} \left(f_2 + 4f_3 + f_4^{(0)} \right), \quad \text{Where}$$

$$f_4^{(0)} = f(x_4, y_4^{(0)}) = \frac{1}{2}(1 + x_4^2)y_{4,p}^2 = \frac{1}{2}(1 + (0.4)^2)(1.2771)^2 = 0.9459$$

$$\therefore y_{4,c}^{(1)} = 1.12 + \frac{0.1}{3} (0.6522 + 4(0.7979) + 0.9459) = 1.2796$$

Second improvement: Put $r=1$ in eqn(2)

$$y_{4,c}^{(2)} = y_2 + \frac{h}{3} \left(f_2 + 4f_3 + f_4^{(1)} \right), \text{ Where}$$

$$f_4^{(1)} = f(x_4, y_4^{(1)}) = \frac{1}{2} \left(1 + x_4^2 \right) y_{4,c}^{(1)} = \frac{1}{2} \left(1 + (0.4)^2 \right) (1.2796)^2 = 0.9496$$

$$\therefore y_{4,c}^{(2)} = 1.12 + \frac{0.1}{3} (0.6522 + 4(0.7979) + 0.9496) = 1.2797$$

$$\text{Similarly } y_{4,c}^{(3)} = 1.2797$$

Since $y_{4,c}^{(2)}$ & $y_{4,c}^{(3)}$ are the same up to four decimal places

$$\mathbf{y(0.4)=1.2797}$$

SPH5107 – COMPUTER PROGRAMMING

Unit - 4

Overview of C

A brief history

C is a programming language developed at “AT & T's Bell Laboratories” of USA in 1972. It was written by Dennis Ritchie (Fig 2).



Fig 2. Dennis Ritchie

The programming language C was first given by Kernighan and Ritchie, in a classic book called “The C Programming Language, 1st edition”. For several years the book “The C Programming Language, 1st edition” was the standard on the C programming. In 1983 a committee was formed by the American National Standards Institute (ANSI) to develop a modern definition for the programming language C. In 1988 they delivered the final standard definition ANSI C.

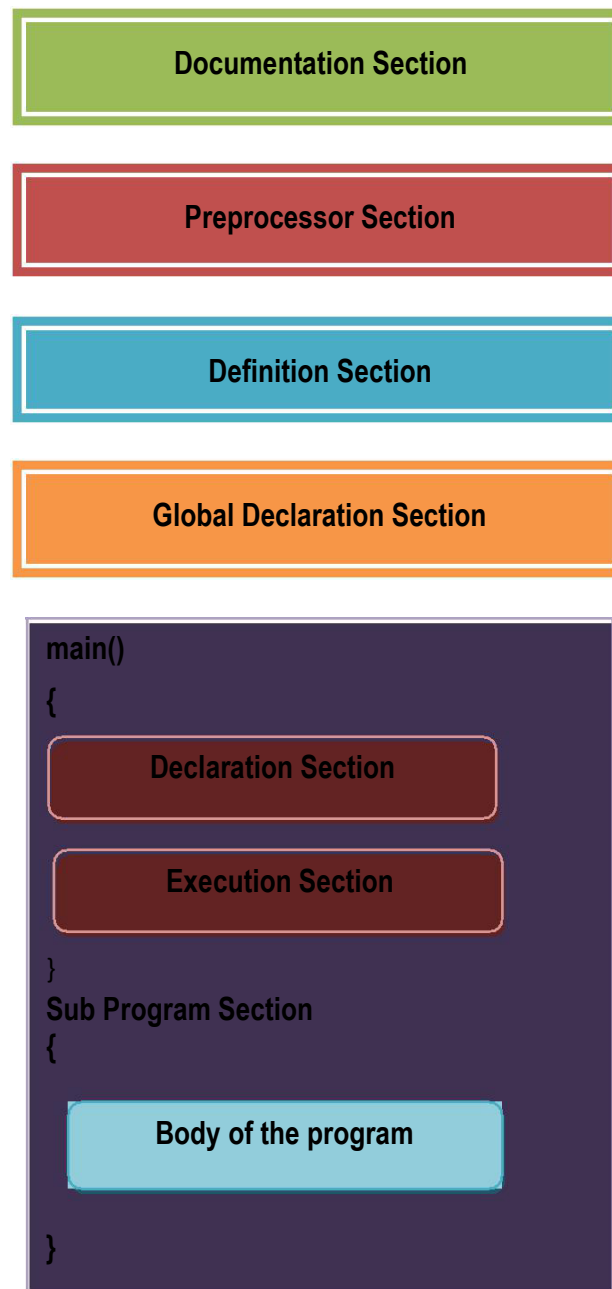
Features of C

- Portability
- Modularity
- Extensible
- Speed
- Mid-level programming language
- Flexibility
- Rich Library

Advantages of C

1. A C program written in one computer can easily run on another computer without making any change
2. It has variety of data types and powerful operators
3. A C program is a collection of functions supported by the C library. So we can easily add our own functions to C library. Hence we can extend any existing C program according to the need of our applications
4. Since C is a structured language, we can split any big problem into several sub modules. Collection of these modules makes up a complete program. This modular concept makes the testing and debugging easier

Structure of a C program



Documentation Section:

- It consist of a set of comment lines
- The comment lines begins with /* and ends with */ or a single set of // in the beginning of the line
- These lines are not executable
- Comments are very helpful in identifying the program features

Preprocessor Section:

- It is used to link system library files, for defining the macros and for defining the conditional inclusion
- Eg: #include<stdio.h>, #include<conio.h>, #define MAX 100, etc.,

Global Declaration Section:

- The variables that are used in more than one function throughout the program are called global variables
- Should be declared outside of all the functions i.e., before main().

main():

Every 'C' program must have one main() function, which specifies the starting of a 'C' program. It contains the following two parts

Declaration Part:

- This part is used to declare the entire variables that are used in the executable part of the program and these are called local variables

Execution Part:

- It contains at least one valid C Statement.
- The Execution of a program begins with opening brace '{' and ends with closing brace '}'
- The closing brace of the main function is the logical end of the program

Sub Program section:

- Sub programs are basically functions are written by the user (user defined functions)
- They may be written before or after a main () function and called within main () function
- This is optional to the programmer

Constraints while writing a C program

- All statements in 'C' program should be written in lower case letters. Uppercase letters are only used for symbolic constants
- Blank space may be inserted between the words. Should not be used while declaring a variable, keyword, constant and function
- The program statements can be written anywhere between the two braces following the declaration part
- All the statements should end with a semicolon (;)

Example Program

/* addition.c - To find the average of two numbers and print them out together with their average */

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

```
void main( )
```

```
{
```

```
    int first, second;
```

```
    float avg;
```

```
    printf("Enter two numbers: ");
```

```
    scanf("%d %d", &first, &second);
```

```
    printf("The two numbers are: %d, %d", first, second);
```

```
    avg = (first + second)/2;
```

```
    printf("Their average is %f", avg);
```

```
}
```

Compilation and Execution of C program

1. Creating the program
2. Compiling the Program
3. Linking the Program with system library
4. Executing the program

Creating the program:

- Type the program and edit it in standard 'C' editor and save the program with .c as an extension.
- This is the source program

Compiling (Alt + F9) the Program:

- This is the process of converting the high level language program to Machine level Language (Equivalent machine instruction) -> Compiler does it!

- Errors will be reported if there is any, after the compilation
- Otherwise the program will be converted into an object file (.obj file) as a result of the compilation
- After error correction the program has to be compiled again

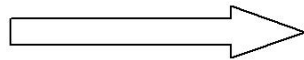
Linking the program with system Library:

- Before executing a c program, it has to be linked with the included header files and other system libraries -> Done by the Linker

Executing the Program:

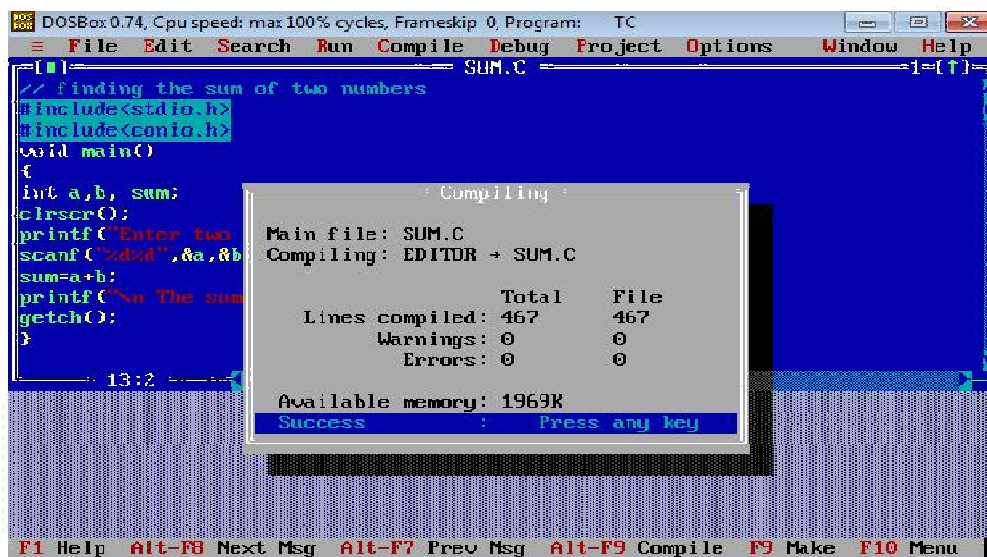
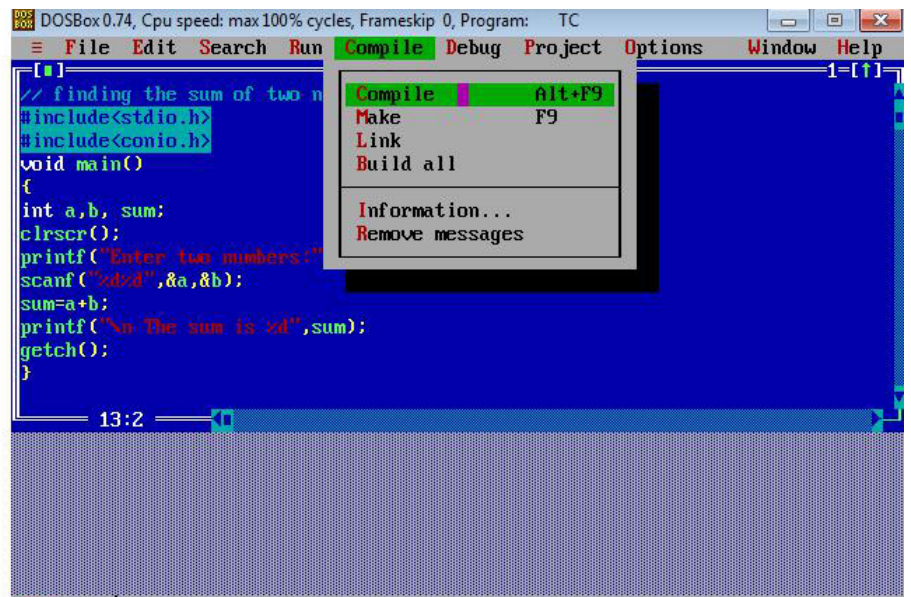
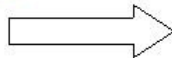
- This is the process of running (Ctrl + F9) and testing the program with sample data. If there are any run time errors, then they will be reported.

Creating the program

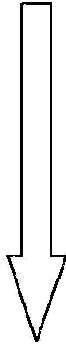
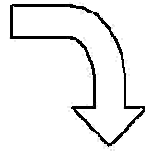


```
// finding the sum of two numbers
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b, sum;
    clrscr();
    printf("Enter two numbers:");
    scanf("%d%d",&a,&b);
    sum=a+b;
    printf("\n The sum is %d",sum);
    getch();
}
```

Compilation and Linking



Executing



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
File Edit Search Run Compile Debug Project Options Window Help
// finding the sum of
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b, sum;
    clrscr();
    printf("Enter two numbers:");
    scanf("%d%d",&a,&b);
    sum=a+b;
    printf("\n The sum is %d",sum);
    getch();
}
13:12
F1 Help | Make and run the current program
```

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter two numbers:5 3

The sum is 8_
```

The above illustration provides a lucid description of how to compile and execute a C program.

C Tokens

C tokens, Identifiers and Keywords are the basic elements of a C program. C tokens are the basic buildings blocks in C. Smallest individual units in a C program are the C tokens. C tokens are of six types. They are,

1. Keywords (eg: int, while),
2. Identifiers (eg: main, total),
3. Constants (eg: 10, 20),
4. Strings (eg: "total", "hello"),
5. Special symbols (eg: (), {}),
6. Operators (eg: +, /, -, *)

1. Keywords

Keywords are those words whose meaning is already defined by Compiler. They cannot be used as **Variable Names**. There are **32 Keywords** in C. C Keywords are also called as **Reserved words**. There are 32 keywords in C. They are given below:

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

2. Identifiers

Identifiers are the names given to various program elements such as variables , arrays & functions. Basically identifiers are the sequences of alphabets or digits.

Rules for forming identifier name

- The first character must be an alphabet (uppercase or lowercase) or an underscore.
- All succeeding characters must be letters or digits.
- No space and special symbols are allowed between the identifiers.
- No two successive underscores are allowed.
- Keywords shouldn't be used as identifiers.

3. Constants

The constants refer to fixed values that the program may not change or modify during its execution. Constants can be of any of the basic data types like an integer constant, a floating constant and a character constant. There is also a special type of constant called enumeration constant.

Eg:

Integer Constants- 45, 215u

Floating Constants- 3.14, 4513E-5L

Character Constants- \t, \n

4. Strings

A string in C is actually a one-dimensional array of characters which is terminated by a null character '\0'.

Eg:

```
char str = {'S', 'A', 'T', 'H', 'Y', 'A', 'B', 'A', 'M', 'A'}
```

5. Special Symbols

The symbols other than alphabets, digits and white spaces for example - `[] () {} , ; : * ... = #` are the special symbols.

6. Operators

An Operator is a symbol that specifies an operation to be performed on the operands. The data items that operators act upon are called operands. Operators which require two operands are called Binary operators. Operators which require one operand are called Unary Operators.

Types of Operators

Depending upon their operation they are classified as

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators

5. Increment and Decrement Operators
6. Conditional Operators
7. Bitwise Operators
8. Sizeof() Operators

Arithmetic Operators

Arithmetic Operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus.

S.NO	Operators	Operation	Example
1	+	Addition	A+B
2	-	Subtraction	A-B
3	*	multiplication	A*B
4	/	Division	A/B
5	%	Modulus	A%B

Rules For Arithmetic Operators

1. C allows only one variable on left hand side of = eg. $c=a*b$ is legal, but $a*b=c$ is not legal.
2. Arithmetic operations are performed on the ASCII values of the characters and not on characters themselves
3. Operators must be explicitly written.
4. Operation between same type of data yields same type of data, but operation between integer and float yields a float result.

Example Program

```
#include <stdio.h>
int main()
{
int m=40,n=20, add,sub,mul,div,mod;
add = m+n;
sub = m-n;
mul = m*n;
div = m/n;
mod = m%n;
```

```

printf("Addition of m, n is : %d\n", add);
printf("Subtraction of m, n is : %d\n", sub);
printf("Multiplication of m, n is : %d\n",
mul); printf("Division of m, n is : %d\n",
div); printf("Modulus of m, n is : %d\n", mod);
}

```

Output

Addition of m, n is : 60
 Subtraction of m, n is : 20
 Multiplication of m, n is : 800
 Division of m, n is : 2
 Modulus of m, n is : 0

Relational Operators

Relational Operators are used to compare two or more operands. Operands may be variables, constants or expression

S.NO	Operators	Operation	Example
1	>	is greater than	m > n
2	<	is less than	m < n
3	>=	is greater than or equal to	m >= n
4	<=	is less than or equal to	m <= n
5	==	is equal to	m == n
6	!=	is not equal to	m != n

Example Program

```

#include
<stdio.h> int
main()

```



```

{
int m=40,n=20;
if (m == n)
{
printf("m and n are equal");
}
else
{
printf("m and n are not equal");
}
}

```

Output

m and n are not equal

Logical Operators

Logical Operators are used to combine the results of two or more conditions. It is also used to test more than one condition and make decision.

S.NO	Operators	Operation	Example	Description
1	&&	logical AND	(m>5)&&(n<5)	It returns true when both conditions are true
2		logical OR	(m>=10) (n>=10)	It returns true when at-least one of the condition is true
3	!	logical NOT	!((m>5)&&(n<5))	It reverses the state of the operand “((m>5) && (n<5))” If “((m>5) && (n<5))” is true, logical NOT operator makes it false

Example Program

```

#include <stdio.h>
int main()
{

```

```

int a=40,b=20,c=30;
if ((a>b )&& (a >c))
{
printf(" a is greater than b and c");
}
else
if(b>c)
printf("b is greater than a and
c"); else
printf("c is greater than a and b");
}

```

Output

a is greater than b and c.

Conditional Operator

It itself checks the condition and executed the statement depending on the condition.

Syntax:

Condition? Exp1:Exp2

Example:

X=(a>b)?a:b

The ‘?:’ operator acts as ternary operator. It first evaluate the condition, if it is true then exp1 is evaluated, if condition is false then exp2 is evaluated. The drawback of Assignment operator is that after the ? or : only one statement can occur.

Example Program

```

#include <stdio.h>
int main()
{
int x,a=5,b=3;
x = (a>b) ? a : b ;
printf("x value is %d\n", x);
}

```

Output

x value is 5

Bitwise Operators

Bitwise Operators are used for manipulation of data at bit level. It operates on integer only.

S.NO	Operators	Operation	Example	Description
1	&	Bitwise AND	X & Y	Will give 1 only when both inputs are 1
2		Bitwise OR	X Y	Will give 1 when either of input is 1
3	^	Bitwise XOR	X ^ Y	Will give 1 when one input is 1 and other is 0
4	~	1's Complement	~X	Change all 1 to 0 and all 0 to 1
5	<<	Shift left	X<<Y	X gets multiplied by 2^Y number of times
6	>>	Shift right	X>>Y	X gets divided by 2^Y number of times

Example Program

```
#include <stdio.h>
main()
{
int c1=1,c2;
c2=c1<<2;
printf("Left shift by 2 bits c1<<2=%d",c2);
}
```

Output

Left shift by 2 bits c1<<2=4

Special operators:

sizeof () operator:

1. Sizeof operator is used to calculate the size of data type or variables.
2. Sizeof operator will return the size in integer format.
3. Sizeof operator syntax looks more like a function but it is considered as an operator in c programming

Example of Size of Variables

```
#include<stdio.h>
int main()
{
    int ivar = 100;
    char cvar = 'a';
    float fvar = 10.10;
    printf("%d",  sizeof(ivar));
    printf("%d",  sizeof(cvar));
    printf("%d",  sizeof(fvar));
    return 0;
}
```

Output :

2 1 4

In the above example we have passed a variable to size of operator. It will print the value of variable using sizeof() operator.

Example of Sizeof Data Type

```
#include<stdio.h>
int main()
{
    printf("%d", sizeof(int));
    printf("%d", sizeof(char));
    printf("%d",
    sizeof(float)); return 0;
}
```

Output :

2 1 4

In this case we have directly passed an data type to an sizeof.

Example of Size of constant

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

```

int main()
{

    printf("%d", sizeof(10));
    printf("%d", sizeof('A'));
    printf("%d",
    sizeof(10.10)); return 0;
}

```

Output :

2 1 4

In this example we have passed the constant value to a sizeof operator. In this case sizeof will print the size required by variable used to store the passed value.

Example of Nested sizeof operator

```

#include<stdio.h>
int main()
{
    int num = 10;
    printf("%d",
    sizeof(sizeof(num))); return 0;
}

```

Output:

2

We can use nested sizeof in c programming. Inner sizeof will be executed in normal fashion and the result of inner sizeof will be passed as input to outer sizeof operator.

Innermost Sizeof operator will evaluate size of Variable “num” i.e 2 bytes Outer Sizeof will evaluate Size of constant “2” .i.e 2 bytes

Comma(,) Operator:

1. Comma Operator has Lowest Precedence i.e it is having lowest priority so it is evaluated at last.
2. Comma operator returns the value of the rightmost operand when multiple comma operators are used inside an expression.
3. Comma Operator Can acts as –
 - **Operator** : In the Expression
 - **Separator**: Function calls, Function definitions, Variable declarations and Enum declarations

Example:

```
#include<stdio.h>
void main()
{
    int num1 = 1, num2 =
    2; int res;
    res = (num1, num2);
    printf("%d", res);
}
```

Output

2

Consider above example

int num1 = 1, num2 = 2; // In variable Declaration as separator
 res = (num1, num2); // In the Expression as operator

In this case value of rightmost operator will be assigned to the variable. In this case value of num2 will be assigned to variable res.

