(I) In all cases, we have used member fn such getdata() to provide initial values to the private member variable.

for eg → A.getdata();

All these function calls are used with appropriate objects that have already been created; These fns can't be used to initialize the member variable at time of creation of their objects.

for eg
```
int m = 20; // at time of declaration of variable
float x = 5.75;
```

So we should be able to initialize a class type variable when it is declared.

(II) Illy when variable goes out of scope, the compiler automatically destroys the variable. It has not happened with objects we have studied.

(III) A constructor is special member fn which enables an object to initialize itself when it is created. because it constructs the values of data member of class.

(IV) for eg

(a)
```
class SBS
{
long int PinCode;
char Address[20];
char city[10];
public: SBS()
{
Pincod = 152004;
Address = {"SBSSTC"};
city = {"P2R"};
};
void main()
{
SBS S; // constructor called;
```

⇒

(b) outside the class;
```
SBS:: SBS() // default Construct
{
}
```

Characteristics of Constructor ⇒

(i) It should be declared in public section
(ii) They are called automatically when object are created.
(iii) They don't have return type, not even void.
(iv) they can have default arguments (with no arguments)
(v) It has same name as class name, it is member fn of class whose body can be written inside or outside the class.
(vi) User can pass more than one argument

Default Constructor ⇒ that accepts no arguments.

```
Class A
{
    int m;
    int n;
Public: A()
    {
m=10;
n=20;
    };
A a;
```

```
Class A
{ int m, n;
Public: A(int a, int b)
    {
     m=a;
     n=b;
    }
};
A a(10,20); // constructor with
            Default Arguments.
```

Types of Constructor

(i) Default Constructor →
(ii) Parameterized Constructor
(iii) copy constructor

For eg →

```
class cube
{
int a;
};
void main()
{
cube c;
cout << c.cube;
}
```
(a)

O/P ⇒ 0

```
Class cube
{
int a;
Public: Cube()
    {
     a=10;
    }
};
void main()
{
cube c;
cout << c.cube;
}
```
(c.a) (10)

① if we do not define a construct explicitly, the compiler will provide a default constructor implicitly.

Note 1 → if user want to execute some code / instruction immediately after obj is created, you can place that code inside the body of constructor.

(11) Parameterized Constructor

```
Class cube
{
int a;
Public:
cube(int x)
  {
    a = x;
  }
};
void main()
{
cube C1(10);
cube C2(20);
Cout << C1.a;
Cout << C2.a;
}
```

nested Constructor / more than one argument

```
a, b;

cube (int x, int y) sum (int x, int y, int z)
{ cout << x+y;      {
  a=x;              cout << x+y+z;
}                   }
```

Student fee.

```
Class Student
{
int Max_Marks;
long int rn;
float fee;
Public: student (int a, long int b, float c)
        {
          Max_Marks = a;
          rn = b;
          fee = c;
        }
};
void main()
{
Student S(50, 12011716, 15000);
}
Student S = Student(50, 12011716, 15000);
```

Implicit call → A a(20, 30);

Sum s(30, 30.5); // Shorthand Method

Explicit call → A a = A(20, 30);

Sum S = Sum(20, 30);

name of constructor is explicitly provided to invoke it

Copy Constructor ⇒ (i) to initialize the value of an object by copying values of another object of same class.

(ii) It is form of (class name &).

For eg → A a; // default constructor

A b = a; // Copy Constructor

(iii) It takes a reference to an object of same class as an argument.

For eg

```
class A
{
Private : int a;
          float b;
Public: A(int x, float y)
        {
         a = x;
         b = y;
        }
      A(A &m)
        {
        a = m.a;
        b = m.b;
        }
void show()
 {
  cout << a;
  cout << b;
 }
};
void main()
{
clrscr();
A m1(20, 15.5);
A m2 = m1;
A m3(m1);
m1.show();
m2.show();
m3.show();
}
```

(iv) If user do not define a copy constructor, then compiler will automatically create it & it is public.

(v) Dynamic Constructor => (1) So the value provided at run time is called Dynamic Constructor.

(vi) dynamically initialize the object.

```
class abc
{
int age;
float Marks;
Public: abc(int a, float b)
        {
         age=a;
         Marks=b;
        }
        abc (abc &m)
        {
        age = m.age;
        Marks = m.Marks;
        }
};
void main()
{
  int x;
  float y;
  cin>>x
  cin>>y;
  abc m1(x,y); // dynamic initialization of variable
  cout<<x<<y;
  getch();
}
```

copy constructor must receive its arguments pass by reference. Otherwise copy constructor call results in infinite recursion

Constructor Overloading →

① More than one constructor with different argument

Practice WAP which reads a complex No., copy that into another by using copy constructor.

Ans

```
class complex
{ int r, i;
public: complex(int a, int b)
        {
          r=a;
          i=b;
        }
complex(complex &c)
{
   r=c.r;
   i=c.i;
}
void show()
{
cout<<r<<"+i"<<i;
}
};
void main()
{
complex c1(10,15);
complex c2(c1); // copy
c1.show();
c2.show();
getch();
}
```

Default Copy Constructor

not written

⇒ default copy constructor

Constructor with default Argument

```
class complex
{
int x, y;
Public: complex (int a, int b=0)
        {
          x=a;
          y=b;
        };
```

⇒ default value of y

complex c1(6); ⇒ x = 6, y = 0

complex c2(6,5); ⇒ x = 6, y = 5

Que

created by a constructor.

(ii) name of 2d is same as constructor but it is preceded by a(~)

(iii) 2d automatically called by compiler upon exit from program to clean up storage which is not taken by object.

(iv) objects are destroyed in reverse order of creation.

```
complex()
{
x = 10;
y = 20;
}
~complex()
{
}
```

(v) no arguments are passed in destructor.

Dynamic Constructor → are used with new & delete keyword

Initializer list →

local & global class →

nested class →

abstract class →

empty class →

bit field & class →

Constant functions →

Initializer List → (I) In constructor, initialization of data member is performed inside body of constructor.

(II) C++ provides an alternative way for initializing the data member of object in the constructor known as Initializer list

(III) It is placed b/w parameter list & opening brace of body of constructor. List begins with colon(:) & is seperated by commas.

For eg
```
int length;
int breadth;
rectangle (int a, int b): length(a), breadth(b)
{ }
```

(IV) It allows initialization of data member at time of their creation which is more efficient as values are assigned before the constructor even starts to execute.

```
class Rectangle
{
int length;
int breadth;
Public: rectangle();
        rectangle(int, int);
        void Area();
};
Rectangle::rectangle(int a, int b): length(a), breadth(b)
{ }
void Rectangle:: Area()
{
int area;
area = length * breadth;
cout << area; }
```

```
void main()
{
clrscr();
Rectangle r(10, 30);
r.Area();
getch();
}
```

nested class

```
class student
{
 Private: int rollno;
          char name[20];
 Public:  class date
          {
          Private:
          int dd, mm, yy;
          Public:
          void show();
          void read();
          } dob;
    void read();
    void show();
    };
    void main()
    {
    student S1;
      S1.read();
      S1.show();
      getch();
    }
void student :: date :: read()
        {cin>>dd>>mm>>yy;
        }
void student :: date :: show()
        {
        cout<<dd<<mm<<yy;
        }
void student :: read()
        {
        cin >> rollno >> name;
        cout<<" dob of stu"; dob.read();
        }
void student :: show()
        {
        cout<<rollno<<name;
        dob.show;
        }
```

It is useful for implementing a powerful data structure such link list and tree.