06 August 2018

Characteristics of Procedural Oriented Programming Para-

1) Emphasis on doing things (algorithms)
2) Large programs are divided into smaller program known as function
3) Most of the functions share global data
4) Data move openly around the system from fn to fn
5) fn transform data from one form to another
6) Employs top down approach in program design.

Object

Object is a identifiable entity with some characteristic and behaviour
↳ represented by data ↳ Represented by function

Characteristics of Object Oriented Programming Paradigms

1) Emphasis on data rather than procedure
2) Programs are divided into what known as object
3) fn that operate on data of an object are tied together in data structure
4) Data is hidden and cannot be accessed by external function
5) Objects may communicate with each other
6) New data & fn can be easily added whenever necessary.
7) Employs bottom up approach

class

class is known as group of objects which shares common properties and relationship.
A class provides template of the object and any instance of class is object.

August 2013

BASIC CONCEPT OF OOPS

i) Abstraction : Abstraction refers to the act of representing the essential features without including background details.

ii) Encapsulation : The wrapping of data and function (that operate on the data) into a single unit (called classes) known as encapsulation.

iii) Modularity : Is the property of the system that has been decomposed into the set of loosely coupled and highly cohersive module
It is process of partitioning the program into individual components
1) To reduce complexity of large programs
2) It creates the well documented, defined boundaries within the system.

iv) Inhe
thi
clas
are
inhe

v) Poly
to
na

iv) Inheritance : It is capability of one class of things to inherit properties from another class. The class from which properties are inherited is called base class and class inheriting property is base class

v) Polymorphism : It is property of message to perform different action under one name

13 August 2012.

BEGINNING WITH C++.

C++ was developed by the Bjarne Stroustrup at AT&T laboratory.

Comments

Those lines in C++ program that are too be ignored by compiler while execution.

1) Single line comment, //.
2) Multiple line comments, /*....*/

Output Operator

1) cout

cout << Here < is bitwise left shift operator and << is insertion operator or put to operator.

Input Operator

1) cin

Console input

Extraction or get from operator

Factorial Program

```
# include <iostream.h>
void main()
{ int a,b;
  cin >>a;
  if (a=0) {b=1;}
  else if (a>=0)
    { for (int i=0; i<a; i++)
```

```
        b=b*a;
    cout<<"factorial is "<<b;}
    else.
    {cout<<"factorial does not exist";}
    cout<<factorial<<}
```

Generated by CamScanner from intsig.com

CLASSES & OBJECTS.

DATA TYPES.

1) Fundamental or built in type
   Integer, float, double, char, void
2) User defined data types
   Array, structure, union, enumeration, class

LIMITATIONS OF STRUCTURE

1) Structure variables are not treated as built in data types
2) Structure variables do not provide the funtionality of data hiding

CLASS

Class specification contains 2 parts
i) class declaration
2) class function definition.

```
Class
{ private :  → accessibility label
    data declarations
    function declarations
  public:
    data declarations
    function declarations
};
```

21 Aug 2019

Date:

CREATING CLASS OBJECTS

student x, y, z;
class { } x, y, z;

Accessing class members
x.input(); y.input();
x.rollno; ✓ x:rollno; X

Defining functions
- Inside class
- Outside class (scope resolution)

INLINE FUNCTION

Inline does not works at
- switch, loops, if else
- Having return type and don't return
- Having recursive fus.
- static variables
- too large code
- Non returning but contains return
- They can't use static variables.

Inline void student :: Input()
{ .... }.

21 Aug 2019 **NESTING OF MEMBER FUNCTIONS**

Date:

```
class student
 {
  int rollno;
  char name[25];
  public:
  void input()
  {
  cin >> rollno >> name;
  {
  void display()
  { input(); // nesting fn.
   cout << rollno << name;
   {
  };
```

cascading of I/o operator

When 1 fn of class calls another fn

ARRAY WITHIN CLASS

```
const int size = 10;
class array
{ int a[size];
};
```

Date:

STATIC DATA MEMBER

1) The static variable is initalized to zero when the 1st object of its class is created. No other initalization is permitted.
2) The only one copy of this member is created and all the objects will share. that copy. no matter how many objects are created
3) It is visible only within class. but its lifetime is erdire program

```
class item
 { static int count ;
    int number;
    public :
    void getdata (int a)
     { number=a;
       count++; }
     void getcount(void)
     { cout <<" count is " << count ;
      }
  };
int item :: count
  void main()
  { item a, b, c;
    clrscr();
    a.getcount();
    b.getcount();
```

```
c.getcount();
a.getdata(100);
b.getdata(200);
c.getdata(300);
cout<<"After reading data";
a.getcount();
b.getcount();
c.getcount();
}
```

④ Inline functions can't use static variable ∵ they are defined outside the class.

