SPLC
(Software Development Life Cycle)
8 Steps —

1. Programme Specification

Problem Definition Requirement Analysis

Requirements of Software Development
A) Skilled man-power (most crucial)
B) Infra-structure Requirement
C) Feasibility Study

2. Logic & Algorithm Building

Logic is a thought process

The following tools can be used to design logic or algorithm —
A) Flow charting (make flow chart on paper)
   b) pictorial representation of logic
B) Pseudo code (false code)
   b) simple English like language to design a logic
C) DFD (Data Flow Diagram)
   This explains the idea of project in detail

3. Testing and Debugging of the logic

We have 2 methods to check logic
A) Independent inspection
B) Structural walkthrough
4. Coding
Converting logic into a real computer programme using a programming language.

5. Testing and Debugging
3 types of errors:
(A) Syntax - typographical errors in syntax
(B) Logical - errors in running loop etc. and this error is never shown on screen.

We check logical error by Dry Run.
(C) Run Time - local memory related error such as memory full due to operation of many programs simultaneously.

Function
It is the collection of statements to a specific task or it is the sub-program of a program.

6. Implementation
It includes user training.
A program is implemented in any of 3 ways-
(A) Internet - can be accessed by anybody
(B) Intranet - can be accessed within an organisation within a campus
(C) Extranet - e.g. Amigone can be accessed by anybody but access is authorised.
3. Documentation
   - internal (all the logics should be documented)
   - external (eg: user manual).

3. Maintenance
   Mutual understanding b/w company and the software making company.

C++
   It is extension of C. C++ is an object oriented programming language.
   Object is any real thing in the world even an idea.

   Property - work on required (abstraction)
   - characteristic of an object.

# Object-Oriented Programming Language
   1. Semi OOP = C++
   2. Fully OOP = Java (we have to have object)

   C++ is Semi OOP because program can be written without object.

# Class
   Every object belongs to a class and it has property of that class. Before object, we have to make a class.
   
   A class is a set of behaviour & properties.
Object

The inheritance of the class

Inheritance

- Most important feature
- Addition of features from old class

Polymorphism

- Overloading on operators, we can add different data types. They are e/a operator overloaded.
- They are under polymorphism
- (A) Operator Overloading
- (B) Functional Overloading

The implementation of operator Overloading and functional Overloading is called Polymorphism.

What is Encapsulation or Data Hiding?

- There are 4 building blocks of any programming language:
  - Object/class, Inheritance, Polymorphism, Encapsulation & Abstraction

# Major Loops of C++

while (i)

for

3

3
Conditions

```
if ()
else
else if ()
\}
\}
\)
\)
```

Nested

```
if ()
\}
\)
```

```
# BASIC PROGRAMS

cout = console output
cin = console input

In C we use printf();
in C++ we use cout << ""; 

In C we have scanf();
in C++ we have cin >> 
```
Write a C++ program to accept 2 numbers and find out the sum.

```cpp
#include <iostream>

void main()
{
    int x, y, sum = 0;
    cout << "Enter the first number: " << x;
    cin >> x;
    cout << "Enter the second number: " << y;
    cin >> y;
    sum = x + y;
    cout << "Sum is " << sum;
}
```

- `Cin = console input output`
- `In = get output to new line in C`.
- In C++, we use `endl`.

Write the C++ program to accept 5 numbers and display the biggest.

```cpp
#include <iostream>

void main()
{
    int n;
    int big = 0;
    int i = 0;
    while (n <= 5)
    {
        cout << "Enter the Number" << i;
        cin >> n;
    }
```
if (n > big)
    big = n;
    i++;
End

Execution of loop
This is c/a Day Run.
Variable      Output

- n: Enter The Number
  big = 0
  i = 1

- n = 5
  big = 5
  i = 2

- n = 9
  big = 9
  i = 3

- n = 6
  big = 9
  i = 4

- n = 11
  big = 11
  i = 5

- n = 2
  big = 11
  i = 6

Biggest no. is 11.
Do the above program using while loop and for loop.

