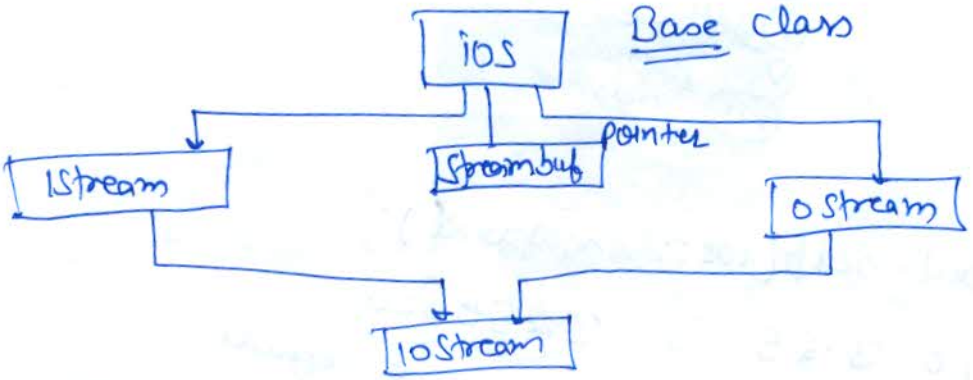
C++ Stream classes



(A) Unformatted → I/O operation

(1) unformatted input stream →

(a) cin >> c → skip white space

(b) cin.get() — char c;
cin.get(c)

(c) cin.getline() — char c[10];
cin.getline(c, 10)

(d) read()

(2) unformatted O/P stream →

(a) cout <<

(b) cout.putl() — cout.getput(c)

(c) cout.write — cout.write(c, 10)

(3) unformatted I/O →

cin & cout & both

formatted I/O function

I/O → IOS class function & flag → (a) width() — cout.width(w);
cout<<a;

(b) precision() → cout.precision(2);

(c) fill() → cout.fill('a');
cout.width(10);
cout<<a;

(d) setf()

cout.setf(arg1, arg2)

cout.setf(ios::left, ios::adjustfield)

scientific floatfield

(II) setprecision()
(III) setfill()
(IV) dec, hex, oct
(V) setbase
(VI) endl
(VII) ends

for eg

(I) cout.setf(ios::showpoint);
i.e 3.25 ⇒ 3.250000
6 Default

(II) cout.setf(ios::showpos); Show +Sign
i.e + 275.500

(III) cout.fill('*');
cout.setf(ios::left, ios::adjustfield);
cout.width(10);
cout<<"Name BENU";

N	A	M	E		B	E	N	U	*

(IV)
```
int size=30;
char S[30];
cout<<"Enter Name of stu";
cin>>S;
cout<<S;
cin.getline(S,size);
cout.write(S,size);
```
Space
Amar Singh
⇒ Amar
Amar Singh

---

cout<<setioflags(ios::left)<<setw(20)<<d<<endl;

Constants

① const int model = 90; ✓
② const int v[] = {1,2,3}; ✓
③ const int x; ✗ error

for eg
```
void f()
{
  model = 200;   ✗
  v[2]++;        ✗
}
```

Constant & pointer

① char *const p // const pointer to char
② char const *p // pointer to const char
③ const char *const p // const pointer to const

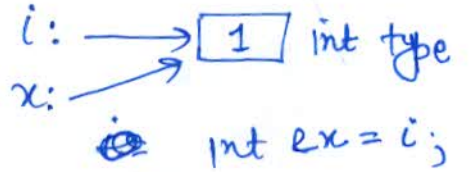
for eg
```
int a = 1;
Const int c = 2;
Const int *p = &c; //
```

References → It is alternative name for an object. It must initialize the reference.

for eg →①
```
int i = 1;
int &x = i;   // x and i now refer to same int
int b = 1;    // b = 1;
x = 2;        // i = 2;
```

②
```
int i = 1;
int &r1 = i;   // r1 is initialized
int &r2;       // error
```

③
```
int i = 0;
int &r1 = i;   // both refer to same region i.e int
r1++;          // i is incremented to 1
```



int &x = i;

Arrays → (i) Collection of elements of same type.
(ii) It is indexed from 0 to (N-1)

i.e

(i) float b[3] // array of three float.

