INDEX

Object Oriented Programming with C++

Principles of object oriented programming & control structures

Principles of object oriented programming

Object oriented programming paradigm:-

* OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.
* It ties data more closely to the functions that operate on it.

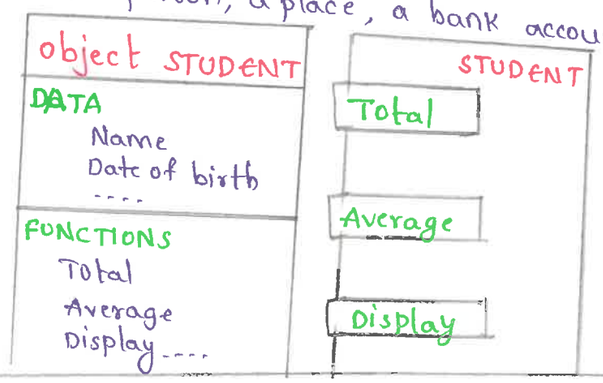
Features of object oriented programming are:

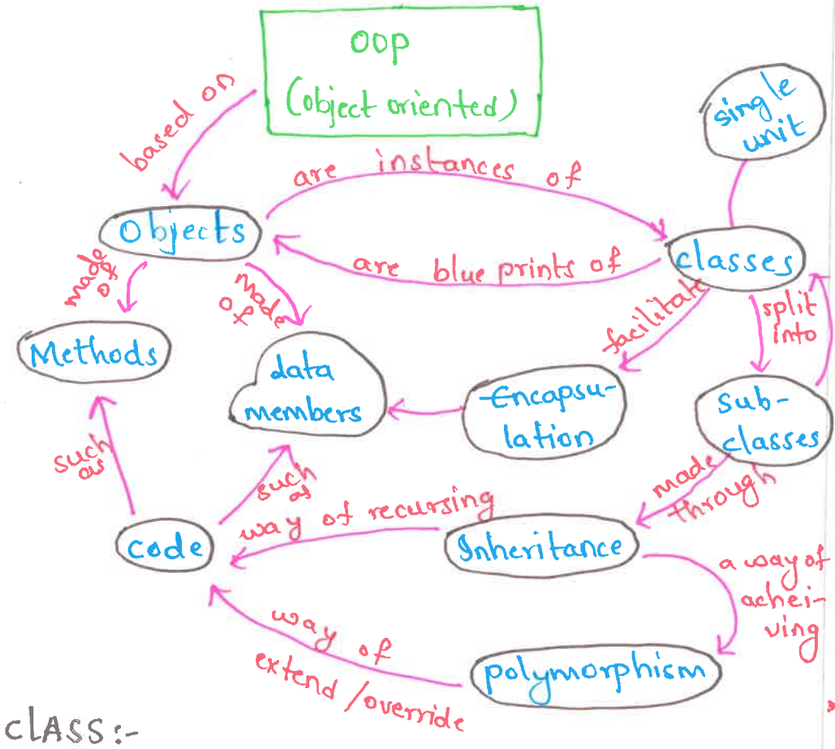
* Emphasis is on data rather than procedure
* programs are divided into what are known as objects.
* Data structures are designed such that they characterize the objects.
* Data is hidden and cannot be accessed by external functions.

Basic Concepts of object-oriented programming:-

Objects:- objects are basic runtime entities in an object oriented system.

* It represents person, a place, a bank account,





CLASS:-

* It is a blue print of object. A class is thus a collection of objects of similar type.
* For example, mango, apple and orange are members of the class fruit.

Abstraction:-

* The Act of representing essential features without including the background details.

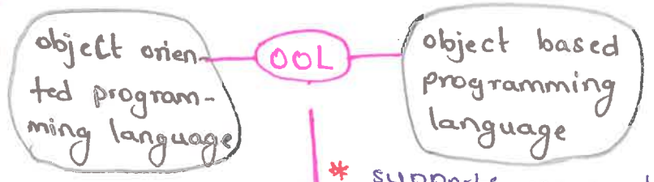
Encapsulation:-

* The wrapping up of data and functions into a single unit is known as encapsulation.
* The data is not accessible to outside world, and only those functions which are wrapped in the class can access it.

Inheritance:- Inheritance is the process by which objects of one class acquire the properties of objects of another class.

Polymorphism:- It means the ability to take more than one form.

Object Oriented languages:-



* incorporates OBPL along with inheritance & dynamic binding
* language that support oops are c++, small-talk, python, Java.

* supports encapsulation & object identity.
* data hiding & access mechanisms.
* Automatic intialization & clear up of objects.
* operator over loading.

Application of oops:-

→ based on Real time systems.
→ simulation and modeling.
→ object-oriented databases
→ Hyper text, hypermedia and expertext
→ AI and expert systems.
→ Neural networks and parallel programming.
→ Decision support and office automation systems.
→ CIM/CAM/CAD systems.

TOKENS

→ Tokens are smallest individual units in a program.

→ Group of characters that logically belong together.

Operators (+, −, *, /)

special symbols to perform arithmetic & logical tasks applied to variables & objects.

key word (do, while, void)

reserved words that has predefined meanings to compiler.

STRINGS ("terry")

group of characters

CONSTANT (45, 21, a)

value that can never be changed.

SPL characters (,), }, {, %, ^)

compiler has special meaning for special characters

IDENTIFIERS (pi, Num)

variables functions arrays, classes all SYMBOLIC NAMES.

Keywords

→ Reserved words that can't be used as variable name / constant.

→ 32 keywords in C++.

⇒

auto break case char const continue default do double else Goto extern for float int if long register return short signed sizeof static struct switch typedef union unsigned void while

C++ data type :-

→ variables use data type during declaration to restrict the type of data.

→ what a kind of variables, a data can store → with different amount of memory.

3 types

→ PRIMITIVE / PRIMARY

↳ built in / predefined and can be used directly by users.

* INTEGER → 4 bytes, 'int' keyword
* CHARACTER → 1 byte, 'char' keyword
* BOOLEAN → TRUE / FALSE
* FLOATING PT → 4 bytes, decimal values
* DOUBLE FLOAT → 8 bytes, double precision
* VOID → value less, for functions.
* WIDE CHARACTER → (2 or 4 bytes).