Write a C++ program to accept a number and display the table till that number.

```cpp
#include <cstdio.h>

void main()
{
    int n;
    
    int i, j;
    int res;
    
    cout << "Enter the number: ";
    cin >> n;
    
    for (i = 1; i <= n; i++)
        for (j = 1; j <= 10; j++)
            res = i * j;
            cout << res << endl;
}
```

---

**Day Run**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Table of 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 3</td>
<td>1</td>
</tr>
<tr>
<td>i = 1</td>
<td>2</td>
</tr>
<tr>
<td>j = 2</td>
<td>3</td>
</tr>
<tr>
<td>res = 2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>


i = 11  \text{(then it will go to other loop.)}

n = 3

i = 2

j = 1

res = 2

\text{Table 8.2}

\text{Similarly table 8.3.}

\textbf{Concept of}

\# \textbf{class and object}

\textbf{Write a C++ program \& will accept employ number, name and salary and display the same by using the concept of class and object.}

\# \textbf{include <conio.h>}

\textbf{class prop}

\{

\textbf{int emp no; \hspace{1cm}} \textbf{There are technically c/a}
\textbf{char ename[10]; \hspace{1cm}} \textbf{data members.}
\textbf{float salary;} \hspace{1cm} \textbf{also c/a characters of class}
\}

\textbf{- By default all the data members and member functions}

\textbf{Public:}

\textbf{void getdata()}

\}
- This will take emp no, name & salary from class & display on output screen.
- Public is written with colon not semi-colon.
- If we don't write public, it will be private and can't be used outside class.

```cpp
Cout << "Enter the Emp no.: ";
Cin >> emp no;
Cout << "Enter the name : ";
Cin >> eName;
Cout << "Enter the Salary : ";
Cin >> salary;

void showdata() 
{
    Cout << " Employee No. : " << emp no;
    Cout << " Name : " << eName;
    Cout << " Salary : " << Salary;
}

} // class close

void main() 
{
    emp e;
    // this is e/a object/name of object
    e.getdata();
e.showdata();
}
```

- If there are 2 objects say e1 & e2, we'll do e1.getdata(), e1.showdata(), e2.getdata(), e2.showdata();
* Modify the above program to calculate HRA (House rent allowance) 15% and travelling allowance (TA) 12% of the salary. Display HRA, TA, Bonus along with emp no, name and salary for each employee.

Hint

```c
emp e;
e . getdata();
e . HRA();
e . TA();
e . Bonus();
e . total();
e . showdata();
```

OR (include all this in class)

```c
emp e ;
e . getdata();
e . perks ();
e . showdata();
```

```c
eg: - 3
void perks ()
{
    HRA = Salary * 15%  
    TA = Salary * 12%  
    Bonus = Salary * 9%  
    Total = Salary + HRA + TA + Bonus;
}```
class
{
    public (can be used outside class)
    {
    }
    private (can't be used outside class)
    {
    }
}

Class and Object

Constructor

1. Constructor is a member function of the class having the same name as class.
2. The constructor function will be called automatically when the object is created.
3. The constructor function is used to initialize the values of the variables attached with object.

```cpp
#include <iostream.h>
#include <conio.h>
#include <string.h>

class emp
{
    int empno;
    char ename[10];
    float salary;
    public:
        emp()
        {
            empno = 1001;
        }
    }
```
4. Constructor will not return any values, even void.

```cpp
    Strocy (ename,"Amit");
    Salary = 10000;
```

```cpp
    void getdata ()
    {
        cout << "enter the emp no : ";
        cin >> emp no;
        cout << "enter the name : ";
        cin >> e name;
        cout << "enter the salary ";
        cin >> salary;
    }