Examples of comma operator:

Type 1 : Using Comma Operator along with Assignment

```
#include<stdio.h>
int main()
{
    int i;
    i = 1,2,3;
    printf("i:%d\n",i);
    return 0;
}
```

Output:

i:1

Explanation:

i = 1,2,3;

1. Above Expression contain 3 comma operator and 1 assignment operator.
2. If we check precedence table then we can say that "Comma" operator has lowest precedence than assignment operator
3. So Assignment statement will be executed first .
4. 1 is assigned to variable "i".

Type 2 : Using Comma Operator with Round Braces

```

#include<stdio.h>
int main()
{
    int i;
    i = (1,2,3);
    printf("i:%d\n",i);
    return 0;
}

```

Output:

i:3

Explanation:

i = (1,2,3);

1. Bracket has highest priority than any operator.
 2. Inside bracket we have 2 comma operators.
 3. Comma operator has associativity from Left to Right.
 4. Comma Operator will return rightmost operand
- i = (1,2,3) Assign 3 to variable i.

Type 3 : Using Comma Operator inside printf statement

```

#include<stdio.h>
#include< conio.h>
void main()
{
    clrscr();
    printf("Computer", "Programming");
    getch();
}

```

Output:

Computer

You might feel that answer of this statement should be “Programming” because comma operator always returns rightmost operator, in case of printf statement once comma is read then it will consider preceding things as variable or values for format specifier.

Type 4 : Using Comma Operator inside Switch cases.

```

#include<stdio.h>
#include< conio.h>
void main()
{

```

```
int choice = 2 ;
switch(choice)
{
    case 1,2,1:
        printf("\nAllas");
        break;

    case 1,3,2:
        printf("\nBabo");
        break;

    case 4,5,3:
        printf("\nHurray");
        break;
}
}
```

Output :

Babo

Type 5 : Using Comma Operator inside For Loop

```
#include<stdio.h>

int main()
{
    int i,j;
    for(i=0,j=0;i<5;i++)
    {
        printf("\nValue of J : %d",j);
        j++;
    }
    return(0);
}
```

Output:

Value of J : 0
Value of J : 1
Value of J : 2
Value of J : 3
Value of J : 4

Type 6 : Using Comma Operator for multiple Declaration

```
#include<stdio.h>
int main()
{
int num1,num2;
int a=10,b=20;
return(0);
}
```

Note : Use of comma operator for multiple declaration in same statement.

Variable:

- A variable is an identifier that is used to represent some specified type of information within a designated portion of the program.
- A variable may take different values at different times during the execution

Rules for naming the variable

- A variable name can be any combination of 1 to 8 alphabets, digit, or underscore
- The first character must be an alphabet or an underscore (_).
- The length of variable should not exceed 8 characters length, and some of the 'C' compiler can be recognize upto 31 characters.

Data Types in C

C has a concept of 'data types' which are used to define a variable before its use. The definition of a variable will assign storage for the variable and define the type of data that will be held in the location.

The value of a variable can be changed any time.

C has the following basic built-in datatypes.

- int
- float
- double
- char

The bytes occupied by each of the primary data types are

Data type	Description	Memory bytes	Control String	Example
Int	Integer Quantity	2 bytes	%d or %i	int a=12;
Char	Single Character	1 bytes	%C	char s='n';
float	Floating Point	4 bytes	%f	float f=29.777

Double	Double precision floating pointing no's	8 bytes	%lf	double d= 5843214
--------	---	---------	-----	-------------------

Scope of a variable

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable cannot be accessed. There are three places where variables can be declared in C programming language:

1. Inside a function or a block is called **local** variable,
2. Outside of all functions is called **global** variable.
3. In the definition of function parameters which is called **formal** parameters.

Local Variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function. Local variables are not known to functions outside their own. Following is the example using local variables. Here all the variables a, b and c are local to main() function.

```
#include <stdio.h>

main ()
{
    /* local variable declaration */
    int a, b, c;

    /* actual initialization */
    a = 10;
    b = 20;
    c = a + b;
    printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
}
```

Global Variables

Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables:

```
#include <stdio.h>

/* global variable declaration */
int g;

main ()
{
    /* local variable declaration */
    int a, b;

    /* actual initialization */
    a = 10;
    b = 20;
    g = a + b;

    printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
}
```

PRECEDENCE AND ASSOCIATIVITY OF OPERATORS

If an arithmetic expression is given, there are some rules to be followed to evaluate the value of it. These rules are called as the priority rules. They are also called as the hierarchy rules. According to these rules, the expression is evaluated as follows;

Rule 1 :- If an expression contains parentheses , the expression within the parentheses will be performed first. Within the parentheses , the priority is to be followed.

Rule 2 :- If it has more than parentheses , the inner parenthesis is performed first.

Rule 3:- If more than one symbols of same priority , it will be executed from left to right.

C operators in order of precedence (highest to lowest). Their associativity indicates in what order operators of equal precedence in an expression are applied

Operator	Operation	Associativity	Priority
() [] . -> ++ --	Parentheses Brackets (array subscript) Dot operator Structure operator Postfix increment/decrement	left-to-right	1
++ --	Prefix increment/decrement	right-to-left	2

+ ! (type) * & sizeof	- ~ Type cast Pointer Address Determine size in bytes	Unary plus/minus Not operator, complement operator operator	
* / %	Multiplication/division/modulus	left-to-right	3
+ -	Addition/subtraction	left-to-right	4
<< >>	Bitwise shift left Bitwise shift right	left-to-right	5
< <= > >=	Relational less than less than or equal to Relational greater than greater than or equal to	left-to-right	6
== !=	Relational is equal to is not equal to	left-to-right	7
&	Bitwise AND Bitwise exclusive	left-to-right	8
^	Bitwise exclusive OR	left-to-right	9
	Bitwise inclusive OR	left-to-right	10
&&	Logical AND	left-to-right	11
	Logical OR	left-to-right	12
?:	Ternary conditiona	right-to-left	13
= += *= %= ^= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left	14
,	Comma	left-to-right	15

Example for evaluating an expression

Let X = 2 , Y =5 then the value of the expression

$((Y - 1) / X) * (X + Y)$ is calculated

as:- $(Y - 1) = (5 - 1) = 4 = T1$

$(T1 / X) = (4 / 2) = 2 = T2$

$(X + Y) = (2 + 5) = 7 = T3$

$(T2 * T3) = (2 * 7) = 14$

The evaluations are made according to the priority rule.

Type conversion in expressions.

Type conversion is the method of converting one type of data into another data type.

There are two types of type conversion.

1. Automatic type conversion

2. Type casting

Automatic type conversion

- This type of conversion is done automatically. The resultant value of an expression depends upon the operand which occupies more space, which means the result value converted into highest data type.
- The compiler converts all operands into the data type of the largest operand.
- This type of type conversion is done implicitly, this method is called as implicit type conversion.

Eg.1

```
float a,b,c;a=10,b=3;
```

```
c=a/b
```

output= > c= 3.3 {4 bytes(float) (All the variables are same datatype) Eg.2

```
int a,b,c; a=10,b=3; c=a/b;
```

output= >c=3{2 bytes(int)} Eg.3

```
int a; float b,c; a=10,b=3;
```

```
c=a/b;
```

output=> c=3.3 {4 bytes(float) highest datatype is float}

Type casting

➤ This method is used, when user wants to change the type of the data. General Format for type casting is

(datatype)operand

Eg.1

```
int x=10, y=3; z=(float)x/y;(ie z=10.0/3;)
```

output=>z=3.3(float) Eg:2

```
int x=10,y=3; z=x/(float)y;(ie z=10/3.0;)
```

output=>3.3(float)

- The type of the x is not changed, only the type of the value can be changed
- Since the type of conversion is done explicitly, this type conversion is called as explicit type conversion

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:

1. All integer types to be converted to float.
2. All float types to be converted to double.
3. All character types to be converted to integer

Input and Output statements

In 'c' language several functions are available for input/output operations. These functions are collectively known as the standard I/O library.

1. Unformatted input /output statements
2. Formatted input /output statements

Unformatted Input /Output statements

These statements are used to input /output a single /group of characters from/to the input/output devices. Here the user cannot specify the type of data that is going to be input/output.

The following are the Unformatted input /output statements available in 'C'.

Input	Output
getchar()	putchar()
getc()	putc()
gets()	Puts()

single character input-getchar() function:

A getchar() function reads only one character through the keyboard. **Syntax:** char variable=getchar();

Example:

```
char x;
```

```
x=getchar( );
```

single character output-putchar() function:

A putchar() function is used to display one character at a time on the standard output device.

Syntax: putchar(charvariable);

Example:

```
char x;  
putchar(x);
```

the getc() function

This is used to accept a single character from the standard input to a character variable.

Syntax: character variable=getc();

Example:

```
char c;  
c=getc( );
```

the putc() function

This is used to display a single character variable to standard output device.

Syntax: putc(character variable);

Example:

```
char c;  
putc(c );
```

the gets() and puts() function

The gets() function is used to read the string from the standard input device.

Syntax: gets(string variable);

Example:

```
gets( s);
```

The puts() function is used to display the string to the standard output device. **Syntax:** puts(string variable);

Example:

```
puts( s);
```

Proram using gets and puts function

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

```

main()
{
char scientist[40];
puts("Enter name");
gets(scientist);
puts("Print the Name");
puts(scientist);
}

```

output:

```

Enter Name:Abdul Kalam
Print the Name:Abdul Kalam

```

Formatted input /output statements

The function which is used to give the value of variable through keyboard is called **input function**. The function which is used to display or print the value on the screen is called **output function**.

Note : - In C language we use two built in functions, one is used for reading and another is used for displaying the result on the screen. They are scanf() and printf() functions. They are stored in the header file named stdio.h.

General format for scanf () function

scanf("control string", &variable1, &variable2,.....)

The control sting specifies the field format in which the data is to be entered. %d –integer

%f – float

%c- char

%s –

string

%ld – long integer

%u – Unsigned Integer

Example:

scanf("%d",&x) – reading an integer value, the value will be stored in x

scanf("%d%f",&x,&a) - reading a integer and a float value In the above scanf () function , we don't use any format. This type of Input is called as the Unformatted Input function.

Formatted Input of Integer

The field speciation for reading the integer number is:

%wd

Where The percentage sign(%) indicates that a conversion specification follows. w – is the field width of the number to be read. d will indicates as data type in integer number.

Example:

```
scanf("%2d %5d", &num1,&num2);
```

data line is 50 31425

the value 50 is assigned to num1 and 31425 is assigned to num2. suppose the input data is as follows

31425 50 , then the variable num1 will be assigned 31 and num2 will be assigned to 425 and the 50 is unread.

An input field may be skipped by specifying * in the place of field width.

Example the statement `scanf("%d %*d %d",&a,&b);` will assign the data 123 456 789 as follows: 123 is assigned to a , 456 skipped because of * and 789 to b

Output Function : To print the value on the screen or to store the value on the file, the output functions are used. `printf()` is the function which is use to display the output on the screen. The General format of the `printf()` function is

```
printf("control string",variable1,variable2,.....);
```

Example

`printf("%d",x)` – printing the integer value x.

`printf("%d%f", x,a)`- printing a integer and float value using a single `printf` function.

Formatted Output of Integer :Similar to formatted input , there is a formatted output also to have the output in a format manner.

In this control string consists of three types of items.

- Characters that will be printed on the screen as they appear
- Format specification that define the output format for display of each item
- Escape sequence characters such as
 \n – new line

\b – back space
 \f – form feed
 \r – carriage return
 \t – horizontal tab
 \v – vertical tab

The format speciation is as follows %wd

Where w – is the field width of the number to be write . d will indicates as data type in integer number.

Examples:

Printf(“%d”,9876); // output:

9876 printf(“%6d”,9876);

output:

1 2 3 4 5 6

		9	8	7	6
--	--	---	---	---	---

printf(“%-6d”,9876); **output:**

1 2 3 4 5 6

9	8	7	6		
---	---	---	---	--	--

printf(“%06”,9876);

output:

1 2 3 4 5 6

0	0	9	8	7	6
---	---	---	---	---	---

Formatted input of Real(float) Numbers:

. The field speciation for reading the real number is:

%w.pf

Where w – is the field width of the number to be read . p indicates the number of digits to be read after the decimal point f – indicates that data type in float(real) number.

Example

scanf(“%2.1f %5.2f”,&num1,&num2);

data line is 50.1 31425.20

the value 50.1 is assigned to num1 and 31425.20 is assigned to num2.

An input field may be skipped by specifying * in the place of field width.

Example: the statement `scanf("%f %*f %f", &a,&b);` will assign the data 12.3 4.56 78.9 as follows: 12.3 is assigned to a , 4.56 skipped because of * and 78.9 to b.

Formatted output of Real(float) Numbers:

The field speciation for reading the real number is:

%w.pf

Where w – is the field width of the number to be read . p indicates the number of digits to be displayed after the decimal point f – indicates that data type in float(real) number.

Example:

Float y = 98.7682

`Printf(" %f ", y);` // output: 98.7682

`printf("%7.2f ",y);`

output:

1	2	3	4	5	6	7
		9	8	.	7	6

`printf("%-7.2f ",y);`

output:

1	2	3	4	5	6	7
9	8	.	7	6		

Formatted input of Single characters or strings:

The field speciation for reading the character

strings: %ws or %wc

where,

%c is used to read a single character.

Example:

Char name;

`Scanf("%c", &name);` \\ I / P : a

```
Char name[20];
```

```
Scanf("%s",&name); \\ I / P : sathyabama
```

Printing of a Single Character:

The field speciation for reading the character
strings: %ws or %wc

where,

%s – A sequence of characters can be displayed.

%c – A single character can be displayed.

The character will be displayed right-justified in the field of w, left-justified by placing a minus sign before the integer w.

Example:

```
Char x = 'a';
```

```
Char name[20] = "anil kumar gupta";
```

```
Printf("%c", x); // output: a
```

```
Printf("%s",name); // output: anil kumar gupta
```

```
Printf("%20s", name);
```

Output:

1	2	3	4	5	6	6	8	9	10	11	12	13	14	15	16	17	18	19	20
				a	n	i	l		k	u	m	a	r		g	u	p	t	a

```
Printf("%-20.10s", name);
```

Output:

1	2	3	4	5	6	6	8	9	10	11	12	13	14	15	16	17	18	19	20
a	n	i	l			k	u	m	a	r									

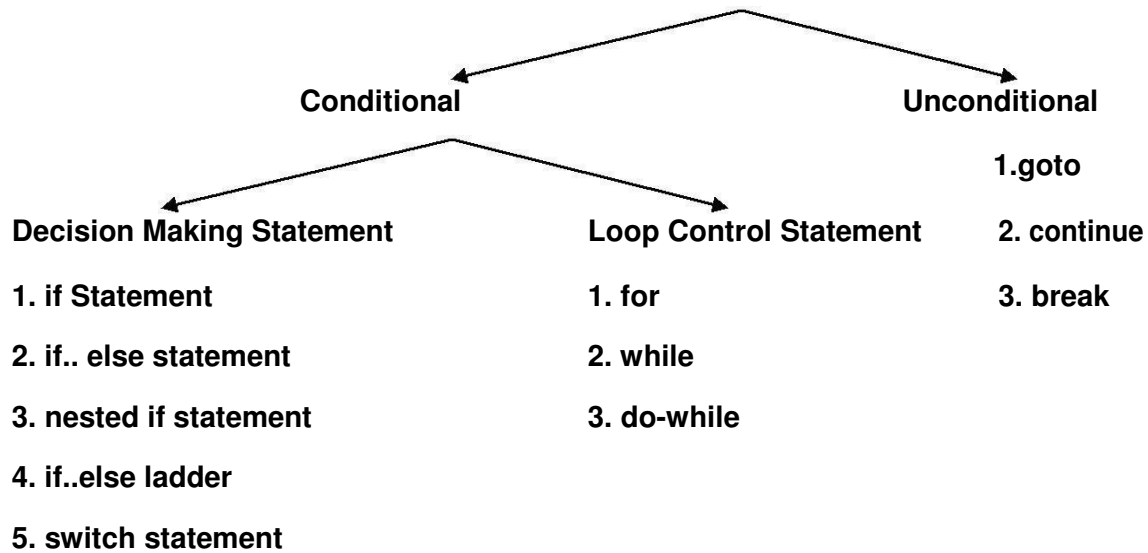
```
Printf("%.5s", name);
```

Output:

g	u	p	t	a
---	---	---	---	---

CONTROL STRUCTURE

Control Statements

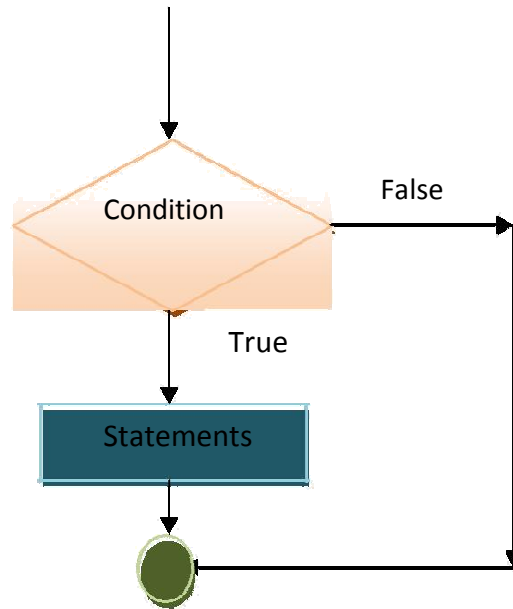


CONDITIONAL STATEMENT

Decision Making Statement

If Statement:

- ♣ The if statement is a decision making statement.
- ♣ It is used to control the flow of execution of the statement and also used to the logically whether the condition is true or false
- ♣ It is always used in conjunction with condition.



Syntax:

```
if(condition)
{
    True statements;
}
```

- ♣ If the condition is true, then the true statements are executed.
- ♣ If the condition is false then the true statements are not executed, instead the program skips past them.
- ♣ The condition is given by relational operators like ==,<=,>=,!=,etc.

Example 1: //program to check whether the entered number is less than 25

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    clrscr();
    printf("Enter one
value"); scanf("%d",&i);
    if(i<=25)
        printf("The entered no %d is <
25",i); getch();
}
```

Output:

```
Enter one value 5
The entered no 5 is < 25
```

Example 2: //program to calculate the sum and multiplication using if Statement

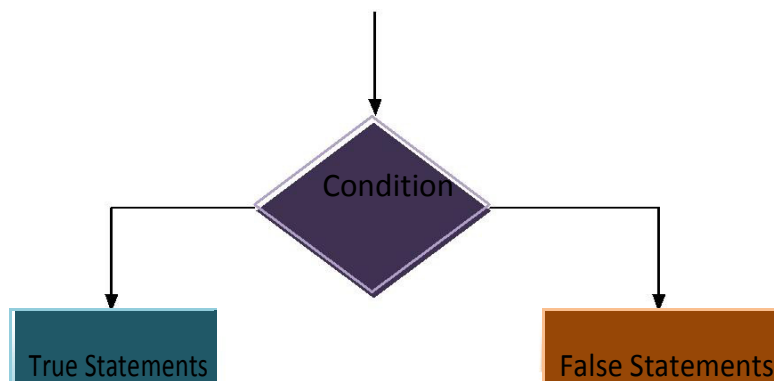
```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,n;
    clrscr();
    printf("Enter two values");
    n=scanf("%d%d",&a,&b);
    if(n==2)
    {
        printf("the sum of two numbers : %d",a+b);
        printf("the product of two numbers:%d",a*b);
    }
    getch();
}
```

Output:

```
Enter two value 5 10
the sum of two numbers : 15
the product of two numbers : 50
```

if.. else statement:

- ♣ It is basically two way decision making statement and always used in conjunction with condition.
- ♣ It is used to control the flow of expression and also used to carry the logical test and then pickup one of the two possible actions depending on the logical test.
- ♣ If the condition is true, then the true statements are executed otherwise false statements are executed.
- ♣ The true and false statements may be single or group of statements.



Syntax:

```
    If (condition)
        True statements;
    else
        False statements;
```

Example 1: //program to find the greatest of two number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    printf("Enter two value");
    scanf("%d%d",&a,&b);
    if(a>b)

        printf("The given no %d is greatest",a);
    else
        printf("The given no %d is greatest",b);
}
```

Output:

```
Enter two value 5 10
The given no 10 is greatest
```

Nested if..else Statement:

When a series of if_else statements are needed in a program, we can write an entire if_else statement inside another if and it can be further nested. This is called nesting if.

Syntax:

```
    if(condition 1)
    {
        if(condition 2)
        {
            True statement 2;
        }
        else
            False statement 2;
    }
    else
        False statement 1;
}
```


Example 1: //program to find the greatest of three numbers.

```
#include <stdio.h>

int main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;

    /* check the boolean condition */
    if( a == 100 )
    {
        /* if condition is true then check the following */
        if( b == 200 )
        {
            /* if condition is true then print the following */
            printf("Value of a is 100 and b is 200\n" );
        }
    }
    printf("Exact value of a is : %d\n", a );
    printf("Exact value of b is : %d\n", b );

    return 0;
}
```

Output:

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

If_else Ladder:

1. Nested if statements will become complex, if several conditions have to be checked.
2. In such situations we can use the else if ladder .