→ DERIVED → derived from built-in

* FUNCTION → FunctionType Name (parameter);
* ARRAY → datatype Name [SIZE];
* POINTER → datatype of var_Name;
* REFERENCE → datatype & Name;

→ USER-DEFINED → defined by users

* CLASS → building block of C++
* STRUCTURE → Group items of diff. types
* UNION → assign names to constants
* ENUM → all members share the same memory location.

Type Compatibility :-

→ C++ is very strict with regard to type compatibility as compared to C.

→ For instance, C++ defines int, short int, and long int as three different types.

→ It must be cast when their values are assigned to one another.

VARIABLES

```
class terry
{
  public :
    static int a;   → static
    int b;          → instance
  public :
    funct( );
    {
      int c;        → local
    };
```

```
int age = 20;
 ↑    ↑     ↑
data  variable  value
type  name
```

OPERATORS PRECEDENCE AND CONTROL STRUCTURES

OPERATORS IN C++

→ Operators tells compiler to do specific math / logical manipulation.

BINARY →
- +, -, *, /, % → Arithmetic
- !, =, <, >, <=, == → Relational
- && , || , ! → bit wise
- =, +=, -=, /=, %= → Assignment

UNARY OPERATOR → ++ , --

TERNARY OPERATOR :

variable = Expression 1 ? Expression 2 : Expression 3

NOTE :- Expression 1 is the condition to be evaluated.

Expression 2 will be executed & result returned if condition of expression 1 is false.

Expression 3 also executed and result returned

TYPE CONVERSION

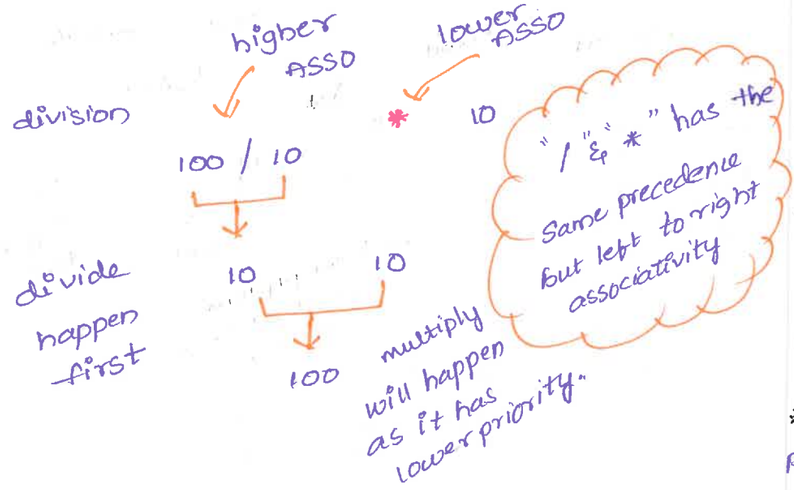
- It is a process of converting one data to another data type.
- Conversion take place low to high or high-low
  - Implicit Conversion
  - Explicit conversion

Implicit conversion :- the type of conversion is done automatically by the compiler.

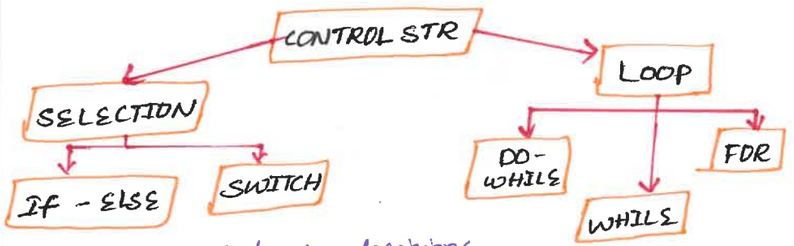
Explicit Conversion :- manually changes data one type to another type. This is explicit conversion

OPERATOR ASSOCIATIVITY :

- It specifies whether an expression contains multiple operators with the same precedence.
- An operand is grouped with the one on its left or the one on its right.



CONTROL STRUCTURES



→ Repeat a code / take decisions.
→ use a compound statement / block.

C++ If-ELSE

If statement test condition and if is executed if condition is true, else part is executed if condition is false.

```
syntaxe :
    if(condition)
   { statement ;
   }
```

```
else
  { statement ;
  }
```

C++ SWITCH :

* switch executes one statement from multiple conditions.
* It is like if-else-if statement ladder in C++.

```
switch (expression){
case value 1 :
break ;
case value n :
break ;
default ;
break ;
}
```

C++ WHILE LOOP

* while is used to iterate a part of program several times.
* Iteration not fixed means, use while loop

```
while (condition)
{
//code to be executed
}
```

C++ do-WHILE LOOP

* It is used to iterate a part of program several times.
* It executed atleast once, whether the condition is true or false.

```
do
{
//code to be executed
}
while (condition);
```

C++ FOR-LOOP

* If iteration is fixed then for loop is used
* In for loop initialisation condition and increment / decrement done in same line.

```
for (init ; con ; incr/decr)
{
// code to be executed
}
```

C++ NESTED FOR LOOP

* for loop inside another for loop known as nested for loop.
* Inner loop executed fully when outer loop is executed one time.

```
int main ( ) {
    for(int i=1 ; i<=2 ; i++){
     for(int j=1 ; j<=2 ; j++){
       cout << "HI";
      }
    }
}
```

output :

HI
HI
HI
HI

FUNCTIONS IN C++

The Main Function :-

* The Main Function is the starting point for the execution of the program

Syntax :-
```
Void main ( )
{
}
```

example :-
```
main ( )
{
// main programs statements
}
```

Function prototyping :-

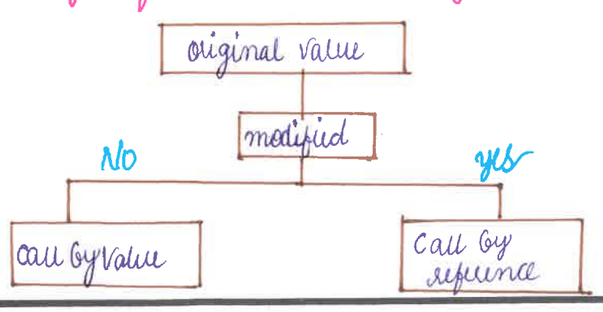
* Function prototype describes the function interfaces to the compiler.
* Tells to the compiler about the number of arguments required.