    void showdata ()
    {
        cout << "employee no : " << emp no;
        cout << " name : " << e name;
        cout << " salary : " << salary;
    }
    // class close
    void main ()
    {
        emp e1;  // for constructor
        e1. showdata();
        e1. getdata ();
        e1. showdata ();
    }
```
5. There are 3 types of constructors available.
   (a) Default constructor
   (b) Parameterize constructor
   (c) Copy constructor

Parameterize constructor
Public:
emp(int x, char name, float sal);
&
emp no = x;

emp e1 = 1001, "Amit", 10000);

8. What do you mean by copy constructor. Explain with examples
Destructors

Destructor is a member function of the class having
the same name as the class but start with a 'u' sign.


It is used to release the memory space occupied
by the object. It is called automatically when
an object go out of space.

Design of Destructors through class

class:

Public:

Void getdata()

Void showdata()

x()
```c
x();
{
    cout << "memory is free";
}

void main()
{
    x();
    O1.ShowData();
    O1.GetData();
    O1.ShowData();
    // This is calling a function function1();
    T = function1();
}
```

Functions

We use functions to:
- avoid redundancy, simplify the program and
to manage memory space.

1. Function is the core collection of instructions to
do a specific task.
2. The function may or may not return a value.
   If not returning any value, we can call it
does procedure.
3. If the function is not returning any value,
than the return type is void.
4. Functions are used to reduce the redun-
dancy of the code.
5. The functions are used to do the better memory management.

\[ \text{PC} = 2\text{KB} \]

Run time error.

6. If a big program is divided into multiple small units (functions), it is easy to maintain to the software / program.

In other words, by using functions, we can increase the maintainability.

Function can be called with two methods:

(i) call by reference
(ii) call by value

Call by Reference

(i) The call by reference method of passing arguments (parameters)

\[ \text{eg: } \text{R.s} = \text{sum}(x, y); \]

Arguments (passing arguments) to a function, copy the reference of an argument into the formal parameter.

(ii) Inside the function, the reference is used to access actual argument used in the call

(c) can be changed with reference / formal parameters.

This means that changes made to the parameter affect the past argument.
Call by value

This method copies the actual value of a argument into the formal parameters of the function.
In this case, changes made to the parameters inside the function have no impacts on the argument.

```cpp
main()
{
    int x;
    Res = Sum
        int y;
    Res = Sum (x, y);
}

int Sum(int a, int b)
{
    a = a * 2;
    b = b * 25;
}
```

Here `x` & `y` are functions & `a`, `b` are arguments.

Write a C++ program to accept a number and calculate the factorial by using a function method call by value.

```cpp
#include <iostream.h>
#include <conio.h>
void main()
{
    int n, fact = 0;
    cout << "Enter the number";
Cin >> x;

int fact = factorial (x);

cout << factorail is " << fact;

int factorial (int n)
{
    int mes = 1, i = 1;
    for (i=1; i<=n; i++)
    {
        mes = mes * i;
    }
    return mes;
}

Pointer Variable:

Pointers is a memory variable, which is used to store the address of other variables of same type.

int *p; (pointer variable)  
int xj; (normal variable)

p = &x;  
int x = 10;  
*p = &x;  
*x = x * 2;
Write a C++ program to accept 2 numbers & swap the values using call by reference method

```cpp
#include <iostream.h>
#include <conio.h>

void main()
{
    int x, y;
    cout << "Enter the first no.: ";
    cin >> x;
    cout << "Enter the second no.: ";
    cin >> y;
    cout << "Value before swapping ";
    cout << "x: " << x;
    cout << " y: " << y;
    
    void swap (int *i, int *s)
    {
        int t;
        t = *i;
        *i = *s;
        *s = t;
    }

    swap (&x, &y);
    cout << "Value after swapping ";
    cout << "x: " << x;
    cout << " y: " << y;
}
```
* Design a C++ program to explain the function call & mechanism call by reference.

(LAB)

Enter details of 5 books and calculate total price.

```cpp
class Book
{
    char name[20];
    char author[20];
    float price;
public:
    void getDetails() { 
        cout << "Enter the name, author, price: ";
        cin >> name >> author >> price;
    }
    void showDetails() {
        cout << name << author << price;
    }
    float getPrice() {
        return price;
    }
    void main() {  
        // Array of objects
        Book B[5];
        cout << "Enter details of 5 books: ";
        for (int i = 0; i < 5; i++)
            B[i].getDetails();
        cout << "Total price is: ";
    }
};
```
float TP = 0;
for (i = 0; i < 5; i++)
    TP = TP + B[i].getPrice();

cout << TP;

Array of Objects
Array in the collection of multiple data of same type.