Syntax:

```
if(condition 1)
{
    if(condition 2)
    {
        True statement 2;
    }
    elseif(condition 3)
    {
```

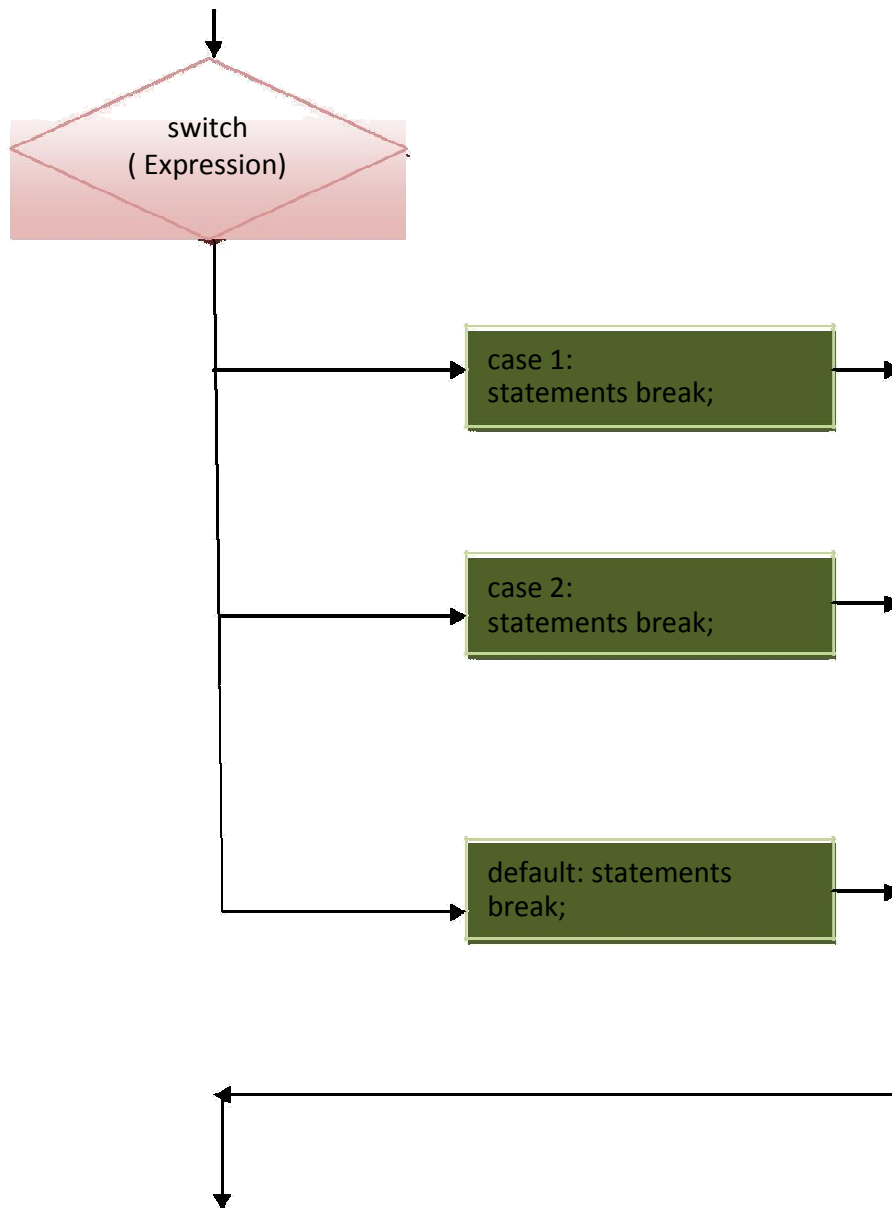
```
        True statement 3;
    else
        False statement 3;
    }
else
    False statement 1;
}
```

Switch Statement

- The switch statement is used to execute a particular group of statements from several available groups of statements.
- It allows us to make a decision from the number of choices.
- It is a multi-way decision statement.

Rules for writing switch () statement.

1. The expression in switch statement must be an integer value or a character constant.
2. No real numbers are used in an expression.
3. Each case block and default block must be terminated with break statement.
4. The default is optional and can be placed anywhere, but usually placed at end.
5. The 'case' keyword must terminate with colon(:).
6. Cases should not be identical.
7. The values of switch expression is compared with the case constant expression in the order specified i.e., from top to bottom.



Syntax:

```
switch(expression)
{
    case 1:
        state
        ment;
        break;
    case 2:
        state
        ment;
        break;
```

```
        default: statement;
            break;
    }
```

// program to print the give number is odd / even using switch case statement.

```
#include<stdio.h>
#include<conio.h> void main()
{
    int a,b,c;
    printf("Enter one value");
    scanf("%d",&a); switch(a%2)
    {
        case 0:
            printf("The given no %d is even",
                a); break;
        default :
            printf("The given no %d is odd",
                a); break;
    }
}
```

Output:

Enter one value 5
The given no 5 is odd

Unconditional statement

Break statement

5. The break statement is used to terminate the loop.
6. When the keyword break is used inside any loop, control automatically transferred to the first statement after the loop.

Syntax:

```
break;
```

//program to print the number upto 5 using break statement

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    for(i=1;i<=10;i++)
```

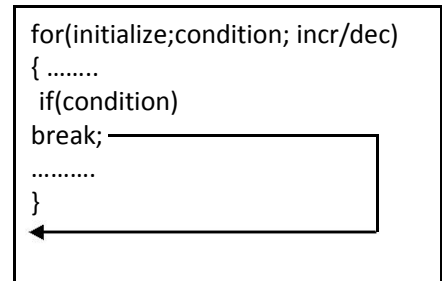
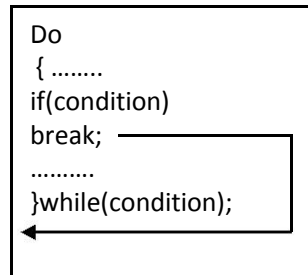
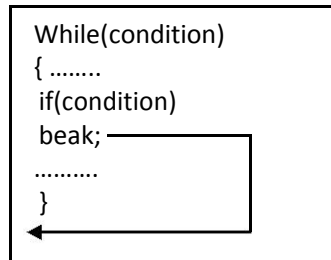
```

{
if(i==6)
break;
printf("%d",i);
}
}

```

Output:

1 2 3 4 5



Continue Statement

- In some situation, we want to take the control to the beginning of the loop, bypassing the statement inside the loop which have not been executed, for this purpose the continue is used.
- When the statement continue is encountered inside any loop, control automatically passes to the beginning of the loop.

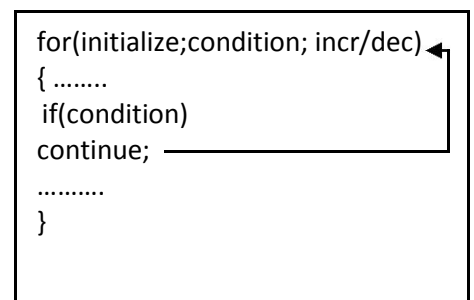
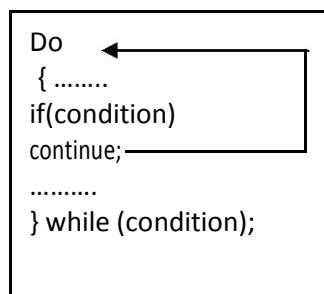
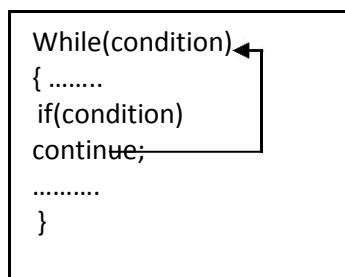
Syntax:

continue;

```

While(condition)
{
.....
if(condition)
continue;
.....
}

```




Difference between break and continue

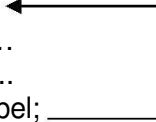
Break	Continue
Break statement takes the control to the outside of the loop	Continue statement takes the control to be beginning of the loop
It is also in switch statement	This can be used only in loop statements
Always associated with if condition in loop	This is also associated with if condition

Goto Statement:

- C provides the goto statement to transfer control unconditionally from one place to another place in the program.
- A goto statement can change the program control to almost anywhere in the program unconditionally.
- The goto statement require a label to identify the place to move the execution.
- The label is a valid variable name and must be ended with colon(:).

Syntax:

1. goto label; 

2. label: 

/* program to print the given both number is equal or not*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
printf("Enter the numbers");
scanf("%d%d",&a,&b);
if(a==b)
    goto equal;
else
{
    printf("%d and %d are not
    equal",a,b); exit(0);
}
equal: printf("%d and %d are equal",a,b);
```

```
}
```

Output:

```
Enter the numbers    4    5
4 and    5 are not equal
```

```
Enter the numbers    5    5
5 and    5 are equal
```

LOOPING STATEMENTS

A loop statement allows us to execute certain block of code repeatedly until test condition is false.

There are 3 types of loops in C programming:

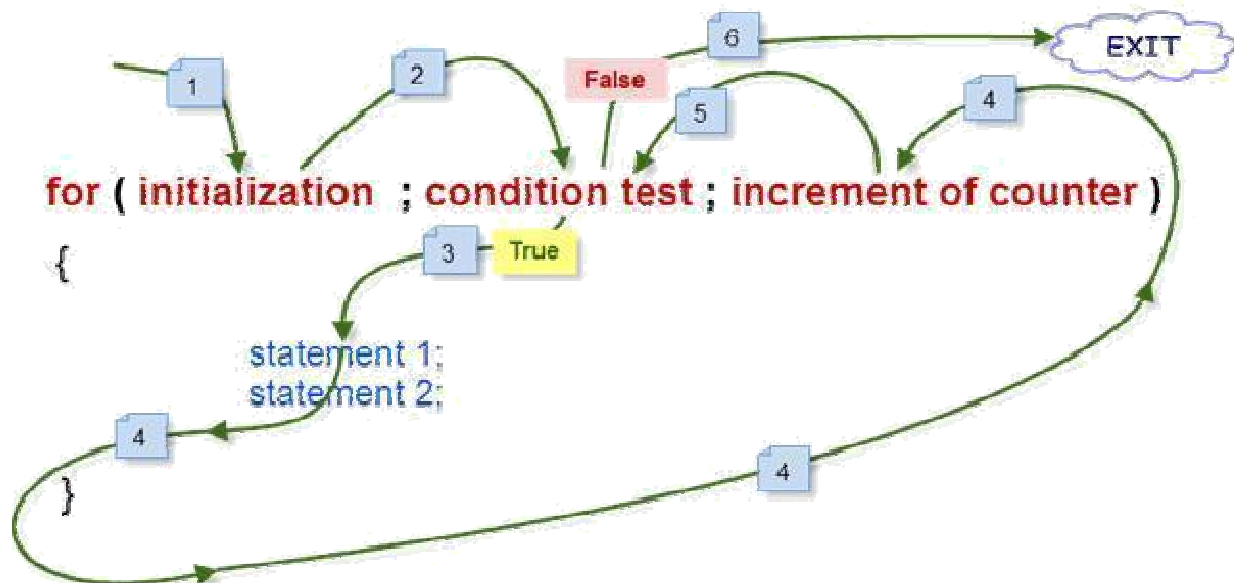
- ☐ for loop
- ☐ while loop
- ☐ do...while loop

for loop:

The syntax for a for loop is

```
for ( variable initialization; condition; variable update )
{
Code to execute while the condition is true
}
```

The initialization statement is executed only once at the beginning of the for loop. Then the test expression is checked by the program. If the test expression is false, for loop is terminated. But if test expression is true then the code/s inside body of for loop is executed and then update expression is updated. This process repeats until test expression is false.



for loop example

**Write a program to find the sum of first n natural numbers where n is entered by user.
Note: 1,2,3... are called natural numbers.**

```

#include <stdio.h>
void main(){
int n, count, sum=0;
printf("Enter the value of
n.\n"); scanf("%d",&n);
for(count=1;count<=n;++count) //for loop terminates if count>n
{
    sum+=count; /* this statement is equivalent to
    sum=sum+count */
}
printf("Sum=%d",sum);

}
  
```

Output

```

Enter the value of
n. 19
Sum=190
  
```


In this program, the user is asked to enter the value of n . Suppose you entered 19 then, count is initialized to 1 at first. Then, the test expression in the for loop, i.e., $(count \leq n)$ becomes true. So, the code in the body of for loop is executed which makes sum to 1. Then, the expression $++count$ is executed and again the test expression is checked, which becomes true. Again, the body of for loop is executed which makes sum to 3 and this process continues. When count is 20, the test condition becomes false and the for loop is terminated.

/* C program to check whether a number is prime or not. */

```
#include <stdio.h>
int main()
{
    int n, i, flag=0;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    for(i=2;i<=n/2;++i)
    {
        if(n%i==0)
        {
            flag=1;
            break;
        }
    }
    if (flag==0)
        printf("%d is a prime number.",n);
    else
        printf("%d is not a prime number.",n);
    return 0;
}
```

Output

```
Enter a positive integer: 29
29 is a prime number.
```

This program takes a positive integer from user and stores it in variable n . Then, for loop is executed which checks whether the number entered by user is perfectly divisible by i or not starting with initial value of i equals to 2 and increasing the value of i in each iteration. If the number entered by user is perfectly divisible by i then, $flag$ is set to 1 and that number will not be a prime number but, if the number is not perfectly divisible by i until test condition $i \leq n/2$ is true means, it is only divisible by 1 and that number itself and that number is a prime number.

FUNCTIONS

LIBRARY FUNCTIONS

Definition

C Library functions are inbuilt functions in C language which are clustered in a group and stored in a common place called Library. Each and every library functions in C executes explicit functions. In order to get the pre- defined output instead of writing our own code, these library functions will be used. Header file consists of these library functions like Function prototype and data definitions.

3. Every input and output operations (e.g., writing to the terminal) and all mathematical operations (e.g., evaluation of sines and cosines) are put into operation by library functions.
4. The C library functions are declared in header files (.h) and it is represented as [file_name].h
5. The Syntax of using C library functions in the header file is declared as "#include<file_name.h>". Using this syntax we can make use of those library functions.
6. "#include<filename.h>" command defines that in C program all the codes are included in the header files followed by execution using compiler.
7. It is required to call the suitable header file at the beginning of the program in terminal in order to use a library function. A header file is called by means of the pre-processor statement given below,

`#include<filename.h>`

Whereas the filename represents the header file name and #include is a pre - processor directive. To access a library function the function name must be denoted, followed by a list of arguments, which denotes the information being passed to the function.

Example

In case if you want to make use of printf() function, the header file <stdio.h> should be included at the beginning of the C program.

```
#include <stdio.h>
int main()
{
/* NOTE: Error occurs if printf() statement is written without
using the header file */

    printf(" Hello World");
}
```

The „main() function□ is also a library function which is called at the initial of the program.

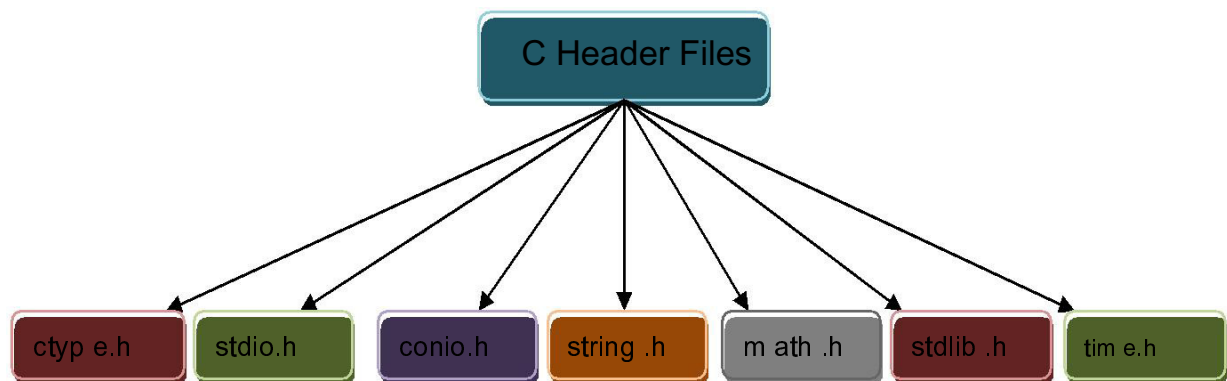
Example

To find the square root of a number we use our own part of code to find them but this may not be most efficient process which is time consuming too. Hence in C programming by declaring the square root function `sqrt()` under the library function "math.h" will be used to find them rapidly and less time consuming too. Square root program using the library functions is given below:

Finding Square root Using Library Function

```
#include <stdio.h>
#include <math.h>
int main(){
    float num,root;
    printf("Enter a number to find square
    root."); scanf("%f",&num);
    root=sqrt(num); /* Computes the square root of num and stores
    in root. */
    printf("Square root of
    %.2f=%.2f",num,root); return 0;
}
```

List of Standard Library Functions in C Programming



Adding User Defined functions in C library:

- In C Programming we can declare our own functions in C library which is called as user-defined functions.
- It is possible to include, remove, change and access our own user defined function to or from C library functions.
- Once the defined function is added to the library it is merely available for all C programs which are more beneficial of including user defined function in C library function
- Once it is declared it can be used anywhere in the C program just like using other C library functions.
- By using these library functions in GCC compilers (latest version), compilation time can be consumed since these functions are accessible in C library in the compiled form.
- Commonly the header files in C program are saved as "file_name.h" in which all library

functions are obtainable. These header files include source code and this source code is further added in main C program file where we include this header file via “#include <file_name.h>” command.

Steps for adding user defined functions in C library:

Step 1:

For instance, hereby given below is a test function that is going to be included in the C library function. Write and save the below function in a file as “addition.c”

```
addition(int a, int b)
{
    int sum;
    total =a + b;
    return sum;
}
```

Step 2:

Compile “addition.c” file by using Alt + F9 keys (in turbo C).

step 3:

A compiled form of “addition.c” file would be created as “addition.obj”.

Step 4:

To add this function to library, use the command given below (in turbo C). c:\> tlib math.lib + c:\ addition.obj
+ represents including c:\addition.obj file in the math library. We can delete this file using – (minus).

Step 5:

Create a file “addition.h” and declare sample of addition() function like below. int addition (int a, int b);
Now “addition.h” file has the prototype of function “addition”.

Note : Since directory name changes for each and every IDE, Kindly create, compile and add files in the particular directory.

Step 6:

Here is an example to see how to use our newly added library function in a C program.

```
4.    include <stdio.h>
        User defined function is included here.
5.    include “c:\\addition.h”
int main ( )
{
```

```

    int total;
    // calling function from library
    total = addition (10, 20); printf
    ("Total = %d \n", total);
}

```

Output:
Total = 30

- Source code checking for all header files can be checked inside “include” directory following C compiler that is installed in system.
- For instance, if you install DevC++ compiler in C directory in our system, “C:\Dev-Cpp\include” is the path where all header files will be readily available.

Mostly used header files in C:

C library functions and header files in which they are declared in conio.h is listed below:

S.No	Header file	Description
1	stdio.h	A standard input/output header file where Input/ Output functions are declared
2	conio.h	Console input/output header file
3	string.h	String functions are defined in this header file
4	stdlib.h	The general functions used in the C program is defined in this header file.
5	math.h	Mathematical related functions are defined in this header file.
6	time.h	Time and clock allied functions are defined in this header file.
7	ctype.h	Every character managing functions are declared in this header file
8	errno.h	This header file contains Error handling functions.
9	assert.h	Diagnostics functions are declared in this header file.

C – conio.h library functions

The entire C programming inbuilt functions that are declared in conio.h header file are given below. The source code for conio.h header file is also given below for your reference.
List of inbuilt conio.h file C functions:

S.no	Function	Description
1	clrscr()	This function is used to clear the output screen.
2	getch()	It reads character from keyboard
3	getche()	It reads character from keyboard and echoes to o/p screen
4	textcolor()	This function is used to change the text colour
5	textbackground()	This function is used to change text background

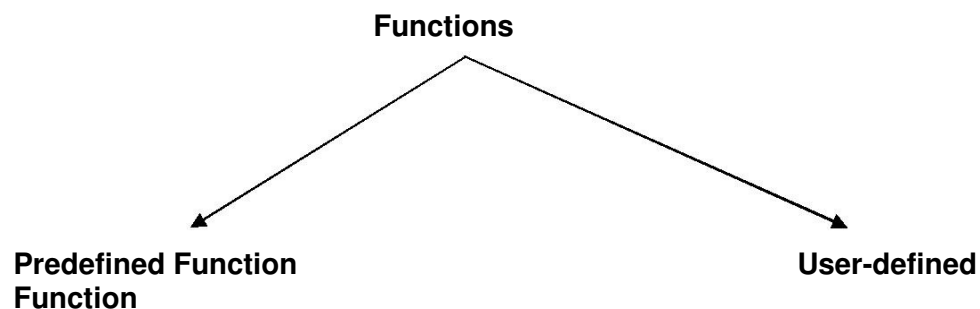
C – stdio.h library functions

Inbuilt functions of C declared in stdio.h header file are given below.