syntax :- `type functionname (argument-list);`

Example :- `float Volume ( int x, float y, float z);`

Note :-
- `float Volume (int x, float y, z);` is illegal
- `Void display ( );` // function with no arguments

Call by Reference and call by Value :-



Call by Reference :- In call by reference, original value is modified because of pass reference (address)

Example :-
```
Void swap (int* x, int *y)
{
int t;
t = *x ; /* value assigned x to t */
*x = *y ; /* put the value at y to x */
*y = t ; /* put the value at t to y */
}
int main ( )
{
int x = 500, y = 100;
swap (&x, &y);
Cout << " x is " << x << endl ;
cout << " y is " << y << endl ;
return 0;
}
```

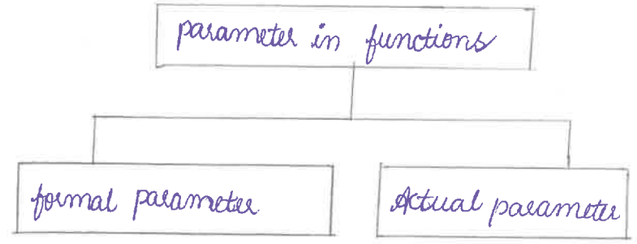
output :- x is 100
y is 500

Call by Value :- In call by value, the value of function parameter is changed for the current function only.

Example :-
```
Void change (int data)
int main ( )
{
int data = 3 ;
change (data);
Cout << data ;
}
Void change (int data)
{
data = 5;
}
```

output :- Value of data is 3.

parameters in a function :-



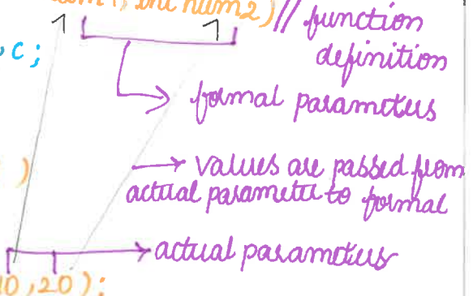
Formal parameter :-

- parameter written in a function definition is called formal parameter
- They are used in the function header.
- It receives the values that are passed to the function.
- It is treated as local variable.

Actual parameters :-

- parameter written in the function call is called Actual parameter.
- Numbers, expressions or even function calls are used.
- It is used in the fuction call.
- Values are passed to the function definition

```
Void add (int num1, int num2) // function definition
{
int a, b, c;
c = a + b;
}
int main ( )
{
add (10, 20);
return 0;
}
```



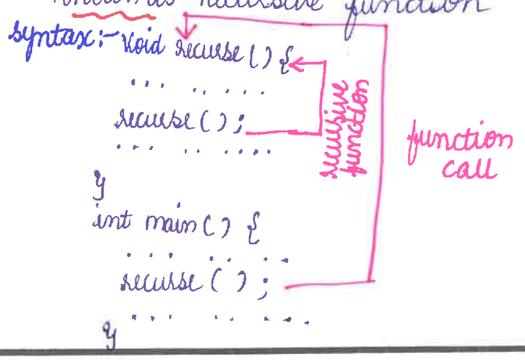
passing Array to function :-

- Arrays are passed through function definition from function call.
- Different data type array values can be passed

Example :-
```
void print max (int arr [5]);
int main ( )
{
int arr [5] = {25, 10, 5 4, 15, 40};
// array values are defined
print max (arr1); // passing array to function
}
Void print max (int arr [5]);
{
int max = arr [0]; // function definition
for (int i = 0 ; i < 5, i++)
{ if (max < arr [i])
{
max = arr [i];
}
}
cout << " maximum element is: " << max
}
```

C++ Recursive Function :-

- A function that calls itself is known as Recursive function



Return by reference

A C++ function can return reference Similar array as it returns a pointer

```
# include<iostream>
int n;
int &test();
int main()
{
test()=5;
cout <<n;
return 0;
}int &test()
{
return n;}
```

output: 5

Inline function

It is a function that is expanded in line when it is called by inline function.

Syntax:-

```
(inline function-header
{function body
}
```

Ex:-

```
inline int cube (int s)
{
return s*s*s;
}
int main()
{
cout<<"The cube is 3 is:"
<<cube(3)<<"\n";
} return 0;
```

output:- The cube of 3 is: 27

Default argument in C++:

A default value is a value in the function declaration automatic-ally assigned by the computer

Syntax:-

```
int sum (int x, int y, int z=0, int two);
```

Ex:-

```
float amount (float Principal, int Period, float
rated =0.15);
value =amount (5000,7);          //one arg missing
value = amount (5000,5,0.12);// error arg missing
```

Function overloading

When Two or more functions can have the same name but with different Parameter is used in Program is called function overloading.

Syntax:-

```
//Declarations
int add (int a, int b);
int add (int a, int b, int c);
double add (double x, double y);
double add (int p, double q);
double add (double p, int q);
//function calls
count << add (5,10);
count << add (5,10.0);
count << add (12.5,7.5);
```

Ex:-

```
void Print (float f);
{
}
void Print (int a)
{ }
int main ( )
{ Print (2.5);
  print (4);
}
```

classes and objects

Specifying a class:-

A class is used to specify the form of an object. It combines data representation and methods for manipulation.

Class declarations

```
class class_name
{
Private:
  variable decl;
  function decl;
Public:
  variable decl;
  function decl;
};
```

Ex:-

```
class item
{
  int number;
  float cost;
Public;
  void getdata(int,float);
  void Putdata(void);
}; //ends with semicolon
```

Accessing data members

Data members of objects, of a class can be accessed using the direct members access operator(.).

