CAP444
OBJECT ORIENTED PROGRAMMING USING C++

Unit 1

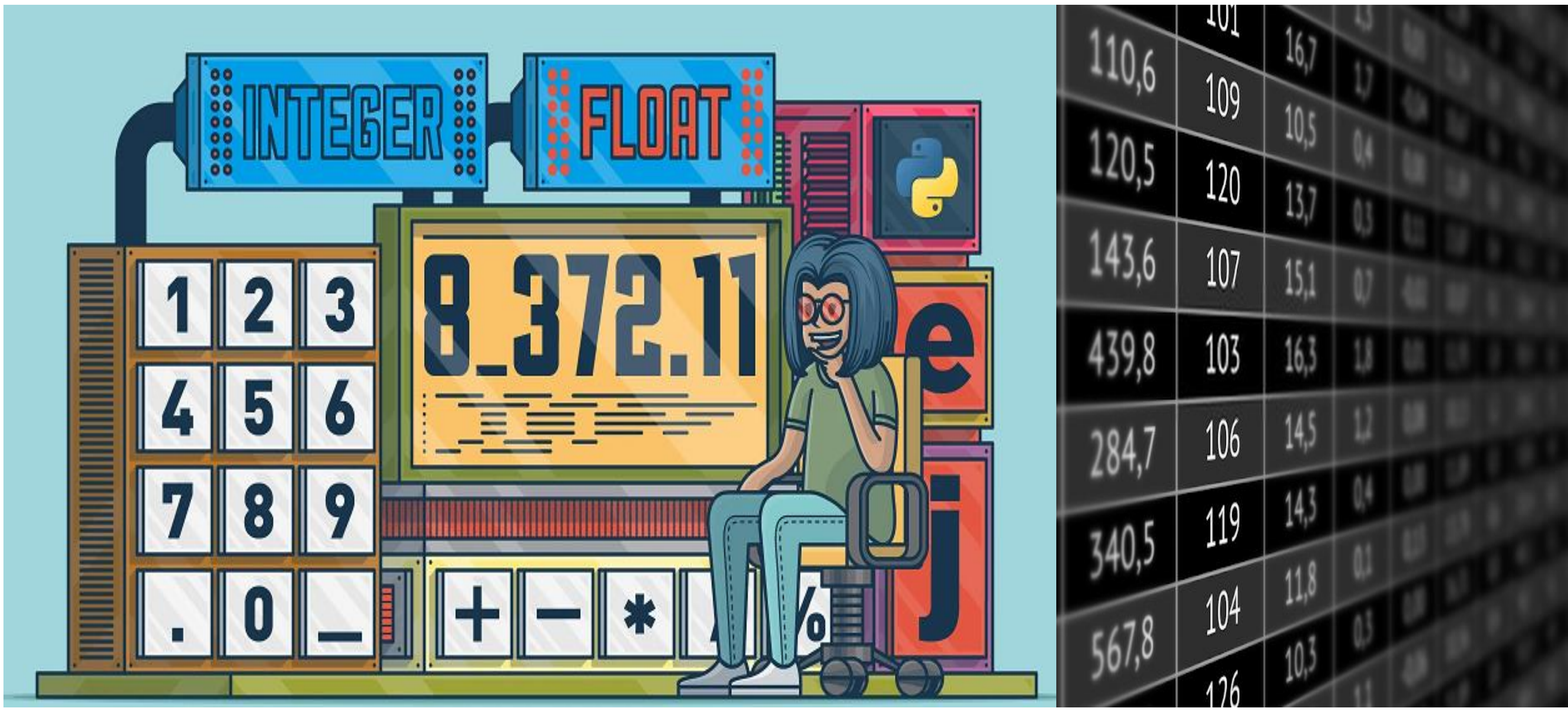
Created By:
Kumar Vishal
(SCA), LPU

Topics Covered

Principles of OOP :

- basic concepts of object oriented programming,
- object oriented languages,
- classes and objects,
- access specifiers,
- constructors: types of constructors,
- multiple constructor in a class,
- destructors,
- functions overloading,
- friend function,
- inheritance: types of inheritance

Basic concepts of object oriented programming



Data types

User-define type

- Class
- Structure
- Union
- Enum
- Typedef

Built-in-type

- Integer
- Character
- Boolean
- Floating Point
- Double
- Void
- Wide Character

Derived type

- Function
- Array
- Pointer
- Reference

Primitive Data Types: These data types are built-in or predefined data types and used to declare variables.

Primitive data types available in C++ are:

- Integer(int)
- Character(char)
- Boolean(bool)
- Floating Point(float)
- Double Floating Point(double)
- Valueless or Void(void)
- Wide Character(wchar_t)

Wide Character: Wide character data type is also a character data type but this data type has size greater than the normal 8-bit datatype.

Derived Data Types: The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types.

These are:

- Function
- Array
- Pointer
- Reference

Abstract or User-Defined Data Types: These data types are defined by user itself.

- Class
- Structure
- Union
- Enumeration or Enum
- Typedef

Data type	Size(in byte)	Range
char	1 =8 bits (2^8)	-128 to 127 or 0 to 255
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	4=32 bits (2^{32})	-2,147,483,648 to 2,147,483,647
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
float	4	
double	8	
long double	12	

We can display the size of all the data types by using the sizeof() operator

Memory representation

1 Byte= 8 Bits

128	64	32	16	8	4	2	1
0	1	0	0	0	0	0	1

A(65)=01000001

char ch=65;

Or

char ch='A'

Char is occupying 1 Byte memory

How to find out range?