S.no	Function	Description
1	printf()	This function is used to print the character, string, float, integer, octal and hexadecimal values onto the output screen
2	scanf()	This function is used to read a character, string, numeric data from keyboard.
3	getc()	It reads character from file
4	gets()	It reads line from keyboard
5	getchar()	It reads character from keyboard
6	puts()	It writes line to o/p screen
7	putchar()	It writes a character to screen
8	clearerr()	Clears the error indicators
9	f open()	All file handling functions are defined in this header file.
10	f close()	closes an opened file
11	getw()	reads an integer from file
12	putw()	writes an integer to file
13	f getc()	reads a character from file
14	fputc()	writes a character to file
15	f putc()	writes a character to file
16	f gets()	reads string from a file, per line at a time
17	f puts()	writes string to a file
18	f eof()	finds end of file
19	f getchar	reads a character from keyboard
20	f getc()	reads a character from file
21	f printf()	writes formatted data to a file
22	f scanf()	reads formatted data from a file
23	f getchar	reads a character from keyboard
24	f putchar	writes a character from keyboard
25	f seek()	moves file pointer position to given location
26	SEEK_SET	moves file pointer position to the beginning of the file
27	SEEK_CUR	moves file pointer position to given location
28	SEEK_END	moves file pointer position to the end of file.
29	f tell()	gives current position of file pointer
30	rewind()	moves file pointer position to the beginning of the file
31	putc()	writes a character to file
32	sprintf()	writes formatted output to string
33	sscanf()	Reads formatted input from a string
34	remove()	deletes a file
35	fflush()	flushes a file

Functions

- A function is a group of statement that is used to perform a specified task which repeatedly occurs in the main program. By using function, we can divide the complex problem into a manageable problem.
- A function can help to avoid redundancy.
- Function can be of two types, there are
 1. Built-in Function (or) Predefined Function (or) Library Function
 2. User defined Function



Difference between Predefined and User-defined Functions

Predefined Function	User-defined function
Predefined function is a function which is already defined in the header file (Example: math.h, string.h, etc)	User- Defined function is a function which is created by the user as per requirement of its owner
Predefined Function is a part of a header file, which are called at runtime	User- Defined function are part of the program which are compiled at runtime
The Predefined function name is given by the developer	User- Defined function name created by the user
Predefined Function name cannot be changed	User defined Function name can be changed

User Defined Functions

- The function defined by the users according to their context (or) requirements is known as a user defined function.
- The User defined function is written by the programmer to perform specific task (or) operation, which is repeatedly used in the main program.

- These functions are helpful to break down the large program into a number of the smaller function.
- The user can modify the function in order to meet their requirements.
- Every user define function has three parts namely
 - Function Declaration
 - Function Calling
 - Function Definition

Need for user-defined function

- While it is possible to write any complex program under the function, and it leads to a number of problems, such as
 - The problem becomes too large and complex.
 - The user can't go through at a glance
 - The task of debugging, testing and maintenance become difficult.
- If a problem is divided into a number of parts, then each part may be independently coded and later it combined into a single program. These subprograms are called functions, it is much easier to understand, debug and test the program.

Merits of User-Defined Function

5. The length of the source program can be reduced by dividing it into smaller functions
6. It provides modularity to the program
7. It is easy to identify and debug an error
8. Once created a user defined function, can be reused in other programs
9. Function facilitates top-down programming approach
10. The Function enables a programmer to build a customized library of repeatedly used routines
11. Function helps to avoid coding of repeated programming of the similar instruction

Elements of User-Defined Function

5. **Function Declaration**
6. **Function Call**
7. **Function Definition**

Function Declaration

Like normal variable in a program, the function can also be declared before they defined and invoked

Function declaration must end with semicolon (;)

A function declaration must declare after the header file

The list of parameters must be separated by comma.

The name of the parameter is optional, but the data type is a must.

- If the function does not return any value, then the return type void is must.
If there are no parameters, simply place void in braces.

- The data type of actual and formal parameter must match.

Syntax:

Return_type function_name (datatype parameter1, datatype parameter2,...);

Description:

Return type	:	type of function
Function_name	:	name of the function
Parameter list or argument list	:	list of parameters that the function

can convey.

Example:

int add(int x,int y,int z);

Function Call

The function call be called by simply specifying the name of the function, return value and parameters if presence.

Syntax: function_name();
function_name(parameter);
return_value =function_name (parameter);

Description:

function_name	:	Name of the function
Parameter	:	Actual value passed to the calling function

Example

```
fun();
fun(a,b);
fun(10,20);
c=fun(a,b);
e=fun(2.3,40);
```

Function Definition

- It is the process of specifying and establishing the user defined function by specifying all of its element and characteristics.

Syntax:

Return_type function_name (datatype parameter1, datatype parameter2)

Example 1

```
#include<stdio.h>
#include<conio.h>
void add(); //Function Declaration void sub();//Function Declaration
void main()
{
    clrscr();
    add(); //Function call
    sub(); //Function call
    getch();
}
void add()      //Function Definition
{
    int a,b,c;
    printf("Enter two values");
    scanf("%d%d",&a,&b); c=a+b;
    printf("add=%d",c);
}
void sub()      //Function Definition
{
    int a,b,c;
    printf("Enter two values");
    scanf("%d%d",&a,&b);
    c=a-b;
    printf("sub=%d",c);
}
```

Example 2 :

```
//Program to check whether the given number is odd or even
#include<stdio.h>
#include<conio.h>
void oddoreven()
{
    printf("Enter One value");
    scanf("%d",&oe);
    if(oe%2==0)
        printf("The Given Number%d is even"); else
        printf("The Given Number %d is odd");
}
void main()
{
    clrscr();
    oddoreven();
    getch();
}
```

}

Function Parameter

- The Parameter provides the data communication between the calling function and called function.
- There are two types of parameters.
 - **Actual parameter:** passing the parameters from the calling function to the called function i.e the parameter, return in function is called actual parameter
 - 3. **Formal parameter:** the parameter which is defined in the called function i.e. The parameter, return in the function definition is called formal parameter

Example:

```
main()
{
    .....
    .....
    Fun(a,b);
    .....
    .....
}
Fun(int x,int y)
{
    .....
    .....
}
```

Where

a,b are the actual parameters

x,y are formal parameter

Example Program

```
#include<stdio.h>
#include<conio.h>
void add(int,int); //Function Declaration
void sub(float,int); //Function Declaration
void main()
{
    clrscr();
    add(3,4); //Function call
    sub(2.5,5); //Function call
    getch();
}
void add(int a,int b)//Function Definition
{
    int c;
    c=a+b;
    printf("add=%d",c);
}
void sub(float a, int b) //Function Definition
```

Output:

```
add=7
sub=-2.500000
```

```

{
float c;
c=a-b;
printf("sub=%f",c);
}

```

Example 2:

```

//program for factorial of given
number #include<stdio.h>
#include<conio.h> void main()
{
int fact(int);
int f;
clrscr();
printf("Enter one value");
scanf("%d",&f);
printf("The Factorial of given number %d is
%d",f,fact(f)); getch();
}
int fact(int f)
{
if(f==1)
return 1;
else
return(f*fact
(f-1));
}

```

Output:
Enter one value 5
The Factorial of given
number 5 is 120

Function Prototype (or) Function Interface

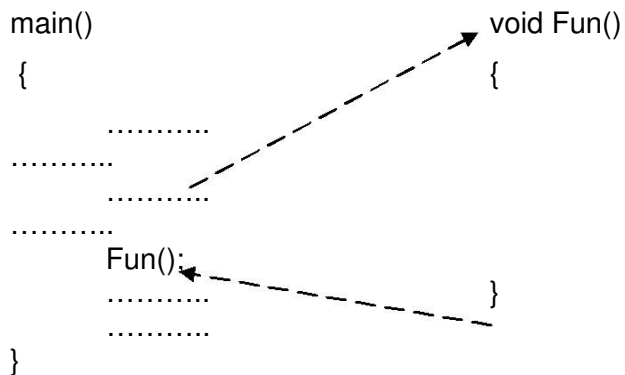
- The functions are classified into four types depends on whether the arguments are present or not, whether a value is returned or not. These are called function prototype.
- In 'C' while defining user defined function, it is must to declare its prototype.
- A prototype states the compiler to check the return type and arguments type of the function.
- A function prototype declaration consists of the function's return type, name and argument. It always ends with semicolon. The following are the function prototypes
 - **Function with no argument and no return value.**
 - **Function with argument and no return value.**
 - **Function with argument and with return**

value. ○ Function with no argument with return value.

Function with no argument and no return value

- In this prototype, no data transfer takes place between the calling function and the called function. i.e., the called program does not receive any data from the calling program and does not send back any value to the calling program.

Syntax:-



The dotted lines indicates that, there is only transfer of control, but no data transfer.

Example program 1

```
#include<stdio.h>
#include<conio.h>
void mul();
void main()
{
    clrscr();
    mul();
    getch();
}
void mul()
{
    int a,b,c;
    printf("Enter two values");
    scanf("%d%d",&a,&b); c=a*b;
```

Output:

Enter two values 6 4
mul=24

```

        printf("mul=%d",c);
    }

```

Example program 2

//Program for finding the area of a circle using Function with no argument and no return value

```

#include<stdio.h>
#include<conio.h>
void circle();
void main()
{
    The area of circle 78.500000 circle();
}
void circle()
{
    int r;
    float cir;
    printf("Enter radius");
    scanf("%d",&r);
    cir=3.14*r*r;
    printf("The area of circle is %f",cir);
}

```

Output:

Enter radius 5

Function with argument and no return value

- In this prototype, data is transferred from the calling function to called function. i.e., the called function receives some data from the calling function and does not send back any values to calling function
- It is one way data communication.

Syntax:-

```

main()
{
    .....
    Fun(a,b);
    .....
}
void Fun(x,y)
{
    .....
}

```

Example program 1:

```

#include<stdio.h>
#
i

```

```
#include<conio.h>
```

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

```
void main()
```

```
{
```

```
    clrscr();
```

```
    int a,b;
```

```
    printf("Enter two values");
```

```
    scanf("%d%d",&a,&b);
```

```
    add(a,b);
```

```
    getch();
```

```
}
```

```
void add(int x,int y)
```

```
{
```

```
    int c;
```

```
    c=x+y;
```

```
    printf("add=%d",c);
```

```
}
```

The solid lines indicate data transfer and dotted line indicates a transfer of control.

a and b are the actual parameters

x and y are formal parameters

Output:

Enter two values 6 4

add=10

Example program 2:

//Program to find the area of a circle using Function with argument and no return value

```

#include<stdio.h>
#include<conio.h>
void circle(int);
void main()
{
    int r;
    clrscr();
    printf("Enter radius");
    scanf("%d",&r);
    circle(r);
}
void circle(int r)
{
    float cir;
    cir=3.14*r*r;
    printf("The area of circle is
%f",cir); getch();
}

```

Output:

```

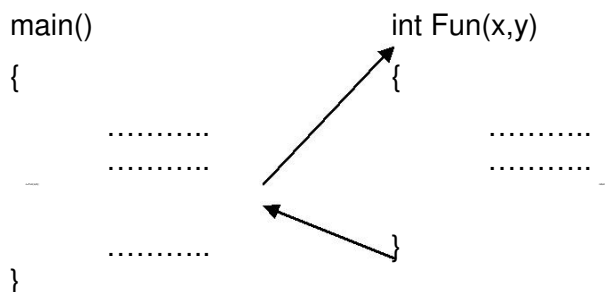
Enter radius 5
The area of circle 78.500000

```

Function with argument and with return value.

- In this prototype, the data is transferred between the calling function and called function. i.e., the called function receives some data from the calling function and sends back returned value to the calling function.
- It is two way data communication

Syntax:-



The solid lines indicates data transfer takes place in between the calling program and called program

a,b are the actual parameter

x,y are formal parameter

Example program 1:

```

#include<stdio.h>
#include<conio.h>
void add(int,int);

```

Output:

```

Enter two values 6 4
Add=10

```



```

void main()
{
    clrscr();
    int a,b,c;
    printf("Enter two values");
    scanf("%d%d",&a,&b);
    c=add(a,b);
    printf("Add=%d",c); getch();

}
void add(int x,int y)
{
    int m;
    m=x+y;
    return m;
}

```

Example Program 2

// Program to find the area of a circle using Function with argument

and with return value

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

```
#include<conio.h> float
```

```
circle(int); void main()
```

```

{
    int r; clrscr();
    printf("Enter radius");
    scanf("%d",&r);
    printf("the area of
    circle is
    %f",circle(r)); getch();
}

```

Output:

```

Enter radius 5
the area of circle 78.500000

```

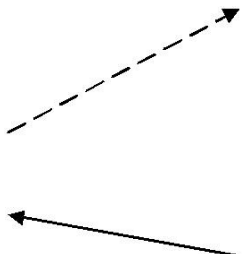
```
float circle(int r)
{
    float cir;
    cir=3.14*r*r;
    return cir;
}
```

Function with no argument with return value

- In this prototype, the calling function cannot pass any arguments to the called function, but the called program may send some return value to the calling function.
- It is one way data communication

Syntax:-

```
.....
.....
.....
.....
Fun();
return(z);
.....
}
```



Example program 1

```
#include<stdio.h>
#include<conio.h>
int add();
void main()
{
    clrscr();
    int z;
    z=add();
    printf("Add=%d",z);
    getch();
}
int add()
```

NOTE: The dotted line indicates a control transfer to the called program and the solid line indicates data return to the calling program

Output:

Enter two values 6 4
Add=10

Example Program 2

// Program to the area of a circle using no argument with a return value

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

```
#include<conio.h>
```

```
float circle();
```

```
void main()
```

```
{
```

```
    clrscr();
```

```
    printf("the area of circle is
```

```
    %f",circle()); getch();
```

```
}
```

```
float circle()
```

```
{
```

```
    float cir; int r;
```

```
    printf("Enter radious"); scanf("%d",&r);
```

```
    cir=3.14*r*r;
```

```
    return cir;
```

```
}
```

Output:

Enter radius 5

the area of circle 78.500000

Parameter Passing Methods (or) Passing Arguments to Function

- Function is a good programming style in which we can write reusable code that can be called whenever required.
- Whenever we call a function, the sequence of executable statements gets executed. We can pass some of the information (or) data to the function for processing is called a parameter.
- In 'C' Language there are two ways a parameter can be passed to a function. They are
 - **Call by value**
 - **Call by reference**

Call by Value:

- This method copies the value of the actual parameter to the formal parameter of the function.
- Here, the changes of the formal parameters cannot affect the actual parameters, because formal parameter are photocopies of the actual parameter.
- The changes made in formal arguments are local to the block of the called function. Once control returns back to the calling function the changes made disappears.

Example Program

```
#include<stdio.h>
#include<conio.h>
void cube(int);
int cube1(int);
void main()
{
    int a;
    clrscr();
    printf("Enter one values");
    scanf("%d",&a);
```

Output:

```
Enter one values 3
Value of cube function is 3
Value of cube1 function is 27
```

```

        printf("Value of cube function is=%d", cube(a));
        printf("Value of cube1 function is =%d", cube1(a
        )); getch();
    }
    void cube(int x)
    {
        x=x*x*x;
        return x;
    }
    int cube1(int x)
    {
        x=x*x*x;
        return x;
    }

```

Call by reference

- Call by reference is another way of passing parameter to the function.
- Here the address of the argument is copied into the parameter inside the function, the address is used to access arguments used in the call.
- Hence, changes made in the arguments are permanent.
- Here pointer is passed to function, just like any other arguments.

Example Program

```

#include<stdio.h>
#include<conio.h>
void swap(int,int);
void main()
{
    int a=5,b=10;
    clrscr();
    printf("Before swapping a=%d b=%d",a,b);

```

Output:

Before swapping a=5 b=10
After swapping a=10 b=5

```

        swap(&a,&b);
        printf("After swapping a=%d
        b=%d",a,b); getch();
    }
    void swap(int *x,int *y)
    {
        int *t;
        t=*x;
        *x=*y;
        *y=t;
    }

```

Nesting of function call in c programming

If we are calling any function inside another function call, then it is known as Nesting function call. In other words, a function calling different functions inside is termed as Nesting Functions.

Example:

// C program to find the factorial of a number.

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

```

    //Nesting of functions
    //calling function inside another
    function //calling fact inside
    print_fact_table function

```

```
void print_fact_table(int);           // function declaration
```

```
int fact(int);                       // function declaration
```

```
void main()                          // main function
```

```

{
    print_fact_table(5);              // function call
}

```

```
void print_fact_table(int n)          // function definition
```

```

{
    int i;
    for (i=1;i<=n;i++)

```

```

    printf("%d factorial is %d\n",i,fact(i)); //fact(i)-- function call
}

int fact(int n)                                // function definition
{
    if (n == 1)
        return 1;
    else
        return n * fact(n-1);
}

```

Output:

```

1 factorial is 1
2 factorial is 2
3 factorial is 6
4 factorial is 24
5 factorial is 120

```

Recursion

A function calling same function inside itself is called as recursion.

Example: // C program to find the factorial of a number.

```

#include <stdio.h>
int fact(int); // function declaration
void main()    // main function
{
    printf("Factorial =%d",fact(5)); // fact(5) is the function call
}
int fact(int n)    // function definition
{
    if (n==1) return 1; else
    return n * fact(n-1); // fact(n-1) is the recursive function call
}

```

Output:

```

Factorial = 120

```

Discussion:

For 1!, the function returns 1, for other values, it executes like the one below:

When the value is 5, it comes to else part and calculates like this,

$$\begin{aligned} &= 5 * \text{fact}(5-1) &&= 5 * \text{fact}(4) \\ &= 5 * 4 * \text{fact}(4-1) &&= 5 * 4 * \text{fact}(3) \\ &= 5 * 4 * 3 * \text{fact}(3-1) &&= 5 * 4 * 3 * \text{fact}(2) \\ &= 5 * 4 * 3 * 2 * \text{fact}(2-1) &&= 5 * 4 * 3 * 2 * \text{fact}(1) \\ &= 5 * 4 * 3 * 2 * 1 \quad (\text{if } (n==1) \text{ then return } 1, \text{ hence we get } 1) \\ &= 120 \end{aligned}$$

ARRAYS

Introduction:

So far we have used only single variable name for storing one data item. If we need to store multiple copies of the same data then it is very difficult for the user. To overcome the difficulty a new data structure is used called arrays.

An array is a linear and homogeneous data structure

An array permits homogeneous data. It means that similar types of elements are stored contiguously in the memory under one variable name.

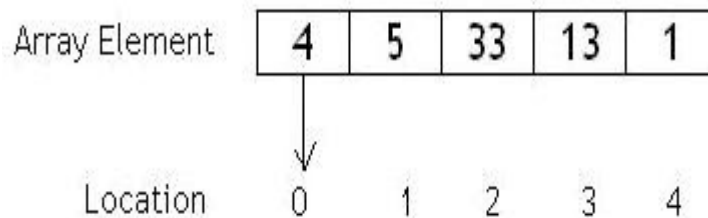
An array can be declared of any standard or custom data type.

Example of an Array:

Suppose we have to store the roll numbers of the 100 students then we have to declare 100 variables named as roll1, roll2, roll3, roll100 which is a very difficult job. Concept of C programming arrays is introduced in C which gives the capability to store the 100 roll numbers in the contiguous memory which has 100 blocks and which can be accessed by single variable name.

- ☐ C Programming Arrays is the **Collection of Elements**
- ☐ C Programming Arrays is collection of the Elements of the **same data type**.
- ☐ All Elements are stored in the **Contiguous memory**
- ☐ All elements in the array are accessed using the subscript variable (index).

Pictorial representation of C Programming Arrays



The above array is declared as `int a [5];`

a[0] = 4; a[1] = 5; a[2] = 33; a[3] = 13; a[4] = 1;

In the above figure 4, 5, 33, 13, 1 are actual data items. 0, 1, 2, 3, 4 are index variables.

Index or Subscript Variable:

1. Individual data items can be accessed by the name of the array and an integer enclosed in square bracket called subscript variable / index
2. Subscript Variables helps us to identify the item number to be accessed in the contiguous memory.

