OOP's:

Access modifier :— • Public • Private • Protected
↓
by default implemented.

Public :- can accessed in to class & outside of the class
Private - can accessed in to the class but not outside class
Protected similar to private, but it can access by the child class and it is also accessible in the class.

Getter / Setter :— In case of private we can access the methods or variables with this methods.

Structure padding :— CPU access 1 word at a time. depend on the CPU architecture.
(32-bit) 4-byte. & 64-bit processor - 8-byte

Dynamic and static object creation :—

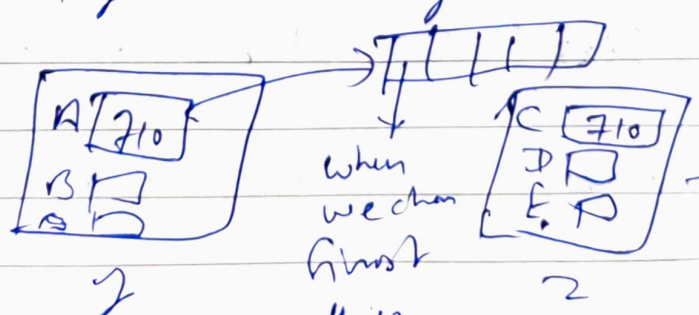
Constructor :— default constructor is then - like - object class
⇒ we can create user defined constructor

Parameterized constructor :— — (This keyword - store add of object)

Copy constructor :- (Pass by reference) —
default copy constructor removed once you create your own copy constructor.

Shallow and deep copy :—

• default constructor are shallow copy - it copy the only address of the location



— Shallow copy.

if reflect to the 2nd automatically.

In case of deep copy we create different character array and ~~create~~ copy it in to the new.

Copy assignment operator:— $a = b$ object.

Destructor:— use for deallocate the memory.
default destructor is present until we don not create any destructor by itself.

⇒ Destructor name is same as class name
⇒ No return type.
⇒ No input parameter.

In case of destructor (~) sign is used to differentiate b/w both constructor & destructor.

⇒ for statically created object destructor automatically called.
but for dynamically created objects we have ~~to~~ call it manually - (delete object).

Constant Keyword. (can in form creation, data members, object creation).
⇒ const type variable -
const int a;

Initilization list:— Used to initilize the data member of class.

e.g. - class test { int a; int b;
public:
test (int i, int j) : a(i), b(j) { const } {
body.

Static keyword :- it is use to create those data member which belongs to class and for access those ~~data~~ members we don't ~~to~~ need to create an object.

initialization -

→ Scope resolution obeying

data type classname :: datamember=value;

Static function :- no need of creating object to call static function.

⇒ It is just call by the name.

- It have not this keyword.

- Static function can access only static member.

(Data or information hiding) **Encapsulation** :- Wraping up data members & function

Fully encapsulated class → All the data member marked private.

Adv: - (1) Data hide → Security

(2) If we want we can make class read-only.

(·) Code reusability

(·) Unit testing.

Inheritance :- Inherting the member of parent class to child.

Syntax - class parent-class {

};

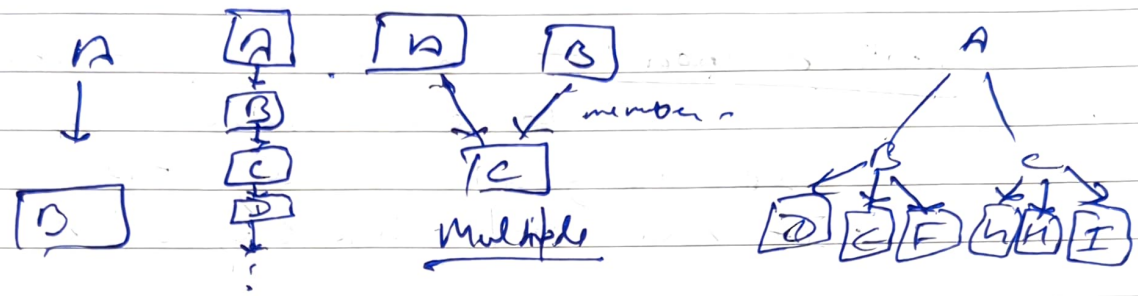
class child class : access modifier Parant_class {

};

Base class member Access specifier	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	N/A	N/A	N/A

Type—

1) Single In ⇒ Child class inherits properties (members) of Parent.
2) Multilevel → A → B → C
3) Multiple
4) Hybrid ⇒ Combination more than one type of inheritance
5) Hierarchical



Single Multilevel Hierarchical.

Inheritance Ambiguity! — e-2

```
class A {
  Void abc {
      }
}
class B {
  Void abc {
      }
}

class C : Public A, Public B {
          Public
      }
int main() {
    C obj;
}   obj.abc();  (Ambiguity! whether A or B)
```

For resolving ambiguity we use scope resolution operator :-

eg :-

→ inherited classes

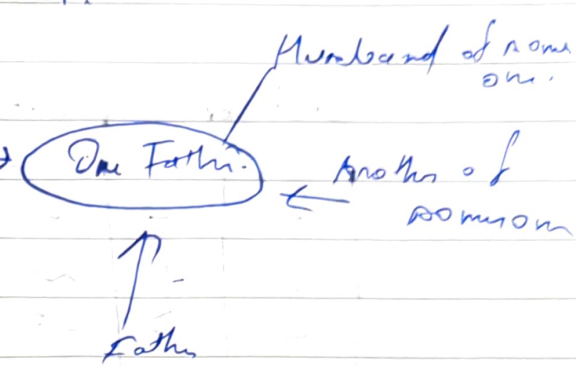
obj . A :: fun ();

obj . B :: Fun ();

or member.