(ii) char *a[3] // an array of 3 pointers to char
a[0] – a[3]

Array initializers →

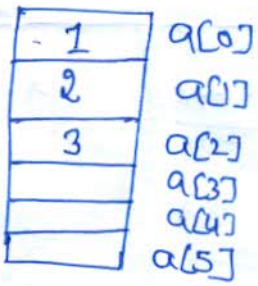
(1) int a[] = {1, 2, 3, 4}

char v[] = {'a', 'b', 'c'}

(2) char v[3] = {'a', 'b', 'c', 'd'} // toomany cha

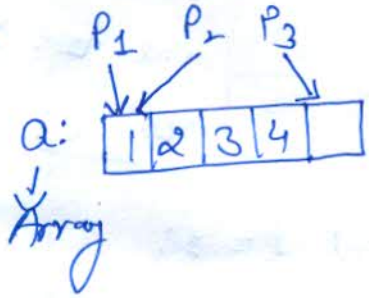
char v[3] = {'a', 'b', 'c'} // ✓

for eg int a[3] = {1, 2, 3}
↓
Size, 3 elements

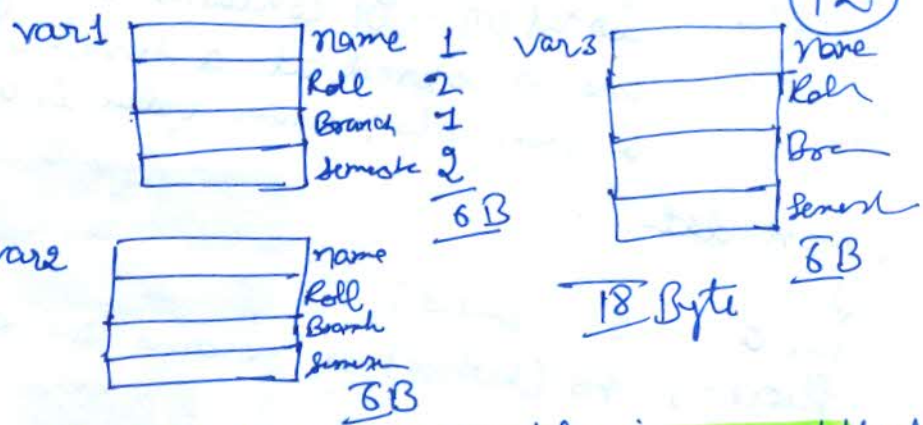


Pointer into Arrays

```
int a[] = {1,2,3,4};
int *P1 = a;
int *P2 = &a[0];
int *P3 = &a[4];
```



```
Struct student
{
char name[80];
int roll_no;
char Branch;
int semester;
};
```



(I) var1, var2, var3;

Each & every variable is given diffrent space in memory while creating variable

(II) Struct student var1, var2, var3;

(III) Student var1, var2, var3;

Accessing member of structure →

(I)
```
Student var1;
var1.name = "Akash";
var1.Roll_No = 130;
```

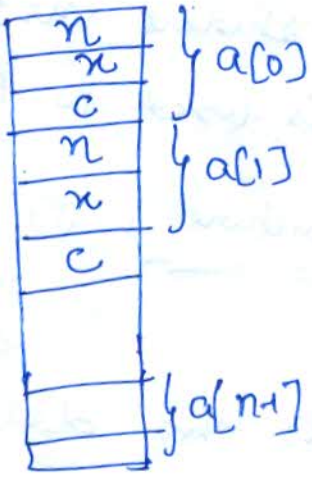
(II) Student var1 = {'Akash', 130, "CSE", 5};

(III) Student var1 = {"Akash", 130, "CSE"};

data member is initialized to zero

Array of Structure

```
Struct A
{
 int n;
 float x;
 char c;
};
A a[10];
```



a is array of 10 elements each element is structure type.