Syntax:-

```
object-name, function-name (actual-argument);
```

For example, the function call statement

```
x.getdata (100,75.5);
```

Member function:-

The function which is declared inside a class is called member function.

```
Class Box
{
  Public:
    double len,b,h;
    double getVolume(void);
};
```

Classes and objects

Scope resolution operators:-(::)

with this operator, it is used to define the member function outside the class

Syntax:-

```
return-type class-name ::function-name (arg,decl)
{ function body }
```

Ex:-

```
void getdata ( );
//function definition outside class.
void item ::get data (int a,float b)
{  number =a ;
   cost =b;
}
```

Array's within a class

- Arrays can be declared as member of class.
- Arrays can be declared as Private, Public or Protected number of class

Syntax:-

```
#include <iostream.h>
  class  array,demo
  { int  a [b];
    Public:
     void getelts ();
  };
```

Ex:-

```
#include <iostream>
using namespace std;
cost a=50 ;
class ITEMS
{ int itemcode (m);
  float itemprice(m);
   int count;
Public:
    void CNT(void)(count=0)
    void get item(void);
    void displaysum (void);
    void remove (void) ;
    void display item (void);
};
```

- In the class arraydemo and in the class items a [ ] and item code [m], itemprice [m] is declared.

Static member function

*A static member function shares the single copy of the members function to any number of class object.

Syn* A static function can have access to only other static members (function or variables) declared in the same class.

*A static member function can be called using the class name (instead of it object) as follows:

class-name :: function-name:

Syntax:-

```
Class Demo
{ Private :
    Static int x;
    Static int y;
  Public :
    Static Void Print()
    { }
};
```

Ex:-

```
#include <iostream>
using name space std;
class test;
{
   int code;
   static int count
Public:
   void setcode (void)
   {
   code = ++count;
   }
   void Showcode(void);
  {
  count <<"object number";
  }
Static Void showcount(void);
{
cout <<"count :"<<count<<"\n";
}  };
int test ::count;
int main()
{
test ::showcount();
// calling static function.
}
```

NOTE:-

- Static members are initialized to zero, when the first object of its class is created.
- only one copy of that member is created for the entire class, and shared by all the objects of class.
- This visible only within the class.

Array of object

*An array of type class can be created.

*Array of class type is also known as "array of object".

Syntax:-

```
class books
{ char title [30];
  float Price;
 Public:
     Void getdata();
     Void Putdata();
};
 int main ()
{
 books book [3];}
//array of object created
```

Ex:-

```
Class employee
{
   char name [30];
   float age;
Public:
    void getdata (void);
    void Putdata (void);
};
Employee manager[3];
Employee foreman[15];
Employee worker [15];
```

C++ friend function

If a function is defined as a friend function in c++ then protected and Private data member of class can accessed

Syntax:-

```
Class B
{
friend datatype
      functionname();
};
```

Ex:-

```
Class A
{ int x ;
Public :
  A()
  { x=10;
}
friend class B ;//friend};
Class B {
Public :
void display (A &t){
cout <<"value of x"<<t.x );};
void main (){
A a ; B b ;
b.display (a);
return 0 ;}
```

output:-

The value of x=10

CONSTRUCTOR & DESTRUCTOR

Constructor

- A Constructor is a special member function, whose fasle is to intialize the member of its class.
- Allocate memory for object
- Automatically called when object is created
- Declared in Public section & doesn't return any value.
- Syntax:-

```
ClassA
{ Public:
    int x;
    ACL;
};
```

Types of constructor

- Default constructor
- Parameterized constructor
- copy constructor

Default constructor :-

- It accepts no arguments
- If no default constructor is defined means, it will execute, by the help of compilers.

Syntax:-

```
ClassA
  { Public:
      A(); -> Default
              constructor
  };
```

Parameterized constructor

- when a constructor has a Parameter, it is called Parameterized constructor.
- It will supply orgument, when the time of creation of object is done

Syntax:-

```
ClassA
 {
   A(int,int);
 Void main()
 {
   A a;
   A a(5,4);
 }
```

```
#include <iostream.h>
class Point
  {int x,y;
   Public:
     Point(intx,inty)
     { x=x1; y=y1;
     }
    int getx() {return x;}
    int gety() {return y;}
    };
    int main ()
   { Point P1(10,15);
cout<<"P1.getx() <<p1.gety()
  return 0;
}
```

copy constructor

- A copy constructor is a member function that initialize an object using another object of the same class.
- Copy constructer takes a reference to an object of the same class of an argument.

Ex:-

```
#include <iostream.h>
 Class sample
 { int id;
    Public:
        void init(int x)
        { id=x;
        }
     void display()
      {
      Count<<"ID"<<id;
      }
 };
int main()
{
 Sample obj1;
  obj1 init(10)
  obj1. display();
 Sample obj2(obj1);
 obj2. display();
 return 0;
}
```

Dynamic constructor

when allocation of memory is done dynamically using dynamic memory allocators "New" in a constructor is known as dynamic constructor

Constructor overloading

- we can have more than one Constructor in a class with the same name as long as such has a different list of orgument
- overloaded constructor have the same name & differ by number of orguments.

Destructors

- A Destructor is a spacial function as a constructor
- It destroys the class object created by constructor

Example:-

```
#include <iostream.h>
 Class Test
{ Public:
   Test()
  {
  Cout<<"constructor executed"
  }
 ~Test()
{ cout<<"destructor executed";
};
int main()
{test t;
return 0;
}
```

OPERATOR OVERLOADING

Operator Overloading

— Relation to the class to which the operator is applied.

Syntax of operator overloading

```
return type c.n :: operator (arg)
{
   ≡
}
```

Steps for operator overloading

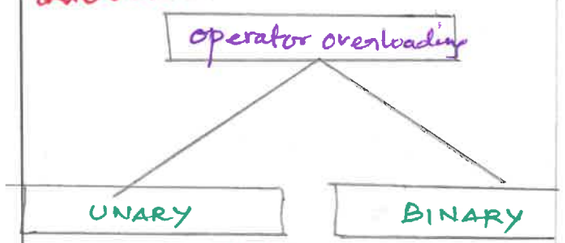
1. Create a class defined by type
2. Declare the operator ()
3. implement the required function.

Eg: op x (or) x op

for unary x op y

for binary operator op(x)

Types of operator overloading



Unary operator loading :-

- Just one operand
- if it minus, change the sign of operand when applied into basic data.