Polymorphism :-

to many forms.



Types →

Static Polymorphism → (1) Compile time Polymorphism —
(i) Function Overloading
(ii) Operator Overloading.

We tell that, this entity, exist in how many forms on compile time.

i) Function overloading :- Writing same functions in multiple forms.

⇒ return type fun_name (Parameters) { }

for function overloading we have to must change parameters (it may be change in the parameter's data type.)

eg. int fun (int a) { }
int fun (char a) { } → it work
change in this is not work (same).

⇒ It ~~gives the priority to the same class fun~~ (which present into that class)

Operator Overloading :→ Defining the functions of operator in such a way that they perform desire output.

List of operator that can be overloaded in C++;

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new[]	delete	delete[]

List of operator that can not be overloaded :—

::	.*	.	?:

Syntax –

→ this → current object (1st operand).
2nd operand

return type operator + ( ) ;
{
}

eg – void operator + (B &ob) { int val1 = this->a;
int val2 = ob.a;
int m = val1;
}

main { B ob1, ob2;
ob1.a = 4;
ob2.a = 7;
}

2) Runtime Polymorphism (Dynamic Poly...):—

(i) Method Overriding:—

Conditions :—

i) Method of parent class & the method of child class must have same name & same parameters.

(ii) It is possible through inheritance only.

It gives the priority of the method of the (class of object).

Abstraction :— Implementation hiding & showing only essential information.

Adv —

You can only make changes to your data or function & no one else can.

- It makes the application secure by not allowing any one else to see the background details
- Increases the reusability of the code
- Avoids duplication of your code.

Virtual function :—

Point to drive class —

```
class Bam {
    int h;
  void func (   )
    {

    }
}
class child: Public Bam {
        int  h;
      void fun( ) { }
}
int   main( )
{
  ~~Bam class~~
    Bam *bamclass pointer;
    Bam obj;
    child obj, child;
      bamclass pointer = &obj; → Pointing bam
                                 class pointer to derived
                                 class.
}
      bamclass pointer → fun( );
```

→ it run the bam class function, how ever it point to the child class.

→ If we make child class pointer and point to child class object then it run the child class

⇒ For solving this problem which is base class pointer point to the child class object but it run the method of base class

⇒ We make the base class function virtual. So it run the child class or (pointed class) method.

Virtual function:

```
virtual int fun ( ) {
```

Pure virtual function:—

```
virtual void display ( ) = 0; // do nothing function
                                 or pure virtual function
```

Abstract base class:— Those classes who made by derived in to another class and then (Child) function by run by it was in pure virtual function.

⇒ Abstract class are not use to create objects. (It function are created to override (in child classes) which are pure virtual function.

⇒ In abstract class, at least one pure virtual function are present.

⇒ In pure Abstract / Interface class all the fun^n are pure virtual fun^n.

Basic Concept in OOP :-

1) Classes :- Basic template for creating objects.
2) Objects :- Basic run time entities (memory allocation)
3) Data Abstraction & encapsulation - Wrapping data and functions into single unit.
4) Inheritance :- Properties of one class can be inherited into others.
5) Polymorphism - ability to take more than one forms.
6) Data binding :- Code, which will execute is not known until the program run.
7) Message Passing - Object. message (information) call format.

Benefits of OOP

- Better code reusability using objects and inheritance.
- Principal of data hiding helps build secure systems.
- Multiple object can coexist without any interference.
- Software complexity can be easily managed.

Pointer — use to store address of variables
64-bit system have 8-bit pointer size.
32 - " " " " " 4 - " " " "

Pointer addition —

```
int *p;        // p=4
p = p+1        // p = p+1 => p = 8
```

↳ 1 block

```
int a[10];
```

a ⟺ &a[0]

*(a+1) → a[1]

double pointer :— → use to create 2D Array

```
int *p;        → address of p
int **pp;      → address of pointer
int x;
p = &x;
pp = &p;
pp → &p
*pp ⇒ &p
**pp ⇒ x;
```

- Compilation error.
- Memory limitation.
- Run time error.
  (i) illegal access of memory. (Segmentation fault)
  (ii) Undefined behavior (%) in CPP not allow (not negative index) allow in CPP
- Time Limit exceed (calculate only execution time) (codeforces / std cout ...) ⇒ increase compilation time

Calculation of execution time :—

Previous Page ⟸

a.cpp [a.opp] a.out → input [execution] o/p

compilation time

execution time or Runtime

First need next page →

{ { } } → it can access { but not access block variable (but int not available out side)

⇒ **No of iterations per second:—**

if your computer to compute - look on online websites — in 1 second 10^7 to 10^8 iteration runs

Ex - 10^7 iteration → 1 second
10^8 " → 10 sec
10^9 " → 100 sec
10^{10} " → 1,000 sec
10^{11} " → 1,0000 sec

$2^a = N$

$\log_2(2^a) = \log_2(N)$

$a \log_2 2 = \log_2 N$

$a = \log_2(N)$

$O(\log(n))$

⇒ Sum of N over all test cases is $< 10^7$ valid to run.

Variable sharing → [Global variable ✗ (Reference variable)
Inline function (good for online coding)
Default argument (←)

const int var = 5;

⇒ 2D Array →

```
int ** arr = new int * [n]
for (int i=0; i<n; i++)
{
  arr[i] = new int [n];
}
```

include <iostream>
↳ preprocessor directives

macro ⇒ # define PI 3.14 (int)

{ } → it also

{
it can use global block variable
}
it can not use it nested block variable.
}

{ } it can not create another block variable

Global variable not declare into Stack.