An array of object represent a single record in the memory. If you want more than one record of class type, we have to create an array of objects.

Write a C++ program to accept the following details of the 10 employees and display it into the tabular format (Emp no, name, job, salary).

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

class employee
{
    int emp_no;
    char name[10];
    char job[10];
    int salary;
public:
    void getdata();
};
```
cout << "enter the emp no":
cout << "enter the name":
cout << "enter the job":
cout << "enter the salary":
void showdata()
{
    cin >> "enter the emp no":
cin >> "name":
cin >> "job":
cin >> "salary":
void main()
{
    employee e[10];
    for (int i = 0; i < 9; i++)
    {
        // code to get details of the 10 employees.
        e[i].getdata();
    }
    for (i = 0; i < 9; i++)
    {
        // code to show the details of the employees
        e[i].showdata();
    }
}
Scope Resolution Operators
To design the body of functions outside the class, but first we need to define it inside the class. (ref. last program).

```c++
#include <iostream.h>

int salary;

public:
void get_data();
void show_data();

void employee::get_data()
{
}

void employee::show_data()
{
}

void main()
{

}
```

Write a C++ program to accept the following details of 10 students (roll no., name, marks for 3 subjects) and display the report card of all 10 students in the following tabular format.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Roll No.</th>
<th>Name</th>
<th>Comp.</th>
<th>Bio</th>
<th>Phy.</th>
<th>Total Marks</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

average marks of the class: 0. 
Inheritance
Creating a new class by using a previous class or Dividing one class using the base class

$\text{f1, f2} \xrightarrow{A} \text{Base only f1 and f2 are can be accessed}$

$\text{f3} \xrightarrow{B} \text{derived f1 also & f3 both can be accessed by using 01.}$

Types of Inheritance

1. Simple
   If a class is derived or created by using already existing class then that is called as simple inheritance.

```
[A] \rightarrow \text{Base class}
\downarrow
[B] \rightarrow \text{Derived class}
```

eq: Write a C++ program that will accept emp no., name, job and basic salary of the employee and calculate HRA: 10% of the basic, TA: 3% of the basic salary, Bonus: 5% of the basic salary. Display the total salary.

```cpp
#include <conio.h>

#include <iostream.h>

class emp
{
    // emp data's details
};
```
Public:
   int empNo;
   char Name[10];
   char Job[10];
   int Basic;

Private:

void getData()
{
    cout << "Enter empNo: " << endl;
    cin >> empNo;
    cout << "Enter emp name: " << endl;
    cin >> name;
    cout << "Enter Job: " << endl;
    cin >> Job;
    cout << "Enter Salary: " << endl;
    Basic = 10000;
    HRA = 10 * Basic;
    TA = 0.1 * Basic;
    Bonus = 0.05 * Basic;
    Total = Basic + HRA + TA + Bonus;
}

void showdata()
{
    cout << "emp no: " << empNo;
cout << "Name: " << Name;  
cout << " Job: " << Job;  
cout << " Total Salary: " << Total;  
3.3; 
void main() {  
  perks e_;  
e_. get data();  
e_. cal - perks;  
e_. show data();  
}  

2. Multi-level inheritance  
\[
\begin{array}{c}
A \rightarrow \text{Base} \\
\downarrow \\
B \rightarrow \text{Derived} \\
\downarrow \\
C \rightarrow \text{Derived}
\end{array}
\]

Deriving a class using existing derived earlier class. C has all the basic features of the Base.
C has the features of both A & B but
B has only features of A.

3.  
\[
\begin{array}{c}
A \\
\downarrow \\
C \\
\downarrow \\
B
\end{array}
\]

4. Hierarchical inheritance  
\[
\begin{array}{c}
A \\
\downarrow \\
B \\
\downarrow \\
C
\end{array}
\]

5.  
\[
\begin{array}{c}
A \\
\downarrow \\
D \\
\downarrow \\
C
\end{array}
\]
Multiple Inheritance

Internal Exam → External Exam

Base - I  Base - II

 Derived