Syntax:

```
return type class name :: operator -()
{
   statements;
}
```

example:

```
class space
{  int x,y,z;
 public:
      void getdata(int a, int b, int c);
      void display (void);
      void operator-();  };
void space :: getdata
              (int a, int b, int c)
 { x=a; y=b; z=c; }
void display ()
 { cout<< x <<y<<z; }
void space :: operator -()
 {  x=-x; y=-y; z=-z; }
void main()
{    space s;
   s.getdata(10, -20, 30);
    cout<< s;
    s.display();
    -s;
    s.display(); }
```

I/p: 10, -20, 30 — O/p: -10, 20, -30

Binary operator overloading

Adding two numbers using operators (+)

- one argument must be in the operator function

Syntax:

```
return type class name :: operator +(arg)
{
   statements;
   = }
```

Example:

```
class complex
{ float x, y;
 public:
    complex ();
    complex(float x, float i)
 { x=x; y=i; }
    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 (void)
 { cout<< x<< "+i" <<y << "\n"; }
void main()
{   complex c1, c2, c3;
    c1. complex (2.5, 3.5);
    c2. complex (1.6, 2.7);
    c3 = c1+c2;
    c3.display (); }
```

Overloading Binary using friends

1. Replace the member function by declaring **friend** function
2. complex operator + (complex a, complex b)
   { return complex (a.x + b.x), (a.y + b.y)); }

Manipulation of String

```
String :: String (const char *s)
{  len = strlen(s);
   p = new char (len+1);
   strcpy (p, s); }
String :: String (const String &s)
 {   len = s.len;
   p = new char [len+1];
   strcpy (p, s.p);
```

Rules for overloading operators

* only existing can be overloaded.
* New operators cannot be created
* overload operator must have one operand (atleast)
* we cannot change the basic meaning of operator.
* must follow the syntax and rules
* cannot be overridden
* Some operators cannot be overloaded

  size of, . *, ::, ? :
* friend function cannot be used for certain operators

  =, (), { }, ->
* unary operators, take no explicit arguments and return no explicit values, but friend function take one argument.
* Binary operator, take one argument and return one argument when overloaded, but friend function it Take 2 arguments.

MIN. REQ	UNARY	BINARY
ARGUMENTS	NIL	1
FRIEND FUNCTION	1	2

TYPE CONVERSION

— different types of mixed in exp.

* Basic to type to class type
* class type to Basic type
* one class type to another class type.

* * * * * * * * * *

INHERITANCE & POINTERS

Defining Inheritance :

* the mechanism is to derive a new class.

syntax for derived class

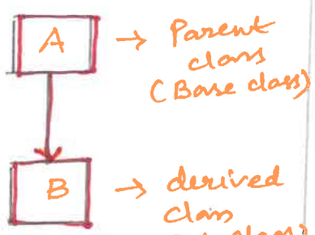
```
class  new class name : access specifi  old class name
{
    Variable decl;
    function decl;
};
```

example :-

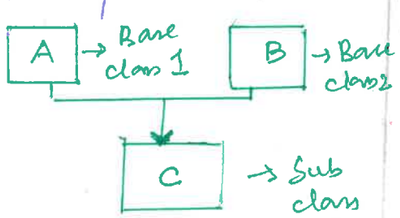
```
class B : public A
{     int x;
  public:
      void disp(); };
```

TYPES OF INHERITANCE

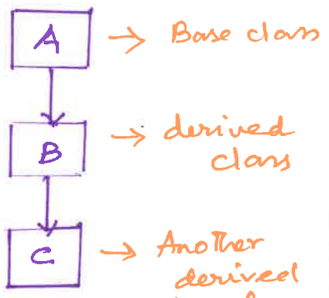
1. Single Inheritance
   * a derived class with only one base class



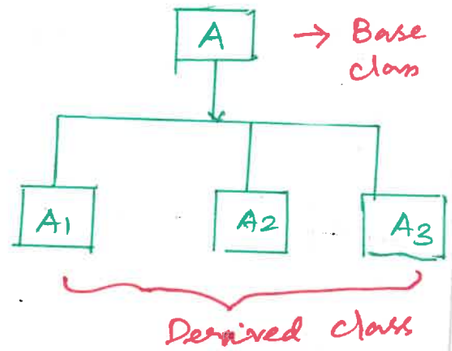
2. Multiple Inheritance
   * one derived class with many base classes



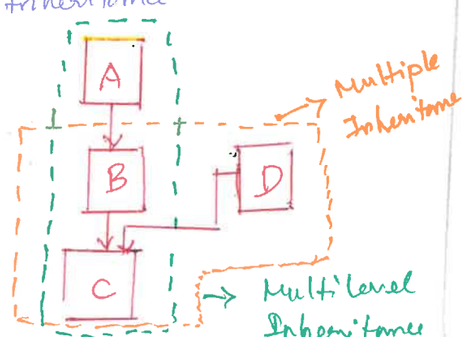
3. Multilevel Inheritance
   * deriving a class from another derived class



4. Hierarchical Inheritance
   * one class may be inherited more than one derived class.

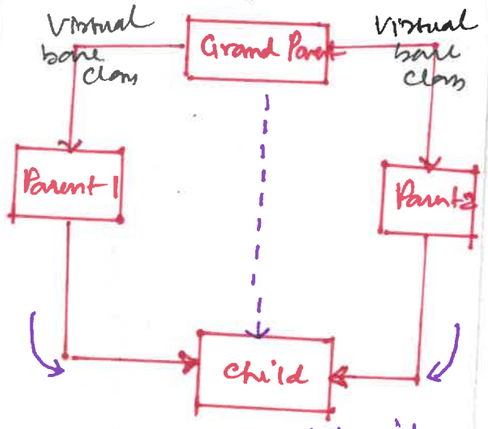


5. Hybrid Inheritance
   * Combination of more than one type of inheritance
   * Multi level + multiple Inheritance.



VIRTUAL BASE CLASS

* the duplications of inherited members due to these paths can be avoided by making the common base class as Virtual base class.



→ Multipath Inheritance

```
class A
{ ≡ };          // grand parent
class B1 : virtual public A
{ ≡ };          // parent 1
class B2 : public virtual A
{ ≡ };          // parent 2
class C : public B1, public B2
{ ≡ };          // only one copy of A will be inherited.
```

ABSTRACT BASE CLASS

* Not used to create an object.
* act as base class inherited by the other class.
* it is built-in, provide by based upon the program.

Syntax :-

```
abstract class <class name>
{
    ≡
};
```

→ abstract is keyword.

* it can inherit only the abstract methods (or) functions.