STATIC MEMBER FUNCTION

1) Static function can have access to only other static members declared in the class
2) Static member function can be called with the class name instead of object name.

23 Aug 2012

```
class_name :: function_name
```

ARRAY OF OBJECTS

OBJECT AS FUNCTION ARGUMENT

1) A copy of the entire object is passed to function (call by value)
2) Only the address of object is passed to function (call by reference)

Date:

Functions returning object

```
class obj
{ int x; };
obj fun()
{ obj z=0;
  return z;
}
```

Constant Member function.

They are those function which do not alter any of the data defined in the class.

```
void display() const
{ cout <<     cout <<
}
```

27 Aug 2012.

FRIEND FUNCTION

- Accesses private members of two classes

Eg:-

```
class ABC
{
  public:
  friend void xyz(void);
};
```

1) Friend function definitation doesn't use keyword friend or scope resolution function, as it is not a member of class.

2) A function can be friend to any no. of class
3) A friend function has full access right to private members of the class
4) friend function is not in scope of class to which it has been declared as friend
5) It cannot be called using object of that class.
6) It can be invoked like a normal function
7) It usually has object as arguments

Friend function as member of class

```
class X
{ public:
    int fun()
  };

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

Making all Member functions of X as friend of Y

```
class X
{ public: };
class Y
  { public:
      friend class X;
  };
```

GLOBAL CLASSES.

class defined inside a function

- When class is defined inside a function or block, such classes are called local classes.

local classes can use global variable and static variables inside the function but cannot use automatic local variables. The global variables should be used with scope resolution.

Some restriction on local classes → (1) cannot have its own static data members and static member functions.

(2) The member functions of local class must be defined inside the class.

(3) Enclosing function can't excess private member of local class.

POINTER TO MEMBERS OF A CLASS

```
Class A
{ private:
    int m;
  public:
    void show();
};
int A::*p;
ip = &A::m;
```

If we use class objects

```
A x;
int *p = &x.m;
```

3/Sept/2012

Date:

CONSTRUCTOR

It is a special function of class, with same name as that of class and is used for initalizing data members. It is called automatically whenever a object of class is created.

Characteristics

1) Declared in public section.
2) Invoked automatically when object is created.
3) Have no return type, not even void and therefore cannot return any value.
4) Like other C++ function, they can have default arguments
5) constructor cannot be virtual.
6) We cannot refer to their addresses
7) They cannot be inherited, though a derived class can call base class constructor
8) They make Implict call to operator NEW and DELETE, when the memory is allocated

TYPE OF CONSTRUCTOR

1) Default constructor :- No argument constructor compiler creates itself.

```
⇒ class ABC
  { int a,b;
    public:
      ABC() { a=0; b=0; }
  };
```

2) Parametric Constructor :-
Constructor which accepts parameter

```
⇒ class ABC
  { int a,b;
    public:
      ABC (int t1, int t2);
  };
  ABC :: ABC (int t1, int t2)
  { a=t1; b=t2; }
  void main()
  { clrscr();
      ABC X(50,60); // Implict call.
      ABC Y(100,200);
      ABC Z; Z= ABC(10,50); // Explict call
  }
```

3) Copy Constructor :-
It initalizes value of object with another object.

```
⇒ class ABC
  { int a,b;
    public:
```

```
ABC(ABC &t); // Reference call.
ABC(int t1, int t2);
{; ABC::ABC(ABC&t) {a=t.a; b=t.b;
ABC::ABC(int t1, int t2)
{ a=t1; b=t2;}
void main()
{ ABC x(10,20);
  ABC y(x); // Implict
  ABC z; //Explict
  z=x;
}
```

DYNAMIC CONSTRUCTOR

It allocates memory to data members of class dynamically

```
class string
{ char *name;
  int length;
  public:
  string();
  string(char*s);
};
string::string()
{ length=0;
  name=new char[length+1];
}
string::string(char*s)
```

```
length = strlen(s);
name = new char(length+1);
strcpy(name, s)
}
void main()
{ string S;
string S2("KING");
char *p="MIT";
string S3(p);
}
```

DESTRUCTOR

It is special member fn of class, used to deallocate memory allocated by constru.

~ABC()

Destructor never takes any argument

Calling :→