Characteristics of an array:

2. The declaration `int a [5]` is nothing but creation of five variables of integer types in memory instead of declaring five variables for five values.
3. All the elements of an array share the same name and they are distinguished from one another with the help of the element number.
4. The element number in an array plays a major role for calling each element.
5. Any particular element of an array can be modified separately without disturbing the other elements.
6. Any element of an array `a[]` can be assigned or equated to another ordinary variable or array variable of its type.
7. Array elements are stored in contiguous memory locations.

Array Declaration:

Array has to be declared before using it in C Program. Array is nothing but the collection of elements of similar data types.

Syntax: <data type> array name [size1][size2].....[sizen];

Syntax Parameter	Significance
Data type	Data Type of Each Element of the array
Array name	Valid variable name
Size	Dimensions of the Array

Array declaration requirements

Requirement	Explanation
Data Type	Data Type specifies the type of the array. We can compute the size required for storing the single cell of array.
Valid Identifier	Valid identifier is any valid variable or name given to the array. Using this identifier name array can be accessed.

Size of Array	It is maximum size that array can have.
---------------	---

What does Array Declaration tell to Compiler?

- ☐ Type of the Array
- ☐ Name of the Array
- ☐ Number of Dimension
- ☐ Number of Elements in Each Dimension

Types of Array

- ☐ **Single Dimensional Array / One Dimensional Array**
- ☐ **Multi Dimensional Array**

Single / One Dimensional Array:

- ☐ Single or One Dimensional array is used to represent and store data in a linear form.
- ☐ Array having only one subscript variable is called **One-Dimensional array**
- ☐ It is also called as **Single Dimensional Array** or **Linear Array**

Single Dimensional Array Declaration and initialization:

Syntax for declaration: <data type> <array name> [size];

Examples for declaration: `int iarr[3]; char carr[20]; float farr[3];`

Syntax for initialization: <data type> <array name> [size] = {val1, val2, ..., valn};

Examples for initialization: `int`

`iarr[3] = {2, 3, 4};`

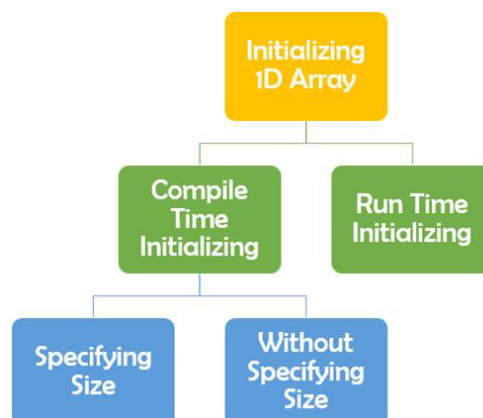
`char carr[20] = "program"; float`

`farr[3] = {12.5, 13.5, 14.5};`

Different Methods of Initializing 1-D Array

Whenever we declare an array, we initialize that array directly at compile time.

Initializing 1-D Array is called as compiler time initialization if and only if we assign certain set of values to array element before executing program. i.e. at compilation time.



Example Program

```
#include <stdio.h>

int main()
{
    int num[] = {2,8,7,6,0};
    int i;
    for (i=0;i<5;i++) {
        printf("\n Array Element num [%d] = %d",i, num[i]);
    }
    return 0;
}
```

Output:

```
Array Element num[0] = 2
Array Element num[1] = 8
Array Element num[2] = 7
Array Element num[3] = 6
Array Element num[4] = 0
```

Accessing Array

- We all know that array elements are randomly accessed using the subscript variable.
- Array can be accessed using array-name and subscript variable written inside pair of square brackets [].

Consider the below example of an array

51	32	43	24	5	26
2001	2003	2005	2007	2009	2011

In this example we will be accessing array like this

arr[3] = Forth Element of Array

arr[5] = Sixth Element of Array

whereas elements are assigned to an array using below way

```
arr[0] = 51;   arr[1] = 32;   arr[2] = 43;   arr[3] = 24;   arr[4] = 5;   arr[5] = 26;
```

Example Program1: Accessing array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int arr[] = {51,32,43,24,5,26};
int i;
for(i=0; i<=5; i++) {
printf("\nElement at arr[%d] is %d",i,arr[i]);
}
getch();
}
```

Output:

Element at arr[0] is 51

Element at arr[1] is 32

Element at arr[2] is 43

Element at arr[3] is 24

Element at arr[4] is 5

Element at arr[5] is 26

How a[i] Works?

We have following array which is declared like `int arr[] = { 51,32,43,24,5,26};`

As we have elements in an array, so we have track of base address of an array. Below things are important to access an array.

Expression	Description	Example
arr	It returns the base address of an array	Consider 2000
*arr	It gives zeroth element of an array	51

Expression	Description	Example
*(arr+0)	It also gives zeroth element of an array	51
*(arr+1)	It gives first element of an array	32

So whenever we tried accessing array using arr[i] then it returns an element at the location*(arr + i)

Accessing array a[i] means retrieving element from address (a + i).

Example Program2: Accessing array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int arr[] =
{51,32,43,24,5,26}; int i;
for(i=0; i<=5; i++) {
printf("\n%d %d %d %d",arr[i],*(i+arr),*(arr+i),i[arr]);
}
getch();
}
```

Output:

```
51 51 51 51
32 32 32 32
43 43 43 43
24 24 24 24
5 5 5 5
26 26 26 26
```

Operations with One Dimensional Array

- ☐ Deletion – Involves deleting specified elements form an array.
- ☐ Insertion – Used to insert an element at a specified position in an array.
- ☐ Searching – An array element can be searched. The process of seeking specific elements in an array is called searching.

5. Merging – The elements of two arrays are merged into a single one.
6. Sorting – Arranging elements in a specific order either in ascending or in descending order.

Example Programs:

1. C Program for deletion of an element from the specified location from an Array

```
#include<stdio.h>
int main() {
int arr[30], num, i, loc;
printf("\nEnter no of
elements:"); scanf("%d", &num);
//Read elements in an array
printf("\nEnter %d elements :",
num); for (i = 0; i < num; i++) {
scanf("%d", &arr[i]);    }
//Read the location
printf("\nLocation of the element to be deleted
:"); scanf("%d", &loc);
/* loop for the deletion */
while (loc < num) {
arr[loc - 1] =
arr[loc]; loc++; }
num--; // No of elements reduced by 1
//Print Array
for (i = 0; i < num; i++)
printf("\n %d", arr[i]);
return (0);
}
```

Output:

```
Enter no of elements: 5
Enter 5 elements: 3 4 1 7 8
```

Location of the element to be deleted: 3

3 4 7 8

2. C Program to delete duplicate elements from an array

```
int main() {
    int arr[20], i, j, k, size;
    printf("\nEnter array size:
"); scanf("%d", &size);
    printf("\nAccept Numbers: ");
    for (i = 0; i < size; i++)
        scanf("%d", &arr[i]);
    printf("\nArray with Unique list:
"); for (i = 0; i < size; i++) {
        for (j = i + 1; j < size; j++) {
            if (arr[j] == arr[i]) {
                for (k = j; k < size; k++) {
                    arr[k] = arr[k + 1]; } size--;
            }
        }
        j++;
    }
    for (i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    return (0);
}
```

Output:

Enter array size: 5

Accept Numbers: 1 3 4 5 3

Array with Unique list: 1 3 4 5

3. C Program to insert an element in an array

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

```
int main() {
```



```

int arr[30], element, num, i,
location; printf("\nEnter no of
elements:"); scanf("%d", &num);
for (i = 0; i < num; i++)
{ scanf("%d", &arr[i]); }
printf("\nEnter the element to be
inserted:"); scanf("%d", &element);
printf("\nEnter the location");
scanf("%d", &location);
//Create space at the specified location
for (i = num; i >= location; i--) {
arr[i] = arr[i - 1]; }
num++;
arr[location - 1] = element;
//Print out the result of insertion
for (i = 0; i < num; i++)
printf("n %d",
arr[i]); return (0);
}

```

Output:

Enter no of elements:

5 1 2 3 4 5

Enter the element to be inserted: 6

Enter the location: 2

1 6 2 3 4 5

4. C Program to search an element in an array

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

```
int main() {
```

```
int a[30], ele, num, i;
```

```
printf("\nEnter no of elements:");
```

```
scanf("%d", &num);
```

```

printf("\nEnter the values :");
for (i = 0; i < num; i++) {
scanf("%d", &a[i]);    }
    //Read the element to be searched
printf("\nEnter the elements to be searched
:"); scanf("%d", &ele);
//Search starts from the zeroth
location i = 0;
while (i < num && ele != a[i])
{ i++; }
//If i < num then Match
found if (i < num) {
printf("Number found at the location = %d", i + 1);
}
else {
printf("Number not found");
} return (0);
}

```

Output:

Enter no of elements:

5 11 22 33 44 55

Enter the elements to be searched:

44 Number found at the location = 4

5. C Program to copy all elements of an array into another array

```

#include<stdio.h>
int main() {
int arr1[30], arr2[30], i, num;
printf("\nEnter no of
elements:"); scanf("%d", &num);
//Accepting values into Array
printf("\nEnter the values:");

```

```

for (i = 0; i < num; i++) {
scanf("%d", &arr1[i]);    }

/* Copying data from array 'a' to array 'b */
for (i = 0; i < num; i++) {
arr2[i] = arr1[i];    }

//Printing of all elements of array
printf("The copied array is:");
for (i = 0; i < num; i++)
printf("\narr2[%d] = %d", i, arr2[i]);
return (0);
}

```

Output:

Enter no of elements: 5

Enter the values: 11 22 33 44 55

The copied array is: 11 22 33 44 55

6. C program to merge two arrays in C Programming

```

#include<stdio.h>

int main() {

int arr1[30], arr2[30],
res[60]; int i, j, k, n1, n2;

printf("\nEnter no of elements in 1st
array:"); scanf("%d", &n1);

for (i = 0; i < n1; i++) {
scanf("%d", &arr1[i]); }

printf("\nEnter no of elements in 2nd
array:"); scanf("%d", &n2);

for (i = 0; i < n2; i++)
{ scanf("%d", &arr2[i]);
} i = 0;

j = 0;

k = 0;

```

```

// Merging starts
while (i < n1 && j < n2)
{ if (arr1[i] <= arr2[j])
{ res[k] = arr1[i];
i++;
k++; }
else {
res[k] =
arr2[j]; k++;
j++; }
}

/*Some elements in array 'arr1' are still remaining where as the array
'arr2' is exhausted*/
while (i < n1) {
res[k] =
arr1[i]; i++;
k++; }

/*Some elements in array 'arr2' are still remaining where as the array
'arr1' is exhausted */
while (j < n2) {
res[k] =
arr2[j]; k++;
j++; }

//Displaying elements of array
'res' printf("\nMerged array is:");
for (i = 0; i < n1 + n2;
i++) printf("%d ", res[i]);
return (0);
}

Enter no of elements in 1st array:
4 11 22 33 44

```

Enter no of elements in 2nd array: 3

10 40 80

Merged array is: 10 11 22 33 40 44 80

Bisection Method

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
float f,x,a,b;
printf("Enter the value of a ::");
scanf("%f",&a);
printf("Enter the value of b ::");
scanf("%f",&b);
do
{
x = (a+b)/2.00;
f = (x*x*x)-(x)-1; // any equation can be put here
if (f>0)
b=x;
else
a=x;
printf("\n\n\t a = %f  b = %f  f = %f",a,b,f);
}while(b-a>0.0001);
printf("\n\t The Root of the equation is %f",x);
getch();
}

/* Output */
```

```

Turbo C++ IDE
Enter the value of a ::1
Enter the value of b ::2

a = 1.000000    b = 1.500000    f = 0.875000
a = 1.250000    b = 1.500000    f = -0.296875
a = 1.250000    b = 1.375000    f = 0.224609
a = 1.312500    b = 1.375000    f = -0.051514
a = 1.312500    b = 1.343750    f = 0.082611
a = 1.312500    b = 1.328125    f = 0.014576
a = 1.320312    b = 1.328125    f = -0.018711
a = 1.324219    b = 1.328125    f = -0.002128
a = 1.324219    b = 1.326172    f = 0.006209
a = 1.324219    b = 1.325195    f = 0.002037
a = 1.324707    b = 1.325195    f = -0.000047
a = 1.324707    b = 1.324951    f = 0.000995
a = 1.324707    b = 1.324829    f = 0.000474
a = 1.324707    b = 1.324768    f = 0.000214
The Root of the equation is 1.324768

```

Newton Raphson Method In C Programming

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int user_power,i=0,cnt=0,flag=0;
int coef[10]={0};
float x1=0,x2=0,t=0;
float fx1=0,fdx1=0;
void main()
{
    clrscr();
    printf("\n\n\t\t\t PROGRAM FOR NEWTON RAPHSON GENERAL");
    printf("\n\n\n\t ENTER THE TOTAL NO. OF POWER:::: ");
    scanf("%d",&user_power);
    for(i=0;i<=user_power;i++)
    {
        printf("\n\t x^%d::",i);
        scanf("%d",&coef[i]);
    }
    printf("\n");
    printf("\n\t THE POLYNOMIAL IS ::: ");
    for(i=user_power;i>=0;i--)//printing coeff.
    {
        printf(" %dx^%d",coef[i],i);
    }
    printf("\n\tINTIAL X1---->");
    scanf("%f",&x1);
    printf("\n *****");
    printf("\n ITERATION  X1  FX1  F'X1 ");
}
```

```

printf("\n
*****");
do
{
    cnt++;
    fx1=fdx1=0;
    for(i=user_power;i>=1;i--)
    {
        fx1+=coef[i] * (pow(x1,i)) ;
    }
    fx1+=coef[0];
    for(i=user_power;i>=0;i--)
    {
        fdx1+=coef[i]* (i*pow(x1,(i-1)));
    }
    t=x2;
    x2=(x1-(fx1/fdx1));
    x1=x2;
    printf("\n %d      %.3f %.3f %.3f ",cnt,x2,fx1,fdx1);

}while((fabs(t - x1))>=0.0001);
printf("\n\t THE ROOT OF EQUATION IS %f",x2);
getch();
}
/*****OUTPUT*****/
*****/

```

```

Turbo C++ IDE

PROGRAM FOR NEWTON RAPHSON GENERAL

ENTER THE TOTAL NO. OF POWER::: 4
x^0:: -10
x^1:: -1
x^2:: 0
x^3:: 0
x^4:: 1

THE POLYNOMIAL IS ::: 1x^4 0x^3 0x^2 -1x^1 -10x^0
INITIAL x1---->2

*****
ITERATION  x1      FX1      F'X1
*****
1          1.871    4.000    31.000
2          1.856    0.383    25.197
3          1.856    0.005    24.565
4          1.856    0.000    24.557
THE ROOT OF EQUATION IS 1.855585_

```

Lagrange's Interpolation Method For Finding F(X) In C Programming

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float x[10],y[10],temp=1,f[10],sum,p;
    int i,n,j,k=0,c;
    clrscr();
    printf("\nhow many record you will be enter: ");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("\n\nenter the value of x%d: ",i);
        scanf("%f",&x[i]);
        printf("\n\nenter the value of f(x%d): ",i);
        scanf("%f",&y[i]);
    }
    printf("\n\nEnter X for finding f(x): ");
    scanf("%f",&p);

    for(i=0;i<n;i++)
    {
        temp = 1;
        k = i;
        for(j=0;j<n;j++)
        {
            if(k==j)
            {
                continue;
            }
            else
            {
                temp = temp * ((p-x[j])/(x[k]-x[j]));
            }
        }
        f[i]=y[i]*temp;
    }
}
```

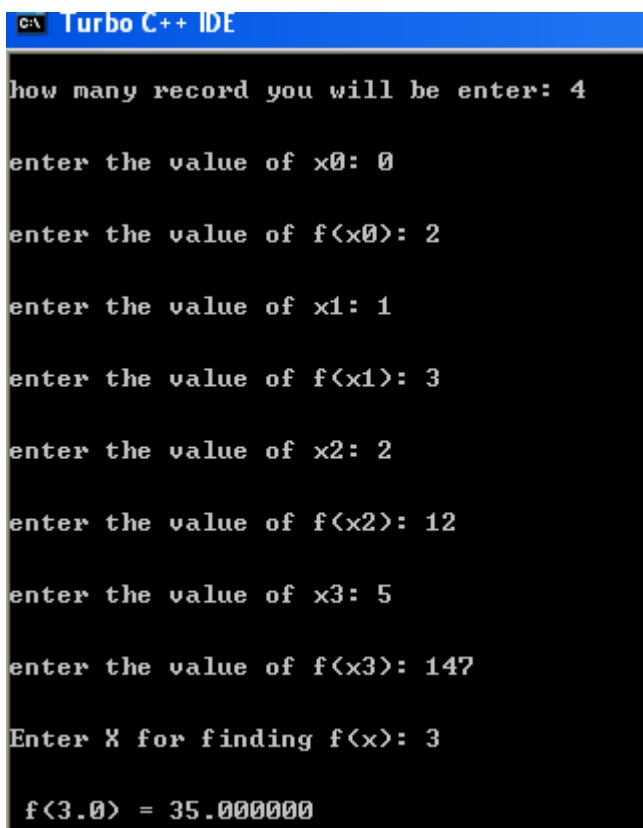


```

for(i=0;i<n;i++)
{
    sum = sum + f[i];
}
printf("\n\n f(%.1f) = %f ",p,sum);
getch();
}

```

/*OUTPUT



The screenshot shows the Turbo C++ IDE window with a black background and white text. The output of the program is as follows:

```

how many record you will be enter: 4

enter the value of x0: 0

enter the value of f(x0): 2

enter the value of x1: 1

enter the value of f(x1): 3

enter the value of x2: 2

enter the value of f(x2): 12

enter the value of x3: 5

enter the value of f(x3): 147

Enter X for finding f(x): 3

f(3.0) = 35.000000

```

.....

Code for SIMPSON'S 1/3 RULE in C Programming

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float x[10],y[10],sum=0,h,temp;

```

```

int i,n,j,k=0;
float fact(int);
clrscr();
printf("\nhow many record you will be enter: ");
scanf("%d",&n);
for(i=0; i<n; i++)
{
    printf("\n\nenter the value of x%d: ",i);
    scanf("%f",&x[i]);
    printf("\n\nenter the value of f(x%d): ",i);
    scanf("%f",&y[i]);
}
h=x[1]-x[0];
n=n-1;
sum = sum + y[0];
for(i=1;i<n;i++)
{
    if(k==0)
    {
        sum = sum + 4 * y[i];
        k=1;
    }
    else
    {
        sum = sum + 2 * y[i];
        k=0;
    }
}
sum = sum + y[i];
sum = sum * (h/3);
printf("\n\n I = %f ",sum);
getch();
}

```

/*_OUTPUT_____

how many record you will be enter: 5

enter the value of x0: 0

enter the value of f(x0): 1

enter the value of x1: 0.25

enter the value of f(x1): 0.8

enter the value of x2: 0.5

enter the value of f(x2): 0.6667

enter the value of x3: 0.75

enter the value of f(x3): 0.5714

enter the value of x4: 1

enter the value of f(x4): 0.5

$I = 0.693250$

TRAPEZOIDAL RULE in C Programming

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float x[10],y[10],sum=0,h,temp;
    int i,n,j,k=0;
    float fact(int);
    clrscr();
    printf("\nhow many record you will be enter: ");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
```

```

printf("\n\nenter the value of x%d: ",i);
scanf("%f",&x[i]);
printf("\n\nenter the value of f(x%d): ",i);
scanf("%f",&y[i]);
}
h=x[1]-x[0];
n=n-1;
for(i=0;i<n;i++)
{
    if(k==0)
    {
        sum = sum + y[i];
        k=1;
    }
    else
        sum = sum + 2 * y[i];
}
sum = sum + y[i];
sum = sum * (h/2);
printf("\n\n I = %f ",sum);
getch();
}

/*

```

OUT PUT

how many record you will be enter: 6

enter the value of x0: 7.47

enter the value of f(x0): 1.93

enter the value of x1: 7.48

enter the value of f(x1): 1.95

enter the value of x2: 7.49

enter the value of f(x2): 1.98

enter the value of x3: 7.50

enter the value of f(x3): 2.01

enter the value of x4: 7.51

enter the value of f(x4): 2.03

enter the value of x5: 7.52

enter the value of f(x5): 2.06

I = 0.099652

Euler's Method in C:

```
#include<stdio.h>
float fun(float x,float y)
{
    float f;
    f=x+y;
    return f;
}
main()
{
    float a,b,x,y,h,t,k;
    printf("\nEnter x0,y0,h,xn: ");
    scanf("%f%f%f%f",&a,&b,&h,&t);
    x=a;
    y=b;
    printf("\n  x\t y\n");
    while(x<=t)
    {
        k=h*fun(x,y);
        y=y+k;
        x=x+h;
        printf("%0.3ft%0.3fn",x,y);
    }
}
```

}

SPH5107 – NUMERICAL METHODS AND COMPUTER PROGRAMMING

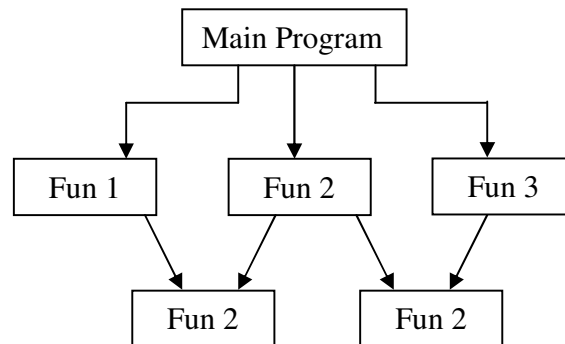
UNIT - 5

NEED FOR OBJECT ORIENTED PROGRAM:

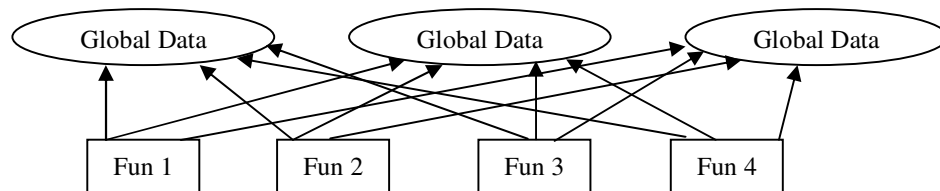
Procedure Oriented Programming (POP)

The high level languages, such as BASIC, COBOL, C, FORTRAN are commonly known as Procedure Oriented Programming.

Using this approach, the problem is viewed in sequence of things to be done, like reading, processing and displaying or printing. To carry out these tasks the function concepts must be used.



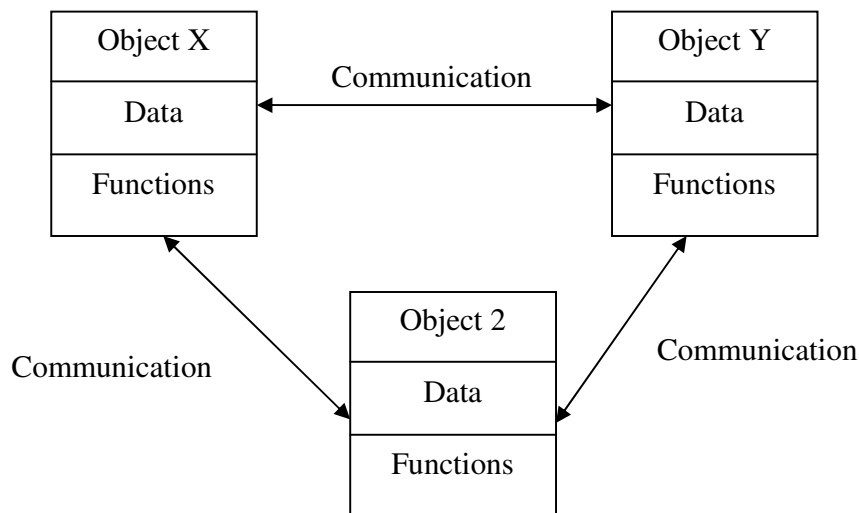
- ◆ This concept basically consists of number of statements and these statements are organized or grouped into functions.
- ◆ While developing these functions the programmer must care about the data that is being used in various functions.
- ◆ A multi-function program, the data must be declared as global, so that data can be accessed by all the functions within the program & each function can also have its own data called local data.



- ◆ Th global data can be accessed anywhere in the program. In large program it is very difficult to identify what data is accessed by which function. In this case we must revised about the external data and as well as the functions that access the global data. At this situation there is so many chances for an error.

OBJECT ORIENTED PROGRAMMING (OOP)

- ◆ This programming approach is developed to reduce the some of the drawbacks encountered in the Procedure Oriented Programming Approach.
- ◆ The OO Programming approach treats data as critical element and does not allow the data to freely around the program.
- ◆ It bundles the data more closely to the functions that operate on it; it also protects data from the accidental modification from outside the function.
- ◆ The object oriented programming approach divides the program into number of entities called objects and builds the data and functions that operates on data around the objects.
- ◆ The data of an object can only access by the functions associated with that object.



Difference between C & C++

S.No	Procedure oriented Programming (C)	Object Oriented Programming (C++)
1.	Programs are divided into smaller sub-programs known as functions	Programs are divided into objects & classes
2.	Here global data is shared by most of the functions	Objects are easily communicated with each other through function.
3.	It is a Top-Down Approach	It is a Bottom-Up Approach
4.	Data cannot be secured and available to all the function	Data can be secured and can be available in the class in which it is declared
5.	Here, the reusability is not possible; hence redundant code cannot be avoided.	Here, we can reuse the existing one using the Inheritance concept

Benefits or Advantages of OOPS

- ◆ The complexity of software can be managed easily.
- ◆ Data hiding concept help the programmer to build secure programs
- ◆ Through the class concept we can define the user defined data type
- ◆ The inheritance concept can be used to eliminate redundant code
- ◆ The message-passing concept helps the programmer to communicate between different objects.
- ◆ New data and functions can be easily added whenever necessary.
- ◆ OOPS ties data elements more closely to the functions that operates on.

Basics of C++ Programming

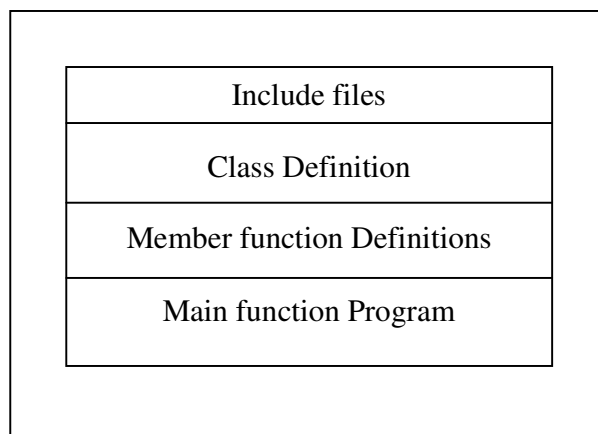
C++ was developed by BJARNE STROUSSTRUP at AT&T BELL Laboratories in Murry Hill, USA in early 1980's.

Strousstrup combines the features of 'C' language and 'SIMULA67' to create more powerful language that support OOPS concepts, and that language was named as "C with CLASSES". In late 1983, the name got changed to C++.

The idea of C++ comes from 'C' language increment operator (++) means more additions.

C++ is the superset of 'C' language, most of the 'C' language features can also applied to C++, but the object oriented features (Classes, Inheritance, Polymorphism, Overloading) makes the C++ truly as Object Oriented Programming language.

Structure of C++ Program



Include files provides instructions to the compiler to link functions from the system library.

Eg: `#include <iostream.h>`

#include – Preprocessor Directive
iostream.h – Header File

- ◆ A class is a way to bind and its associated functions together. It is a user defined datatype. It must be declared at class declaration part.
- ◆ Member function definition describes how the class functions are implemented. This must be the next part of the C++ program.
- ◆ Finally main program part, which begins program execution.

```
main( )  
{  
  
}
```

Program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program.

Input / Output statements

Input Stream

Syntax:

```
cin >> var1 >> var2 >>;
```

cin – Keyword, it is an object, predefined in C++ to correspond to the standard input stream.

>> - is the extraction or get from operator

Extraction operation (>>) takes the value from the stream object on its left and places it in the variable on its right.

Eg:

```
cin>>x;  
cin>>a>>b>>c;
```

Output Stream:

Syntax:

```
cout<<var1<<var2;
```

cout - object of standard output stream

<< - is called the insertion or put to operator

It directs the contents of the variable on its right to the object on its left.

Output stream can be used to display messages on output screen.

Eg:

```
cout<<a<<b;
```

```
cout<<"value of x is"<<x;
cout<<"Value of x is"<<x<<"less than"<<y;
```

Tokens

The smallest individual units in a program are known as tokens.
C++ has the following tokens

- ◆ Keywords
- ◆ Identifiers
- ◆ Constants
- ◆ Strings
- ◆ Operators

Keywords

- It has a predefined meaning and cannot be changed by the user
- Keywords cannot be used as names for the program variables.

Keywords supported by C++ are:

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

The specific C++ Keywords

There are several keywords specific to C++

asn	new	template
catch	operator	this
class	private	throw
delete	protected	try
friend	public	virtual
inline		

Identifiers

Identifiers refer to the names of variables, functions, arrays, classes, etc. created by the programmer.

Rules for naming these identifiers:

1. Only alphabetic characters, digits and underscores are permitted.
2. The name cannot start with a digit.
3. Uppercase and lowercase letters are distinct.
4. A declared keyword cannot be used as a variable name.

(i) Variables:

It is an entity whose value can be changed during program execution and is known to the program by a name.

A variable can hold only one value at a time during program execution.

Eg:

Allowable variable names

i
sum
A_B
A-1B

Invalid names

1_B – 1st letter must be alphabet
\$xy – 1st letter must be alphabet
x+b – special symbol '+' not allowed

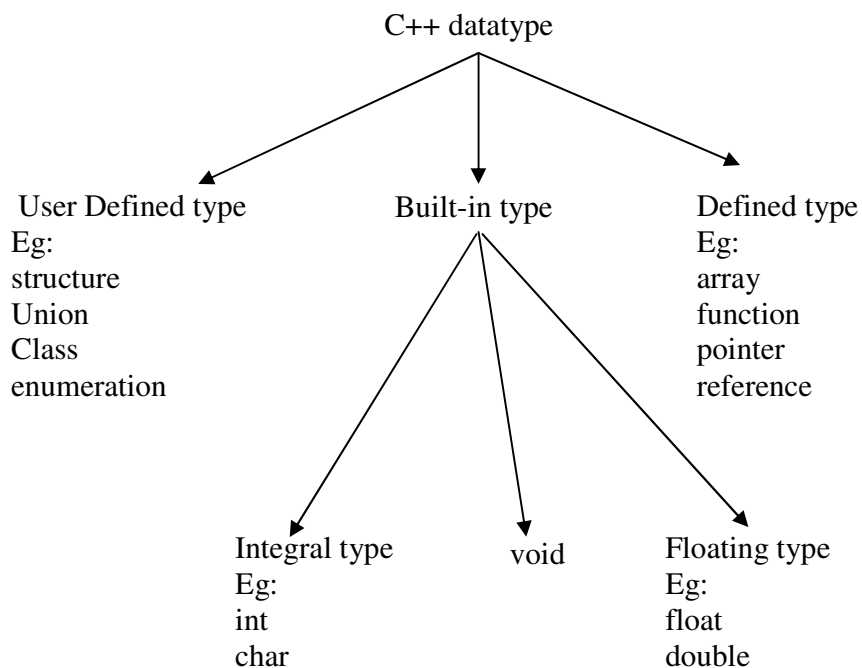
Declaration of Variables

Syntax

datatype variablename;

Datatype

It is the type of data, that is going to be processed within the program



Eg:

```
int x;  
float y,z;  
char a;
```

A variable can be declared anywhere in the program before its first use.

Constants

A quantity that does not change is known as constants.

Types of constants:

- Integer constants - Eg: 123, 25 – without decimal point
- Character constants - Eg: 'A', 'B', '*', '1'
- Real constants - Eg: 12.3, 2.5 - with decimal point

Strings

A sequence of characters is called string. String constants are enclosed in double quotes as follows

“Hello”

Operators

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.

Types of Operators

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment & decrement Operators
6. Conditional Operators
7. Bitwise Operators
8. Special Operators
9. Manipulators
10. Memory allocate / delete Operators

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

Arithmetic Operators

C++ has both unary & binary arithmetic operators.

- Unary operators are those, which operate on a single operand.
- Whereas, binary operators on two operands +, -, *, /, %

Examples for Unary Operators:

1. `int x = 10;`
 `y = -x;` (The value of x after negation is assigned to y ie. y becomes -5.)
2. `int x = 5;`
 `sum = -x;`

Examples for Binary Operators:

1. `int x = 16, y=5;`
 `x+y = 21;` (result of the arithmetic expression)
 `x-y = 11;`
 `x*y=80;`

/ - Division Operator

Eg:

`x = 10, y = 3;`
`x/y=3;` (The result is truncated, the decimal part is discarded.)

% - Modulo Division

The result is the remainder of the integer division only applicable for integer values.

`x=11, y = 2`
`x%y = 1`

Relational Operators

- A relational operator is used to make comparison between two expressions.
 - All relational operators are binary and require two operands.
- <, <=, >, >=, ++, !=

Relational Expression

Expression1 relational operator Expression2


Expression1 & 2 – may be either constants or variables or arithmetic expression.

Eg:

$a < b$ (Compares its left hand side operand with its right hand side operand)
 $10 == 15$
 $a != b$

- An relational expression is always return either zero or 1, after evaluation.

Eg: $(a+b) \leq (c+d)$


 arithmetic expression

Here relational operator compares the relation between arithmetic expressions.

Logical Operators

$\&\&$ - Logical AND
 $!!$ - Logical OR
 $!$ - Logical NOT

Logical operators are used when we want to test more than one condition and make decisions.

Eg: $(a < b) \&\& (x == 10)$

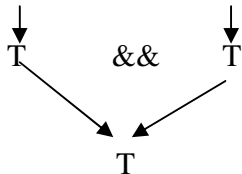
An expression of this kind, which combines two or more relational expressions, is termed as a logical expression.

Like simple relational expressions, a logical expression also yields a value of one or zero, according to the truth table.

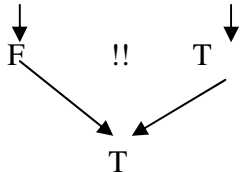
Operand 1	Operand 2	AND	OR	NOT OP1	NOT OP2
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

Eg:

$((10 > 5) \&\& (3 < 13))$



$((11 < 3) !! (10 != 5))$



Assignment Operators

Assignment operators are used to assign the result of an expression to a variable.

Eg: a = 10;
 a = a + b;
 x = y;

Variable operator = operand

OP = is called as shorthand assignment operator.

VOP = exp

is equivalent to v = v op exp

Eg: x+=y; \implies x = x+y;

Increment & Decrement Operators

- Increment ++, this operator adds 1 to the operand
- Decrement --, this operator subtracts 1 from the operand
- Both are unary operators

Eg:

m = 5;
y = ++m; (adds 1 to m value)
x = --m; (Subtracts 1 from the value of m)

Types

- Pre Increment / Decrement OP:
- Post Increment / Decrement OP:

If the operator precedes the operand, it is called pre increment or pre decrement.

Eg: ++i, --i;

If the operator follows the operand, it is called post increment or post decrement.

Eg: ++i, --i;

In the pre Increment / Decrement the operand will be altered in value before it is utilized for its purpose within the program.

Eg: x = 10;
 Y = ++x;

- 1st x value is getting incremented with 1.
- Then the incremented value is assigned to y.

In the post Increment / Decrement the value of the operand will be altered after it is utilized.

Eg: y = 11;
 x = y++;

- 1st x value is getting assigned to x & then the value of y is getting increased.

Conditional Operator

? :

General Form is

Conditional exp ? exp 1 : exp 2;

Conditional exp - either relational or logical expression is used.

Exp1 & exp 2 : are may be either a variable or any statement.

Eg:

(a>b)?a:b;

- Conditional expression is evaluated first.
- If the result is '1' is true, then expression1 is evaluated.
- If the result is zero, then expression2 is evaluated.

Eg: lar = (10>5)?10:5;

Bitwise Operators - Used to perform operations in bit level

Operators used:

&	-	Bitwise AND
	-	Bitwise OR
^	-	Exclusive OR
<<	-	Left shift
>>	-	Right shift
~	-	One's complement

Special Operators

- sizeof
- comma(,)
- size of operators returns the size the variable occupied from system memory.

Eg:

```
var = sizeof(int)
cout<<var; Ans: 2
```

```
x = size of (float);
cout << x; Ans: 4
```

```
int y;
x = sizeof (y);
cout<<y; Ans: 2
```

Precedence of Operators

Name	Operators	Associativity
Unary Operators	-, ++, --, !, sizeof	R → L
Mul, div & mod	*, /, %	L → R
Add, Sub	+, -	L → R
Relational	<, <=, >, >=	L → R
Equality	=, !=	L → R
Logical AND	&&	L → R
Logical OR		L → R

Example

```

x      =10, y = 2m z = 10.5
a      = (x+y) – (x/y)*z;
        = 12 – 5 * 10.5;
        = 12 – 52.5;
a      = -42.5

```

Manipulators

Manipulators are operators used to format the data display. The commonly used manipulators are endl, setw.

endl manipulators

It is used in output statement causes a line feed to be inserted, it has the same effect as using the newline character “\n” in ‘C’ language.

```

#include <iostream.h>
main()
{
    int a=10, b=20;
    cout << “C++ language” << endl;
    cout << “A value: “ << a << endl;
    cout << “B value:” << b << endl;
}

```

O/P:

```

C++ language
A value: 10
B value: 20

```

Setw Manipulator

The setw manipulator is used to specify the field width for printing, the content of the variable.

Syntax: **setw(width);**
where width specifies the field width.

```
int a = 10;  
cout << " A value" << setw(5) << a << endl;
```

Output:

A value 10

Note: Remaining operators will be discussed in 4th unit.

Symbolic Constant

Symbolic constants are constants to which symbolic names are associated for the purpose of readability and ease of handling.

- #define preprocessor directive
- const keyword
- enumerated data type

#define preprocessor directive

It associates a constant value to a symbol and is visible throughout the function in which it is defined.

Syntax:

#define symbol name constant value

Eg

```
#define max_value 100  
#define pi 3.14
```

The value of symbolic constant will not change throughout the program.

const keyword

Syntax:

const datatype var = constant;

Eg:

```
const int max = 100;  
main()  
{  
    char x[max];
```

```

    }
const size = 10;

```

Allowable statement, default symbolic constant type is integer. Here size is of type int.

Reference Variable:

A reference variable provides an alias (alternative name) for a previously defined variable.

For example, if we make the variable sum a reference to the variable total, then sum & total can be used interchangeably to represent that variable.

A reference variable is created as follows:

datatype & ref_name = var_name;

Eg:

```

float total = 100;
float & sum = total;
cout << sum << total;

```

Both the variables refer to the same data object in the memory i.e.

total, sum
100

Type Conversion:

- (i) Implicit type conversion
- (ii) Explicit type conversion

Implicit type conversion

It will be done by the compiler, by following the rule of lower type converted to higher type.

Eg: int y = 10;
 float z = 10.5, x;
 x = y+z; (y is converted to float type by compiler)
 x = 10.0 + 10.5
 x = 20.5 (result var. x is must be float)

Explicit type conversion

It will be performed by the programmer. According to the need of this in the program.

Syntax: datatype (var)

Eg: int y = 10;
 float z = 2.5;(resultant type of y+z is float, that is converted explicitly to int type)
 x = int (y + z);
Now the result is of int type.

CONTROL INSTRUCTIONS

- In real world, several activities are sequenced or repeated based on some decisions.
- Constructing control instructions can program such activities.

Types of control Instruction

1. Sequential Control Instruction
2. Selection Control Instruction
3. Loop Control Instruction
4. Case Control Instruction

SEQUENTIAL CONTROL INSTRUCTION

- Here instructions are executed sequential manner
- That is the same order in which they appear in the program.
- By default the instructions in a program are executed sequentially.

Example:

```
#include<iostream.h>
void main()
{
int a,b;
cout<<"Enter the value of a&b":
cin>>a>>b;
int x=a+b;
cout<<"Sum of a & b is"<<x;
}
```

RUN

```
Enter the value of a& b
10 5
Sum of a & b is 15
```

In the above program instructions are executed one after another, in which they appear in the program.

SELECTION CONTROL INSTRUCTION

- Many times, we want a set of instructions to be executed in one situation, and an entirely different set of instruction to be executed in another situation.

- This kind of situation is dealt in C++ by constructing selection control instructions.

The following Statements are supporting to construct selection control instructions:

1. simple if statement
2. if-else statement
3. Nested if- else statement
4. Else-if statement

1. Simple if statement(one way decision stmt):

It is a powerful decision making statement, which is used to control the sequence of the execution of statements.

Syntax:

```
If(test expression)
{
    statement Block;
}
statement-x;
```

Execution Procedure:

The statement Block may be a single statement or a group of statements. If the test expression is true, the statement block is executed; otherwise the statement block will be skipped and the execution will jump to the statement x. Remember when the condition is true both the statement block and statement x are executed in sequence.

Example:

```
if( category == "sports")
{
    marks = marks + bouns_marks;
}
cout<<marks;
```

The program tests the type of category of the student. If the student belongs to the SPORTS category, then additional bouns_marks are added to his marks before they are printed. For others, bouns_marks are not added.

2. The if-else statement (two way decision stmt)

It performs some action even when the test expression fails.

Syntax:

```
if( test expression)
{
    true block statements;
}
else
{
    false block statements;
```

```
{  
statement x;
```

Execution Procedure:

If the test expression is true, then the true block statement, immediately following the statement is executed; otherwise, the false block statements are executed. In either case, either true or false block will be executed, not both. In both the cases, the control is transferred subsequently to statement x.

Example:

```
# include<iostream.h>  
void main()  
{  
    int age;  
    cout<<"Enter your age";  
    cin>>age;  
    if((age>12)&&(age<20))  
    {  
        cout<<"you are a teen aged person";  
    }  
    else  
    {  
        cout<<"You are not a teen aged person";  
    }  
    cout<<"Program terminated";  
}
```

Run1

```
Enter your age  
16  
You are a teen aged person  
Program Terminated
```

Run 2

```
Enter your age  
23  
You are not teen aged person  
Program Terminated
```

3. Nesting of if else statements(Multi way decision stmt)

When a series of decisions involved, we may have to use more than one if – else statement in nested form as follows:

```
if(test condition 1)  
{  
    if( test condition 2)
```

```

        {
            statement 1;
        }
    else
    {
        statement 2;
    }
}
else
{
    statement 3
}
Statement x;

```

Execution Procedure:

If the condition –1 is false, the statement 3 will be executed; otherwise it continues to perform the second test. If the condition 2 is true, the statement –1 will be evaluated; otherwise statement 2 will be executed and then the control, is transferred to statement x.

Example: Program to find largest of 3 nos

```

#include<iostream.h>
void main()
{
    int a,b,c;
    cout<<" three nos";
    cin>>a>>b>>c;
    If(a>b)
    {
        if(a>c)
        {
            cout<<"a is greatest";
        }
        else
        {
            cout<<"c is greatest";
        }
    }
    else
    {
        if(b>c)
        {
            cout<<"b is greatest";
        }
        else
        {

```



```

        cout<<"c is greatest";
    }
}

```

Run:

Enter 3 nos:

24 56 34

b is greatest

4. else – if statement

There is another way of putting ifs together when multipath decisions are involved. A multipath decision is a chain of ifs in which the statement associated with each else is an if.

It takes the following
general form:

```

if( test condition1)
{
    statement1;
}
else if (test condition2)
{
    statement 2;
}
else if(test condition3)
{
    statement 3;
}
else
{
    statement 4;
}
statement x;

```

Execution Procedure:

This construct is known as else if ladder. The condition evaluated from the top, downwards. As soon as the true condition is found, the statement associated with it is executed and the control is transferred to the statement x (skipping the rest of the ladder). When all the conditions become false, then the final else containing the statement will be executed.

Let us consider an example of grading the student in an academic institution. The grading is done according to the following rules:

Average marks	Grade
80-100	Honours

60-79	First Division
50- 69	Second Division
40- 49	Third Division
0-39	Fail

This grading can be done using the else if ladder as follows:

```
#include<iostream.h>
void main()
{
    int marks;
    cout<<"enter ur marks"
    cin>>marks;
    If(marks>79)
        Grade="Honours";
    else if(marks>59)
        Grade="First Division";
    else if(marks>49)
        Grade="Second Division";
    else if(marks>39)
        Grade="Third Division";
    else
        Grade="Fail";

    cout<<Grade;
}
```

Run

Enter ur mark
67
First Division

LOOP CONTROL INSTRUCTIONS

- Loop causes a section of code to be executed repeatedly until a termination condition is met.
- A program loop therefore consists of two segments, one known as the body of the loop and the other known as the control statement.
- Control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

The following Statements are supporting to construct loop control instructions:

1. while statement
2. do - while statement
3. for statement

while statement(Entry controlled Loop)

While is used when the number of iterations to be performed is not known in advance.

Syntax:

```
while(test condition)
{
    body of the loop
}
Statement x;
```

Execution Procedure:

The test condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop.

Example: Display 1----N numbers

```
#include<iostream.h>
void main()
{
    int n;
    cout<<"how many integers to be displayed";
    cin>>n;
    int I=1;
    while(I<=n)
    {
        cout<<I<<endl;
        I++;
    }
    cout<<"Program Terminated";
}
```

Run

How many integers to be displayed:

5

1

2

3

4

5

Program Terminated

do-while statement (Exit controlled loop stmt)

- Some times, it is desirable to execute the body of a while loop only once,, even if the test expression evaluates to false during the first iteration.

- This requires testing of termination expression at the end of the loop rather than the beginning as in the while loops.
- So, the do- while loop is called bottom tested loop.
- The loop is executed as long as the test condition remains true.

Syntax:

```
do
{
    body of the loop;
} while(test condition);
statement x;
```

Execution Procedure:

On reaching the do statement, the program proceeds to evaluate the body of the loop first. At the end of the loop, the test condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true.

When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.

Example: Display 1----N numbers

```
#include<iostream.h>
void main()
{
    int n;
    cout<<"how many integers to be displayed";
    cin>>n;
    int I=1;
    do
    {
        cout<<I<<endl;
        I++;
    }while(I<=n);
    cout<<"Program Terminated";
}
```

Run

How many integers to be displayed:

5
1
2
3
4
5

Program Terminated

for loop statement

For loop is useful while executing a statement a fixed number of times.

For statement is the compact way to express a loop.

Syntax

```
for(initialization; condition; increment/decrement)
{
    Body of the loop;
}
statement x;
```

Execution Procedure:

The Initialization part is executed only once. Next the test condition is evaluated. If the test evaluates to false, then the next statement after the for loop is executed. If the test expression evaluates to true, then body of the loop is executed. After executing the body of the loop, the increment/ decrement part is executed. The test is evaluated again and the whole process is repeated as long as the test expression evaluates to true.

Example: display numbers 1.....n using for loop

```
#include<iostream.h>
void main()
{
    int n;
    cout<<"how many integers to be displayed";
    cin>>n;
    int I;
    for(I=1;I<=n;I++)
    {
        cout<<I<<endl;
    }
    cout<<"Program Terminated";
}
```

Run

How many integers to be displayed:

5
1
2
3
4
5

Program Terminated

JUMPING STATEMENTS

BREAK Statement

- We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the conditional test.
- The keyword break allows us to do this.

- It is used to terminate the loop. When the keyword break is used inside any c++ loop, control automatically transferred to the first statement after the loop.
- A break usually associated with if statement.
- **Syntax : break;**

Example:

```
#include<iostream.h>
void main()
{
int I;
for(I=1;I<=10;I++)
{
    if(I<=6)
        break;
    cout<<I;
}
}
```

Output: 1 2 3 4 5

Here the cout statement print value of I upto 5 when I reaches 6 the if statement is true. So the break statement transfer the control to the outside of the for loop.

Example: to determine whether a no is prime or not.

// a prime number which is divisible only by 1 and itself.

```
#include<iostream.h>
void main()
{
int n;
cout<<"Enter a number";
cin>>num;

for(I=2;I<num;I++)
{
    if(num%I==0)
    {
        cout<<" number is not prime";
        break;
    }
}
if(num==I)
{
    cout<<"Prime number";
}
}
```

}

- In this program the moment $\text{num} \% I$ turns out to be 0, num is exactly divisible by I, the message “not prime no” is printed and the control breaks out of the while loop.
- Why does the program require the if statement after the while loop at all ?
 - i. it jumped out because the number proved to be not prime.
 - ii. The loop causes to an end because the value of I became equal to num.
- When the loop terminates in the second case, it means that there was no number between 2 & num-1 that would exactly divide num. That is num is indeed a prime.
- If this is true, the program should print out the message “Prime number”.

CONTINUE Statement

- In some programming situation we want to take the control to the beginning of the loop, by passing the statement inside the loop, which have not yet been executed.
- The keyword continue allows us to do this. When the keyword continue is encountered inside a c++ loop, control automatically passes to the beginning of the loop
- Continue statement usually associated with if statement.
- **Syntax: continue;**

Example: sum of the +ve numbers(2, -3,4,-2,9,5)

```
int Sum=0;
for(int I=0;I<10;I++)
{
    cin>>no;
    if(no<0)
        continue;
    sum=sum+no;
}
cout<<sum;
```

In the above program when the entered number is less than 0, then it is -ve number, so move the control to read the next number without performing summation.

Unconditional statement

- goto is the unconditional statement. Which is used to move the control anywhere inside the program.
- Goto require a label in order to identify the place where the control is to be moved. A label is a valid variable name & a colon must follow it.
- The label is placed immediately before the statement where the control is to be transferred.
- General form:

goto label;	label;
.....	statement;

```
.....
label:
statement;
```

```
.....
Goto label;
.....
```

the label can be anywhere in the program either before or after the goto statement.

Example:

```
main()
{
int x,y;
read:
cin>>x;
if(x<0)
    goto read;
y=x*x;
cout<<x<<y;
}
```

CASE CONTROL INSTRUCTION

- The control statement, which allows user to make a decision from the number of choices, is called a switch case statement.
- It allows user to execute a group of statements from several available group of statements.
- **Syntax:**

```
switch( expression)
{
    case value1:
        block1;
        break;
    case value2:
        block1;
        break;
    case value3:
        block1;
        break;
    .....
    .....
    default:
        default block;
}
```

statement x;

Rules to be followed to construct case control instructions

1. The expression in switch statement must be an integer value or a character constant.
2. No case values are identical
3. Each case block must end with break statement.

4. The case keyword must terminate with colon(:)
5. Default is optional.

Execution Procedure:

- When switch is executed, the value of the expression is successively compared against the values value1, value2 –etc.
- If a case is found whose value matches with the value of the expression, then the block of statements that follows the case are executed.
- The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement transferring the control to the statement x following the switch.
- The default is optional case, when present, it will be executed if the value of the expression does not match with any case values.
- If not present action takes place if all matches fails and the control goes to the statement x.

Example: Read a number between 0-9 and print it in words.

```
cout<<"enter any number between 0-9";
cin>>num;
switch(num)
{
    case 0:
        cout<<"ZERO";
        break;

    case 1:
        cout<<"ONE";
        break;
    case 2:
        cout<<"TWO";
        break;
    .....
    .....
    default:
        cout<<"Num is not within 0-9";
}
```

RUN

```
Enter a no between 0-9
9
NINE
```

The input value matches with case 9, so the corresponding block is getting executed. Remaining cases are all skipped from execution.

ARRAYS

What are arrays?

For understanding the arrays properly, let us consider the following program.

```
main()  
{  
    int x;  
    x = 5;  
    x = 10;  
    cout<<x;  
}
```

- This program will print the value of x as 10. Why so? Because when a value 10 is assigned to x the earlier value of x, i.e., 5, is lost. These ordinary variables are capable of holding only one value at a time.
- However, there are situations in which we want to store more than one value at a time in a single variable.
- **For example**, suppose we wish to arrange the percentage of marks obtained by 100 students in ascending order.
- In such a case we have two options to store these marks in memory:
 - (i) Construct 100 variables to store percentage of marks obtained by 100 different student i.e., each variable containing one student's marks.
 - (ii) Construct one variable capable of storing or holding all the hundred values.
- The second one is better. The reason is it would be much easier to handle one variable than handling 100 different variables.
- An array is a group of logically related data items of the same data type addressed by a common name, and all the items are stored in contiguous memory locations.

Array Declaration

Like other normal variables, the array variable must be defined before it use.

Syntax:

Datatype Arrayname[array-size];

Arraysize – indicates the maximum number of elements the array can hold.

Example:

```
int marks[100]; // Integer array of size 100  
float salary[25]; //Floating print array of size 25  
char name[50]; //character array of size 50
```

Accessing Array Elements

- Once an array variable is defined, its element can be accessed by using an index or position.

Syntax:

Arrayname[index];

- To access a particular element in the array, specify the array name followed by an integer constant or variable (array index) enclosed within square braces.
- Array index indicates the element of the array, which has to be accessed.

Example:

name[4]; //Accesses the 5th element of the array name.

- Note that, in an array of N elements, the first element is indexed by zero & the last element of an array is indexed by N-1
- The loop used to read the elements of the array is


```
for(int i=0; i<5; i++)
{
    cin>>name[i];
}
```
- The variable i varies from 0 to N-1.
- Note that, the expression age[i] can also be represented as i[age], similarly, the expression age[3] is equivalent to 3[age].

Array Initialization at Definition (at compile time)

- Arrays can be initialized at the point of their definition as follows:
datatype array-name[size] = {list of values separated by comma};

For instance, the statement,
 int age[5] = {19,21,16,1,50};

- Defines an array of integers of size 5.
- In this case, the 1st element of the array age is initialized with 19, 2nd with 21, and so on.
- The array size is omitted when the array is initialized during at compile time.
 int age[] = {19,21,16,1,50};
- In such a cases, the compiler assumes the array size to be equal to the number of elements enclosed within the curly braces.
- Hence, the above statement, size of the array is considered as five.
 int age[5] = {19,21,16,1,50};
 (or)
 int age[] = {19,21,16,1,50};

Example: Sum of Array Elements

```
#include<iostream.h>
void main( )
{
```

```

int a[10];
cout<<"Enter the no. of elements, max<10>";
cin>>n;
cout<<"Enter elements";
for(int I = 0; I<n; I++)
{
    cin>>a[I];
}
int sum = 0;
for(int I = 0; I<n; I++)
{
    sum = sum + a[I];
}
cout<<"Sum of entered elements"<<sum;
}

```

Two Dimensional Array

Matrix is a two dimensional array & two subscripts are required to access each element.

Declaration:

datatype Array-name[size1][size2];

size1 – no. of rows in a matrix.

size2 – no. of columns in a matrix.

Example:

```
int x[3][3];
```

Representation of 2-D array in memory:

Matrix is allocated into the memory according to row wise (row by row allocation).

Example:

```
A= 1  5
    3  4
```

Accessing 2-D Array elements:

The elements of a 2-D array can be accessed by the following statement

A[i][j]

i – row number

j – column number

Initialization of 2-D Array

A 2-dimensional array can be initialized during its definition.

`datatype matrixname [row size][col size] = {elements of first row, elements of 2nd row...
elements of n-1 row};`

Example:

`int a[3][3] = { 1,2,3,4,5,6,7,8,9}`
or
for more readability each row elements can be grouped:

`int a[3][3] = {{ 1,2,3},{4,5,6},{7,8,9}};`
or
`int a[][3] = {{ 1,2,3},{4,5,6},{7,8,9}};`

Row size can be omitted.

Example: // Read & display a matrix

```
#include<iostream.h>
void main( )
{
    int a[5][5];
    cout<<"Enter row and col value of a matrix max<5>";
    cin>>r>>c;
    cout<<"Enter elements in a Matrix";
    for(int i = 0; i<n; i++)
    {
        for(int j = 0; j<n; j++)
        {
            cin>>a[i][j];
        }
    }
    cout<<"Display of given Matrix";
    for(i = 0; i<n; i++)
    {
        for(int j = 0; j<n; j++)
        {
            cout<<a[i][j]<<setw(5);
        }
        cout<<endl;
    }
}
```

Run:

Enter row and col value of a matrix max<5>
2 2
Enter elements in a Matrix";

```
1 2
2 4
Display of given Matrix
1 2
2 4
```

FUNCTIONS

Introduction

- It is difficult to implement a large program even if it is algorithm is available.
- To implement such a program in an easy manner, it should be split into a number of independent tasks, which can be easily designed, implemented, and managed.
- This process of splitting a large program into small manageable tasks and designing them independently is called Modular Programming.
- A repeated group of instruction in a program can be organized as a function.
- A function is a set of program statements that can be processed independently.

Advantages

- Reduction in the amount of work and development time.
- Program and function debugging is easier.
- Reduction in size of the program due to code Reusability.

FUNCTION COMPONENTS

- Function declaration or prototype
- Function Definition
- Function call
- Function parameters
- Function return statement

1. Function Prototype or Declaration:

It provides the following information to the compiler

- The name of the function
- The type of the value returned(optional, default is an integer)
- The number and type of the arguments that must be supplied in a call to the function.

- **Syntax:**

Return type function_name(argu1,argu2,.....argun);

Return type- specifies the data type of the value in the return statement.

Fun_name- name of the function.

Argu1,argu2...argun – type of argument to be passed from calling function to the called function

- **Examples:**

int max(int,int);

It informs the compiler that the function max has 2 arguments of the type integer. The function max() returns an integer values.

void max();

It informs the compiler that the function max has no arguments, max() is not returning any value.

max();

It informs the compiler that the function max has no arguments. The function max() returns an integer values.

- In function prototype default return type is integer.

2. Function definition:

The function itself is referred to a function definition.

Syntax:

```
return type function_name(argu1,argu2,.....argun) //function declarator  
{  
    function body;  
}
```

- The first line of the function definition is known as function declarator, and is followed by the function body.
- The function delcarator and declaration must use the same function name, the number of arguments, the arguments type and the return type.
- Function definition is allowed to write in a program either above or below the main ().
- If the function is defined before main (), then function declarator is optional.

- **Example:**

```
int max(int x, int y)  
{  
    if(x>y)  
        return(x);  
    else  
        return(y);  
}
```

For this function max() definition, it is declaration must be :
int max(int,int);

3. Function call:

- A function, which gets life only when a call to the function is made.
- A function call specified by the function name followed by the arguments enclosed in parenthesis and terminated by a semi colon.

- **Syntax:**

Function_name(argu1,argu2,.....argun) ;

- If a function contains a return type the function call is of the following form:

var= Function_name(argu1,argu2,.....argun) ;

- **Example:**

c=max(a,b);

- Executing the call statement causes the control to be transferred to the first statement in the function body and after execution of the function body the control is returned to the statement following the function call.

// Greatest among 2 numbers

```
#include<iostream.h>
void main()
{
    int a,b;
    int max(int,int); //function declaration
    cout<<"Enter any 2 integers";
    cin>>a>>b;
    int c= max(a,b);
    cout<<"Greatest is: "<<c;
}

int max(int x,int y)
{
    if (x>y)
        return(x);
    else
        return(y);
}
```

Run:

Enter any 2 integers

40

35

Greatest is: 40

The max() returns the maximum of the parameters a and b. The return value is assigned to local variable c in main ().

4. Function Parameters

- The parameters specified in the function call are known as **actual parameters** and those specified in the function declarator (definition) are known as **formal parameters**
- for example in the main(), the statement c=max(a,b); passes the parameters(actual parameters) a and b to max().
- The parameters x and y are formal parameters.
- When a function call is made, a one to one correspondence is established between the actual and the formal parameters.
- The scope of the formal parameters is limited to its function only.

5. Function Return

- Functions can be grouped into two categories:

- i. A Function does not have a return value (void function)
 - ii. Functions that have a return value.
- The statements: return(x); and return(y); in function max() are called function return statements. The caller must be able to receive the value returned by the function.
- In the statement c=max(a,b)
- The value returned by the function max() returning a value to the caller.

Limitation of return

A key limitation of the return statement is that it can be used to return only one item from a function.

PASSING DATA TO FUNCTIONS

The entity used to convey the message to a function is the function argument. It can be a numeric constant, a variable, multiple variables, user defined data type, etc.

Passing constants as arguments

The following program illustrates the passing of a numeric constant as an argument to a function. This constant argument is assigned to the formal parameter which is processed in the function body

```
// Greatest among 2 numbers
#include<iostream.h>
void main()
{
    int a,b;
    int max(int,int); //function declaration
    int c= max(40,35);
    cout<<"Greatest is: "<<c;
}

int max(int x,int y)
{
    if (x>y)
        return(x);
    else
        return(y);
}
```

```
Run:
Enter any 2 integers
40
35
Greatest is: 40
```

In main(), the statement c=max(40,35); invoke the function max with the constants.

Passing variable as arguments:

Similarly to constants, variables can also be passed as arguments to a function.