Union → union store value of different data types in single location. It contains many different values but only one is stored at a time. When new value is stored at a time, the previous value is automatically erased.

```
union data
{
int a;     2B (lowest)
float b;   4B (highest) → reserved for all types.
};
data d1, d2;
d2.a = 10;
d1.b = 20;
```

Que 1. Differ b/w structure & union.

Limitation in Structure

(I) It does not permit data hiding i.e struct variable can be accessed by any function anywhere in scope.

(II) All members are public.

(III) In C language, struct student var1; ✓

In C++, student var1; ✓

(IV) Mostly it is used to hold data member only.

Extensions to Structure → (I) It combines both data member & function together in single unit

(II) It permits the data hiding.

(III) Data Member ~~can be~~ are prive by default.

(IV) It declared as user defined data type i.e class.

Class → (I) user defined data type
(II) Binding of data and function

Declaration of class in C++ →

```
class class_Name
{
access-specifier1 : member-data1;   → visibility of member;
                    ______ 2;
                    member fn -;
access-specifier2 : member-data;
                    member fn -;
access specifier3 : member-data;
                    member-fn;
};
```

(I) Private → can be accessed only from within class. hidden from outside the class known as data hiding.
(II) Public → Can be accessed only from outside class.

for eg →

Inside the class →

```
Class student
{
Private : char name[80];   → data Mem
          int age;
Protected : int roll_No;
            int semester;   → member fn
Public : void getdata()
   {
   Cout<<"Enter name";
   cin>> name;
   cout<<"Enter age";
   cin>>age;
   cout<<"Enter the semester";
   cin>> semester;
   }
void showdata()
   {
   Cout<<"Name:"<<name;
   };
```

Creation of object

```
Student S1, S2;
    or
S1, S2, S3

void main()
{
student S1, S2;
S1. getddata();
S2. showdata();
getch();
}
```

Accessing Member of class

(I) Private Member can not access outside the class. It can be accessed only by member fn of that class.

(II) Public member can be accessed outside the class such as

```
Object_Name.member_name ;
        AND
Object_Name.function_Name(argument);
```

(III)

```
S1.getdata();
S2.getdata();
S1.showdata();
S2.showdata();
```

```
getdata();        X
S1.age=25;        ✓
```

↓ Private member can not be accessed directly by object bt can be accessed by member fn.

foreg →

```
Public: int age;
};
Student S1;
S1.age=10; ✓
```

foreg → sum of two no. with help of class;

Writing body of Member fn outside class →

```
Class student
{
Private:
Protected:
Public: void getdata();
        void showdata();
};
void student::getdata()
{   }
vod student::showdata()
{   }
```

