Deepak Aryal
C++

Scanned by CamScanner

Downloaded from CSIT Tutor

# C++

## History of C++

→ Developed by B. straustrap at Bell labs around 1980's.

## Compare of procedural (structural) programming and object oriented programming (OOP)

### Charecterstics of procedural programming

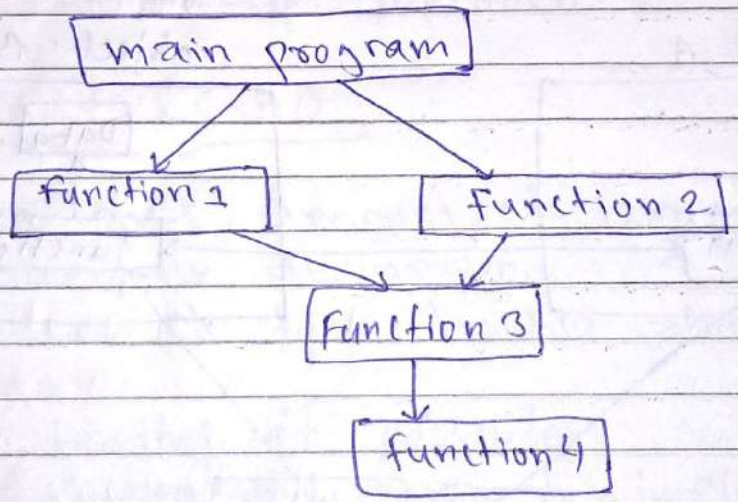



fig:- structural of procedural programing

### Relation of function and data

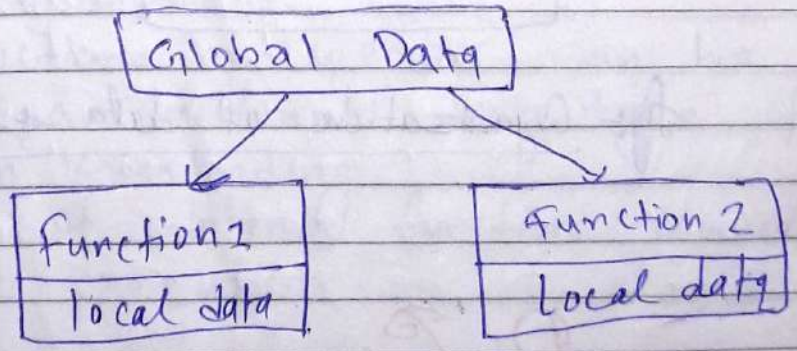



fig:- Relationship of function & data

Scanned by CamScanner
Downloaded from CSIT Tutor

## Charecterstics (procedural)

→ Emphasis is on doing things (algorithm).
→ large programs are divided into smaller programs known as function.
→ Most of the functions share global data.
→ Use top-down approach in the program design.

## Charecterstics of OOP:

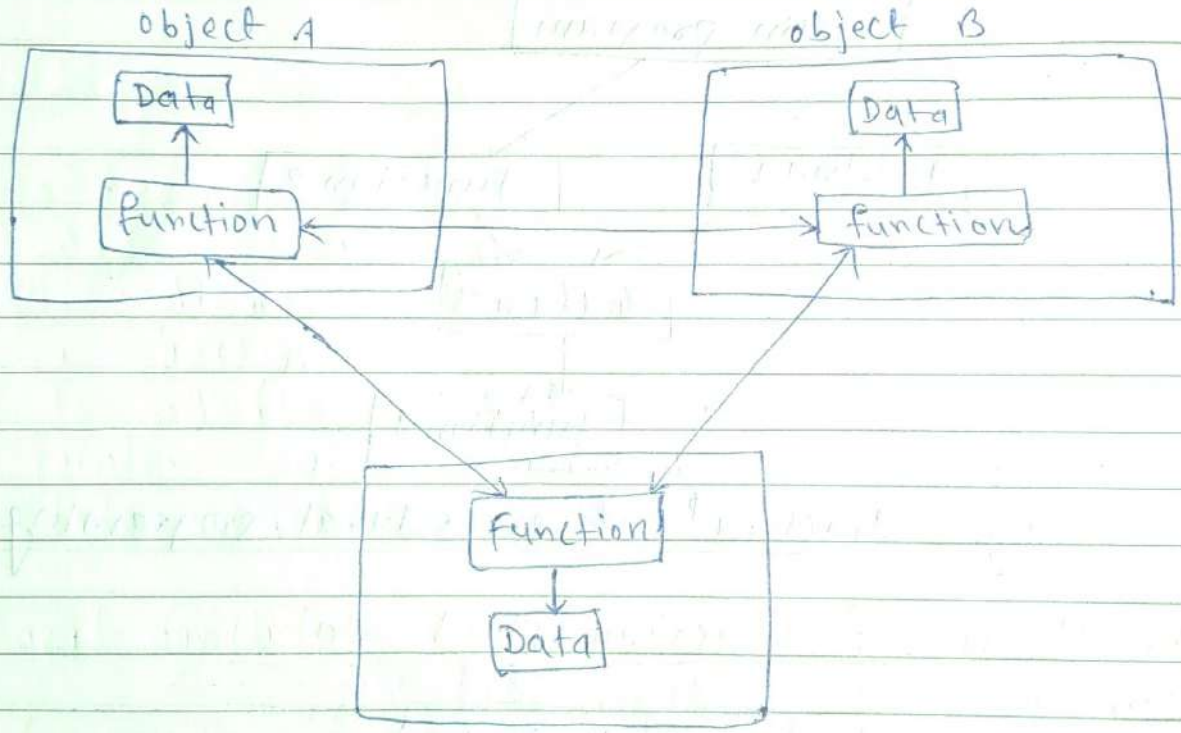



fig:- Organization of data + function in OOP

~~Chase~~

P Po

Scanned by CamScanner
Downloaded from CSIT Tutor

## Charecterstics (non-procedural)

→ Emphasis is on data rather than procedure.
→ Programs are divided into objects.
→ functions that operate on the data of an object are tied together.
→ Data is hidden and can't be accessed by external functions.
→ objects may communicate with each-other through functions.
→ follows bottom-up approach.

## Features (C++)

i) It has vast amount of library functions.
ii) Spot variable declaration.
iii) Don't have to face trouble about format of I/o data.
iv) Dynamic binding if possible.
v) Syntax, if simpler and easy to implement.
vi) It also has the capability to compile structural programming. Hence it is not a pure oop.
vii) facility of passing default argument and refrence arguments.
viii) Some function or operator can be used to operate on different data type function or operator overloading.
ix) Provision of inline function which save the execution.

Note:- Array always passes through the reference.

Scanned by CamScanner
Downloaded from CSIT Tutor

## Data types of C++

- User-defined type
  - Structure
  - Union
  - Class
  - Enumeration
- Built in type
  - Integer
    - int
    - long int
  - Char
  - void
  - floatingpoint
    - float
    - double
- Derived type
  - Array
  - pointer

## Void

→ To specify when it is not returning any value in function.

→ To indicate the empty argument list to a function.

→ To indicate general datatype. A generic pointer can be assigned to any base data type.

## Input function:

→ The input stream provides to read operation with input device (Keyboard). Cin is followed by the symbol >> is called extraction operator (or get) and then with the variable into which the input data is to be stored.

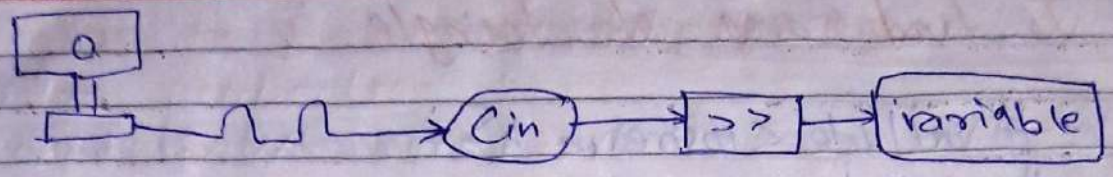
```
Cin >> variable;
```

Object → Cin

extract operator → >>

Scanned by CamScanner

Downloaded from CSIT Tutor



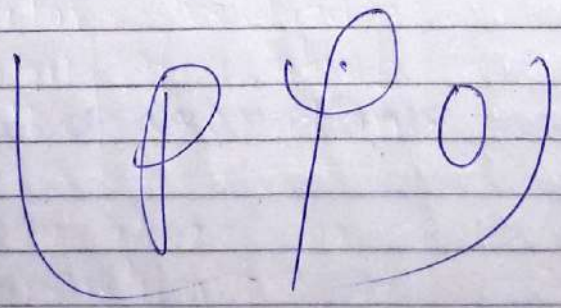

Eg:

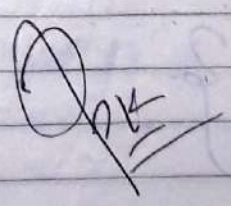
```
#include<iostream.h>
void main()
{
    int i;
    cin >> i;
    cout<<i;
}
```

**Output stream:** It provides to perform to output on screen (standard o/p device). Cout is followed by the symbol << called the insertion or put to operator and with the value of variable displayed on the monitor.

```
Cout << variable;
```

Cout object → Cout
put to operator → <<





Scanned by CamScanner
Downloaded from CSIT Tutor

# To find area of traingle.

```
# include <iostream.h>
# include <math.h>
void main( )
{
   float a, b, c, s, area;
cout <<"\n Enter values:";
 cin >> a>>b >>c;

      s = (a+b+c) /2.0;

if( (a+b)>c && (b+c) >a && (a+b)> b)
  {
  cout"\n traingle formed:";

   area = sqrt (s*(s-a)*(s-b)*(s-c));

   cout << area;
  }
  else
     {
      cout"\n traingle not formed:";

     }

}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

# Program to find quardatic equation.

```
#include <iostream.h>
#include <math.h>
void main()
{
float a, b, c, x1, x2, x, d;

cout << "\n Enter values:";
cin>>a>>b>>c;
d= b*b - 4*a*c;
if(d>0)
{
d= sqrt(d);
x1 = (-b+d)/(2*a);
x2 = (-b-d)/(2*a);

cout << " x1 = " << x1;
cout << " x2 = " << x2;
}

else if (d==0)
{
x = -b/(2*a);
cout << "x = " << x;
}
else
{
cout << "\n Roots are imaginary:";
}

}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## function overloading:-

Overloading refers to the use of the same thing for different purpose. This means, we can perform same function name for different tasks. function overloading can be done in two ways.

i) Different number of arguments
ii) Different kind of argument passed.

→ A function call first matches the prototype having same number of arguments with different types of arguments.

#) Calculating area of circle, rectangle & traingle using function overloading.

```
#include <iostream.h>   #include <math.h>
const float PI = 3.14;
void area(float);
void area(float, float);
void area(float, float, float);
void main( )
{
  float radius;
  cout<<"\n Enter radius:";
  cin >> radius;
  area(radius);
  float le, br;
  cout <<"\n Enter length & breadth:";
  cin >>le>>br;
  area(le, br)
```

Scanned by CamScanner
Downloaded from CSIT Tutor

```
float a, b, c;
cout << "\n Enter a, b & c:";
cin>> a>> b>> c;
area(a, b, c);
}
void area(float r)
{
  cout << PI * r * r;
}
void area(float ll, float bb)
{
  cout << ll * bb;
}
void area(float a, float b, float c)
{
  float s;
  s = (a+b+c)/2.0;
  cout << sqrt (s*(s-a) * (s-b) * (s-c));
}
```

Dipak

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Write a program to swap values of two integers, two characters, two floats & two strings respectively using function overloading.

```
# include<iostream.h>  # include<string.h>
# define MAX 10;
void swap (int , int);
void swap (float, float);
void swap (char, char);
void swap (char[], char[]);
void main()
{
  int a, b;
  cout<<"\n Enter two integers:";
      cin>> a>>b;
      swap (a,b)
  char a, b;
  cin >> a>>b;
  swap (a, b);
  cout << a<< b;
  float m, n;
  cin>>m>> n;
  swap (m,n);
  cout << m<<n;
  char s1[20], s2[20];
   cin>> s1>>s2;
  swap (s1, s2);
  cout<<s1<<s2;
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
void swap (char &&aa, char &&bb)
{
 char temp = aa;
        aa = bb;
        bb = temp;
 }
void swap (int &xx, int &yy)
    {
      int temp = xx;
            xx = yy;
            yy = temp;
    }
  void swap (float &mm, float &nn)
    {
      float temp = mm;
            mm = nn;
            nn = temp;
    }

  void swap (char ss1[20], char ss2[20])
    {
      char temp[20];
        strcpy (temp, ss1);
        strcpy (ss1, ss2);
        strcpy (ss2, temp);
    }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## Method of function declation

### Passing by value

```
void abc(int);
void main()
{
  int x=5;
  abc(x);
  cout<<x;
}
void abc(int y)
{
  y = y*2;
  cout<<y;
}
```

### Passing by refence

```
void abc(int &)
void main()
{
  int x=5
  abc(x);
  cout<<x;
}
void abc (int &y)
{
  y=y*2
  cout<<y;
}
```

### Passing by Refence with Pointer

```
void abc(int*)
void main()
{
  int x=5;
  abc(&x);
  cout<<x;
}
void abc(int *y)
{
  *y = *y*2;
  cout<<*y;
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## Default Argument :-

A function can be called without specifying all its arguments. The function declaration must provide default values for those arguments that have not specified.

```
# include <iostream.h>
void repeat (char='*', int=5);
void main( )
{
  repeat ( );
  repeat('=');
  repeat(+; 4);
}
void repeat (char c, int n)
{
  for (i=0; i<n; i++)
    cout << c;
    cout << "\n";
}
```

→ While using default arguments, parameters without default arguments are placed first and those with default values are placed later (because of the convention of storing the arguments on the stack from right to left).

## Inline function :-

Function save the memory space by avoiding the duplication of the same code at each point of calling.

But functions call involve

Scanned by CamScanner

Downloaded from CSIT Tutor

execution time overload. C++ provides inline functions to reduce function call overhead especially for small functions.

## function overload execution time due to

→ The jump to the function
→ saving Registers
→ Passing arguments onto the stack in the calling program.
→ Removing them from the stack.
→ Restoring registers.
→ returning values to the

## Inline Keywords:

It is actually just request to the compiler. Sometimes the compiler will ignore the request & compile the function as normal function.