For Signed data types:

1.) calculate total number of bits

2.) Calculate -2^(n-1) for minimum range

3.) Calculate (2^(n-1))-1 for maximum range

Unsigned Data Types:

1.)Find number of bits

2.)minimum range is always zero for unsigned data type

3.)for maximum range calculate 2^n-1

Example:

Char : 1 byte: 8 bits=n

Signed: $-2^{(8-1)}$ to $(2^{(8-1)})-1$

=-128 to 127

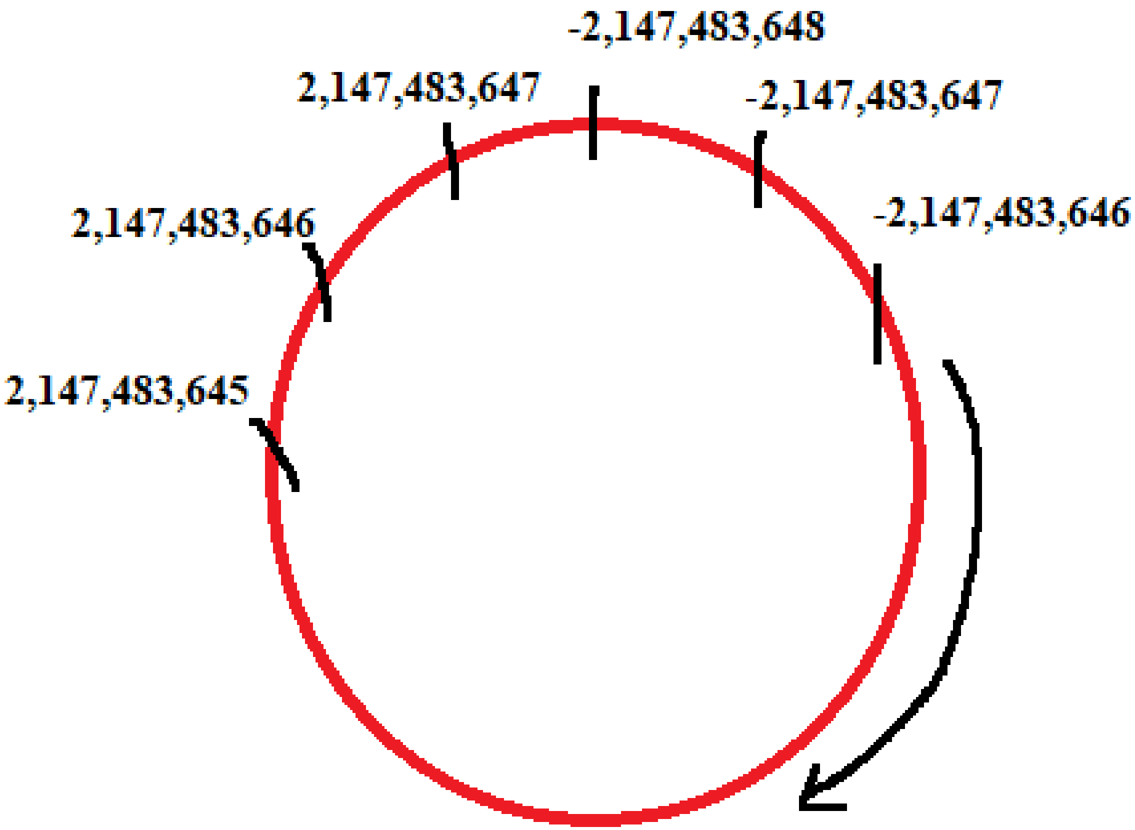
Unsigned:

0 to $2^{(8)}-1$

=0 to 255

Exceeding range...?

intger range: -2,147,483,648 to 2,147,483,647



What will be output?