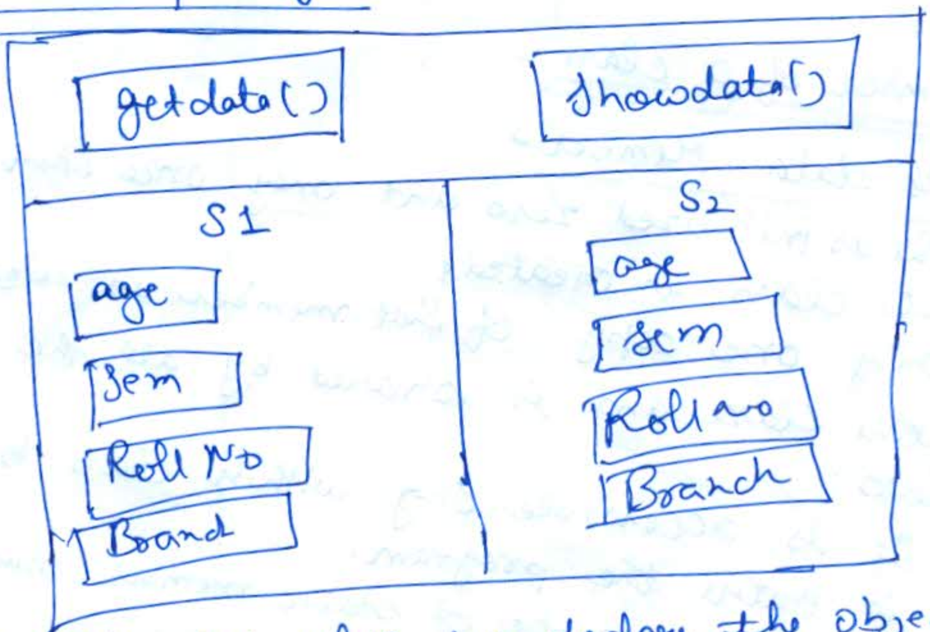
Private member fn as member →

```
class student
{
Private : int age;
          char name[80];
          void getdata();
   Public : showdata();
};

student S1;
S1.getdata(); X error(illegal) // Can't be accessed by object
S1.age = 25;       same           but
void Student :: showdata()
{
    getdata();     // can be called by another member fn
                      of class.
}
```

Memory allocation for object →

for eg →



(I) memory is allocated when we declare the object.

(II) there is seperate memory for each separate object

(III) but all member fn of differ object share same Common memory which declared once at time of declaration of member fn of class.

```
Student S1[5];
S1[5].getdata();
S1[i].showdata();

void main()
{
student S1[5];
for(int i=0; i<=4; i++)   // to enter data of 5 student
 {
 S1[i].getdata();
 }
for(int i=0; i<=4; i++)   // to show data of 5 student.
 {
 S1[i].showdata();
 }
getch();
}
```

Static Member of a class →

(1) Static data Member →

(a) It is initialized zero and only once when first object of its class is created.

(b) Only one copy of that member is created for the entire class and is shared by all the Object of the class.

(c) It is accessible only within class but its lifetime is entire the program.

(d) The type & scope of static member must be defined outside the class.

i.e
```
int count::c;
int count::c=10;
```

Static Member

for eg

```
class counter
{
int n;                          // default private data
static int count;               // static member data
Public: void getdata(int n1)
        {
          n = n1;               // count is zero
          count++;              // 1
        }
        void showdata()
        {
          cout << " count is " << count << endl;
        }
};
int counter:: count;            // declaration of static scope & type of static data outside class.
void main()
{
counter c1, c2, c3;
c1.showdata();                  // 0 //zero
c2.showdata();                  // 0
c3.showdata();                  // 0
c1.getdata(100);                // Calling to fn // count 1
c2.getdata(200);                //               // count 2
c1.showdata();                  // 2
c2.showdata();                  // 2
getch();
}
```

Static data member are stored seperately rather than as an part of object.

Static Member function → (i) Static fn can have access to only other static members (fn or variable) declared in same class.

(ii) Static member fn can be called using class name rather than object.

class Name :: fn-Name;

```
class counter
{
int n;
static int count;
Public: int getdata()
        {
        return (count++)
        };
int counter:: count ;
void main()
{
cout << counter:: getdata();
}
```

Object as function arguments

(1) Copy of the entire object

(11) only address of object is transferred

```
class complex
{
float realp;
float imagp;
Public : void getdata();
         void sum(complex c1, complex c2);
         void output();
};
void complex::getdata()
{
cout<<" Enter real part";
cin>> realp;
cout<<" ------ imaginar part";
cin>> imagp;
}
void complex::sum(complex c1, complex c2)
{
    realp= c1.realp + c2.realp;
    imagp= c1.imagp + c2.imagp;
}
void complex :: output()
{
cout<< realp <<" +i"<< imagp<< endl;
}
```

```
void main()
{
complex c1, c2, c3;
c1.getdata();
c2.getdata();
c3.sum(c1, c2);
c1.output();
c2.output();
c3.output();
getch();
}
```