## #) Example of Inline function

```
# include<iostream.h>
void square (int);
void main()
{
  square (3);
  square (5);
  square (8);
}
```

```
Inline void square(int i)
{
  cout << i* i;
}
```

Compiled program

```
void main()
{
  cout << 3*3;
  cout << 5*5;
  cout << 8*8;
}
```

Code copied here

Scanned by CamScanner

Downloaded from CSIT Tutor

# Class and objects

```
#include<iostream.h>
Class small
{
  private:
    int data;
  public:
    void set(int d)
    {
      data = d;
    }
  void show();
    {
    cout << "\n Data=" << data;
    }
}
void main()
  {
    small obj1, obj2;
    obj1.Set(11);
    obj2.Set(22);
    obj1.show();
    obj2.show();
```

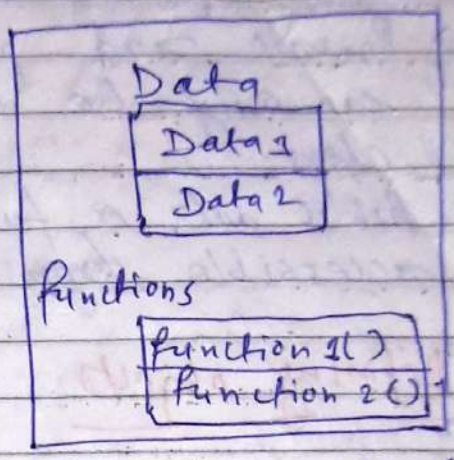
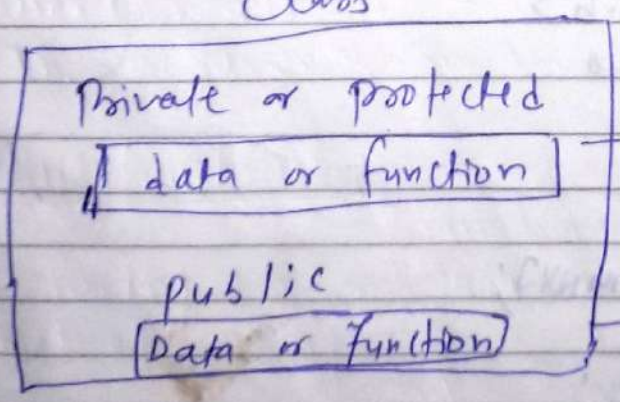



fig: class contain data & function

Class




→ Not accessible from outside of class

→ Accessible from outside of class

Scanned by CamScanner
Downloaded from CSIT Tutor

→ Planning data or function into a single entity is the central idea of OOP.
→ A key function of OOP is data hiding.
→ Private and protected data or function can only be accessible from within the class.
→ Public data or functions on the other hand are accessible from outside the class.

## Defining objects

Small obj1, obj2;
→ Specification for the class small does not create any objects (memory space).
→ It only describe how they will look when they are created.
→ It is the defination that actually creates objects.

#) Write a program to create a class called employee in which data members are name, age & salary. Read the values from keyboard and print on monitor.

```
#define<iostream.h>
# define MAX 20
class employee
{
    Private:
    char name[MAX];
    int age;
    float Salary;
```

Scanned by CamScanner
Downloaded from CSIT Tutor

```
Public:
void input()
{
 cout <<"\n Enter name:";
  cin>> name;
 cout<<"\n Enter your age:";
   cin>> age;
  cout<<"\n Enter your salary:";
   cin>> salary;
  }
   void display ( )
   {
   cout <<"\n name = " <<name;
   cout <<"\n age = " <<age;
   cout <<"\n salary= " <<salary;
   }
};
void main
  {
  employee E
    E. input();
    E. display();
  }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Write a program to create a class called circle in which data members are radius & area then find the area.

```
# include <iostream.h>
~~#define~~ PI 3.14 or const float PI=3.14
class Circle
{
 Private:
 float radius, area;
 public:
  void input ()
   {
    cout<<"\n Enter value of radius:";
    cin>> radius;
   }
  void process ()
   {
    area= PI*radius *radius;
   }
  void display ()
   {
    cout<<"\n Area=" <<area;
   }
  };
 void main()
 {
   Circle c;
   c. Input ();
   c. process ();
   c. display ();
 }
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Write a program to create a class called Account in which following are data & functions:

| Data | functions |
|---|---|
| Name | Input |
| Account no | deposit |
| Account type | withdraw |
| Address | display |
| Balance | |

```
# include <iostream.h>
# define MAX 20
class Account
{
 Private:
char name[MAX], account_no[MAX], account_type[MAX];
char address[MAX];
 float balance;
Public;
 void input()
 {
cout<<"\n Enter your name, address, account
    type and account number:";
 cin>>name>>address>>account_type>>accountno;
 }
 void deposit()
 {
  float x;
 cout<<"\n Enter your previous balance?";
  cin>>balance;
 cout<<"\n Enter amount you want deposit:";
  cin>>x;
 balance = balance+x;
 }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
void withdraw()
{
float amt,k;
  Cout <<"Enter the amount to withdraw";
  cin>> y;
  k = balance-y
  if (k > 5000)
  {
     balance = balance-y;
     else
       {
         Cin >>"In balance insufficient!";
       }
  }};
void display()
{
Cout<<"Name & address =" <<name<< address;
cout <<"Account no & type = " <<account_no<< account type
cout <<" Balance = "<< balance;
}
}

void main()
{
  Account A;
  A. input()
  A. deposit()
  A. withdraw()
  A. display()
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Write a programme to create a class called Rectangle in which data members are length & breadth then find its area.

```
# include <iostream.h>
class rectangle
{
Private:
  float l, b, area;
Public:
  void input ( )
  {
  cout << "\n Enter length.";
  cin>> l;
  cout << "\n Enter breadth";
  cin>>b;
  }
  void process ( )
  {
   area= l*b;
   }
  void display ( )
  {
    cout << "\n Area = " << area;
  }
};
void main( )
{
   Rectangle R;
   R. input ( );
   R. process ( );
   R. display ( );
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Write a program to create a class called Batsman in which data & members function are as follows:

| Data | function |
|---|---|
| i) Batsman name | |
| ii) Address | i) Input |
| iii) No of 4 run made | ii) Update |
| iv) No of 6 run made | iii) display |
| v) Total run made | |

```
#include<iostream.h>
#define MAX 20:
class Batsman
{
  Private:
  Char name[MAX], address[MAX];
  int run4, run6, total
  Public:
    void input( )
    {
    cout <<"\n Enter name:"
    cin>> name;
    cout <<"\n Enter address:";
    cin>> address;
    }
    void update( )
    {
      int run = 0, run1, run2;
    cout <<"\n Enter the 1run mode:";
      cin>> run1;
    cout <<"\n Enter the 2run mode:";
      cin>> run2;
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
cout <<"\n Enter run4 mode:";
cin>> run4;
Cout <<"\n Enter run6 mode:";
cin>> run6;

    run= run1+ run2*2+run4*4 + run6*6;
    total= run;
  }

void display( )
{
    Cout <<"\nBatsman name="<<name;
    cout <<"\n Batsman Address="<<address;
    cout <<"\n 4 run mode= " << run4;
    cout <<"\n 6 run mode= " <<run6;
    cout <<"\n total run = " << total
  }

};
  void main( )
  {
    Batsman B
      B. input( );
      B. update( );
      B. display( );

    }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Write a program using class and objects to simulate result preparation system for a student. The data includes roll_no, name & marks in 3 subjects. The percentage marks & grade are to be calculated as follow:

| Percentage | Grade |
|---|---|
| ≥ 90 | 'A' |
| ≥ 75 ≤ 90 | 'B' |
| ≥ 60 ≤ 75 | 'C' |
| < 60 | 'F' |

```
#include <iostream.h>
#define MAX 30
Class Result
{
 Private:
 Char name;
 int roll_no, mark, C_no, P_no, M_no;
 float P;
 Public:
  void input ()
   {
    cout <<"\n Enter your name:";
     cin>>name;
    cout <<"\n Enter your roll no:";
     cin>> roll_no;
    cout <<"\n Enter the marks in 3 subjects:";
     cin>>C_no>>P_no>>M_no;
  void process ()
   {
     mark = C_no+P_no+M_no;
     P= mark/3.0;
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
cout<<"\n your percent = "<<p;
if(p >= 90)
 {
  cout<<"\n 'A' grade:";
 }
else if( p >= 75 && p =< 90)
 {
  cout<<"\n 'B' grade:";
 }
else if(p >= 60 && p =< 75)
 {
  cout<<"\n 'C' grade:";
 }
else if( p<60)
 {
  cout<<"\n fail:";
 }
else
 {
  cout<<"\n something error";
 }
void display()
 {
  cout<<"\n name ="<<name;
  cout<<"\n roll no="<<roll_no;
  cout<<"\n mark="<<mark;
  cout<<"\n percentage ="<<percent;
 }
};
void main()
{
 Result R();
 R.input();
 R.process();
 R.display();
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

Important

## Constructor

→ Constructor is a member function of a class that is automatically called when an object is created of that class. The constructor has the same name as that of the class name and its main job is to initialize the object to a legal initial value for the class.

### Types of constructors

i) Default constructor
ii) Parameterized constructor
iii) Copy constructor

i) Default Constructors:→ If a class has no explicit constructor defined, the compiler will supply a default constructor. The default constructor provided by the compiler does not do anything. It initialize by the garbage (dummy) value.
Once, the constructor with arguments is used, the default constructor become hidden.
After this, the default constructor cannot be used.

Scanned by CamScanner
Downloaded from CSIT Tutor

## Default constructor

eg:

```
Class A
{
  Private:
    int value;
  Public:
    A(int i)
    {
      value = i;
    }
};
void main()
{
    A obj1(5);   //OK
    A obj2;  //error
}
```

## ii) Parametrized constructor :

Constructor may also take arguments, such constructors are called as parametrized constructor.

eg:

```
Class A
{
  Private:
    int i, j, k;
  Public:
    A(int m)
    {
      i = j = k = m;
    }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## Parameterized constructor

```
A (int x, int y)
{
  i = x;
  j = k = y;
}
A (int x, int y, int z)
{
  i = x;
  j = y;
  k = z;
}
};
void main()
{
  A obj1 (5);
  A obj2 (10, 20);
  A obj3 (1, 2, 3);
}
```

**Output**

| | obj1 | obj2 | obj3 |
|---|---|---|---|
| i | 5 | 10 | 1 |
| j | 5 | 20 | 2 |
| k | 5 | 20 | 3 |

## iii) Copy Constructor:

The parameters of constructor can be any of data types except an object of its own class as a value of parameter.

Hence, declaration of the specification is an error.

```
Class A
{ private:
  int i, j;
Public:
  A(A obj); //error
};
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## Copy constructor

→ However, a class's own object can be passed as a reference parameter.

→ Thus, the class specification shown below is valid.

```
Class A
{
  Private:
  int i,j;
  Public:
  A(A &obj)
  {
    // copy constructor
  }
};
```

→ Such a constructor having a reference to an instance of its own class as an argument is known as copy constructor.

## Characteristics of Constructors

→ Called automatically when the objects are created.

→ No, return type not even void.

→ Can not be inherited by a derived class can call the base class constructor.

→ Constructor can not be static.

→ Default constructor and copy constructor are generated by compiler where needed.

→ Constructors are in public.

#

Scanned by CamScanner

Downloaded from CSIT Tutor

## Destructor

→ It is used to destroy the objects that have been created by the constructor.

→ A destructor is also a member function whose name is the same as the class name but is preceded by tilde (~).

→ A destructor takes no arguments & no return type.

→ It is also automatically called by a compiler when an object is destroyed.

eg

```
#include<iostream.h>
Class A
{
  Private:
   int i, j;
  Public:
   A()
   {
     Cin >> i >> j;
   }
   ~A()
   {
     Cout << "n object is destroyed :";
   }
};
void main()
{
    A obj1;
    {
      A obj2;
    }
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## Differences between Constructor & Destructor

→ Arguments can not be passed to the destructor.

→ Only one destructor can be declared for a given class, hence destructor can not be overloaded.

→ Destructor can be virtual, while constructor can not be virtual.

## #) Objects in function Arguments

```
#include <iostream.h>
 Class Distance
  {
    Private:
     int feet;
     float inches;
   Public:
     Distance( )
      { }
     Distance (int ft, float in)
      {
         feet = ft;
         inches = in;
      }
     void get ()
      {
       cout <<"\n Enter data:";
       cin>> feet >> inches;
     }
     void show ();
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
void add (Distance, Distance);
};
void Distance :: add (Distance d22, Distance d33)
{
  inches = d22.inches + d33.inches;
  feet = 0;
  if (inches >= 12.0)
  {
    inches = inches - 12.0;
    feet = feet + 1;
  }
  feet = feet + d22.feet + d33.feet;
}
void distance :: show ()
{
  cout << "\n feet = " << feet << "\t inches = " << inches;
}
void main()
{
  Distance dist1, dist3;
  Distance dist2 (", 6.25);
  dist1.get()
  dist3.add (dist1, dist2);
  dist3.show();
}
```

## Syntax for member function outside class

```
void Distance :: add (distance d22, Distance d33)
```

Return type

Name of class of which function is member

function name

function argument

## Returning objects from function

```
#include<iostream.h>
Class Distance
{
 Private:
  int feet;
  float inches;
 Public:
  Distance()
  {
   feet = 0;
   inches = 0.0;
  }
  Distance (int ft, float in)
  {
   feet = ft;
   inches = in;
  }
   void get()
  {
   cin>>feet>>inches;
  }
  void show()
  {
   cout<< feet<< inches;
   Distance add(distance)
  };
Distance Distance:: add(Distance d22)
{
 Distance temp;
 temp.inches = inches + d22.inches;
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
if (temp.inches >= 12.0)
{
 temp.inches = temp.inches - 12.0;
 temp.feet = 1;
}
temp.feet = temp.feet + feet + d22.feet;
return temp;
};
void main()
{
  Distance dist1, dist3;
  Distance dist2(11, 6.25);
  dist1.get();
  dist3 = dist1.add(dist2);
  dist1.show();
  dist2.show();
  dist3.show();
}
```

Q) Create a class called Time in which data members are hour, minute and second. Create three objects t1, t2 & t3. Read the values of t1 and t2. Obtain value t3 by adding t1 and t2. Perform this by passing object as argument and taking object as return.

P.T.O

Scanned by CamScanner

Downloaded from CSIT Tutor

```
#include<iostream.h>
class Time
{
 Private:
 int hour, min, sec;
 Public:
   Time ( )
   {
    hour = 0;
    min = 0;
    sec = 0;
   }
   Time(int h, int m, int s)
    {
      hour = h;
      min = m;
      sec = s;
   }
   void get ()
    {
      Cin >> hour >>min>>sec;
   }
   void Show( )
    {
      Cout << hour <<min <<sec;
   Time add (Time)
   };
   Time Time:: add (Time t22)
   {
    time = temp;
    temp. sec = sec + t22.sec
```

Q. 9.

Scanned by CamScanner

Downloaded from CSIT Tutor

```
if(temp.sec >= 60.0)
{
    temp.sec = temp.sec - 60.0;
    temp.min = 1;
}
temp.min = min + t22.min
if (temp.min > 59)
{
    temp.min = temp.min - 60.0;
    temp.hour = 1;
}
temp.hour = hour + t22.hour;
return hours;
};
void main()
{
    Time t1, t2, t3
    Time t2(1, 15, 20);
    t1.get();
    t3 = t1.add(t2);
    t1.show();
    t2.show();
    t3.show();
```

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Static Class members:→ There may be static data members and static member functions in a class.

#) Static data members:→ They are usually maintained to store values common to the entire class. Static data is only the copy of the data to maintain for the entire class, which is shared by all the objects of that class.

Two things are needed for making a data member static:-

i) Declaration within the class defination
ii) Defination outside the class

Example of static data

```
Class A
{
 Private:
  static int count;
  int x, y;
};
int A :: count;
void main()
{

  A obj1, obj2, obj3;
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## Static data

→ Static data is used by all the objects. It can be accessed by all objects. It doesn't belong to only one object. If you want to assign value to the Static data, then assign value else default value is zero.