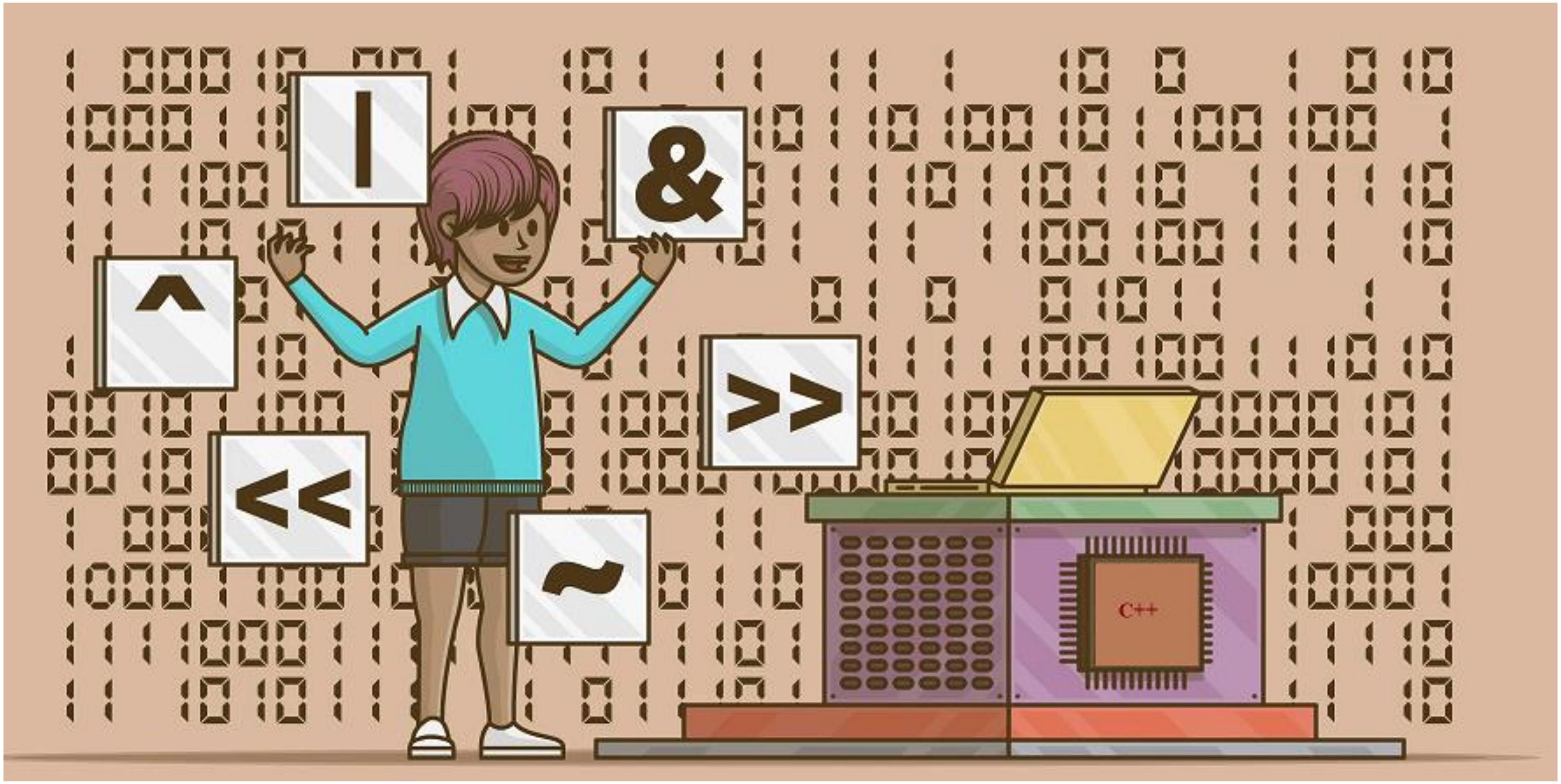
```
#include <iostream>
using namespace std;
int main()
{
    int num=2147483648;
    cout <<num<< endl;
    return 0;
}
```

A. 2147483648
B. - 2147483648
C. Error
D. None

Data type modifiers are:

- Signed
- Unsigned
- Short
- Long

Today we are going to learn about……?



Operators

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators
- Increment /decrement operators
- insertion operator/ extraction operator

Arithmetic operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Bitwise operators

Operator	Description
&	AND Operator
|	OR Operator
^	XOR Operator
~	Ones Complement Operator
<<	Left Shift Operator
>>	Right Shift Operator

AND Operator (&)

If both side bit is on result will be On

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Steps to solve:-

- **a = 12 (find binary form:1100)**
- **b = 25 (find binary form:11001)**

How to find Binary:

64	32	16	8	4	2	1	
		0	1	1	0	0	12
		1	1	0	0	1	25
			1	0	0	0	8

a & b=

01100 (12)

11001   (25)

01000   (8) Ans.

What will be output?

```
#include <iostream>

using namespace std;

int main()
{
   int a=20;
   int b=25;
   cout<<(a&b);
return 0;
}
```

A. 15
B. 16
C. 20

OR Operator (|)

If any side bit is on result will be On

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

Steps to solve:-

- **a = 12 (find binary form:1100)**
- **b = 25 (find binary form:11001)**

How to find Binary:

64	32	16	8	4	2	1	
		0	1	1	0	0	12
		1	1	0	0	1	25
		1	1	1	0	1	29

a | b=

01100 (12)

11001 (25)

11101 (29) Ans.

What will be output?

```
#include <iostream>

using namespace std;

int main()
{
    int a=20;
    int b=15;
    cout<<(a|b);
return 0;
}
```

A. 31
B. 32
C. 22
D. 32

XOR Operator (^)

If both side bit is opposite result will be On

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Steps to solve:-

- **a = 12 (find binary form:1100)**
- **b = 25 (find binary form:11001)**

How to find Binary:

64	32	16	8	4	2	1	
		0	1	1	0	0	12
		1	1	0	0	1	25
		1	0	1	0	1	21

a ^ b=

01100 (12)

11001 (25)

10101 (21) Ans.

Left Shift Operator(<<)

a=10 (1010)

a<<1

1010.0

10100(20) Ans.

a<<2

1010.00

101000(40) Ans.

Right Shift Operator(>>)

a=10 (1010)

a>>1

1010.

101(5) Ans.

a>>2

1010.

10(2) Ans.

What will be output?

```
#include <iostream>
using namespace std;
int main()
{
  int a=15;
  cout<<(a>>1);
return 0;
}
```

Options:

A. 5
B. 6
C. 7
D. 8

Increment/Decrement Operator

++: Increment

++x

--: Decrement

--x

```
int main()
{
    int a=10;
    a++;
    cout<<a;
    return 0;
}
```

What will be output?

```
#include <iostream>

using namespace std;

int main()
{
   int a=10;
   int c=a++;
   cout<<c;

   return 0;
}
```

```
#include <iostream>
using namespace std;

int main()
{
   int a=10;
   int c=++a;
   cout<<c;

   return 0;
}
```

What will be output?

```
#include<iostream>
 using namespace std;

 int main()
 {
    int x = 5, y = 5, z;
    x = ++x; y = --y;
    z = x++ + y--;
    cout << z;
    return 0;

 }
```

insertion operator(<<):