```
string :: string()
{ delete name; }
```

CONSTANT OBJECT

```
void main()
{ const ABC X(M, N)
}
```

↳ they are not altered

const obj. and it can call const. fns - like void m() const;

18 September 2012

Date:

OPERATOR OVERLOADING

Not applicable

1) CLASS MEMBER ACCESS OPERATOR (. OR .*)
2) SCOPE RESOLUTION OPERATOR (::)
3) SIZE OPERATOR (sizeof)
4) CONDITIONAL OPERATOR (?:)

Syntax :-

```
returntype classname::operator op
                (Parameter list)
{ // body
}
```

Difference.

1) friend function will have one argument for unary operator & two argument for binary operator
2) while Member fn will have no arguments for unary operator & one argument for binary operator

The process of overloading involves 3 steps :-

1) Create a class to define user defined data types
2) Declare a operator function in public part of class (member/friend)

3) Define operator fn to implement reqd. fn.

eg :- Vector

for overloading - operator using member fn ⇒ return type operator - () → If - is unary operator

⇒ Vector operator - (arg) → If is binary operator

for friend fn.

⇒ return type operator = (arg 1) = U.O.

⇒ Vector " = (arg 1, arg 2) = B.O.

19/09/2012 ① Rules to overload operator are:

1) Only existing operators can be overloaded, new operators can't be created.
2) Overloaded operator must have atleast one operand. that is of user defined data type.
3) We can't change meaning of operator eg :- ⊕ can't be overloaded for ⊖
4) both overload operator, follow the syntax rule of original operators, they cannot be overridden
5) There are some operators which can't be overloaded. (.* or .) (sizeof) (? :) (::)

6) some operator can't be overloaded using friend function. they can be overloaded using member function (assignment (=), parantheses(()), bracket ([]), indirectional (→))

7) unary operator, overloaded by means of member function, take no explict argument & return no explict value. but those overloaded by friend fu., takes one reference argument, i.e. object of class

8) Binary operator, overloaded through a member function, takes one explict argument and those which are overloaded through friend function takes two explict argument.

9) While overloading binary operator through member fu. the left hand operand must be object of relevant class.

10) Binary arthimatic operations such as addition, multiplication, subtract, division must explict return a value. They must not attempt to

Date:

change their own argument

Overloading unary operator through member function (-).

```
class xyz
{ int a,b,c;
  public:
  xyz()
  {}
  xyz(int t1, int t2, int t3)
  { a=t1; b=t2; c=t3; }
  void operator -()
  { a=-a;
    b=-b;
    c=-c;
  }
  void display()
  { cout<<a<<b<<c; }
};
  void main()
{ clscr();
  xyz w1, w2, w3;
  w1 = xyz(-5, 6, 8)
  w2 = xyz(-4, 5, 7)
  w3 = xyz(-10, 20, 30)
  w1.display();
  w2.display();
  w3.display();
```

```
-w1, -w2, -w3;
w1.display();
w2.display();
w3.display();
}
```

Overloading unary operator through friend function

```
class xyz
{ int a, b, c;
  public:
    xyz(int t1, t2, t3)
      { a=t1; b=t2; c=t3; }
  friend void operator-(xyz &)
  void display()
    {
    }
};
Void operator-(xyz &t)
  { t.a=-t.a;
    t.b=-t.b;
    t.c=-t.c;
  }
void main()
{ xyz w1, w2, w3;
  w1 = xyz(-5, 6, -8);
  w2 = xyz(-4, 5, 7);
  w1.display
  w2.display
```

-W1, -W2, -W3;

---;

}

Overloading binary operator through member function.

```
class complex
{
 float real;
 float imag;
  public:
  complex()
   {
   }
  complex (float a, float b)
  { real = a;
     imag = b; }
  void display {cout << "real" << " + " << imag}
complex operator+(complex c)
{ complex t;
   t.real = real + c.real;
   t.imag = imag + c.imag;
   return t; }
};
void main()
{
  complex C1, C2, C3;
```

Date:

4/09/2012

```
void
    c1 = complex (10.0, 20.5);
    c2 = complex (20.0, 10.7);
    c3 = c1 + c2;
    c1.display();
    c2.display();
    c3.display();
}
```

One
and
* E
* De

cl
z

cl
{

Yw
(1)
(2)

* In
bas
of
acc
Bu
obj
be

04/09/24

Date:

INHERITANCE

One class inherit properties from another class.

* Base class - class from which properties are inherited
* Derived class - Inherit properties of base class