→ In order to count members of the objects in a program, static data is used.

## #) Static Member function:-

↳ A member function that accesses only the static member of a class, it may be declared as static.

↳ This can be done by writing keyword 'static' before the function declaration in the class.

↳ A static member function can access only static members of the same class.

↳ A static member function is called by using the class name instead of its member.

```
Class name :: functionname ();
```

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Example: Program to keep count of created objects using static members.

```
#include<iostream.h>
class A
{
Private:
int no;
float price;
static int count;
Public:
void get(int i, float f)
{
no = i;
price = f;
count ++;
}
void display()
{
cout<<no<<price;
}
static void show()
{
cout<<count;
}
};
int A::count;
void main()
{
A obj1, obj2;
obj1.get(10, 50.0)
obj2.get(20,10.0)
A::show();
obj1.display();
obj2.display();
}
```

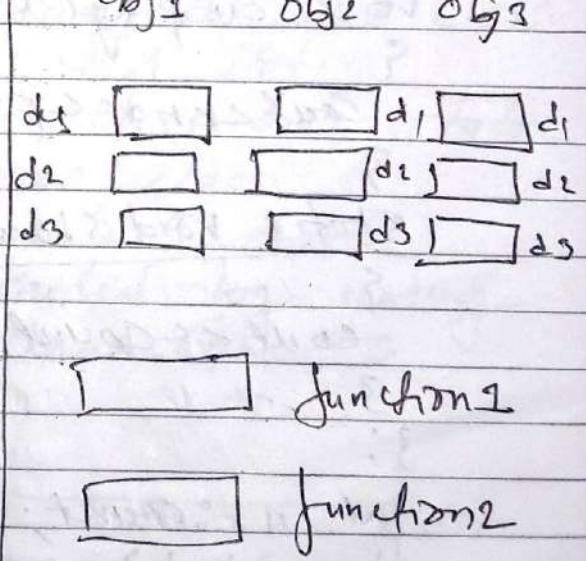
Scanned by CamScanner

Downloaded from CSIT Tutor

# Classes, Objects and Memory

→ All the objects in a given class have the same member functions. The member functions are created and placed in memory only one when they are defined in the class specifier. The data items however hold different values, so there must be a separate instance of each data item in each object. Therefore data is placed in memory when each object is defined.

```
Class A
{
 Private:
 int d1, d2, d3;
 Public:
  void function1 ()
   { }
  void function2 ()
   { }
};
void main()
{
   Obj1, obj2, Obj3;
}
```




Scanned by CamScanner
Downloaded from CSIT Tutor

#) WAP to create a class called Room in which data items are length, breadth and height. Create an array of objects of room. And find its area & volume.

```
#include<iostream.h>
Class Room;
{
  Private:
  float l, b, h, area, volume;
  public:
   void input( )
   {
     cin>>l>>b>>h;
   }
   void process( )
   {
     area= l*b;
     volume= l*b*h;
   }
   void display ( )
   {
     cout<<"\n Area:"<<area;
     cout<<"\n volume:"<<volume;
   }
};
void main( )
{
  int i;
  Room R[50];
for (i=0; i< 50; i++)
{
  R[i].input();
  R[i].process();
  R[i].display();
}
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

#) WAP using class & objects to simulate result preperation system for students. Data include name & marks in 3 subjects. The percentage & grade system is

| Percent | Grade |
|---|---|
| ≥90 | 'A' |
| ≥75<90 | 'B' |
| ≥60<75 | 'C' |
| <60 | 'F' |

```
# include <iostream.h>
# define MAX 20
Class Result
 {
  Private:
  float P_no, C_no, M_no, mark, per;
  Char name, roll;
  Public:
  void get ( )
   {
    cout <<"\n Enter your name";
    cin>> name
    cout <<"\n Enter your roll.no;
    cin>> roll;
    cout<<"\n Enter your marks in 3 subjects;
    cin>>P_no>> C_no>>M_no;
   }

  vid process ( )
   {
```

Scanned by CamScanner

Downloaded from CSIT Tutor

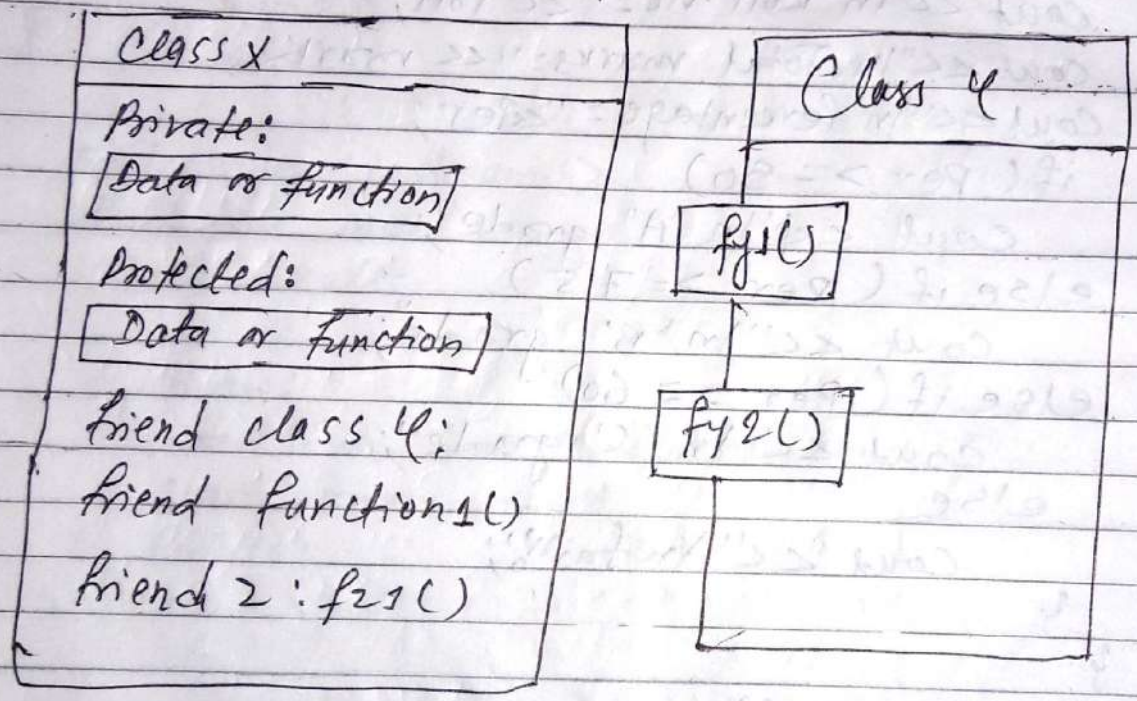
```
mark= Pno+ Cno+ Mno;
Per = (mark * 100)/300;
}
void display()
{
  cout <<"\n Name =" << name;
  cout <<"\n Roll no=" << roll;
  cout <<"\n Total marks=" << mark
  cout <<"\n Percentage = "<<Per;
  if( per >= 90)
     cout <<"\n 'A' grade";
  else if ( per >= 75)
     cout <<"\n 'B' grade";
  else if ( per >= 60)
     cout <<"\n 'C' grade";
  else
     cout <<"\n fail";
  }
};
  void main()
  {
    int i;
    Result R[MAX];
  for (i= 1; i<=20; i++)
    {
      R[i]. get();
      R[i]. process();
      R[i]. display();
    }
  }
```

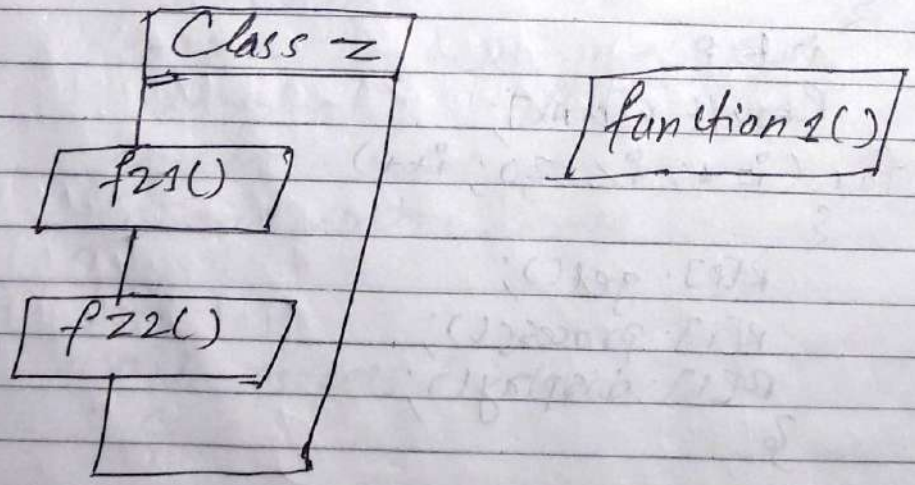
Scanned by CamScanner

Downloaded from CSIT Tutor

# #) Friend function

→ A friend function is a non-member function that is granted access to a class private and protected members and a friend class is a class whose member function can access another class private and protected members.




Scanned by CamScanner
Downloaded from CSIT Tutor

## #) Example of friend function

```
#include <iostream.h>
class ABC
{
  Private:
  float x, y;
  Public:
   void get()
   {
      cin >> x >> y;
   }
  friend float average(ABC);
  };
  float average(ABC obj)
   {
   return ((obj.x + obj.y)/2.0);
   }
  void main()
   {
    ABC obj1;
    obj1.get();
    float a;
    a = average(obj1);
    cout << a;
   }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## #) Things must be Known while using friend function:

i) A function may be declared friend of more than one class.
ii) It doesn't have the class scope.
iii) It doesn't require object for calling it.
iv) To use class private data, we have to use dot (.) operator.
v) Usually, it has object as argument

~~function~~
## friends as bridge

→ When a function requires to operate an object of different two classes, we declare the same function as a friend in both the classes, taking objects of two classes as arguments so that the function can operate on their private data.

eg: WAP to use a common friend function to exchange private values of the class.

```
#include <iostream.h>
class B;
class A;
{
 Private:
 int value_1;
```

Scanned by CamScanner
Downloaded from CSIT Tutor

```
Public:
void input (int a)
{
   value_1 = a;
}
void display()
 {
  cout << value_1;
 }
friend void exchange(A&, B&)
};
class B:
private:
 int value_2;
Public:
 void input(int a)
 {
   value_2 = a;
 }
 void display()
 {
   cout << value2;
 }
friend void exchange(A&, B&);
};
void exchange ( A&x, B&y)
{
   int temp = x.value_1;
   x.value1 = y.value_2;
   y.value_2 = temp;
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
void main ( )
{
  A obj1;
  B obj2;
  obj1.input(100);
  obj2.input(200);
  Cout<<"In Before:";
   obj1.display();
   obj2.display();
   exchange (obj1,obj2);
   obj1.display();
   obj2.display();
}
```