The cout is used in conjunction with stream insertion operator (<<) to display the output on a console

extraction operator (>>):

The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

Control structure

- Conditional structure: if and else
- Selective structure: switch case
- Iteration structures (loops): while, do while, for
- Jump statements: break, continue, goto

if and else

```
if(condition)
{
//Statements(execute when condition true)
}
else
{
//Statements(execute when condition false)
}
```

Switch ...case

For menu options:

```
switch(choice)
{
case 1:
break;
default:
}
```

What will be output?

```
#include <iostream>
using namespace std;
int main()
{
    int a=10;
   switch(a)
   {
      case 10:
      cout<<“Hi";
      case 11:
      cout<<“Hello";
   }
    return 0;
}
```

A. Hi
B. Hello
C. HiHello
D. None

While loop

The syntax of a while loop in C++ is −

```
while(condition)
 {
   statement(s);
}
```

```
#include <iostream>
using namespace std;

int main ()
 {
     int a = 10;
     while( a < 20 )
     {
     cout<< a << endl;
     a++;
       }
   return 0;
}
```

Do While loop: at least one time will be execute

The syntax of a do while loop in C++ is −

```
do {
   statement(s);
}
while( condition );
```

```
#include <iostream>
using namespace std;

int main ()
 {
      int a = 10;
  do
      {
       cout<< a << endl;
      a++;
     } while( a > 20 );
   return 0;
}
```

For loop:

The syntax of a for loop in C++ is –

```
for ( initialization; condition; increment )
{
   statement(s);
}
```

Jump statements: break, continue, goto

break: It breaks the current flow of the program at the given condition.

continue: It continues the current flow of the program and skips the remaining code at specified condition.

goto: It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label.

What will be output?

```
#include <iostream>
using namespace std;
int main()
{
   for(int i=1;i<=5;i++)
   {
      if(i==3)
         continue;
      cout<<i;
   }
   return 0;
}
```

A. 12345
B. 123
C. 1245
D. None

go to Jumping Statement

```
#include <iostream>
 using namespace std;
 int main()
 {
 ineligible:
       cout<<"You are not eligible to vote!\n";
       cout<<"Enter your age:\n";
       int age;
       cin>>age;
       if (age < 18){
              goto ineligible;
       }
       else
       {
              cout<<"You are eligible to vote!";
       }
 }
```

Class

Class is a collection of similar types of objects.

For example: Fruits is class of mango, apple , orange etc.

Fruits


micro SD 16GB
BP-6M
NOKIA
M2
8GB

Class

Class is a collection of data members and member functions.

Example:

```
class Employee{
//data members
// member functions
}
```

Employee
employeeId employeeName
getDetails() setDetails()

Object

Object is an instance of a Class.

```
class Employee
{
        int employeeId;
        char employeeName[20];
        public:
        void getDetails(){}
        void setDetails(){}
};
int main()
{
Employee e1; // e1 is a object
return 0;
}
```

In this code which option is correct?

```
class student
{
public:
int regno;
void getStudentDetails();
}
```

A: regno is data member

B. getStudentDetails() is member function

C. above both options are correct

access specifier

- private
- public
- protected
- By default, all members of a class are private if you don't specify an access specifier.

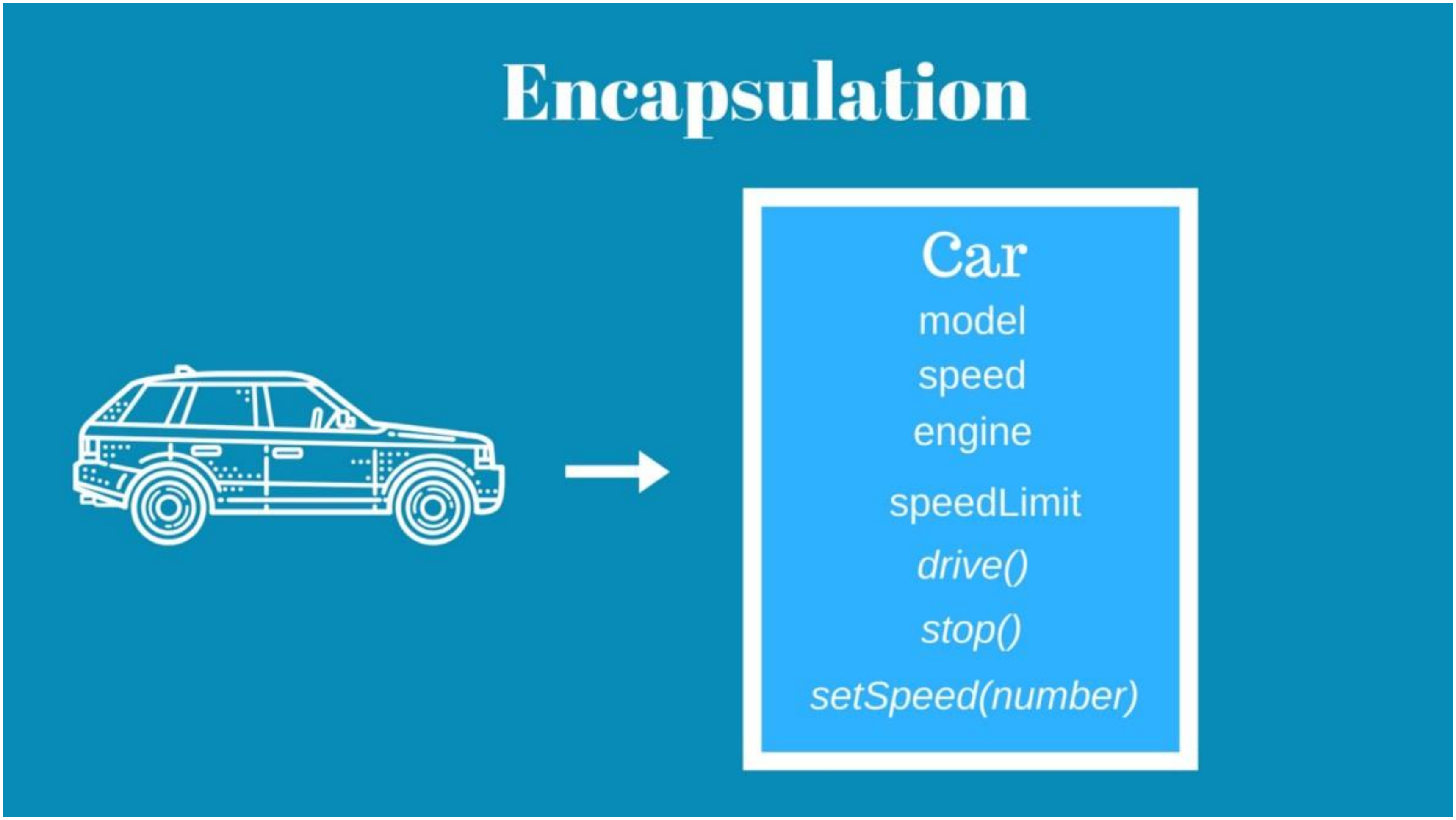
Specifiers	within same class	in derived class	outside the class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