Return by function as object

①
```
complex complex::sum(complex c1)
{
  complex temp;
  temp.realp = c1.realp + realp;
  temp.imagp = c1.imagp + imagp;
  return(temp);
}
void main()
{
 complex x, y, z;
 x.getdata();
 y.getdata();
 z= y.sum(x);
 x.output();
 y.output();
 z.output();
 getch()
}
```

```
y.sum(x);
c1.realp = x.realp
realp = y.realp
```

②
```
complex complex::sum(complex c1, complex c2)
{
  complex c3;
  c3.realp = c1.realp + c2.realp;
  c3.imagp = c1.imagp + c2.imagp;
  return(c3);
}
x.getdata();
y.getdata();
z= sum(x,y);
z.output();
```

Constant Member fns :

```
class constant
{
 int data;
 Public: void assign()
       {
        data=20;
       }
       void changedata() const
       {
        data = 50; // error
       }
       void showdata() const
       {
        cout<<data<<endl;
       }
       };
       void main()
       {
       constant c1;
       c1.assign();
       c1.changed
       c1.show ;
       }
```

access to private data member of class.

(iii) when two classes want to share common fn among both object of two classes then fun is made to be friendly with both classes.

Properties (i) It is not in the scope of class to which it has been decared as ~~class~~friend.

(ii) It is not called by object of class.

(iii) It is invoked ~~like~~ Normal function without any object

(iv) It is not as public/private i.e It is non-member function

(v) It has object as arguments.

(vi) It uses dot operator to accen the private member through object.

```
class employee
{
  float gP;
  float salary;
  Public: void setvalue()
          {
            gP = 1400;
            salary = 10000;
          }
  friend float total(employee e); // non-member fn
};
float total(employee e)
{
  return (e.gp + e.salary);
}
int main()
{
  employee d;
```

```
d.setvalue();
Cout<< total(d) <<endl;
return 0;
}
```

Normal fn

Member fns of one class can be friend of another class

```
class X                          class Y
{                                {
____                             ____
int fun();                       friend int X::fun();
};                               };

class Z
{
____
friend class X; //all member fn of class A are
};                               friend to Z;
```

friend fun of two different classes

```
class A
class B
{
int x;
Public: void get_data()
        { x=10;
        }
      friend void max(A a, B b);
};
class A
  {
  int y;
  Public : void getdata()
           {
           y=20;
           }
         friend void max(A a, B b);
  };

void max(A a , B b)
    {
      if (a.x > b.y)
       cout << a.x;
       else
       cout << b.y;
    }
int main()
  {
     A aa;
     B bb;
     aa.getdata();
     bb.getdata();
      max(aa, bb);
```

Friend class

```
class A
class B
{
  ___
  ___
  ___
  ___
friend A; // friend class of B, A can use all members of class B
};
class A
{
___
___
___
___
___
};
int sum(B t);
};
int A :: sum(B t)
    {
      int S;
      S = t.x + t.y / y
     return(S);
    }
void main()
{
 A a;
 B b;
a.getdata();
b.getdata();
 b.sum(a);
getch()
}
};
```

local class →

Friend Class

```
class A;
class B
{
int x;
Public: void getdata()
        {
        x=10;
        }
 Friend A; // A can use all member fn of class B
};

class A
{
int y;
Public: void getdata()
        {
        y=20;
        }
    int sum(B t);
};
int A :: sum(B t)
  {
   int S;
   S = t.x + y;
   return(S);
  }
void main()
 {
 A a;
 B b;
 a.getdata();
 b.getdata();
 cout<< a.sum(b)
 getch();
 }
```

class A

class B

class A