#) Write a program to sum private data of different two classes using friend.

```
#include <iostream.h>
Class B;
Class A
{
  Private:
  int value1
  Public:
    void input (int a)
    {
      value1= a;
    }
  void display()
    {
     Cout <<value1;
    }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
friend int sum( A&, B&);
};
class B
{
 Private:
  int value2;
 Public:
  void input(int a)
  {
   value2 = a;
  }
  void display ()
  {
   cout<<value2;
  }
  friend int sum( A&, B&);
};
int sum( A& x, B& y)
  {
   int sum;
   sum = x.value1 + y.value2;
  }
void main ()
  {
   A obj1;
   B obj2;
   obj1.input (5);
   obj2.input (7);
   sum ( obj1, obj2);
   Cout << "\n sum="
   Cout<<sum;
  }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## #) Object Oriented programing concept (feature):-

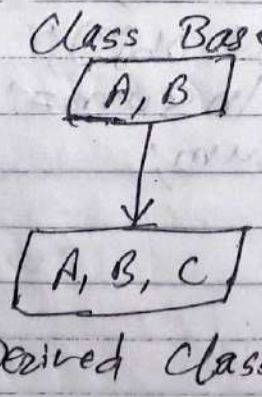
1) **Abstraction:→** It is the technique of creating new data types. To create the models of real life objects are known as user defined data types or abstract data types.

2) **Encapsulation:→** It is a techique by which the member function (methods) and the data (or attritube) on which it acts are associated into a single unit that keeps them safe from external modification.

In C++, it is supported by class. The instance of the class is known as object.

3) **Inheritance:→** The method of creating new classes called derived class from existing class called base-Class to create a hierarchy.

A derived class inherit some or all charecterstics of base class and may add some charecterstics of its own.




Scanned by CamScanner
Downloaded from CSIT Tutor

4) Polymorphism: → It is a mechanism by which a single function or operator can be used to operate on different ways depending upon the type of data passed to it.

function overloading and operator overloading are examples of the polymorphism.

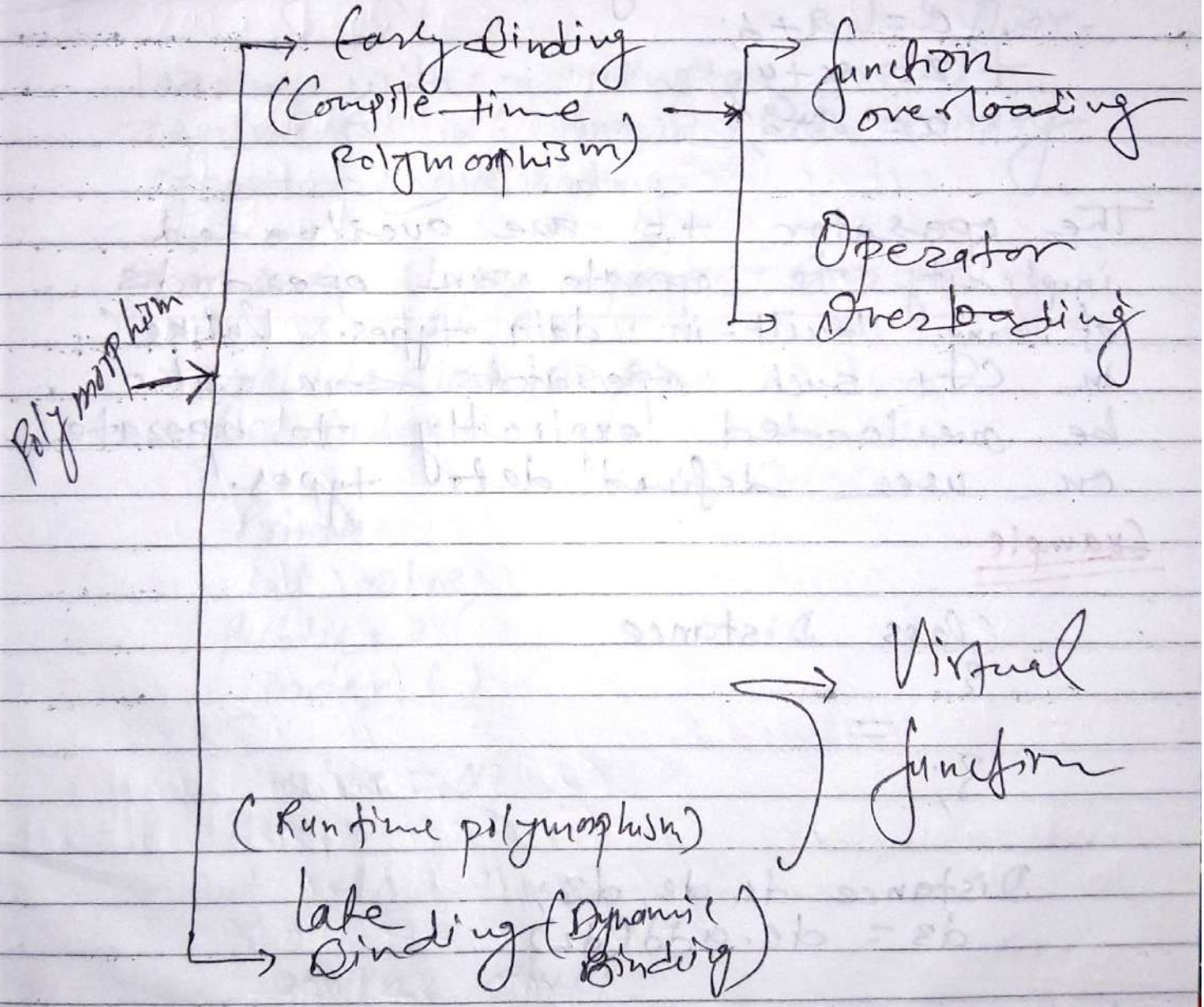



fig: Polymorphism Concept

Scanned by CamScanner
Downloaded from CSIT Tutor

# Operator Overloading

→ The operators such as +, −, +=, > etc are designed to operate only on the standard data types (built-in) to the structured language such as C.

```
int a = 1, b = 2, c;
float x = 1.1, y = 2.2, z;
c = a+b;
z = x+y;
x = a+b;
```

The operator +, = are overloaded implicity to operate on operands of any built-in data types. Unlike C, in C++ such operators can also be overloaded explicitly to operate on uses defined data types.

## Example

```
Class Distance
{
    =
};

Distance d1, d2, d3;
d3 = d1.add(d2);
```

In C++, by overloading + operator the above statement can be changed to an easily readable form.

$\boxed{d3 = d1 + d2}$

Scanned by CamScanner

Downloaded from CSIT Tutor

## Types of operator overloading

i) Unary operator Overloading :- Overloading without explicit arguments to an operator function is known as Unary operator overloading.

ii) Binary operator overloading :- Overloading with a single explicit arguments is known as binary operator overloading.

### Example of Unary operator Overloading

```
#include <iostream.h>
class index
{
  Private:
  int value;
 Public:
  index ()
  {
   value = 0;
  }
 int get()
  {
   return value;
  }
 Index operator ++ ()
 {
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
index temp;
 temp = value+1;
 temp.value = value;
 return temp;
}
};

void main()
{
   Index idx1, idx2;
   cout << idx1.get();
   cout << idx2.get();
   idx1 = idx2++;
   idx2++;
   cout << idx1.get();
   cout << idx2.get();
   int i = 5;
     cout << i;
       ++i;
     cout << i;
}
```

## Operator Keyword :

The keyword "operator" indicates that the operator symbol followed by it is to be overloaded.

**Syntax:**

```
Returntype operator operatorsymbol(arg1, arg2 ----)
{
    = Body of operator function
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

# #) Binary operator overloading

```
#include<iostream.h>
Class Complex
{
  float x, y;
  Public:
  complex ()
   { }
  Complex(float real, float imag)
   {
     x= real;
     y=imag;
   }
  complex operator +(complex);
   void display( );
   };
  Complex Complex:: operator + (complex c)
   {
    Complex temp;
    temp.x= x+ c.x
    temp.y = y+ c.y;
     return temp;
   }
  void complex:: display( )
   {
    cout<<"\nx="<<x;
    cout<<"\ny="<<y;

}
```

P.P.S

Scanned by CamScanner

Downloaded from CSIT Tutor

```
void main()
{
    Complex obj1(2.5, 3.5)
    complex obj2(1.6, 2.7);
    Complex obj3;
    obj3 = obj1 + obj2;
    obj3.display();
}
```

Deepak
3rd
CA+
CSIT
Anyal

Scanned by CamScanner
Downloaded from CSIT Tutor

#) Write a program to concatenate two strings by overloading '+' binary operator.

```
# include <iostream.h>
# include <string.h>
const int size = 30;
class String
{
Private:
char str[SIZE];
Public:
String ()
{
strcpy(str, '\0');
}
String (char mystring[SIZE])
{
strcpy (str, mystring);
}
void display()
{
cout << str;
}
String operator + (string s)
{
String temp;
strcpy(temp.str, str);
strcat(temp.str, s.str);
return temp;
}
};
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
void main()
{
    string obj1("we study");
    string obj1("in ASM");
    string obj3;
    obj3 = obj1 + obj2;
    obj1.display();
    obj2.display();
    obj3.display();
}
```

DPK

DPK

C++

Scanned by CamScanner

Downloaded from CSIT Tutor

# Overloading binary Comparision Operator

```
# include <iostream.h>
Class Distance
{
  Private:
    int feet;
    float inches;
  Public:
    Distance ()
    {
      feet=0;
      inches=0.0;
    }
    Distance( int ft, float in)
    {
      feet= ft;
      inches= in;
    }
    void get ()
    {
      cin>> feet> inches;
    }
    int operator < (Distance)
  };
int Distance:: operator < (Distance d2)
{
  float f1 = feet + inches/12.0;
  float f2 = d2.feet + d2.inches/12.0;

  return (f1 < f2)? 1 : 0;
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
void main()
{
  Distance dist1;
  dist1.get();
  Distance dist2;
  dist2.get();
  dist1.display();
  dist2.display();
 if(dist1 < dist2)
   {
     ~~cout<<"\n Dist 1 less than Dist2"~~
     cout<<"\nDist1 less than Dist2";
   }
 else
     cout<<"\nDist2 less than Dist1";
 }
```

## Prog to overload unary operator (—)

```
# include <iostream.h>
Class A
{
 Private:
     int i, j;
 Public:
     void input()
     {
       cin>>i>>j;
     }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

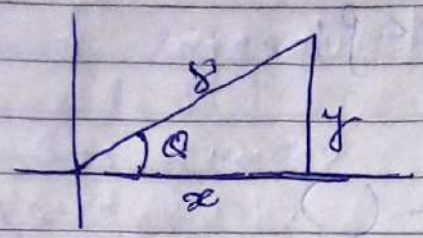
```
void display()
{
    cout << i << j;
}

void operator -()
{
    i = -i;
    j = -j;
}
};
void main()
{
    A obj;
    obj.input();
    obj.display();
    -obj;   //obj.operator-();
    obj.display();
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## #) Adding Polar co-ordinates




$$\sin\theta = \frac{y}{r}$$

$$\Rightarrow y = r\sin\theta \quad \text{--- (i)}$$

$$\cos\theta = \frac{x}{r}$$

$$\Rightarrow x = r\cos\theta \quad \text{--- (ii)}$$

dividing eqn (i) by (ii)

$$\tan\theta = \frac{y}{\theta}$$

$$\Rightarrow \theta = \operatorname{atan}(y/x)$$

squaring and adding (i) & (ii)

$$x^2 + y^2 = r^2$$

$$r = \sqrt{x^2+y^2}$$

## #) Write a program to overload (+) binary operator to add two polar coordinates.

```
# include <iostream.h>
# include <math.h>
class polar
{
  float radius, angle;
float getx()
{
  return radius * cos(angle);
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
float gety()
{
  return radius * sin(angle);
}
Public:
Polar()
{
  radius = angle = 0.0;
}
Polar(float r, angle a)
{
  radius = r;
  angle = a;
}
void display()
{
  cout << radius << "\t" << angle;
}
Polar operator + (Polar obj)
{
  Polar temp;
  float x = getx() + obj.getx();
  float y = gety() + obj.gety();
  temp.radius = sqrt(x*x + y*y);
  temp.angle = atan(y/x);
  return temp;
};
void main()
{
  Polar P1(10.0, 0.3)
  Polar P2(10.0, 1.57)
  Polar P3;
  P3 = P1 + P2;
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

# #) Comparision string

// overload '==' operator to compare strings.

```
# include <iostream.h>
# include <string.h>
class string
{
    Char str1[20];
    void input()
    {
        cin>> str1;
    }
    Public():
    string ()
    { }
    string (char a[20])
    {
        strcpy (str1, a);
    }
    int operator == (string obj)
    {
        int s;
        s = strcmp(str1, obj.str1)
        if (s != 0)
        {
            return 0;
        }
        else
        return 1;
        }
    };
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
void main()
{
    String s1.input();
    String s2.input();
if ( s1 == s2)
{
 cout << "\n Strings are Equal";
}
else
    {
      cout << "\n Strings are not Equal";
    }

}
```

// overloading assignment operator "+=".

```
#include <iostream.h>
Class Distance
{
 Private:
  int feet;
  float inches;
 Public:
  Distance()
  {
    feet = 0;
    inches = 0.0;
  }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
Distance (int ft, float in)
  {
    feet = ft;
    inches = in;
  }
void display()
  {
  cout << feet << "ft" << inches;
  }
void operator += (Distance d2)
  {
    feet += d2.feet;
    inches += d2.inches;
    if(inches >= 12.0)
     {
       inches -= 12.0;
       feet++;
     }
   }
     void get()
     {
      cin>>feet>>inches;
      }
     };
     void main()
      {
        Distance dist1;
        dist1.get();
        dist1.display();
       Distance dist2("",6.25);
        dist1+=dist2;
        dist1.display();
     }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## Difference between operator +=() and operator +():

→ In binary operator '+' a new object of type distance has to be created and returned by the function, and should be assigned to 3rd distance object.

```
dist3 = dist2 + dist2;
```

### Overload binary '+'

```
# include <iostream.h>
class Distance
{
    int feet;
    float inches;
Public:
    Distance()
    {
        feet = 0; inches = 0;
    }
    Distance (int ft, float in)
    {
        feet = ft;
        Inches = in;
    }
    Void display()
    {
        Cout << feet << ~~distance~~ inches;
    }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
Distance operator + ( Distance d2)
  {
    Distance temp;
  temp.feet = feet + d2.feet;
  temp. inches= inches + d2.inches;
    if (inches >= 12.0)
     {
   temp.inches = temp.inches = 12.0;
     temp.feet++;
     }
     return temp;
     }
     void get ()
      {
        cin>> feet>> inches;
      }
     };
   void main()
   {
     Distance dist1, dist2, dist3;
      dist1. get();
      dist2. get();
      dist1. display();
      dist2.display();
    dist3 = dist1 + dist2;
      dist3. display();

   }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## # Data Conversion

i) Conversion between objects and different classes
ii) Conversion between objects and built in data types.

→ Normally, when the value of one object is assigned to another of the same types, the values of all the member data items are simply copied into the new objects.

→ The compiler doesn't need any special instruction to use '=' for the assignment of user defined objects.

→ But, what happens when the variable on different sides of the '=' are of different types?

→ Compiler handles the conversion of basic types, which it does automatically.
→ The compiler doesn't handle things automatically when conversion between basic types and user defined types and conversion between different user defined types.

Note:→ Built in data types is converted to user defined data type by 1 argument constructor.

Scanned by CamScanner
Downloaded from CSIT Tutor

## Conversion between Objects & basic types:

→ When we want to convert between user defined data types and basic types or vice-versa, the compiler can not do it, we must write conversion function.

```
# include <iostream.h>
Const float MTF = 3.28;
Class Distance
{
  Private:
   int feet;
   float inches;
   Public:
   Distance()
   {
     feet = 0;
     inches = 0.0;
   }
   Distance (float meter)
   {
     float f = MTF * meter;
     feet = (int) f;
     inches = 12 * (f - feet);
   }
     Distance (int ft, float in)
     {
       feet = ft;
       inches = in;
     }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
void display()
{
cout << "feet:" << feet << "ft" "inches:" << inches;
}
operator float()
{
float fr = inches/12.0;
fr = fr + (float) feet;
return fr/MTF;
}
};
void main()
{
float m;
cin>>m;
Distance dist1(ft);
dist1.display();
Distance dist2(5,10.2);
m=(float)dist2;
cout<<"\n meter=" << meter;
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## #) Conversion built in data type to the User defined data type

```
# include <iostream.h>
Class A
{
  Private:
    int x, y, z;
  Public:
    A( int m)
    {
     x= m/2;
     y= m *2;
      z= m+10;
    }
    void display()
     {
       cout <<x<<y<<z;
     }
  };
   void main()
   {
     int mm;
     cin >> mm;
   A obj(mm);
   obj. display()
   }
```

Note:- To convert built in to User defined one argument constructor is defined.