CONSTRUCTOR IN DERIVED CLASS

* Object of derived class is created then get executed
* No default constructor present in derived class,
* Then create constructor in child class & construct in parent class.

syntax :-

```
derived constructor (arg1, arg2 .. argN) : myD1
{
    base1 (arg1),
    base2 (arg2),
    ...
    baseN (argN)
{ Body of derived constructor
    ≡
}
```

MEMBER CLASS

* deriving certain properties into another class.
* it takes a view that an object can be collection of other object.

```
class alpha { ... };
class beta { ... };
class gamma
{ alpha a;
  beta b; };
```

POINTERS

→ derived data type
→ address of the another variable

Declaring a pointer :

```
datatype *var;
int a, *ptr1, *ptr2;
ptr1 = &a;
ptr2 = &ptr1;
```

'this' pointer

* this pointer points to the object for which this function,

```
this -> a = 123;
```

POINTERS TO OBJECT

* it is used for object creation

```
item x;
item *it_ptr;
ptr -> show();
```

POINTERS TO DERIVED CLASS

* it not only for base class but also derived class

```
B *cptr;
B d;
D d;
cptr = &b;
cptr = &d // cptr points to object d;
```

→ this points to object d.
→ this pointer can be access following syntax.

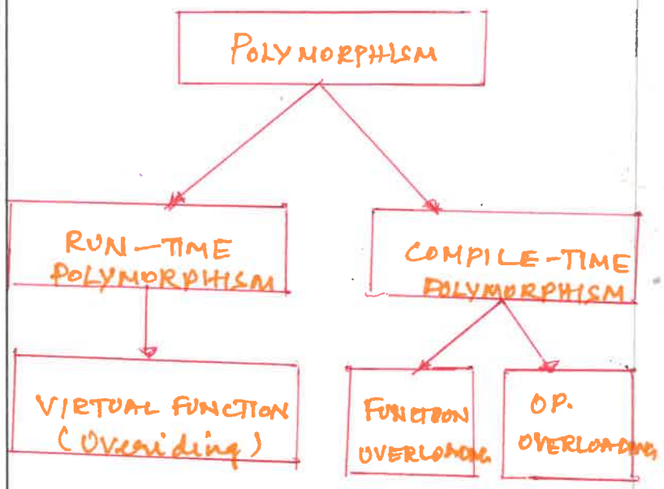
```
class ABC
{ int a;
≡ };
this->a = 123;
```

POLYMORPHISM

POLYMORPHISM

- Single name given multiple meaning.

TYPES OF POLYMORPHISM

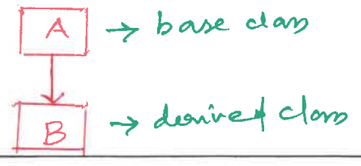


VIRTUAL FUNCTION

* it is a member function
* derived within base class and re-defined in derived class.
* ensure that calling correct function is called for object
* Same function name and same argument list in base and derived class.

it is called,

* Run-time polymorphism
* virtual keyword used in base class.
* resolve of function call at run time.



→ base class

→ derived class

```
class A                                   Virtual keyword in function
{ public:
    virtual void print()        // Same function name
    { cout<<" Am in base class"; }
    void show()
    { cout << " Show class"; } };

class B
    { public:
        void print()            // Same function name
        { cout<<" am in derived class"; }
    };
```

COMPILE TIME POLYMORPHISM

1. Function overloading
   → same function name with different argument list

```
void area(int);
void area(float);
void area(float, int);
```

2. Operator overloading
   * change the basic meaning of the operator.

```
void operator <symbol> (arg);
```

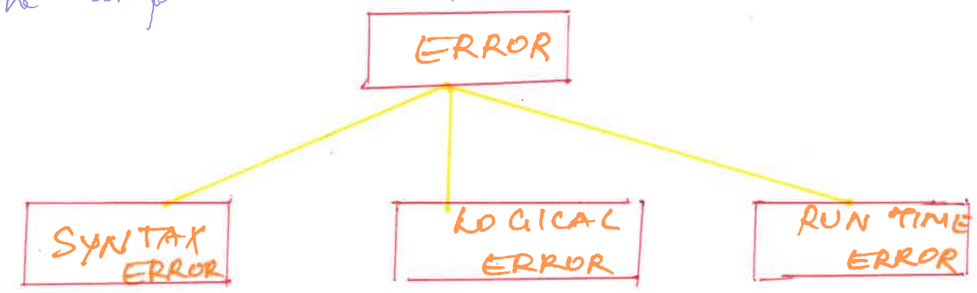
PURE VIRTUAL FUNCTION

* Some times implementation of all function cannot be provided in base class.
* we must override which it in derived class, otherwise it is become abstract class
* ~~write code~~ " DO NOTHING FUNCTION "

```
class Test
{ public:
    virtual void show() = 0;
};
```

EXCEPTION HANDLING

- to handle the run-time error during the compilation of Program.



EXCEPTION HIERARCHY

EXCEPTION HANDLING

Exception

* during execution of program, distrupts the normal flow of program.
* Runtime anamolies
* unusual condition like divide by zero, out of range

Exception Mechanism

* find the problem (hit exception)
* inform about its occurance
* Receive the error information
* Take proper action.

Why exception ?

* separation of error handling from Normal code.
* Functions & Methods handle any type of exceptions.
* Grouping of Error types.

HANDLING THE EXCEPTION

* try
* catch
* finally

Try → generate exception, Throws from inside the block.

Throw → raising an exception done by "throw" expression.

catch → actin to be taken when exception occurs.

finally → its for user consideration
→ solution for users.

Syntax:

1. divide by zero
2. out of range
3. Bad allocation

```
try
{ statements;
  ==
}
catch ( type of argument)
{
  ==
}
```

Multiple catch exception:

* handle the different exceptions differently,
* must include `catch` statement with different declarations.

Syntax:

```
try
{ ==
}
catch (arg1)
{ ==
}
catch (arg2)
{ ==
}
...
catch (arg n)
{ ==
}
```

Example:

```
int main()
{ int number=10, ans=0;
try { ans = number/0; }
catch (int i)
{ cout << " divide by zero"; } }
```

RE-THROWING EXCEPTIONS

* where we have inner & outer try-catch statements (Nested try)
* exception thrown from inner catch block to outer catch block.

```
try { ==
      throw val;
    }
catch (arg)
{ throw
  ==
}
catch ( datatype arg)
{ ==
}
```

(throw excep. value; Re-throw excep.)