```cpp
// Class IE
protected:
int m1, m2;
public:
void Enter_IE() {
    cout << "Enter m1 & m2:\n";
    cin >> m1, >> m2;
}
void print_IE() {
    cout << m1 << m2;
}

// Class EE
protected:
int m3, m4;
public:
```
```cpp
void Enter_IE() {
}

  cout << "Enter m1 & m2:"
  cin >> m1 >> m2

void print_IE() {
  cout << m1 << m2
}

}

class Result : public IE , public EE {
private:
  int SIE, SEE, total;
public:
  void GetResult() {
    SIE = m1 + m2;
    SEE = m3 + m4;
    total = SIE + SEE;
    cout << "Total" << total;
  }
}

void main () {
  Result R;
  R. Enter_IE();
  R. Enter_IE();
  R. print_IE();
  R. print_IE();
  R. GetResult();
```
Function Overloading
It is a feature of C++ where one defines multiple behaviors for same item

TYPES

Function Overloading
operator overloading

or

- polymorphism
- need to define more than 1 task

In C++ same func is performing multiple tasks.
This func can be differentiated by no. of arguments or argument type.

class area

```
void calculate (int a)
int t = a * a;
cout << "Area of Square = " << t;
```

```
void calculate(int a, int b)
int t = a * b;
cout << "Area of rectangle = " << t;
```
void calculate (float a)
{
    float t = 2.14 * a * a;
    cout << "Area of circle: " << t;
}

void main()
{
    area a1;
    a1.calculate (3.4);
    a1.calculate (5);
    a1.calculate (3, 9);
    getch();
}

Write a C++ program to solve equation:

\[ 1 + x^2 + x^3 + x^4 + x^5 = x^n \]

#include <iostream.h>
#include <conio.h>
#include <math.h>

int main()
{
    float x, n;
    cout << "Enter the value of x: ";
    cin >> x;
    cout << "Enter the value for n: ";
    cin >> n;
}
for (i = 1; i <= n, i++)
{
    sum = sum + pow(x, n);
}

1 - x + x^2 - x^3 + x^4 - ... - x^n
for (i = 1; i <= n, i++)
{
    if (i % 2 == 0)
    {
        sum = sum - pow(x, i);
    }
    else
    {
        sum = sum + pow(x, i);
    }
}

1 + 1x + 1x^2 + 1x^3 + ... + x^2
for (i = 1, i <= n, i++)
{
    sum = sum + fact(pow(x, i));
}

fact(int a)
{
}

Return factorial;
}
* Protected Inheritance.

Protected inheritance in deriving from protected base class, then public & protected members of the base class become protected members of the derived class.

And private data members can not be used into even given derived classes.

Private inheritance in deriving from a private base class, then public & protected members of the base class become private into the derived class.
```c
class x{
    private:
    int i;
    int j;
    public:
    protected:
    int j;
    void main() 
x 0;
or fun1();
```

- When we want our private data to be shared by
  a non-member function, then we declare a
  function as friend function of that class. By
  using that, we give it access to private
  data members.

- Single function or entire class can be declared as a
  friend of class.

- A friend function is a known member `funx` of the
  class that has been granted access to all private
  members of the class.

- We simply declare the `funx` within the class by
  prefixing the key word `FRIEND`.

- While designing, the function body or definition,
  must not use the keyword `FRIEND`.

- Definition of friend function is specified outside the
  class body, and is not treated as part of the
  class.

- The major difference between members `funx` &
  friend `funx` in that, member `funx` is accessed
  through object while friend `funx` requires object
  `funx` to be passed as object parameter.
```cpp
class base
{
    int val 1, val 2;
public:
    void getData()
    {
        cout << "Enter the value for val 1":
        cin >> val 1;
        cout << "Enter the value for val 2":
        cin >> val 2;
    }
    friend int mean (base &obj); // friend of declaring class type an object is passed.