```
class B
{ data members;
  member functions; };
class D : accessibility mode B
{ data member;
  member functions; };
```

Three types of Accessibility mode.

1. Private
2. Protected
3. Public

* In private mode, public members of base class, becomes private members of derived class, hence they are accessible to fundions of derived class But they are not accesible to the objects of derived class, and also can't be further inherited.

* In public mode, public members of base class becomes the public member

of derived class, hence they are accessible to the function and object of the derived class.

25/09/12

PROTECTED MODE

In public mode, fn. of base class can be used by anyone. but. in protected mode, only derived class can use those fn, for others it behaves a private members.
But object of base class can access protected. members.

(ii) whenever a derived class is inherited from base class through private visibility mode. All the protected members of base class will become private member in derived class.

(iii) while for deriving properties through public visibility mode, Protected members of base class becomes protected members of derived class

(iv) for protected mode, protected member of base class becomes protected of derived class

Base Class Visibility \ Accessibility Mode	Private	Protected	Public
Private	X	X	X
Protected	Private	Protected	Protected
Public			

3/09/2012

1) Singl
Wh
one

2) Mu
wh
mi

clas

3) Her
whe
Inh

4) Mu

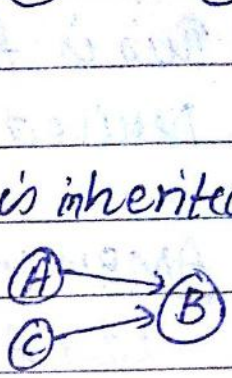
5) Hy
It
typ

28/09/2

TYPES OF INHERITANCE

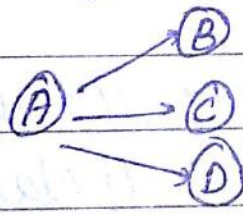
1) Single Inheritance :-
When 1 derived class is inheriting from one base class.

2) Multiple Inheritance :-
when 1 derived class is inherited from multiple base class



class derived_class : visibity base1, visibility base2 ... visibility base n;

3) Hierarchical Inheritance
When many derived class are inheriting from one single base class



4) Multilevel Inheritance

5) Hybrid Inheritance
It is mixture/combination, two or more type of inheritance

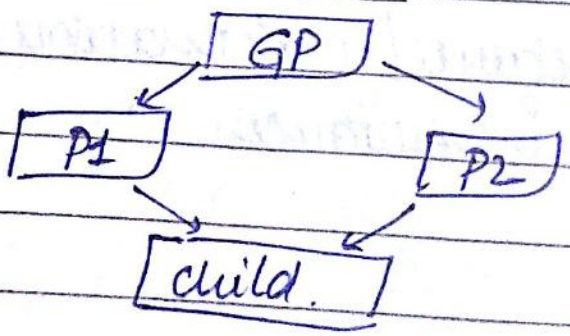
FUNCTION OVERRIDDING

Same name function, same parameter in base as well as derived class. This is function overridding. Derived class overrides base one.

AMBIGUITY RESOLUTION IN INHERITANCE

```
class A
{ public: void xyz(); };
class B
{ public: void xyz(); };
class C : public A, public B
{ public: void xyz(); };
void main()
{ C x;
  x.A::xyz(); // class A.
  x.B::xyz(); // class B
  x.xyz(); // class C
}
```

VIRTUAL BASE CLASS



Date:

child gets 2 set of parameter of GP. using P1 and P2. to prevent this we use virtual base class.

```
class Grandparent
{}
class Parent1 : virtual public Grandparent
{}
class Parent2 : public virtual Grandparent
{}
class child : public parent1, public Parent2
{}
```

ABSTRACT CLASS.

It is the one, that is not used to created the object, it is used to act as base class. It is just a design concept in programming. It acts as a base class on which other class can be built

CONSTRUCTOR IN DERIVED CLASS.

Derived class constructor (Arg 1, Arg 2, ... Arg N);
Base 1 (Arg 1);
Base 2 (Arg 2);
Base 3 (Arg N);
{ Body of derived constructor; }

When object of derived class is created than invoking of constructor of base class and derived class both takes place. Depending on situation, is the order of invoking of these constructor

But, if base class constructor has few parameter, than while created derived class object, we must pass parameters to both derived class and base class

*) In Mul
class
{}
Invokin
Here
*) In ord
tion

*) In m
class
class
class
cono
constru

*) In cas
constru
any n
clas
clas
clas
clas
1

*) for