```
// Greatest among 2 numbers
#include<iostream.h>
void main()
{
    int a,b;
    int max(int,int); //function declaration
    cout<<"enter two integer ";
    cin>>a>>b;
    int c= max(a,b);
    cout<<"Greatest is: "<<c;
}

int max(int x,int y)
{
    if (x>y)
        return(x);
    else
        return(y);
}
```

Run:

Enter any 2 integers

40

35

Greatest is: 40

In main(), the statement c=max(a,b); invoke the function max with the values of a & b

PARAMETER PASSING

- Parameter passing is a mechanism for communication of data and information between the calling function and the called function.
- It can be achieved by either by passing values or address of the variable.
- C++ supports the following 3 types of parameter passing schemes:
 1. **Pass by Value**
 2. **Pass by Address**
 3. **Pass by Reference**

1. Pass by Value

- The default mechanism of parameter passing is called pass by value.
- Pass by value mechanism does not change the contents of the argument variable in the calling function, even if they are changed in the called function.

- Because the content of the actual parameter in a calling function is copied to the formal parameter in the called function. Changes to the parameter within the function will affect only the copy (formal parameters)
- And will have no effect on the actual argument.

• Example:

```
#include<iostream.h>
void swap(int x,int y)
{
    int t;
    cout<<"value of x& y in swap() before exchange";
    cout<<x<<setw(5)<<y<<endl;
    t=x;
    x=y;
    y=t;
    cout<<"value of x& y in swap() after exchange";
    cout<<x<<setw(5)<<y<<endl;
}
```

```
void main()
{
    int a,b;
    cout<<"enter two integers";
    cin>>a>>b;
    swap(a,b);
    cout<<"value of a and b on swap(a,b) in main()";
    cout<<a<<setw(5)<<b;
}
```

Run:

```
enter two integers
30
50
value of x& y in swap() before exchange
30  50
value of x& y in swap() after exchange
50  30
value of a and b on swap(a,b) in main()
30  50
```

Explanation:

- In main(), the statement swap(a,b) invokes the function swap() and assigns the contents of the actual parameters a & b to the formal parameters x & y respectively
- In swap() function, the input parameters are exchanged, however it is not reflected in the calling function; actual parameters a & b do not get modified.

2. Pass by Address:

- C++ provides another means of passing values to a function known as pass by address mechanism.

- Instead of passing the value, the address of the variable is passed.
- In function, the address of the argument is copied into a memory location instead of the value.
- Example:

```
#include<iostream.h>
void swap(int *x,int *y)
{
    int t;
    t=*x;
    *x=*y;
    *y=t;
}
```

```
void main()
{
    int a,b;
    cout<<"enter two integers";
    cin>>a>>b;
    swap(&a,&b);
    cout<<"value of a and b after calling swap() in main()";
    cout<<a<<setw(5)<<b;
}
```

Run:

```
enter two integers
30
50
value of a and b after calling swap() in main()";
50   30
```

Explanation:

- In main(), the statement swap(&x, &y) invokes the function swap and assigns the address of the actual parameters a and b to the formal parameters x & y respectively.
- In swap(), the statement t=*x; assigns the contents of the memory location pointed to by the pointer (address) stored in the variable x. similarly, the parameter y holds the address of the parameter b.
- Any modification to the memory contents using these address will be reflected in the calling function, the actual parameter a & b gets modified.

3. Pass by Reference

- Passing parameters by reference has the **functionality of pass by address and the syntax of pass by value**.
- Any modification made through the formal parameter is also reflected in the actual parameter.
- To pass as argument by reference, the function call is similar to that of call by value.

- In function declarator, those parameters, parameters, which are to be received by reference, must be preceded by the address (&)operator.
- The reference type formal parameters are accessed in the same way as normal value parameters.
- However, any modification to them will also be reflected in the actual parameters.
- Example:

```
#include<iostream.h>
void swap(int &x,int &y)
{
    int t=x;
    x=y;
    y=t;
}
```

```
void main()
{
    int a,b;
    cout<<"enter two integers";
    cin>>a>>b;
    swap(a,b);
    cout<<"value of a and b after swap(a,b) in main()";
    cout<<a<<setw(5)<<b;
}
```

Run:

```
enter two integers
30
50
value of a and b after swap(a,b) in main()
50 30
```

- In main(), the statement swap(a, b); is translated into swap(&a,&b); internally during compilation.
- The function declarator void swap(int &a, int &b) indicates that the formal parameters are of reference type and hence, they must be bound to the memory location of the actual parameters
- Thus any access made to the reference formal parameters in the swap() reflects to the actual parameters.

DEFAULT ARGUMENTS

- Normally a function should specify all the arguments used in the function definition.
- In a c++ function call, when one or more arguments are omitted, the function may be defined to take default values for the omitted arguments by providing the default values in the function prototype.
- Hence the feature of default arguments allows the same function to be called with fewer arguments than defined in the function prototype.
- To establish a default value, the function declaration must be used.

- The compiler checks the function prototype with the arguments in the function call to provide default values to those arguments, which are omitted.
- Default arguments reduce the burden of passing arguments explicitly at the point of the function call.
- **Example:**

```
#include<iostream.h>
void greatest(int = 50,int=25,int =35);
void main()
{
    greatest();
    greatest(10);
    greatest(75,12);
    greatest(15,2,55);
}
void greatest(int x,int y,int z)
{
    if((x>y)&&(x>z))
        cout<<"I st number is greatest";
    else if(y>z))
        cout<<"II nd number is greatest";
    else
        cout<<"III rd number is greatest";
}
```

In the main (), when the compiler encounters the statement greatest(), it is replaced by the statement greatest(50,25,35); Internally substituting the missing arguments. Similarly when the compilers encounters the statement greatest (10); it is replaced by the statement greatest (10, 25, 35); internally substituting the remaining two missing arguments and so on for all the remaining function calls.

- Variable names can be omitted while assigning default values in the prototype.

INLINE FUNCTIONS:

- One of the objectives of using functions in a program is to save memory, which becomes appreciable when a function is likely to be called many times.
- However, every time a function is called, it takes a lot of extra time in executing a series of instructions, for task such as jumping to the function, and returning to the calling function.
- When a function is small, the time required to execute a function is less than the switch time.
- In C++ a new feature called inline function is used to solve the above problem.
- An inline function is a function that is expanded in line when it is called.
- That is, the compiler replaces the function call with the corresponding function code.
- Inline functions are defined as follows

```

inline return type fun_name(arguments)
{
    function body
}

```

- **Example**

```

inline double cube(double a)
{
    return(a*a*a);
}

```

- The above inline function can be called by statements like

```

c=cube(3.0);
d=cube(2.5+1.5);

```

- The keyword **inline** send a request to the compiler. The compiler may ignore this request if the function definition is too long or too complicated and compile the function as a normal function

- **Example:**

```

#include<iostream.h>
inline int square(int num)
{
    return(num*num);
}

```

```

void main()
{
    int n;
    cout<<"Enter a number:";
    cin>>n;
    cout<<"Its square ="<<square(n)<<endl;
    cout<<"square(10) = "<<square(10);
}

```

Run

```

Enter a number: 5
Its square =25
Square(10)=100

```

- In the main(), the statement `cout<<"Its square="<<square(num);` Invokes the inline function `square()`, It will be suitably replaced by the instruction(s) of the body of the function `square()` by the compiler.
- The Execution time of the function `square()` is less than the time required to establish a linkage between the calling function and called function.

PROGRAMS:

1. Program for Fibonacci Series

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int a=0,b=1,i,c,n;
    clrscr();
    cout<<"ENTER THE VALUE FOR 'N' :";
    cin>>n;
    cin>>a>>b;
    cout<<a<<b;
    for(i=3;i<=n;i++)
    {
        c=a+b;
        cout<<c;
        a=b;
        b=c;
    }
    getch();
}
```

2.Program for Perfect Number

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int sum=0,i,n;
    clrscr();
    cout<<"Enter the number :";
    cin>>n;
    for(i=1;i<n;i++)
    {
        if(n%i==0)
            sum=sum+i;
    }
    if(sum==n)
        cout<<" The Number Is Perfect"<<n;
    else
        cout<<" The Number Is Not Perfect"<<n;
    getch();
}
```

3.Program for finding factorial

```
#include<iostream.h>
```



```

#include<conio.h>
int n,i,c;
void fact()
{
int f=n;
if(n==0)
cout<<"The factorial of %d is 1"<<n;
else
for(i=0;i>0;i--)
f*=i;
cout<<"The Factorial of "<<n<<f;
}
void main()
{
clrscr();
cout<<" Enter the number ";
cin>>n;
fact();
getch();
}

```

4.Program for Prime Number

```

#include<iostream.h>
#include<conio.h>

void main()
{
//clrscr();

int number,count=0;

cout<<"ENTER NUMBER TO CHECK IT IS PRIMEOR NOT ";
cin>>number;

for(int a=1;a<=number;a++)
{
if(number%a==0)
{
count++;
}
}

if(count==2)
{
cout<<" PRIME NUMBER \n";
}
else
{
cout<<" NOT A PRIME NUMBER \n";
} //getch();
}

```

5.Program for Armstrong Number

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
clrscr();
int n,m=0,x,y;
cout<<“Enter any three digit numnber:”;
cin>>n;
y=n;
while(n!=0)
{
x=n%10;
m+=pow(x,3);
n=n/10;
}

if(y==m)
cout<<“The number is an Armstrong number”;
else
cout<<“The number is not an Armstrong number”;
getch();
}
```

OBJECT ORIENTED PROGRAMMING IN C++

Characteristics of OOPS

- Programs are divided into known objects
- Builds the data and functions around these objects or entities.

Organization of data and functions in OOP

- Hence object may communicate with each other through functions.
- Now data and functions can be easily added whenever necessary.
- Follows bottom-up approach in program design.

Concepts of OOPS

General concepts of OOPS comprises the following

1. Object

2. Class
3. Data abstraction
4. Inheritance
5. Polymorphism
6. Dynamic Binding
7. Message passing.

Object

Object is an entity that can store data and send and receive messages. They are run time entities they may also represent user-defined data.

When a program is executed the object interacts by sending messages to one another.

Every object will have the data structure called attributes (or property or data) and behavior called operations.

Eg: Consider the object account

Structure	(General format)	Eg:	
Object Name		Account	
Attribute 1 Attribute 2 Attribute N	Structure	Account Number Account Type Name Balance	attribute
Operation 1 Operation 2 Operation N	Behaviour	Deposit() Withdraw() Enquire()	Operation

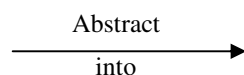
Classes

The objects with the same data structure (attribute) and behaviour (operations) are grouped into a class. All these objects possessing similar properties and grouped into the same unit.

Eg: In the person class all person having similar attributes like Name, Age, Sex and the similar operations like speak, listen, walk. So, boy and girls objects are grouped into the person class.

This should be represented as person objects.

Two different
persons
(Eg: boy, girl)



Person class
Attributes: Name, Age, Sex
Operations: Speak(), Listen(), Walk()

Representation of Class:

```

Class account
{
    private:
        char name[20];
        int accounttype;
        int accountnumber;
        float balance;

    public:
        Deposit( );
        Withdraw( );
        Enquire( );
};

```

Data members

Member functions

In this the account class groups the object such as saving account, current account, etc, Thus, objects having the same structural and behavioral propositions are grouped together to form a class.

The following points on classes can be noted:

1. A class is a template that unites data and operations.
2. A class is a abstraction of the real world entities with similar properties.
3. A class identifies a set of similar objects.

Definition of OOPS:

OOP is a method of implementation in which programs are organized as co-operative collections of objects, each of which represent an instance of some class and whose classes are all members of a hierarchy of classes united through the property called inheritance.

CLASSES AND OBJECTS:

Class Specification:

The class can be described as a collection of data member along with member functions. This property of C++, which allows association of data and functions into a single unit, is called encapsulation. Sometimes classes may not contain any data members or member function called empty classes.

Syntax for class specification

```

      Keyword
      ^
Class classname
{
    //body of a class
}; end of class requires semicolon

```

Userdefined name of the class

More than one object can be created with a single statement as,

```
Class student s1,s2,s3;  
Or  
Student s1,s2,s3.
```

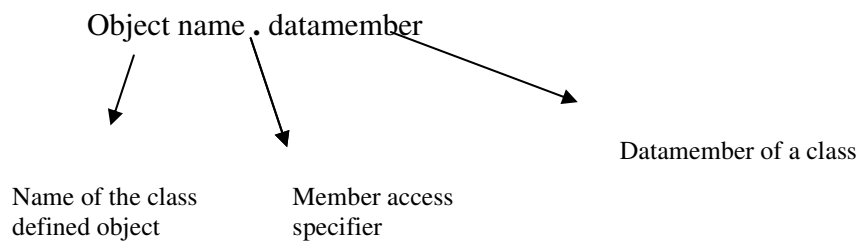
Object can also be created by placing these names immediately after the closing brace.
Thus the definition

```
Class student  
{  
    -----  
    -----  
} s1,s2,s3;
```

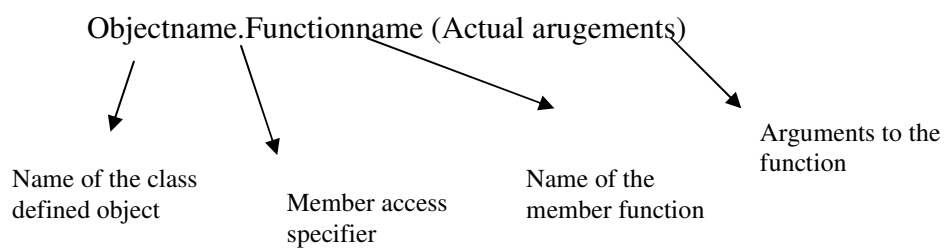
Accessing class members:

Once an object of a class has been created, there must be a provision to access its members. This is achieved by using the member access operator, dot(.).

Syntax: for accessing datamember of a class



Syntax for accessing member function of a class



Eg: s1.setdata(10, "Ram");
s1.outdata();

The object s1 can be used to access the member functions setdata and outdata respectively.

Consider the following program

```
#include <iostream.h>
```

```

#include <string.h>
class student
{
    private:
        int roll_no;
        char name[20];
    public:
        void setdata(int roll_no_in, char name_in)
        {
            roll_no = roll_no_in;
            strcpy(name, name_in);
        }
        void outdata( )      //display data members
        {
            cout<<"rollno = "<< roll_no <<endl;
            cout<<"name = " << name << endl;
        }
};
void main( )
{
    student s1;
    s1.setdata(1, "Ram");
    s2.setdata(10,"Kumar");
    cout<<"Student details . . ."<<endl;
    s1.outdata( );
    s2.outdata( );
}

```

Output:

```

Student details
Rollno = 1
Name = Ram
Rollno = 10
Name = Kumar

```

Defining member function:

The data members of a class must be declared within the body of the class, whereas the member functions of the class can be defined in any one of the following ways.

- Inside the class specification
- Outside the class specification

The syntax of a member function definition changes depending on whether it is defined inside or outside the class specification, but it performs the same operation.

(a) Member function inside the class body:

All the member functions defined within the body of a class.

Eg:

```
Class classname
{
    private:
        int age;
        int setage(int agein);    //member function
        {
            age = agein;        //body of the function
        }
    .....
    public:
        int b;
        void rect( )
        {
            ....    // body of a function
        }
};
```

(b) Member functions outside the class body

To declare function prototype within the body of a class and then define it outside the body of a class. This is done by using the ‘scope resolution operator’ (::). It acts as an identity-label to inform the compiler, the class to which the function belongs.

G.F or Syntax:

```
class classname
{
    ....
    Returntype memberfunction (arguments);    //function declaration
    ....
};
returntype classname :: memberfunction (arguments)    //function definition
{
    //body of the function
}
```

Accessing member functions within the class

A member of a class is accessed by the objects of that class using the dot operator.

Ex:

```
#include <iostream.h>
class number
{
    int num1, num2;    //private by default
    public:
        void read( )
```

```

        {
            cout<<"Enter first number: ";
            cin>>num1;
            cout<<"Enter second number: ";
            cin>>num2;
        }
        int max()
        {
            if(num1>num2)
                return num1;
            else
                return num2;
        }
        //Nesting of member function
        void showmax( )
        {
            cout<<"maximum = "<<max( ) ;
        }
    };
    void main( )
    {
        number n1;
        n1.read( );
        n1.showmax( );
    }

```

Output:

```

Enter first number : 5
Enter second number : 10
Maximum = 10

```

This member function of a class can call any other member function of its own class is called 'nesting of member function'.

Data Hiding:

Data is hidden inside a class, so the unauthorized access is not possible, which is the key feature of OOP.

All the data and functions defined in a class are private by default. Normally the data members are declared as private and member functions are declared as public.

Methods of Data hiding:

- Private
- Public
- Protected

These keywords are called access-control specifiers.

Private members:

In this only the member functions of the same class can access these members. The private members of a class are inaccessible outside the class.

```
class person
{
    private:
        int age;          //private data
        int getage( );    //private function
        . . . .
}
person p1;
a = p1.age;              //cannot access private data //error
p1.getage( );            //Error access
i.e., we can access the private members by using objects.
```

Protected member:

The access control of the protected members is similar to that of private members. The access control of protected members is shown below:

```
Class person
{
    protected:
        . . . .
        int age;          // protected data
        int getage( );    //protected function
        . . . .
};
person p1;
a = p1.age;
p1.getage( );
}          Cannot access protected members
```

Public Members

All data members and function declared in the public section of the class can be accessed without any restriction from anywhere in the program.

Eg:

```
class person
{
    public:
        int age;          //public data
        int getage( );    //public function
```

```

}
person p1;
a = p1.age;           // can access public data
p1.getage( );         // can access public function

```

Nesting of member function

A member function of a class can be called only by an object of that class using a dot operator. In nesting of member function, the member function can be called by using its name inside another member function of the same class is called nesting member function.

Consider the following example

```

#include <iostream.h>
class set
{
    int m,n;
    public:
        void input(void);
        void display(void);
        void largest(void);
};
int set :: largest(void)
{
    if(m>=n)
        return(m);
    else
        return(n);
}
void set:: input (void)
{
    cout<<"input values of m and n" << "\n";
}
void set:: display(void)
{
    cout<<"largest value" << largest( )<< "\n";
}
void main( )
{
    set A:
    A input( );
    A.set( );
}

```

Arrays within a class

The arrays can be used as member variables in a class. That is more than one related variable or data are grouped under the common name,

Eg:

Class abc

```
{
    int a[size];    //a is name of the array size – represents the size of the array
    public:
        void setdata(void);
        void display(void);
};
```

Empty classes (or Stubs)

Main reason for using a class is to encapsulate data and code, it is however, possible to have a class that has neither data nor code. In other words, it is possible to have empty classes.

The declaration of empty classes is as follows:

```
class xyz
{
};
class abc
{
};
```

Such an empty class is also called as stub

Passing Objects arguments:

- It is possible to have functions which accept objects of a class as arguments, just as there are functions which accept other variables as arguments.
- An object can be passed as an argument to a function by the following ways:
 1. Passing object by value, a copy of the entire object is passed to the function
 2. Passing object by reference, only the address of the object is passed implicitly to the function.
 3. Passing object by pointer, the address of the object is passed explicitly to the function

- **Passing Object by value**

In this case a copy of the object is passed to the function and any modifications made to the object inside the function are not reflected in the object used to call the function.

Example:

```
#include<iostream.h>
```

```

class test
{
int m1,m2;
public:
void get()
{
cin>>m1>>m2;
}
void read(test t3)
{
m1=t3.m1;
m2=t3.m2;
}
void display()
{
cout<<m1<<m2;
}
};
void main()
{
    test t1;
    cout<<"Enter Ist object data";
    t1.get();
    cout<<"display Ist object data";
    t1.display();
    test t2;
    cout<<"copy of object1 to object2";
    t2. read(t1);
    cout<<"display 2nd object data";
    t2.display();
}

```

Run:

```

Enter Ist object data
34
56
display Ist object data
34
56
copy of object1 to object2
display 2nd object data
34
56

```

The members of t1 are copied to t2. Any modification made to the data members of the objects t1 and t2 are not visible to the caller's actual parameter

- **Passing objects by Reference:**

Accessibility of the objects passed reference is similar to those passed by value. Modifications carried out on such objects in the called function will also be reflected in the calling function.

Example:

```
#include<iostream.h>
class test
{
int m1,m2;
public:
void get()
{
cin>>m1>>m2;
}
void read(test &t3)
{
m1=t3.m1;
m2=t3.m2;
}
void display()
{
cout<<m1<<m2;
}
};
void main()
{
    test t1;
    cout<<"Enter Ist object data";
    t1.get();
    cout<<"display Ist object data";
    t1.display();
    test t2;
    cout<<"copy of object1 to object2";
    t2.read(t1);
    cout<<"display 2nd object data";
    t2.display();
}
```

Run:

```
Enter Ist object data
34
56
display Ist object data
34
56
copy of object1 to object2
display 2nd object data
34
```

Program for Student Data Base

```
#include
#include
class student
{
int id,mrks;
char nm[20],stm[2];
private:
int st_id;
char name[20];
char strm[2];
int marks;
public:
void dta_entry()
{
clrscr();
cout<<"enter the entries below:-\n";
cout<<"student id: \n";
cin>>id;
cout<<"student name: \n";
scanf("%s",nm);
cout<<"stream: \n";
scanf("%s",stm);
cout<<"marks: \n";
cin>>mrks;
}
void show_details()
{
cout<<" \nstudent name:"<< nm;
cout<<" \nid:"<< id;
cout<<" \nstream:"<< stm;
cout<<" \nmarks:"<< mrks;
cout<<"\n\n";
}
void mod_mrks(int inc)
```

```

    { mrks=mrks+inc;
    cout<<"marks has been incremented.\n";
    }
};

void main()
{
    student st1,st2,st3;
    clrscr();
    st1.dta_entry();
    st2.dta_entry();
    st3.dta_entry();
    clrscr();
    cout<<"the entered records are as follows:\n";
    st1.show_details();
    st2.show_details();
    st3.show_details();
    getch();
    clrscr();
    st1.show_details();
    cout<<"\nmodified records of shobhit:";
    st1.mod_mrks(10);
    st1.show_details();
    getch();
}

```