    int mean (base &obj)
    {
        int m;
        m = 0; val 1 + 01 - val 2;
        return m;
    }

    void main()
    {
        base obj;
        obj. getData();
        cout << "mean value": mean (obj);
    }
};
```
Multiple exception handling
It is possible that a program segment can have more than one condition through an exception. In such cases, we can associate more than one catch statement with a try (must like the condition in a switch statement).

The following structure can be used for multiple catch statements:

```
try
{
    statement(s);
}

catch (type 1 Argument)
{
    // code
}

catch (type 2 Argument)
{
    // code
}
```

Write a C++ program to handle 3 exceptions.
Based on its data types, if user enters 1 (integer) then display a message 'got an integer exception' if user enters '-1' than it should display 'got a double exception', if user enters 1.5 than it should display 'got a float exception'.
(by using try, catch & throw).
Catch / How to catch all the exceptions?

In some situations, the programmer may not be able to anticipate all the possible types of exceptions and in this situation, it is not possible to design independent catch blocks to catch them.

try

    statements(s);

    catch()

    statements(s);

    else

    statements(s);

Inline Function

    void main()

    statement(s)

    sum = add(x, y);

    int add(int i, int j)

    {

    }

- Inline function is a powerful concept i.e. commonly used with the classes. ATE
- All the functions which are designed inside the class become inline functions automatically.
In this process (designing an inline), the compiler place a copy of the code of that function at each point where the function is called at compile time.

Any changes to an in-line function would required all clients of function to be compiled because the compiler would need to replace all the code once again otherwise it will continue with all functionality.

like:
```
myfile.cpp

    sum() {
        y = add (i, i);
    }
```

then include (`# include <myfile.cpp>`) at the beginning of the program.

The meaning of this particular line: to inline a function, or to make a function inline, we need to place keyword `inline` before the function name, and define the function before any calls made to the function.

It is always recommended to have the size of the inline func small (may be 1 or 2 line).

It is totally depend on the compiler whether the func is to be treated as inline or not.
- If the function size is big, then compile simply reject fun in as a inline function.

- The no. of inline func should be less, that’s why it is always recommended to design all the class function outside of the class by using the scope resolution operator because whenever we use inline func, it increases the size of the program. (Beef there is redundancy, and this is the drawback of the func.)

* Constructors in Derived classes.

* Constructor in Inheritance

1. Base class constructors are always called in the derived classes.

2. When ever we create derive class object — first the base class constructor is executed then the derived class constructor finish the execution.

3. Whenever we create any object of a derived class — first the base class constructor is called then the derived class constructor will be called.

4. A constructor plays a very active role in initializing an object.

* Snap NOTE: Constructor plays an import role while using constructors. During inheritance in that as long as base class constructor does not play any argument, the derived classes
need not to have a constructor function

```c++
class a
{
  int x;
  public:
    a()
    {
      x = 10;
      void showdata() { cout << "value of x" << x; }
    }

  class b : public b
  {
    int y;
    public:
      void getdata()
      {
      }
      void showdata()
      {
      }
      void main()
      {
        b b1;
        b1.show(x);
        b1.showdata();
      }

      However, if a base class contains a constructor
      with one or more arguments, then it is
      mandatory for the derived classes to have a
      constructor & pass the arguments to the base
      class constructor.

      While applying inheritance, we usually create
      object using derived class so it make sense.
```
for the derived class to pass arguments to the base class constructors. So this process, the derived class takes the responsibility of supplying the initial values with space class. The constructor of the derived class precedes the entire list of required values as its happening & passes them onto the base class constructor in order in which they are declared in the derived class.