Example:

```
int main()
{ int a=1;
  try { try
        { throw a; }
      catch (int x)
      { cout<<" Exception inner block" }
      throw x; } }
  catch (int x)
  { cout<<" Exception in outer block";
  }
  cout<<" End of Prog ";
}
```

Defining User-defined Exception :

* define own exceptions
* Inherit & overriding.

```
class My : public Exception
{ public:
  const char* what()
  const throw()
  { return "c++";
  } }
```

My → own exception.

C++ Standard Exception

Example - Multiple catch

```
try
 { if (a==1)
    { throw a; ;
   else if (a==2)
     throw 'A';
   else if (a==3)
     throw 4.5;
 }
catch (int a)
 { cout<<" int exception"
catch ( char a)
 { cout<<" char exception" }
 }.
```

* * * *

SAMPLE PSUEDO CODE FOR PROGRAMS

1. Write a program to find the sum of natural numbers ?

Pseudo Code :

Start program
Declare variables n, sum=0 and i
Enter the number for n
for i= 1 to i<=n
Perform operation sum = sum + i
increment i value by one
Print sum
End program

Input :- Enter the no. of terms : 5

Output :- Sum of series : 15

also try with while do while

2. Write a program to print fibonacci series of a number?

Pseudo Code :

Start program
Declare variable n and i = 0
Enter the number for n to generate fibonacci series
While i<n
++ i [increment i value]
if n == 0 or n == 1
Return 0
if (n == 2)
Return 1
else
Return (fib(n-1) + fib(n-2)
Print fibonacci series
End program

input = Enter the no to generate fibonacci's series=6

output = The fibonacci series : 0 1 1 2 3 5

1. Remember the iteration loop
2. apply recursive function

3. Develop a program to check whether the entered user name is valid or not. Get both inputs from user?

Pseudo Code :

Start program
Declare variable a and b
check
if (a == b)
cout "Valid User Name"
else
coub "Invalid User Name"
Endprogram

Input :- Enter the username : Saveetha@789
Re enter the user name : Saveetha@123

Output :- User name invalid.

* Sample Input *
1. Get the String.
2. Compare it.
3. Remember the syntax.

4. Build program to reverse a number using loop ? (Get the input from user).

pseudo code :-

start program
Declare function reversenum (int n)
while (n != 0)
{
digit = n % 10;
n = n / 10;
}
call the reversenum franction from main
End program.

1. Remember the Syntax of while loop

Input :- Number : 14567

Out put :- Revers number : 76541

Hint :- Iteration loops used when repetation of procen happen,
① while ② do...while ③ for loop.

5. Develop a program using function to calculate the simple interest suppose the customer is senior citizen he is offered 12 percent rate of interest ; for all other customers the ROI is 10 percent ?

Pseudo code :-

Start program
Declare variable P, Y, C, i, a ;
Get the value for P, Y
Input choice for Senior or not Senior
switch ( )
{
case 1 :
a = P * Y * 0.12
cout "output of the value for normal person"
}
End Program

* Apply multiple break Statements

* Use default Statement
* use of Break Statement

Sample input

Enter the principal amount : 200000
Enter no. of Years :- 3
is customer senior citizen (Y/n) : n

Sample output

interest : 60000

Also try with the Compound interest

Hint :- ① try with the in^ no. of cases
② Check the default arguments
③ Sample of arithmetic operations
④ Identify the week days & week count, n th day - Programs.

6. Develop a program for matrix multiplication in c++ ?

Pseudo code :-

```
procedure Matrix Multiplication (A,B)
  input  A,B  n*n  matrix
  output C,   n*n  matrix

begin
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      c(i,j) = 0;
    end for
  end for

  for (i=0; i<n; i++)
    for(j=0; j<n; j++)
      for (k=0; k<n; k++)
        c[i,j] = c[i,j] + A[i,k]*B[k,j]
      end for
    end for
  end for
end Matrix Multiplication
```

Inner block

outer block

Hint :-
1. if is array must use for loop

Sample input :-

Enter matrix for A

2 4 1
2 3 9
3 1 8

Enter matrix for B

1 2 3
3 6 1
2 4 7

Sample output :-

product of 2 matrixes is :

16 32 17
29 58 72
22 44 66

2. if matrix, must use 2 loops...

7. Develop a c++ program for electricity bill computation using the tariff given bellow

unit tariff

>100 RS. 1.20 per unit,
>200 RS. 2 per unit, >300 RS. 3 per unit

Pseudo Code :

```
Start program
Declare class ebill
initialize variable cno, Cname, units, bill;
Declare function get() - to get values
    Get the value for cno, Cname, units,
Declare function Putt() - to output value
    output the value for cno, cname,
    units, bill;
Declare function call()
if (units <=100)
        bill = units * 1.20;
    else if (units <=300)
        bill = 100 * 1.20 + (units-100)*2;
    else
        bill = 100*1.20 + 200*2 + (units-300)
        *3;
Declare main()
  Create object for class
  call all the function.
```

Sample input :-

Enter customer Name : Dinesh
Enter no. of units : 500
Customer no : 1
Bill of customer : ₹1120

1. Remember the Syntax of else..if ladder
2. Notify the use of final else part

8. Develop a c++ program to print area of circle using class

Pseudo code :-

```
class Circle
Begin
  Create Radius = 1.0
constructor circle   //called default constructor
Begin
End constructor
constructor circle(Newradius) //calle constructor
Begin
    Radius = New radius
End constructor

Method GetArea()  // compute & return Circle area
Begin
    Return (* Radius * 3.14159).
End Method
End class
```

Rember The Rules
1. for Constructor

Remember the
2. types of Constructor
3. Notify the how memory for objects created
4. how memory alloted.

Sample input :-

Enter the radius : 7

Sample out put :-

The area of circle is : 154

Remember keywords :
1. Check function name
2. public
3. object creation

→ when we use constructor.

} These three are important points.

Write a program to read and print data for student report using Single Inheritance.

Sample Input :-

Enter student name, regno, m1, m2, m3, m4, m5, m6

Step :-

1. Create a base class name as "student"
2. Use a member function getdata() as public access specifier
3. Read the all inputs.
4. Create a derived class as per syntax with name of student1
5. All the values to be inherit from the base class.
6. Calculate the total, average and grade.
7. Create an object for derived class student1 s.
8. Use this object call all the member function of base and derived class.
9. Execute the output.