Scanned by CamScanner

Downloaded from CSIT Tutor

# Write a program to convert built in data types to user-defined data types.

```
# include <iostream.h>
Class Exchange
{
Private:
float Yan, ic, dollar;
Public:
Exchange (float nc)
{
  Yan= nc/102;
  ic = nc*1.6;
  dollar = nc*95.0;
}
void display()
{
  cout<<yan<<ic<<dollar;
}
};
void main()
{
  float nepali;
  cin>> nepali;
  A obj (nepali);
  Obj. display()

}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## #) User defined to Built in

```
# include <iostream.h>
Class A
{
  int x, y, z;
  public:
  A()
  {
   cin>>x>>y>>z;
  }
operator int ()
  {
   int k= x+y+z;
   return k;
  }
  };

  void main()
  {
    A obj;
    int p;

    p = obj;

    Cout<<p;

  }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

# Conversion between objects of different Classes

```
Class A
{
  =
};
A objA;
```

```
Class B
{
  =
};
B objB;
```

$\boxed{\text{ObjA = objB;}}$ ⇒ objA = (A) objB;

destination Source

→ The conversion from one object to another type of object can be done in 2 ways;

i) Conversion routine (function) in source class.

ii) Routine (1 argument constructor) in destination class.

## Implicit Constructor

```
Class A
{
  Private:
  int i, j;
  Public:
    A( int m, int n)
    {
      i = m;
      j = n;
    }
};
```

```
void main()
{
  A obj(10, 20)
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

// Explicit Constructor

```
Class A
{
  Private:
   int i, j;
  Public:
    A()
    { }
   A(int m, int n)
    {
      i = m;
      j = n;
    }
    };

  void main()
    {
      A obj;
      obj = A(10, 20);
    }
```

#) Routine in Source class :→ In this method, we make routine function is the source class. The programme is,

Scanned by CamScanner

Downloaded from CSIT Tutor

// Conversion from polar to rectangle by using conversion function in polar (source).

```
#include <iostream.h>
#include <math.h>
class Rectangle
{
  float xco, yco;
public:
   Rectangle ()
   {
     xco = yco = 0.0;
   }
   Rectangle( float x, float y)
   {
     xco = x;
     yco = y;
   }
   void display ()
   {
   Cout << xco << yco
   }
};

Class polar
{
  Private:
   float radius, angle;
Pubic:
  Public( float r, float a)
{
   radius = r;
```

(P.T.O)

Scanned by CamScanner

Downloaded from CSIT Tutor

```
angle = a;
}
void display()
{
  cout << radius << angle;
}

Operator Rectangle()
{
  float x = radius * cos(angle);
  float y = radius * sin(angle);
  return rectangle(x, y)
}

};

void main()
{
  Rectangle robj;
  Polar pobj(10.0, 0.78)
  robj = pobj;

  Pobj.display();
  robj.display();

}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Conversion function in Destination class:

this method, we use conversion function in the destination class.

// Convert polar to rectangle using routine in rectangle class (destination)

```
#include <iostream.h>
#include <math.h>
class Polar
{
  Private:
   float radius, angle;
  Public:
  Polar()
  {
   radius = angle = 0.0;
  }
  Polar( float r, float a)
  {
   radius = r;
   angle = a;
  }
 void display()
  {
   cout << radius << angle;
  }
  float getR()
  {
   return radius;
  }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
float getA()
  {
    return angle
  }
};

Class Rectangle ()
  {
    Private:
      float xco, yco;
    Public:
      Rectangle ()
      {
        xco = yco = 0.0;
      }
      Rectangle (float x, float y)
      {
        xco = x;
        yco = y;
      }
      Rectangle (Polar P)
        float r = p.getR();
        float a = p.getA();
        xco = r * cos(a);
        y = r * sin(a);
      }
        void display()
        {
          Cout << xco << yco;
        }
      };
      void main()
      {
```

```
Rectangle robj;
Polar Pobj (10.0, 0.78)
robj = Rectangle (Pobj);
robj.display();
Pobj.display();
}
```

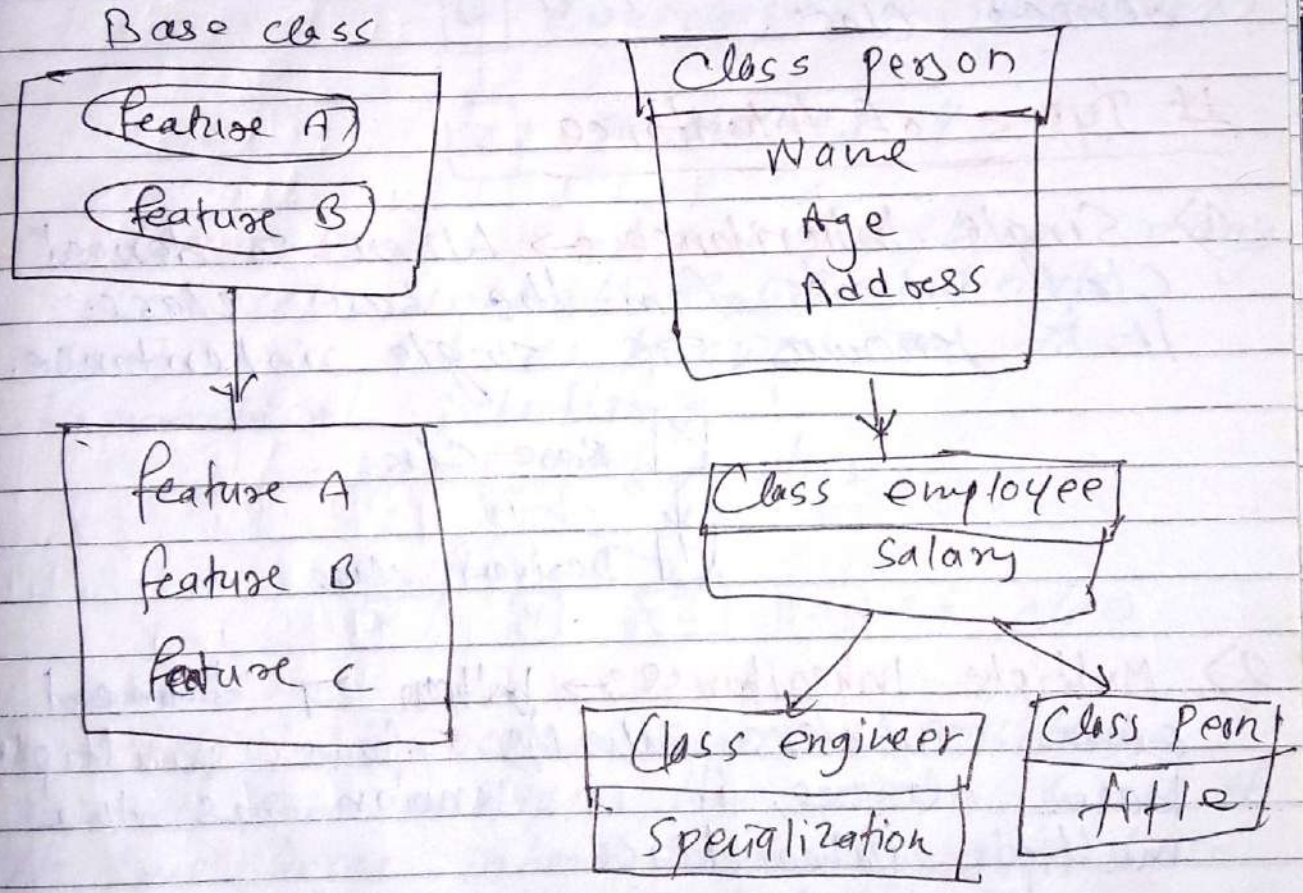
Scanned by CamScanner

Downloaded from CSIT Tutor

# Inheritance :-> Inheritance is the process of creating new classes called derived class from existing class called base class.

→ The derived class can inherit all the capabilities of the base class & can add its own feature in it.

eg:




Scanned by CamScanner
Downloaded from CSIT Tutor

# Advantages of Inheritance

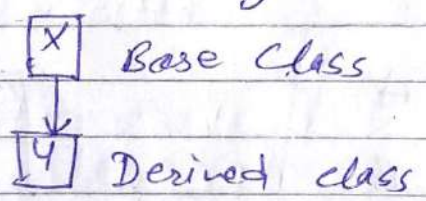
→ It enables code reusability. Once a base class is written and debugged, it can be used in various situations without having to redefine it.

→ Reusing existing code saves time, money and efforts.

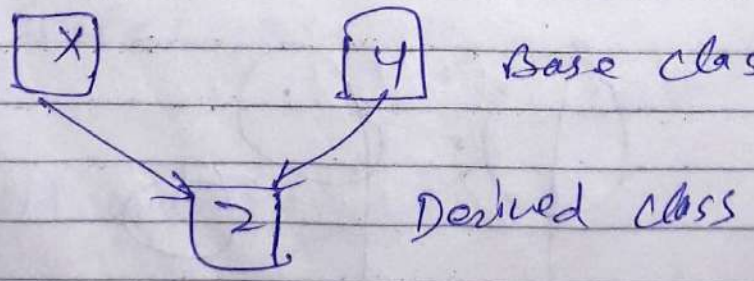
→ Without redefining the old class, we can add new properties to the derived class.

# Types of Inheritance

1) Single Inheritance :→ When a derived class inherits from the base class. It is known as single inheritance.



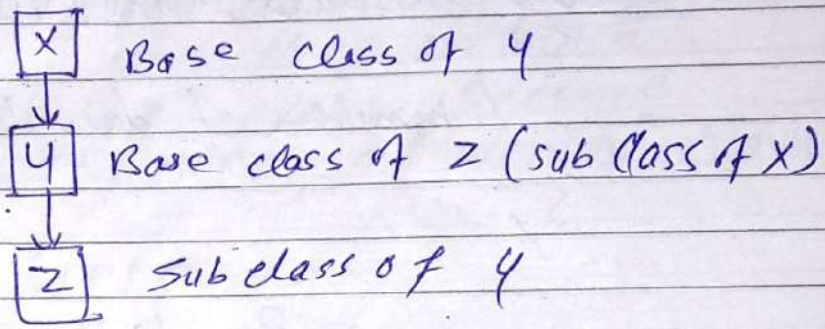

2) Multiple Inheritance :→ When a ~~derived class~~ subclass inherits from multiple base classes, it is known as the multiple inheritance.



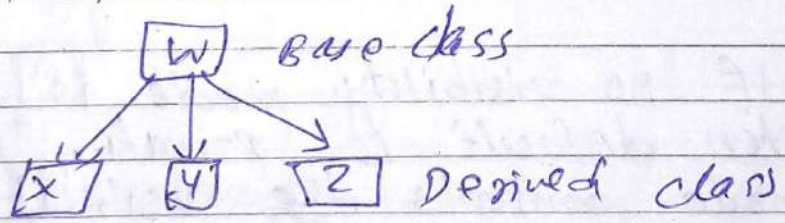

Scanned by CamScanner
Downloaded from CSIT Tutor

types of inheritance

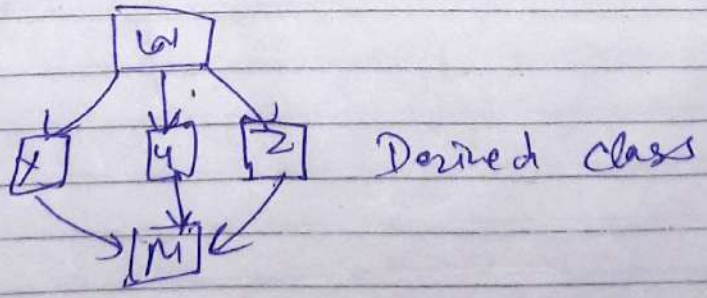
3) Multilevel Inheritance :→ The transitive nature of inheritance is reflected by this form of inheritance. When a subclass (derived class) inherits from a class that itself from another class, it is known as the multilevel inheritance.




4) Hierarchical Inheritance :→ When many subclasses inherit from a single base class, it is known as the hierarchical inheritance.




5) Hybrid Inheritance :→ When a subclass inherits from multiple base classes & all of its base classes inherit from a single base class, this is hybrid inheritance.




Scanned by CamScanner
Downloaded from CSIT Tutor

#) Derived and Base Class:→ A derived class (sub class) has to identify the class from which it is derived. i)

Syntax

i) Single inheritance

```
Class DeriveClassName : VisibilityMode BaseClassName
{
    //members of derived class
};
```

Visibility → Private
→ Public
→ Protected

→ If no visibility mode is specified, then default is private. The visibility mode controls the visibility and availability of inherited base class members in the derived class.

Scanned by CamScanner

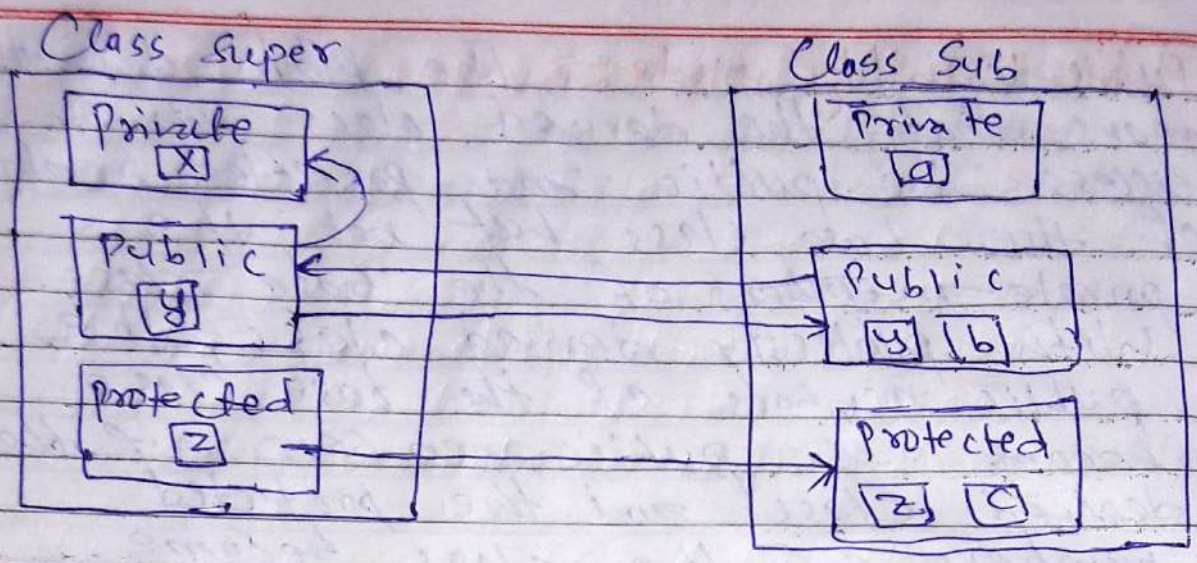
Downloaded from CSIT Tutor

i) Public Visibility Mode :→ The public derivation means that the derived class can access the public and protected members of the base class but not the private members of the base class. With publicity derived class, the public members of the base class become the public members of the derived class and the protected members of the class become the protected members of derived class.