```cpp
#include <iostream.h>

class alpha
{
  int x;
  public:
  alpha (int i)
  {
    x = i;
    cout << "alpha is initialized"; 
  
  void show_x()
  
  cout << "alpha is initialized"; 
  
  cout << "value of x" << x;

class Beta
{
  float y;
  public:
  Beta (float k)
  {
    y = k;
    cout << "Beta is initialized"; 
  }```
```
A what does your constructor behave in inheritance?

3. What does your constructor behave in inheritance?

2. What does your constructor behave in inheritance?

1. What does your constructor behave in inheritance?
Static Data Members in C++

```cpp
class a
{
    static int n;
    public:
        void get_data()
        {
            01 - n = 10
            02 - n = 20
        }
        void show_data()
        {
            01 - n = 20
            02 - n = 30
        }

    void main()
    {
        a a1;
        a1.get_data();
        a1.show_data();
    }
}
```

When we declare the normal variable in a class, each different copies of those data members created with the associated objects.

But in some cases, when we need a common data member throughout that should be same for all objects, we can not do this using normal data members. To fulfill such cases, we need static relevance.

A static variable (data member) is declared
with a keyword `static`. It is also known as class member at only and only single copy of the variable creates for all objects.

How to define a static variable in the class:
```cpp
static Data_Type Variaable_name;
static int x;
```

How to define a static data member:
It should be defined outside of a class with the following syntax:
```cpp
Data_Type class_name :: Variable_name = 10;
int class :: demo :: x = 10;
```

```cpp
class demo
{
static int x;
public:
  static void fun()
  {
    cout << "value of x = " << x;
  
  int demo :: x = 10;
  void main()
  {
    Demo 01;
    01. fun();
    
    Why we use scope resolve operator?
  
  
  
```
Define a class `Travel` in C++ with the description given below:

(i) private data members:
   - `code` of string type
   - `no. of adults`, `type of integer`
   - `no. of children`, `type of integer`
   - `distance`, `type of integer`
   - `total fare`, `type of float`

Private member function of class is given below:

A constructor to assign initial values as follows:
- `code` with a seed null,
- `no. of adult` as zero,
- `no. of children`, `distance`, `total fare` as zero.

A function called `assign fare` calculates and assigns the value of the data member, `total fare` as follows:
- for each adult `₹ 500` if the distance is `> 1000`,
- `₹ 300` if the distance is `> 500 and < 1000`.
- `₹ 200` if distance is less than 500.

For each child, the above fare will be 50%.

For e.g. if distance is 750, no. of adults 3 and no. of children 2, then the total fare should be calculated as:
- `3 x 300 + 2 x 150 = 1200`.

A function `input travel` is to be created to take input values of the data members `code`, `no. of adults`, `no. of children`,
distance and invoke assign face function also.
A punx show travel & display the content of all datamembers of the travel.

# Type of classes in C++
We can have 2 types of classes
1. Global class
2. Local class

Local class in C++
1. A class declared inside a function become local to that function & is called local class in C++.

```cpp
#include <iostream.h>

fun1()
{

class Test
{

data members;

public:
mem per function(s);

fun1();

void main()
{

fun1();

}

```

```
Following are some facts about the local classes:

1. A local class type name can only be used in the enclosing function.

```cpp
# include <iostream.h>

fun 1()
{
    class test
    {
        data members;
        public:
            member function(s);

    }

    void main()
    {
        fun 1();
        test t1;  // Cannot be part of the local class.
    }
}
```

The program 1 will display the compilation errors because the object of the local class can not be declared / created outside of the class.

To remove the compilation error, the object of the local class must be created inside the enclosed function.

```cpp
# include <iostream.h>

class test

fun 1()
{
    class test
```
```cpp

t
Public:
void method ()
{
    cout << "function of local class";
    test t1;
    t1. method ();
}

void main ()
{
    fun 1 ();
}

(2) Local class is can access other local classes of same function. Even local classes access global types variables & functions.
```
method 1
{
    cout << "value of x = " << x;
    for (int i = 0; i < 2; i++)
    {
        method 1();
    }
    int main()
    {
        fun 1();
        return 0;
    }
}