Select correct option

```
class Employee
{
        int employeeId;
        char employeeName[20];
        public:
        void getDetails(){}
        void setDetails(){}
};
int main()
{
Employee e1;
e1. employeeId=1234
cout<< e1. employeeId;
return 0;
}
```

A. 1234
B. 0
C. Compilation error
D. Run time error

How to achieve encapsulation features in C++?



How to achieve encapsulation features in C++?

- Define private data members in a class and
- Define public set and get accessors functions which can be used to access these private data members.

Why Encapsulation?

Increased security of data

Constructor and destructor

Constructors: types of constructors

- A special method which is used to initialize the object
- It is automatically called when an object of a class is created.
- it has the same name as the class name.
- it is always public
- it does not have any return type

Types of constructor:

- Default constructor
- Parameterized constructor
- Copy constructor


49:-
99:-
SELFIEPINNE
99:-


HTC
One
BlackBerry
Internet Devices
Jetpack® 4G LTE
Mobile Hotspot
MiFi® 5510L
Touch to Explore
Choose a Device

Default constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

```
class Employee
 {
  public:
       Employee()
       {
          cout<<"Default Constructor"<<endl;
       }
};
```

Parameterized constructor

A constructor which has parameters is called parameterized constructor.

```
class Employee
 {
    int empId;
    string empName;
   public:
       Employee(int id, string name)
       {
          empId=id;
          empName=name;
       }
};
```

Copy Constructor

Copy Constructor is a type of constructor which is used to create a copy of an existing object of a class.

Syntax:

class-_name**(**class_name **& object_name)**

{

}

To call this: class_name obj1(arguments);

class_name obj2 = obj1;

Destructor

- Destructor is a special member function which destructs or deletes an object.
- A destructor is called automatically when object goes out of scope.
- Destructors have same name as the class preceded by a tilde (~)
- Destructor should not have any parameter
- There can only one destructor in a class
- When a class contains a pointer to memory allocated in class, we should write a destructor to release memory

Which option is correct for defining the destructor?

Option1:

```
~mobile()
    {
 cout<<"destructor called"<<endl;
    }
```

Option2:

```
 ~mobile( string str)
    {
 cout<<"destructor called"<<endl;
    }
```

A. Option1 is correct

B. Option2 is correct

C. Both option is correct

❖ Functions overloading

- same function name but different parameters.
- same function name with different signature
- example of polymorphism(**compile time)**
- overloaded functions should be there in same scope.

What will be output?

```
#include <iostream>
    using namespace std;
    void print(int i)
    {
        cout << i;
    }
    void print(double  f)
    {
        cout << f;
    }
    int main()
    {
        print(5);
        print(500.263);
        return 0;
    }
```

A) 5500.263

B) 500.2635

C) 500.263

Friend function

- It can access all private and protected member of a class
- It can be access without object of the class
- It can define out side of the class scope

Rule:

Prototypes of friend function must be declare inside the class

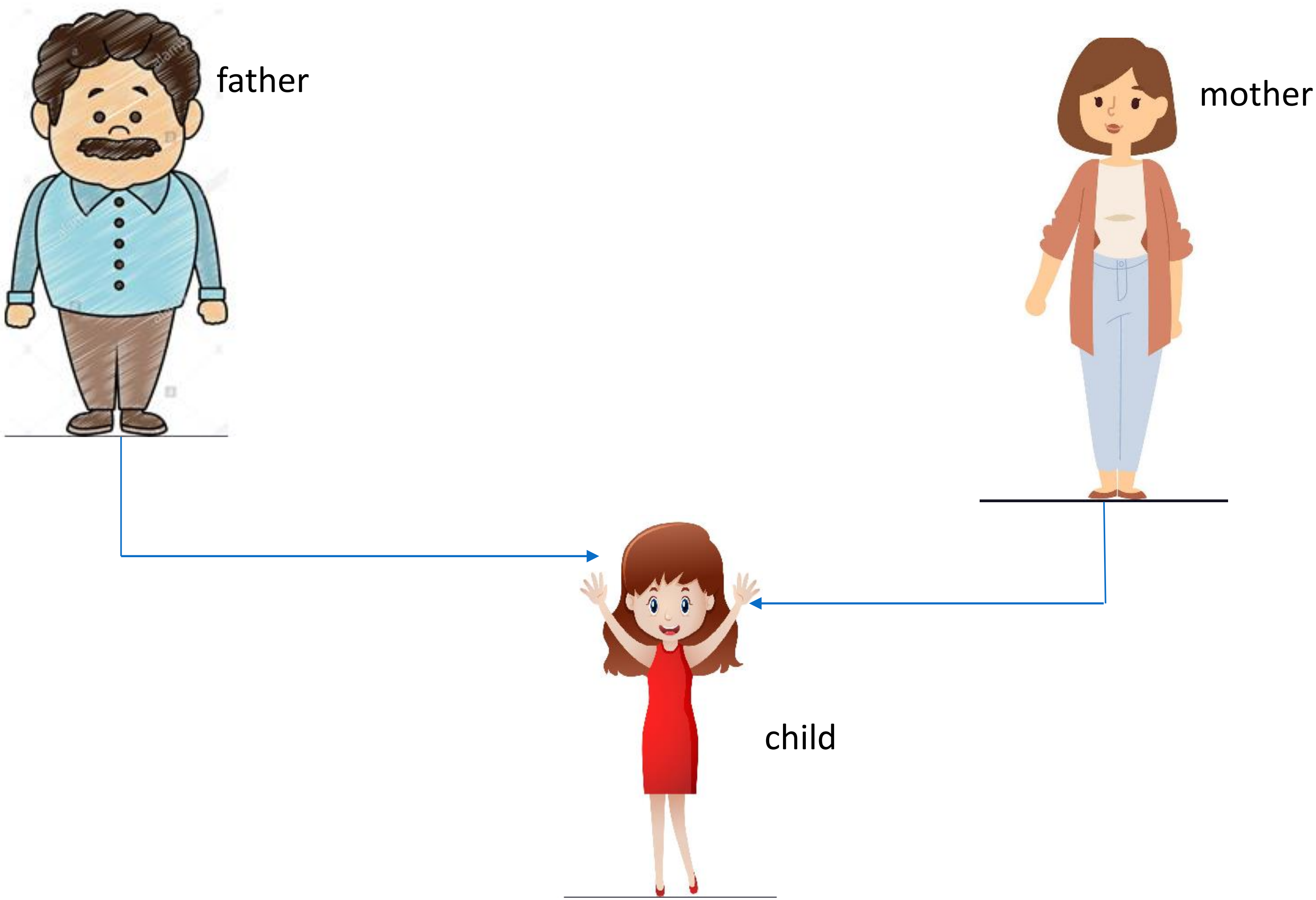
It can be declared either in the private or the public part.

Example : friend function

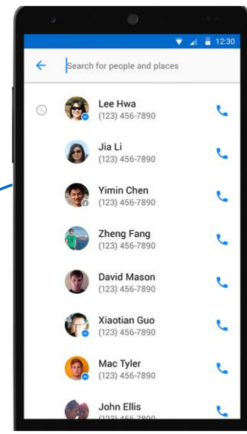
```
#include <iostream>
using namespace std;
class A
{

private:
   int x;
public:
   A()
   {
      x=10;
   }
private:
   friend void newfriend(A &a);
};
```

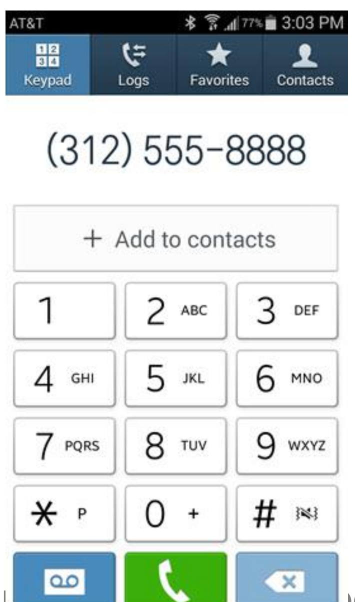
```
void newfriend(A &a)
{
   a.x=20;
   cout<<a.x;
}
int main()
{
A a1;
newfriend(a1);
return 0;
}
```


father
mother
child

More examples of inheritance:



Contact List

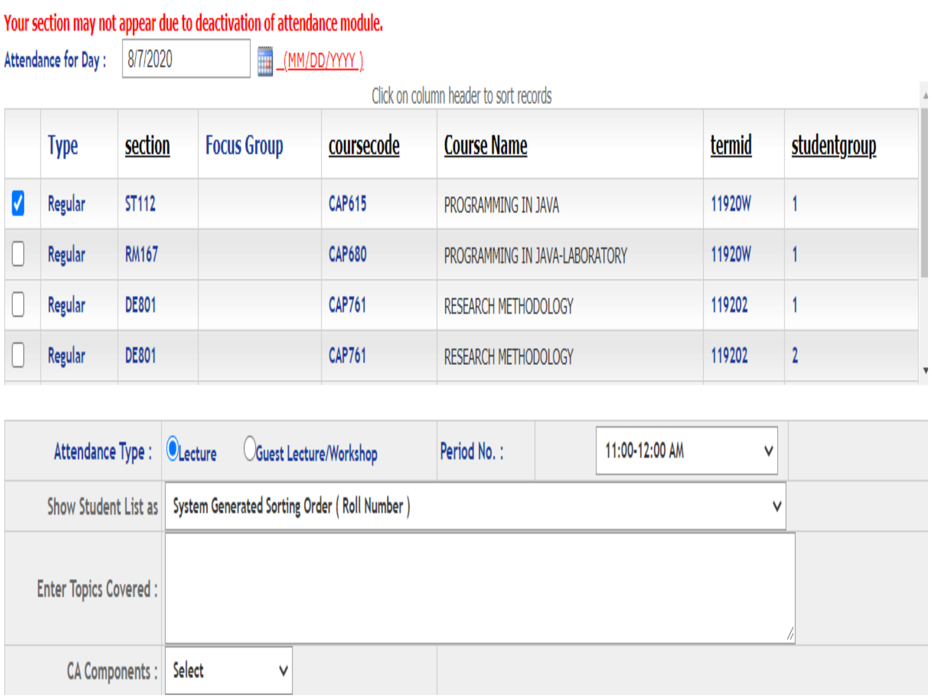


More examples of inheritance:

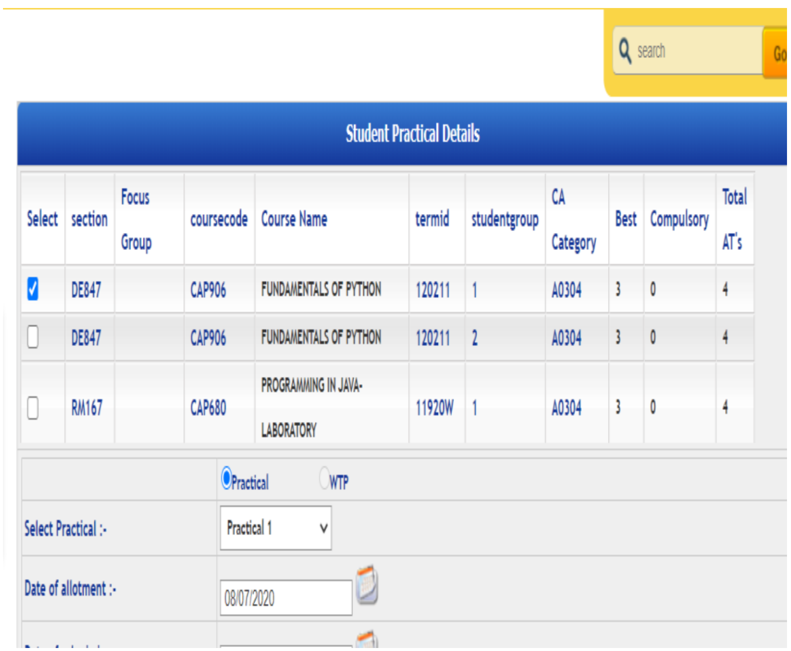
Students

RegistrationNumber	Stream
11601434	CAP - MCA
11604082	CAP - MCA
11616750	CAP - MCA
11907267	CAP - MCA
11908953	CAP - MCA
11909448	CAP - MCA
11909861	CAP - MCA
11910431	CAP - MCA
11910924	CAP - MCA
11911831	CAP - MCA
11814205	CAP - MCA
11915821	CAP - MCA
11915858	CAP - MCA
11916426	CAP - MCA
11900044	CAP - MCA
11901413	CAP - MCA
11901220	CAP - MCA