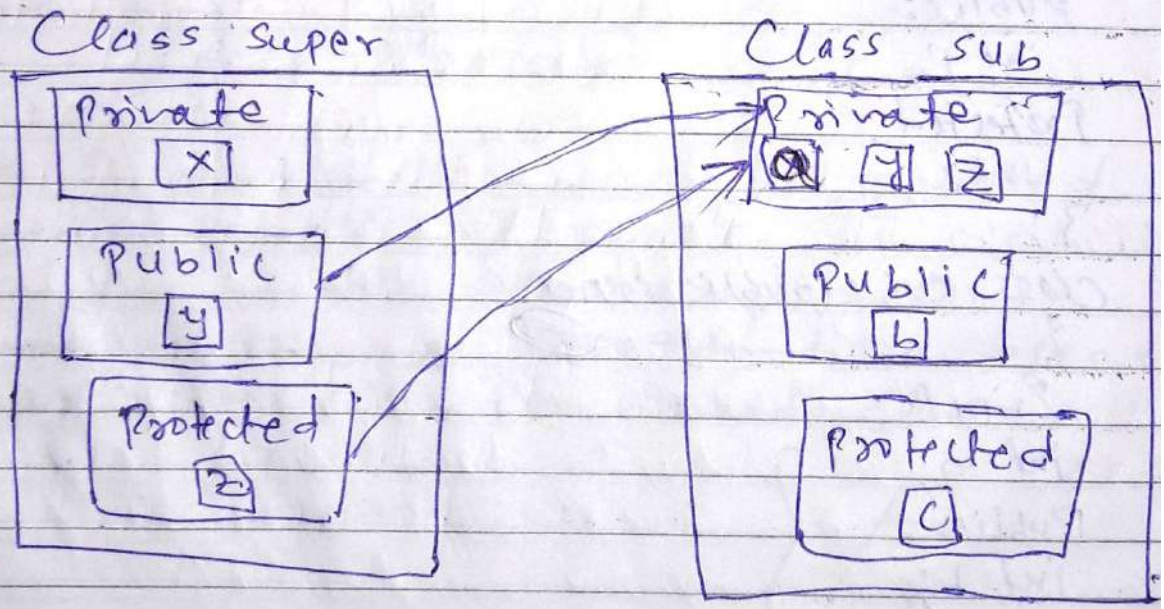
```
Class super
{
Private:
 int x;
Public:
 int y;
Protected:
 int z;
};
class sub : public super
{
 Private:
 int a;
 Public:
 int b;
 protected:
 int c;
};
```

Scanned by CamScanner

Downloaded from CSIT Tutor

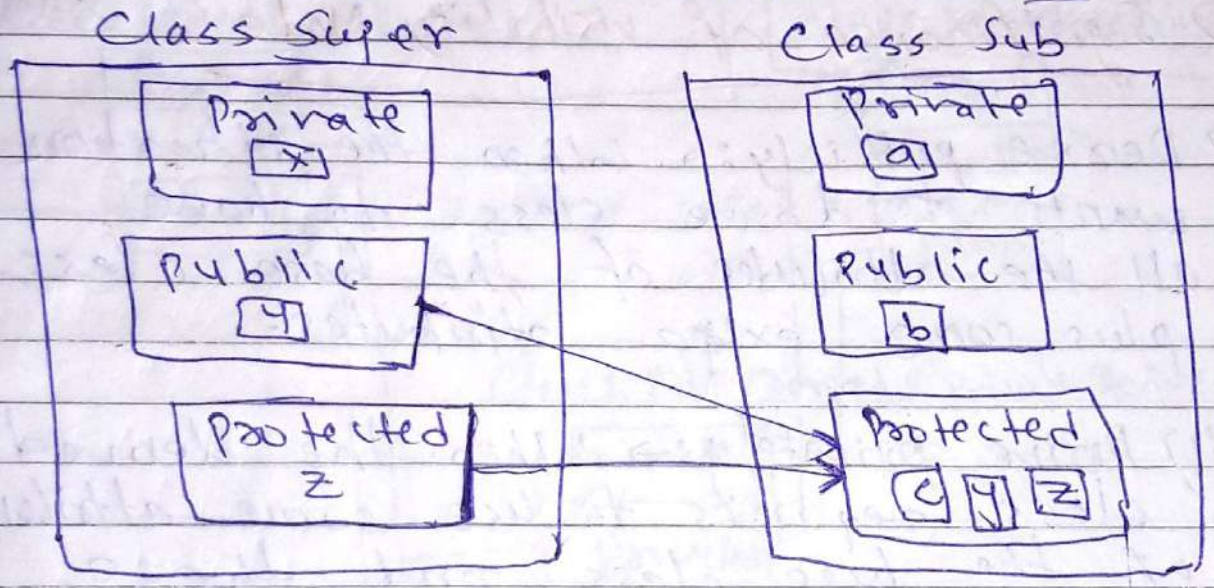



ii) Private Visibility mode :→ With privately derived class, the public & protected members of the base class as the inherited members become private in the derived class, they can not be inherited further.




Scanned by CamScanner
Downloaded from CSIT Tutor

iii) Protected visibility Mode :→ With protected derived class, the public and protected members of the base class become protected members of the derived class, i.e., inherited members are now not available to the outside & can be accessed only through the member function of the derived class. These members can be inherited further.




#) Inheritance & Base Class:- while defining a base class, following things should be considered:-

i) Members intended to be inherited and at the same time intended to be available to every functions, even non members should be declared as public in the base class.

ii) Members intended to be inherited but not intended to be public should be declared as protected.

iii) Members not intended to be inherited should be declared as private.

Scanned by CamScanner
Downloaded from CSIT Tutor

## #) Difference Between Private & Protected:

→ The private and protected members of a class can only be accessed by the member functions and friend functions of the same class. However, protected members can be inherited, whereas private members cannot be inherited.

## #) Significance of visibility Mode

i) Derive publicly:→ When the situation wants to derive class to have all the attributes of the base class, plus some extra attributes.

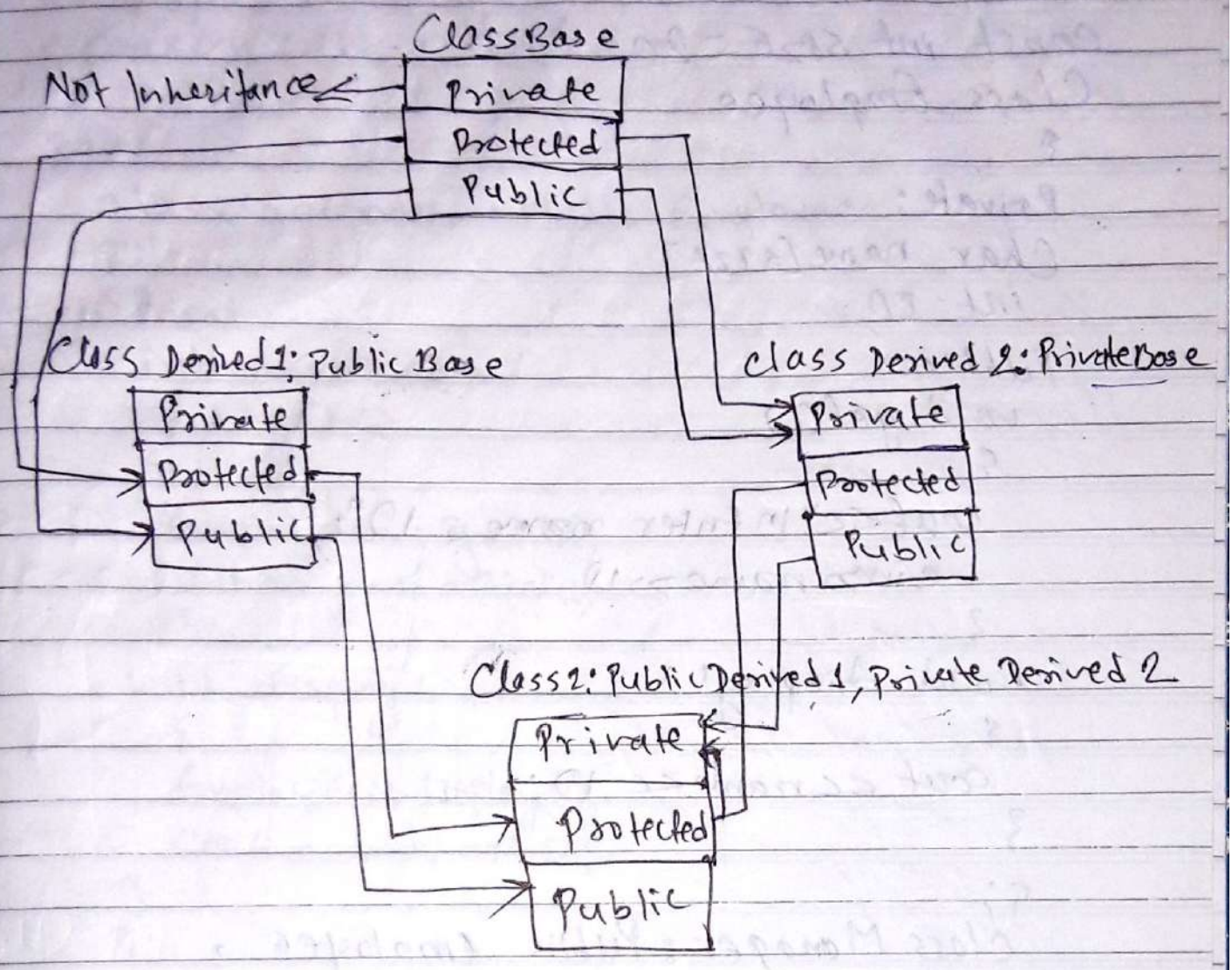
ii) Derive privately:→ When the derived class requires to use some attributes of the base class and these inherited features are intended not to be inherited further.

iii) Derive protectly:→ When the features are required to be hidden from the outside & at the same time, required to be inherited.

Scanned by CamScanner

Downloaded from CSIT Tutor

## #) Relation of Inheritance, visibility mode & type of Inheritance:




→ Thus, if we want that the public and protected members should remain inheritable even beyond the derived class, derived them publicly.

→ Otherwise if you want to end the inheritance for public and protected members, derived them privately.

Scanned by CamScanner
Downloaded from CSIT Tutor

## #) Class Hierarchy

```
#include <iostream.h>
const int SIZE=20;
Class Employee
 {
 Private:
 Char name[SIZE]
  int ID;
 Public:
  void get()
   {
    cout<<"In Enter name & ID";
     cin>>name>>ID;
   }
  void display()
   {
    cout <<name<< ID;
  }
 };
 Class Manager : Public employee .
 {
 Private:
  char title[SIZE];
  float bonus;
 Pubic:
 void get()
  {
   Employee:: get();
   cin>> title>> bonus;
  }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
void display()
{
 Employee:: display();
 cout << title << \t bonus;
}
};
 class Scientist: public Employee
 {
  Private:
  int noofarticle;
  void get()
  {
   Employee:: get();
   cin>> noofarticle;
   }
  void display()
   {
   Employee:: display();
   cout << noofarticle;
  }
  };
  Class labour: public Employee
  { };
void main()
{
 manager m;
 Scientist s;
labour L;
 m.get();
s.get();
L.get();
m.display();
s.display();
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

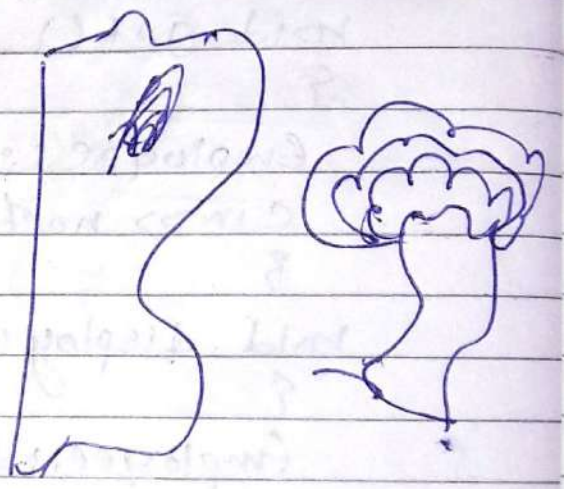
## #) Passing argument to a base class constructor through derived class constructor

Syntax:

```
Derived :: Derived (Datatype x, Datatype y---) : Base(x, y)
{
  =
  =
};
```

Passing arguments from derived class constructor to base class constructor.

```
//example:
    Class Base
    {
      int a;
      float b;
  Public:
    Base (int i, float j)
    {
      a = i;
      b = j;
    }};
class Derived : Public Base
  {
    int m;
  Public:
  Derived (int k, float m) : Base (k, m)
    {
      m = k;
    }
  };
    void main()
    { Derived obj (5, 3.5);
    }
```



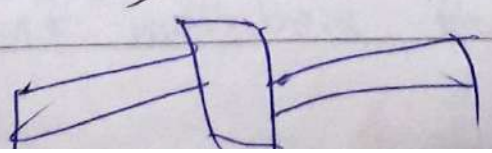
Scanned by CamScanner

Downloaded from CSIT Tutor

#) function Overriding :→ When the same function exists in both the base class & the derived class, the function in the derived class will be executed, not that of base class ~~known~~ function is called as function overriding.

#) Abstract base class (Abstract Class) :→ Class used for only for derived other class is called abstract base class.

```
#include<iostream.h>
Class A
{
 Public:
 void function()
 {
  cout<<"A";
 }};
 class B: public A
 {
  Public:
   void function()
   {
    cout<<"B";
   }
  };
  void main()
  {
   B obj;
   obj.function();
  }
```



Scanned by CamScanner

Downloaded from CSIT Tutor

# Ambiguity in Multiple Inheritance :-> Two base classes having same function name, while a class derived from the classes, but has no function with this name, then there is ambiguity.