Sample output :-

Total marks : 455
Grade : A

Build a code for print address of variable.

Sample Input :

x = 10, y = 20

1. Read x and y value using cin statement.
2. Assign the value to variable using '&' and '*' operators
3. print the address of x and y value.
4. x = &x; y = &y;
5. Execute the output.

Sample output :

x = 0x1115bf
y = 0x1115cf

Write a c++ code for area of square and circle using virtual function

Sample Input :

Enter radius : 5

1. Create a base class circle.
2. Create a member function area() in public.
3. return the value of area()
4. Create a derived class as per syntax.
5. Create a member function as area() in public with virtual.
6. Return the volume of circle.
7. Create a pointers to object in main() b, derived class ob.
8. Assign the object for both base and derived class with '&'
9. Use '->' operator call the member function of area() in base and derive class.
10. Execute the output.

Sample output :

Enter radius : 5
Area of Square : 16
Area of Circle : 78.5

Display the address of each element of an array.

Sample Input :

Not required

1. Create an array

a = { 0, 1, 2, 3, 4 };

2. Get the address for each array value.

&a[0] = 0x1111f ; &a[3] = 0x111f
&a[1] = 0x1112f ; &a[4] = 0x1114

3. Print the address.
4. Execute the output

Sample output :-

a[0] = 0x1111f
a[1] = 0x1112f
a[2] = 0x1113f
a[3] = 0x1114f

used 1 & / operator for getting address of value

C++ Program for concept of Multiple Inheritance. for adding two numbers

Sample Input : x = 5, y = 5

1. Create a base class for add1
2. Use sum() for getting x value.
3. Create another one base class for add2
4. Use sum1() for getting y value.
5. Create derived class for add3.
6. Inherit the x and y value from add1 and add2.
7. Sum of 2 numbers using sum and sum1.
8. Create an object for derived class.
9. call all the member function using object of derived class.
10. Print x and y value and sum of x and y.
11. Execute the Program.

Sample output :-

x value = 5, y value = 5
Sum of x and y = 10.

C++ Code for Hierarchal Inheritance - Employee details

Sample Input :

Enter the employee no, name, hra, da, ta, pf, lic.

1. Create a base class for e1
2. get the input value from input1().
3. Read the empno & name.
4. Create a derived class e2
5. get the input values from input2().
6. Read the values of hra, da, ta, salary.
7. Create a another derived class e3.
8. get the input values from input3().
9. Read the value of pf and l'c
10. Create an object for e3 e;
11. call the member function of e3 and e2 using e.
12. Create an another object and call the member function of e1.
13. Calculate the values of GP, NETPAY and DED.
14. Execute the output.

Sample output :

GP : 41376.00
DED = 1000.00
NP = 40378.00

Write a program to illustrate divide by zero exception

Sample input

x=5, y=0

1. Create a main function for exception
2. try the following statement
   z = x/y;
3. If try block, if any error in statement send to error to catch block.
4. catch will receive the error and generate the exception
5. Finaly given the user defined Solution.
6. Execute the output.

Sample output:

"Divide by zero".

Program to illustrate array index out of bounds Exception.

Sample input:

x[] = {10,20,30,40}

1. Create a main function for exception
2. try the following statement,
   x = x[0] + x[1] / x[4];
3. Catch Block receive the error from the try block.
4. catch say " It is array out of Boundry ".
5. Execute the output.

Sample output: "Array out of Bandry"

C++ program to throw multiple exceptions and define multiple catch statement.

Sample Input : x = 0.0000001

1. Create a main()
2. Use try block, (x < 0.000001)
3. if is small compare to the x value,
4. catch with receive the error from try block.
5. Print "No is too small".
6. Another catch also receive the error if no. is not an integer.
7. Print "Not an integer"
8. final solution given by user-defined statement.
9. Execute the o/p.

Sample output

x = 5;

1. Number is too Small
2. It is not an integer.

final solution,

it is a float value.

C++ Program for calculate the tax following condition

<1,50,000 := No tax

150001 - 300000 = 10%.

300001 - 500000 = 20%.

>500000 = 30%.

use class name Income tax and derived class S1, S2, S3 use TDS() all the class.

Sample Input:

Get Input value as tax.

1. Create base class "Income tax"
2. Call the appropriate values.
3. Create 3 derived classes S1, S2 and S3.
4. Use function TDS() for all base and derived class.
5. Also use virtual keyword all function because base and derived ~~at~~ having same functions.
6. Use Nested if statement for calculate tax.

```
tax += tax * 10/100;
tax += tax * 20/100;
tax += tax * 30/100;
```

7. Print the taxes for all slabs.
8. Create an object for S1, S2, S3.
9. Use pointers to objects for all class, it is a virtual function

```
S1 *s1;
S2 *s2;
S3 *s3;
```

} Pointers to object

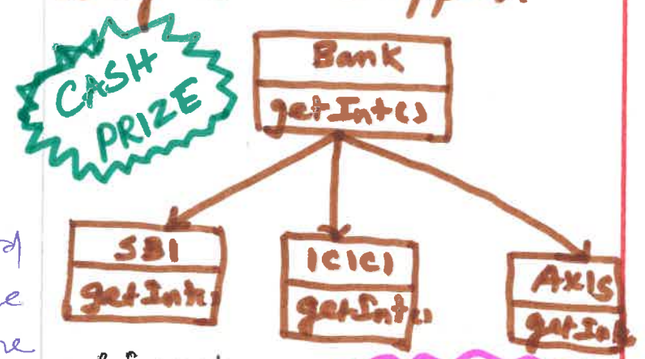
10. call all member function using '->' operator.
12. Close the class.
13. Execute the output

Sample output:

Enter the amount : 1,00,000

No Tax have to print.

Write down code for following diagram. - Snippet.



Snippet:

```
class Bank
{ public:
    void getInt()
    { == }};
class SBI : public Bank
{ public:
    void getInt()
    { == }};
class ICICI : public Bank
{ public:
    void getInt()
    { == }};
class AXIS : public Bank
{ public:
    void getInt()
    { == }};
```

1. Name of Concept?
2. Identify the

3. What is the missing statement?