Students in Attendance Module

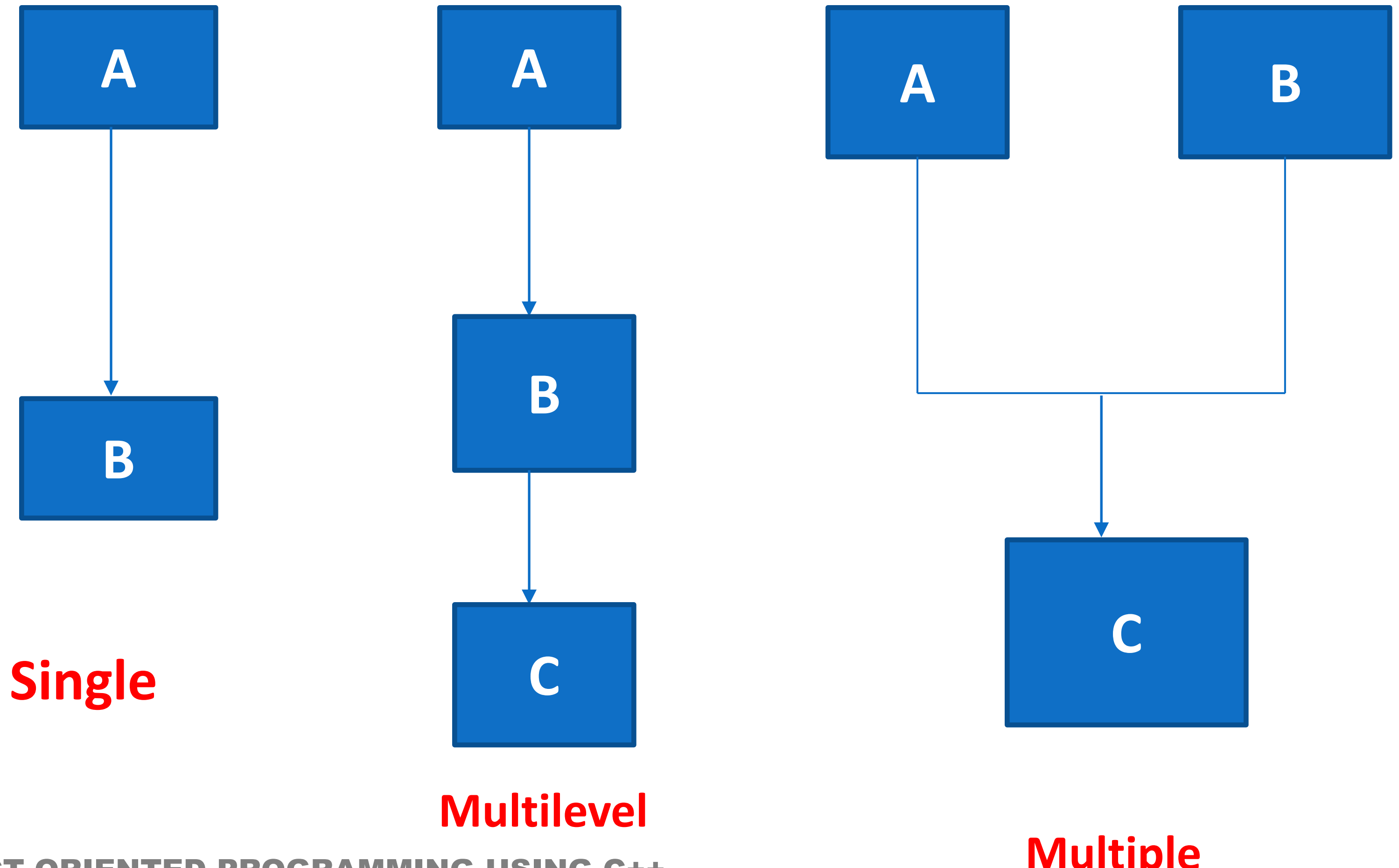


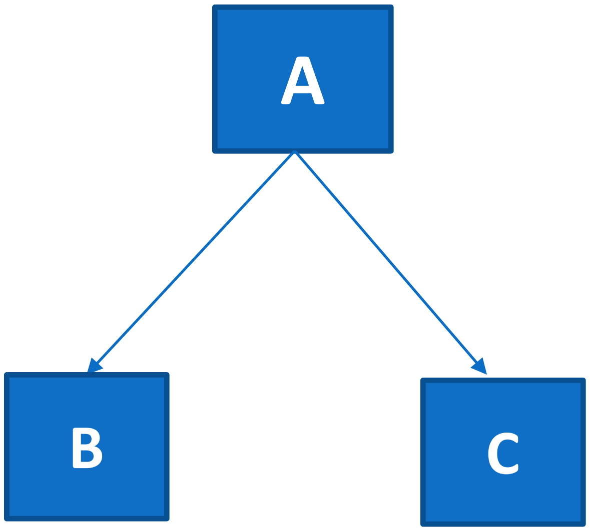
Same Students in CA Module



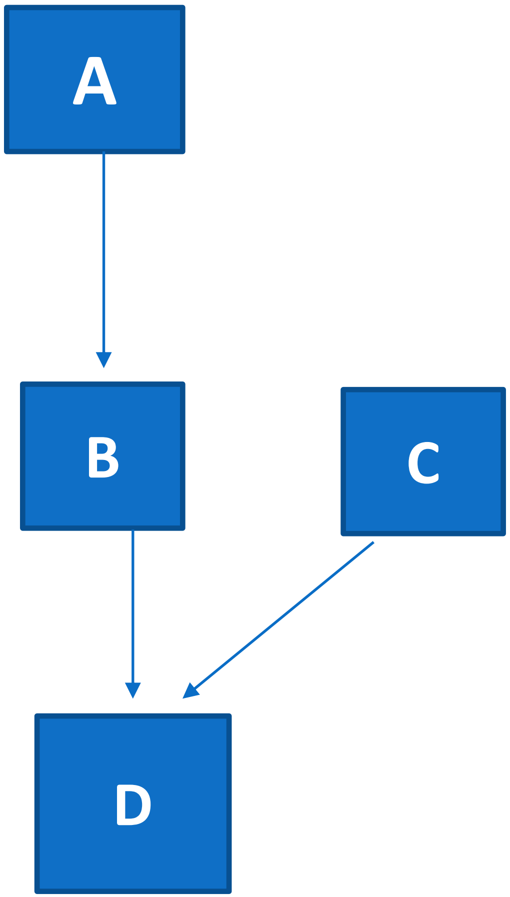
Inheritance: types of inheritance

- One class can hire properties from other class
- Advantages: Reusability





Hierarchical Inheritance



Hybrid Inheritance

Which among the following best defines single level inheritance?

A) A class inheriting a derived class

B) A class inheriting a base class

C) A class inheriting a nested class

D) A class which gets inherited by 2 classes

Single inheritance:

Syntax:

```
class derive_class_name : access_mode base_class_name
{
  //body of derive_class
};
```

Multiple Inheritance:

```
class derive_class_name : access_mode
base_class1, access_mode base_class2, ....
{
  //body of derive_class
};
```

Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

```
class base_class
{
//base class members (x, y)
};
class derive_class : access_Specifier base_class
{
//base class members (x, y)
//derive class members (a,b)
};
```

Which among the following is correct for a hierarchical inheritance?

a) Two base classes can be used to be derived into one single class

b) Two or more classes can be derived into one class

c) One base class can be derived into other two derived classes or more

d) One base class can be derived into only 2 classes

Any Query?