```
# include <iostream.h>
Class A
{
  Public:
  void show()
  {
    cout <<"A";
  }
};
class B
{
  Public:
  void show()
  {
    cout <<"B";
  }
}
Class C: Pubic A, Public B
{ };
void main()
{
  C obj // obj.show(); Ambiguity error

  Obj.A::show();
  Obj.B::show();

}
```

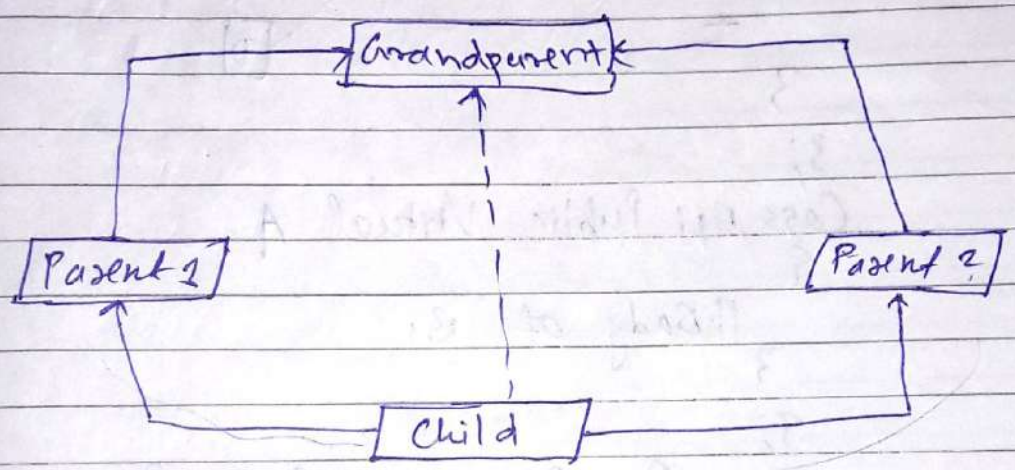
Scanned by CamScanner

Downloaded from CSIT Tutor

## H) Multipath inheritance & virtual Base classes

→ The form of inheritance which derives a new class by multiple inheritance of base classes, which are derived earlier from the same base class is known as the Multiple inheritance.

It involves more than one form of inheritance namely multilevel, multiple & hierarchial as shown in fig.




→ The child class is derived from the base classes Parent 1 and parent 2, which themselves have a common base class grandparent.

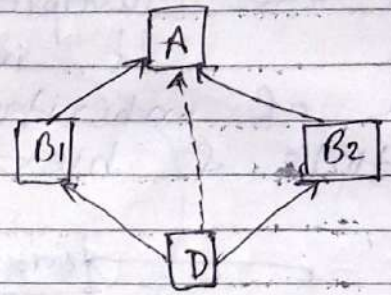
→ The child inherits the properties of the Grandparent via two separate paths as shown. This multipath inheritance introduce compilation problem. The public and protected members of the Grandparent are inherited into the child class twice via Parent1 and Parent2 classes. Therefore, the child would have duplicate sets of members from Grandparent.

Scanned by CamScanner
Downloaded from CSIT Tutor

→ C++ supports important concept called virtual base classes to handle ambiguity cause due to multiple inheritance. It is achived by making the common base class as a virtual base class.

eg;

```
Class A
{
Public:
void function()
  {
    =
  }
};
Class B1: Public Virtual A
{
  //Body of B1
}
};
Class B2: Public Virtual A
{ //Body of B2
}};
Class D: Public B1, Public B2
{
  //Body of Class D
}};
```




→ C++ takes necessary case to see that only one copy of the class is inherited when a class is inherited as virtual irrespect of the number of path that exists.

Scanned by CamScanner
Downloaded from CSIT Tutor

## #) Constructor and Destructor in Multi-level inheritance

```
# include <iostream.h>
Class B
{
  int x;
 Public:
  B()
  {
   Cout <<"Base";
  }
 ~B()
  {
   Cout<<"\n Destroy Base";
  }
};
 Class D1 : Public B
 {
   int y;
 public:
 D1()
 {
   Cout << "Derived1";
 }
~D1()
 {
   cout <<"\n Destroy Derived1";
 }
};
```

P.T.O

Scanned by CamScanner

Downloaded from CSIT Tutor

```
Class D2 : Public D1
{
  int z;
  Public:
   D2()
   {
    cout << " Derived 2";
   }
   ~D2()
   {
    cout << "\n Destroy Derived 2";
   }
};

  void main()
  {
    D2 obj;
  }
```

PTO

Scanned by CamScanner

Downloaded from CSIT Tutor

#) "this" pointer:- The member functions of every objects have pointer named this which points to the object itself. Thus, any member function can find out the address of the object of which it is a member.

```
#include <iostream.h>
class where
{
Private:
char a[10];
Public:
void function()
{
cout<<"In object address="<<this;
}

};
void main()
{
where obj1, obj2, obj3;
obj1. function();
obj2. function();
obj3. function();

}
```

Scanned by CamScanner

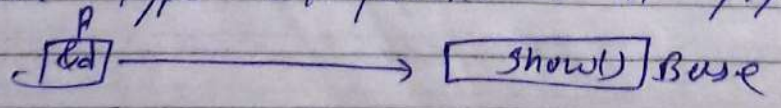
Downloaded from CSIT Tutor

## #) Virtual functions: Virtual means existing in effect but not in reality.

// normal member function accessed with pointer.

```
#include <iostream.h>
Class Base
{
  Public:
  void show()
  {
    cout <<"Base:";
  }
};
Class Derive : Public Base
{
  Public:
  void show()
  {
    cout<<"Derived";
  }
};
void main()
{
  Derive d;
  Base *p;
  p = &d;
  p -> show();
}
```

→ The compiler ignores the contents of the pointer 'P' and choose the member function that matches the type of pointer as in fig.




Scanned by CamScanner
Downloaded from CSIT Tutor

But In

// virtual member function accessed with pointer

```
#include <iostream.h>
Class Base
{
Public:
Virtual void show()
{
cout <<"Base";
}
};
Class Derive: Public Base
{
Public:
void show()
{
cout <<"Derived";
}
};
void main()
{
Derive d;
Base *p;
p= &d;
p->show();
}
```

→ This time, the rule is that the compiler selects the function based on the content of the pointer p, and not the type of pointer.

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Pure virtual function:- It is a virtual function with no function defination body. Since, base class is not executed, body of the virtual function is the base class is removed and equal to zero is added is called pure virtual function.

```
#include <iostream.h>
class Base
{
Public:
Virtual void show()=0;
};
class Derive: Public Base
{
  Pubic:
    void show()
      {
      cout << "Derived";
      }
  };
  void main()
    Derive d;
    Base *p;
      p = &d
      p -> show();
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

# Pointer and object

```
#include <iostream.h>
Class Employee
{
  Private:
  Char name[20];
  int age;
  float salary;
  Public:
    void read()
    {
    cout <<"\n Enter values:";
    cin>> name>> age>> salary;
    }
    void display()
    {
    cout <<"\n Name = " << name;
    cout <<"\n Age = " <<age;
    cout <<"\n Salary= " << salary;
    }
  };
  void main()
  {
    Employee e;
    e.read();
    e.display();
    Employee *p;
    p = &e;
    p → read();
    p → display();
    }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## #) Example of Virtual function

```
#include<iostream.h>
class Graphics
{
 Public:
 virtual void draw()
  {
   cout<<"point";
  }
 };
Class line:public Graphics
{
 Public:
  void draw()
   {
    cout<<"In line";
   }
  };
 Class traingle:Public Graphics
  {
   Public:
   void draw()
    {
     cout<<"In Traingle";
    }
   };
   class rectangle: Public Graphics
  {
   Public:
    void draw()
     {
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
cout << "\n Rectangle";
}
};

void main()
{
    Graphics p;
    line l;
    traingle t;
    rectangle r;

    Graphics *b[] = { &p, &l, &t, &r}

    cout << "\n Draw figure";

    for (int i = 0; i < 4; i++)
    {
        b[i]->draw();
    }
}
```

Deepak
Aryal

Scanned by CamScanner

Downloaded from CSIT Tutor

#) "new" operator:- C++ program provides 'new' operator to obtain block of memory at run time. The 'new' operator obtains memory from the operating system and returns a pointer to its starting point.

```
#include <iostream.h>
#include <string.h>
void main()
{
  Char *p= "we study in college";
  int length= strlen(p);
  Char *ptr;
  ptr= new Char [Length+1];
  strcpy (ptr, p);
  Cout <<"ptr="<<ptr;
  delete ptr;
}
```

Syntax of new operator

```
int *p
```

Data type must agree

```
p = new int [length];
```

Pointer → p
Keyword → new
Size → length

Scanned by CamScanner

Downloaded from CSIT Tutor

// Program using 'new' operator.

```
# include <iostream.h>
Class Distance
{
  Private:
    int feet;
    float inches;
    Public:
     void get ()
     {
      cout << "\n Enter value:";
        cin >> feet >> inches;
     }
     void display ()
     {
       Cout << feet << inches;
     }
     };
     void main()
      {
        Distance d;
        d. get ();
        d. display ();
        Distance * p;
        p = new distance;
        P → get ();
        p → display ();
      }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

// Array of pointers to objects

```
#include <iostream.h>
Class Person
{
Private:
Char name[10];
Public:
void get()
{
cout <<"\n Enter name:";
cin>>name;
}
void display()
{
cout <<"\n Name:"<<name;
}
};
void main()
{
Person *P[100];
int n=0;
Char choice;
do{
P[n]= new Person;
P[n]-> get();
n++;
cout <<"\n another(y/n)?";
cin >> choice
} while (choice==='y');
for (int i=0; i<n; i++)
P[i]-> display();
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

Containership:- When an object is created inside another class, then it is called as Containership.

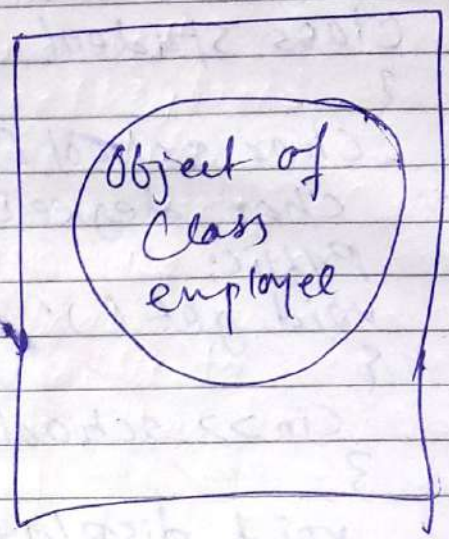
```
#include<iostream.h>
Class student
{
 char school[15];
 char degree[25];
 Public:
 void get()
 {
  cin>> school >> degree;
 }
 void display()
  {
   cout <<school <<"\t"<<degree;
  }
 };
  Class Employee
  {
   char name[15];
   int member;
  Public:
  void get()
   {
    cin>>name>>member;
   }
  void display()
   {
    cout <<name <<"\t"<<member;
   }
  };
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
Class Manager
{
  Char title[15];
  Employee e; //containership
  Student s;//containership
Public:
  void get()
  {
     e.get();
     s.get();
     cin>>title;
  }
  void display()
  {
   e.display();
   s.display();
   cout << title;
  }
};
class Scientist
{
  int noofarticle;
Employee e; //containership
 Student s;
Public:
 void get()
  {
    e.get();
    s.get();
    cin>>noof article;
  }
```

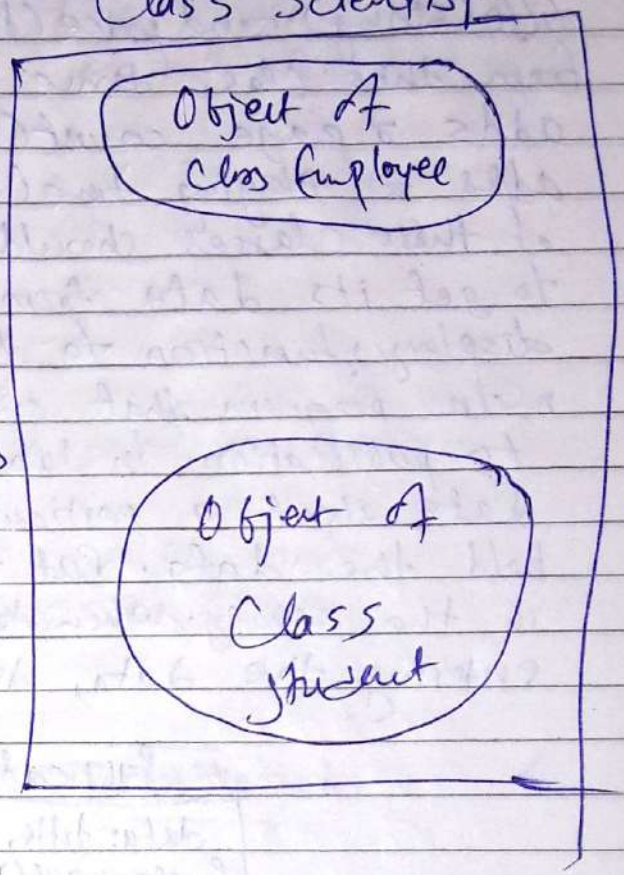
Class labour



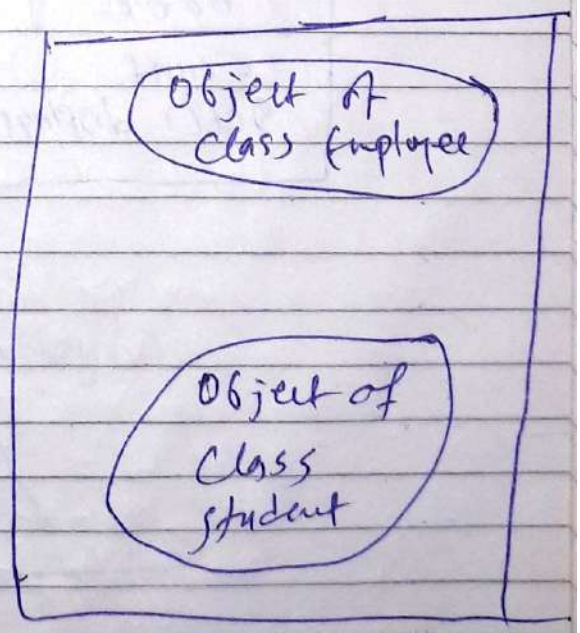

Scanned by CamScanner
Downloaded from CSIT Tutor

```
void display()
 {
  e.display();
  s.display();
  cout << not of article;
  }
 };
 /class labour
  {
   Employee e;//containership
  Public:
  void get()
   {
    e.get();
    }
    void display()
     {
      e.display();
     }
   };
  void main()
   {
   manager mm;
   Scientist ss;
   labour ll;
    mm.get();
    mm.display();
    ss.get();
    ss.display();
    ll.get();
    ll.display();
   }
```

Class Scientist

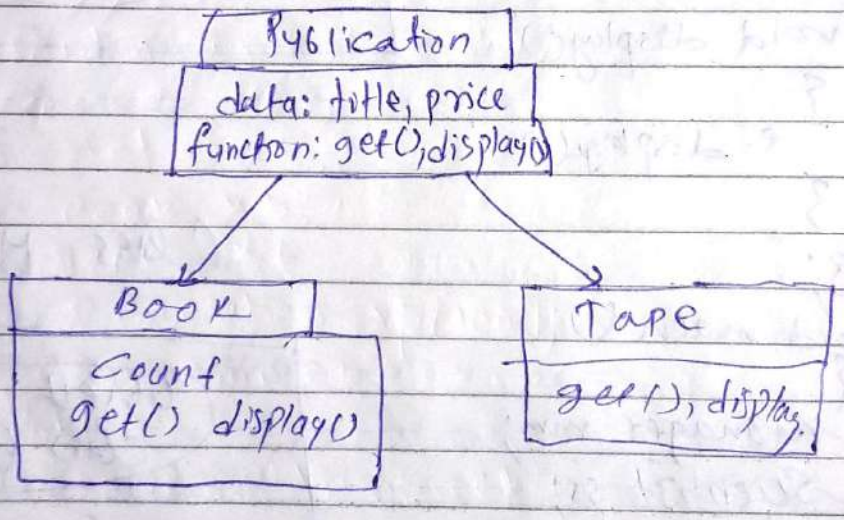



Class Manager




Scanned by CamScanner
Downloaded from CSIT Tutor

#) Create a class called production that stores the title(string) and price(float) of a production. from this class derive two classes BOOK which adds a page count(int) and Tape which adds a playing time(float) in minutes. Each of these classes should have get() function to get its data from user keyboard and display() function to display data. write a main program that creates an array of pointer to publication. In loop, ask the user for data about a particular book or tape to hold the data. Put the pointer to the object in the array. When the user has finished entering the data, display the resulting data.




P.T.O

Scanned by CamScanner
Downloaded from CSIT Tutor

```
⇒ #include<iostream.h>
Class Publication
{
Char title[20]
 float price;
 Public:
 Virtual void get()
   {
    cin>>title>>price;
   }
 void display()
   {
     cout<<title<<price;
   }
 };
 Class Book: Public Publication
 {
   int count;
 Public:
  void get()
  {
    Publication::get();
      cin>>count;
  }
  void display()
  {
    Publication::display();
      cout<<count;
  }
 };
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
Class Tape: Public Publication
{
  float time;
  Public:
  void get()
    {
        Publication :: get()
          cin>> time;
    }
  void display()
  {
    Publication:: display();
     cout << time;
    }
  };
void main()
{
   int n=0;
   Book *b;
   Tape *t
   Publication *p[100];
  Char choice;
  do {
   cout<<"\n object for book or tape";

    cin>> choice;
    if (choice == 'b')
    {
       b = new Book;
       b -> get()
       p[n] = b;
       }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
else
{
    t = new Tape;
      t -> get()
    P[n] = t;
}
n++;
cout << "In Another(Y/n)?";
  cin >> choice;
} while(choice == 'y');

for(int i=0; i<n; i++)
{
    P[i] -> display();
}
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Define a base class shape having data member radius (int). Derive new class called circle and sphere from this class. Write methods to compute the area of circle and sphere.

Area of circle = $\pi r^2$, sphere = $4\pi r^2$

```
#include<iostream.h>
#define PI 3.14
Class Shape
{
  int radius;
  Public:
   void get() {
   cout <<"\n Enter radius:";
    cin radius; }
  };
Class Circle: Public shape {
  Public:
   void display() {
    cout<<PI * radius * radius; }
   };
Class sphere: Public shape {
  Pubic:
   void display() {
    cout << 4 * PI * radius * radius; }};
  void main() {
   Circle c;
   c.get();
   sphere s;
   s.get();
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

**Template :-** C++ supports a ~~various~~ mechanism known as template to implement the concept of generic programing.

→ Template allows us to generate a family of classes or a family of functions to handle different data types.

→ Template classes and functions ~~code~~ eliminate code duplication for different types and ~~thus~~ make the program-development easier and more meaningful.

**Defining function templates :-** With C++ function template, you can write a single general function defination that can be used with a variety of the data types.

→ One advantage of using templates, rather than overloaded function, you simply include a single template defination and the compiler will automatically generate and store any those functions actually called.

**Syntax of function template :-** A function template is prefixed with the keyword template and a list of template type arguments.

→ These template type arguments are called generic data types, not known in declaration, known only at the point of a call to a function template.

Scanned by CamScanner

Downloaded from CSIT Tutor

## Syntax (template function):-

keyword → template <class T, ----> ← Template data types

```
template <class T, ----->
Returntype functionName(arguments)
{
    // Body of template function
}
```

→ A call to a template function is similar to that of a normal function and parameters can be of any data type.

→ When the compiler encounters a call to such functions, it identifies the data type of the parameter and create a function ~~inte~~ internally & makes a call to it.

→ Use of template, avoids the overhead of rewriting functions having body of the same pattern, but operating on different data types.

```
#include <iostream.h>
#include <string.h>
Template <class T>
void swap(T &x, T&y) {
  T temp;
  temp = x;
  x = y;
  y = temp;
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

int = (Final)

```
void swap(char *s1, char *s2)
void main()
{
  int i, j;
  float x, y;
  char ch1, ch2;
  char s1[20], s2[20];
   cin>>i>>j;
  cout<<"In Before:"<<i<<j;
    swap(i, j);
  cout<<"In After:"<<i<<j;
    cin>> x>>y;
    swap(x, y)
     cout<<x<<y;
    cin>>ch1>>ch2;
     swap (ch1, ch2);
    cout<<ch1 <<ch2;
   cout<<"In Enter two strings";
     cin>>s1>>s2;
  swap(s1, s2);
    cout<<s1<<s2;
  }
   void swap (char *p1, char *p2)
   { int length;
length=(strlen(s1)>= strlen(s2))? strlen(p1) :strlen(p2);
char *temp= new char [length+1];
     strcpy (temp, p1)
     strcpy (p1, p2)
      strcpy (p2, temp);
      delete temp;
    }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

# #) Exception Handling :-

An exception is an interruption in the normal flow of program in response to an unexpected or abnormal event.

→ Exception refer to unsual condition in a program. Exceptions are of two kinds namely, synchronous exception and the asynchronous exception.

→ Error such as "out-of-range index" and "overflow" belong to the synchronous type exception.

→ Errors that are caused by events beyond the control of the program are called asynchronous exception.

→ The purpose of the exception handling mechanism is to provide means to detect and report an "exceptional circumstance" so that appropriate action ~~ate~~ can be taken. Error handling code that perform the following tasks.

i) Fixed the problem (Hit the exception)

ii) Inform that an error has occured (throw the exception)

iii) Receive the error information (catch the exception)

iv) Take corrective action (Handle the exception)

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Example of Exception handling; divided by zero

```
#include <iostream.h>
void divide(int x, int y)
{
  if (y != 0)
  {
   cout<<"Divide/ Divisor="<<x/y;
  }
  else
     {
       throw(y);
     }
  }
void main()
  {
   try
     {
       divide (10,5);
       divide (10,0);
     }
   catch(int n)
     {
       cout <<"\n Cannot divide by 0";
     }
  }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## // Program that demonstrate Handling:-

```
#include <iostream.h>
struct unknown { };
void main( )
{
  bool quit = false;
  do
   {
  try
     {
       int pick;
  Cout << "\n Pick an exception to throw:\n";
  cout << "1. Integer\n";
  cout << "2. Float\n";
  Cout << "3. Charecter string\n";
   Cin >> Pick;
   Switch (Pick)
   {
    Case 1:
     throw pick;
      break;
    Case 2:
       throw float (Pick);
       break;
     Case 3:
        throw "I am picked";
          break;
     default:
        throw unknown();
     }
  }
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
Catch (int i)
{
 cerr << "In Integer exception:" << i << endl;
}
catch (float f)
{
 Cout << "In float exception:" << f << endl;
}
catch (char *str)
{
 cerr << "In Charecter string exception:" << str << endl;
}

 Catch(...)
 {
  Cerr << "Handling an unknown exeption";
    quit= true;
  }

 } while(!quit);

}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

# #) Template

→ Generic programing
→ Template function, template class
→ A single function template or single class template without redefing for each and every data. for eg; for swapping, sorting, searching etc.
→ The compiler creates a function or class internally without the user intervention and invoke (call) the same.
→ A call to a template function is similar to that of a normal function & the parameters can be of any data type.
→ The compiler automatically identifies the data type of the arguments passed to the template function & creates a new function and makes an appropriate call.
→ The process of handling template functions by the compilers is totally invisible to the users.

PTO

Scanned by CamScanner

Downloaded from CSIT Tutor

#) Finding maximum of two data items using function template.

```
#include <iostream.h>
template <class T>
T max (T a, T b)
{
  If (a>b)
     return a;
  else
     return b;
}
void main()
{
  char ch, ch1, ch2;
  cin>>ch1>>ch2;
  ch= max(ch1, ch2);
  cout << ch << endl;
  int a, b, c;
  cin>>a>>b;
  c= max (a, b);
  cout<<c<<endl;
  float f1, f2, f3;
  cin>>f1>>f2;
  f3=max (f1, f2);
  Cout << f3;
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## Function & function template

```
#include <iostream.h>
template <class T>
T max (T a, T b)
{
  if (a>b)
      return a;
  else
      return b;
}
Char * max(char *a, char *b);
void main()
{
  char ch, ch1, ch2;
  cin >> ch1 >> ch2;
  ch= max(ch1, ch2);
  cout << ch << endl;
  int a, b, c;
  cin >> a >> b;
  c= max(a,b);
  cout << c << endl;
  char s1[20], s2[20];
  cin >> s1 >> s2;
  float f1, f2, f3;
  cin >> f1 >> f2;
  f3 = max(f1, f2);
  cout << f3;
  cout << max(s1, s2); }
char *max ( char *a, char *b)
{
    return( strcmp(a,b)>0 ? a : b);
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## Multiple Arguments function template

→ All the function template deals with a single generic argument.

→ Declaration of a function template for functions having multiple parameters of different types requires multiple generic arguments.

```
//use of multiple template arguments

struct A
{
    int x;
    int y;
};
struct B
{
    int x;
    double y;
};
template<class T>
void assign A(Ta, Tb, A & s1)
{
    s1.x = a
    s1.y = b;
}
template < class T>
void AssignB(Ta, Tb, B & s2)
{
    s2.x = a;
    s2.y = b;
}

void main()
{
    A s1;
    B s2;
    AssignA(3,4,s1)
    AssignB(3,3.12,s2);
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

Assign B( 3, 3.12, S2);
leads to compilation errors.

→ Since, the template except the first two parameters to be of the same data types and none of them matches.

→ The solution of the problem is, ~~template~~

```
template <class T, class U>
void AssignB(T a, U b, B & s2)
{
    s2.x = a;
    s2.y = b;
}
```

## Class Template

→ Similar to functions, classes can also be declared to operate on different data types. Such classes are called class template.

→ for example, a generic stack class can be created, which can be used for storing data types of integer, real, char e.t.c.

Scanned by CamScanner

Downloaded from CSIT Tutor

```
//stack implementation
    class charstk
    {
      char array[25];
    unsigned int top;
    public:
        charstk();
  void push(char &element);
    char pop();
unsigned int retsize();
  };

  Class Intstack
  {
    int array[25];
    unsigned int top;
    Public:
    void push (int &element);
  int pop();
unsigned int retsize();
  }class Dblstack
  {
    double array[25];
    unsigned int top;
    Public:
    Dblstack( );
    void push(double &element);
     double pop();
    unsigned int retsize();
  }
```

→ The above 3 declaration, a separate class is required for each & every data type.

Scanned by CamScanner

Downloaded from CSIT Tutor

→ Template class declaration enables substitution of code for all three stacks.

```
template <class T>
 Class DblStk
  {
    T array[25];
    unsigned int top;
    Public:
        Datastack();
        void push (T &element);
        T point();
        unsigned Getsize();
  };
```

Scanned by CamScanner

Downloaded from CSIT Tutor

// simple eg of exception for stack (array)

```
#include <iostream.h>
const int MAX = 3;
class stack
 {
   Private:
     int st[MAX];
      int top;
   Public:
      class Range().
       {
         //exception class for stack
       };
     Stack ()
      {
        top = -1;
      }
     void push(int var)
      {
        if(top >= MAX-1)
         {
           throw Range();
         }
        st[++top] = var;
 }
 };
   void main()
   {
     stack s1;
   try{ s1.push(11);
        s2.push(22);
```

Scanned by CamScanner

Downloaded from CSIT Tutor

```
s1.push(33);
s1.push(44); //stack full
cout<< s1.pop()<<endl;
cout<<s1.pop()<<endl;
cout<<s1.pop()<<endl;
cout<<s1.pop()<<endl; //stack empty
}
catch (stack::Range)
{
cout<<"Exception: stack full or Empty"<<endl;
}
cout<<"Normal exit"<<endl;
}
```

Class Range → Here, the body of class is empty, so object of this class have no data and no member function.

→ This name is used to connect a throw statement with a catch block.

→ The Range() statement invoke constructor, which creates an object of this class.

Scanned by CamScanner

Downloaded from CSIT Tutor

// Multiple Exception

```
#include <iostream.h>
const int MAX = 3;
Class stack
{
  Private:
    int st[MAX];
    int top;
  Public:
      class full{};
      class Empty{};
  stack()
  {
    top = -1;
  }
  void push(int var)
  {
    if(top >= MAX-1)
    {
      throw full();
    }
    st[++top] = var;
  }
    int pop()
    {
      if(top<0)
      {
        throw Empty();
      }
      return st[top--];
}};
```

Scanned by CamScanner

Downloaded from CSIT Tutor

multiple exception

```
void main()
{
    Stack S1;
    try
    {
        S1.push(11);
        S1.push(22);
        S1.push(33);
        S1.push(44;) //stack full
    cout << S1.pop() << endl;
    cout << S1.pop() << endl;
    cout << S1.pop() << endl;
    cout << S1.pop() << endl;
    }
    Catch (Stack::Full)
    {
    cout << "Exception: Stack full" << endl;
    }
    Catch(Stack::Empty)
    {
    cout << "Exception: Stack empty" << endl;
    }
    cout << "Normal exit" << endl;
}
```

Scanned by CamScanner

Downloaded from CSIT Tutor

## #) Catch all exception

→ C++ supports a feature to all the exception raised in the try block.

→ The syntax of the catch construct to handle all the exceptions raised in the try block is shown:

```
catch (...)      // 3 dot indicates catch all exceptions
{
    // action for handling all exceptions.
}
```


Compiled By (Noted By)

Deepak Aryal
BSc. CSIT 2069 batch
Asian School of Management & Technology.
Mob [redacted] (IT ~~dang emir~~)



Scanned by CamScanner

Downloaded from CSIT Tutor