```
class Matrix{

Matrix(int r,int c){
arr[0] = new int(c);
arr[1] = new int(c);
}
}
```

```
Matrix m1(2,2);
```

```
Employee arr[size];
// you decide what type of arry to create
new Employee[size];

OR

new Employee*[size];
```

```
class Test{

int num1;
static int num2;

}

Test t1;
```

```
class Employee{
int empid;
double salaray;
};

class Student{
int rollno;
double marks;
};
```

```
class Person{
string name;
string email;
}
```

```
class Date{
int day,month,year;
};

class Time{
int hrs,mins;
};
```

has-a relationship

Association

car has-a engine
Room has-a Wall
Room has-a Window
Student has-a Date
Employee has-a Date

Dependency -> engine

Dependent-> Car

```
// depencency
class Engine{
// data members

};

class Car{
// data members;

// object
Engine e ;// Association
};
```

Car has-a Engine

Types of Association
1. Composition
2. Aggegration

1. Composition
Tight Coupling between two entities

```
class Employee{
int empid;
Date doj;//Composition
Car *c; // Aggegration

}
```

```
class Date{

}
```

Employee has-a DOJ

Composition

2. Aggegration
Loose Coupling between two entities

```
class Car{

}
```

Employee has-a Car

Aggegration

```
class Date{
int day;
int month;
int year;
}
```

12 bytes

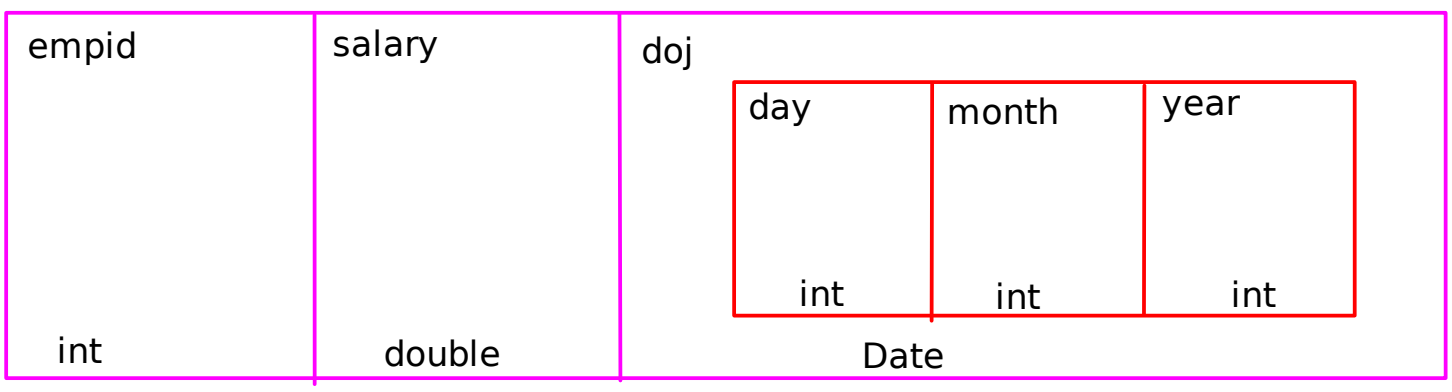
```
class Car{
char name[15];
char color[10];
}
```

25 bytes

```
Employee{
int id;
double salary;
Date doj;
Car *cptr;
}
```

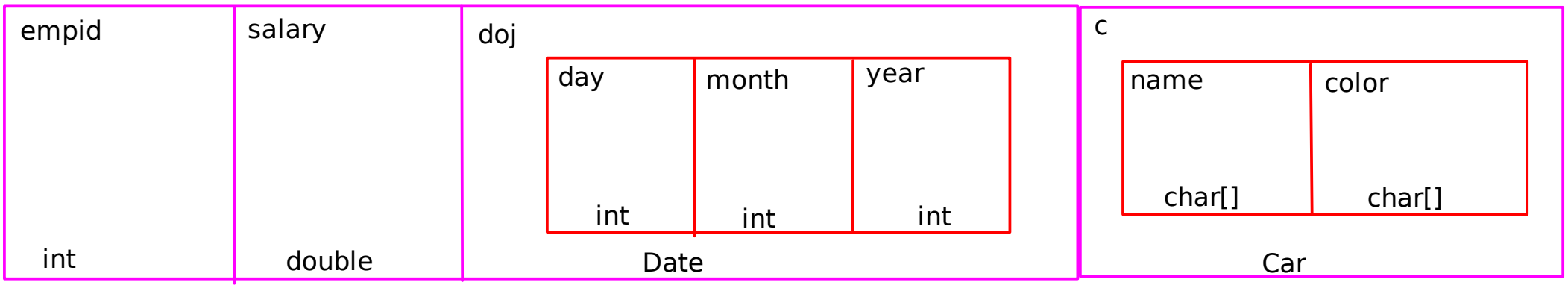
```
Car c;
```

Employee e



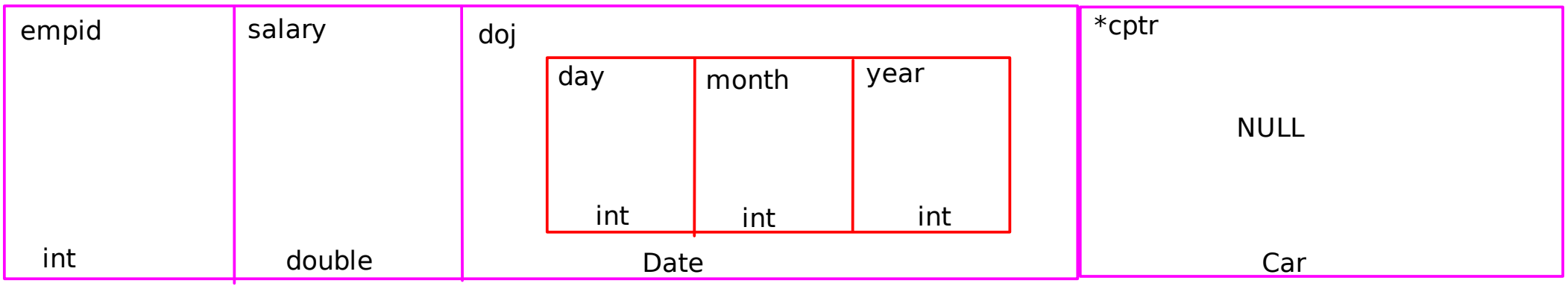
24 bytes

Employee e



24 bytes + 25 bytes of car = 49 bytes

Employee e

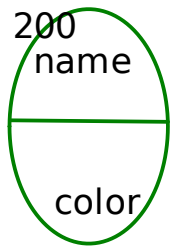


24 bytes + 4 bytes of Car * = 28 bytes

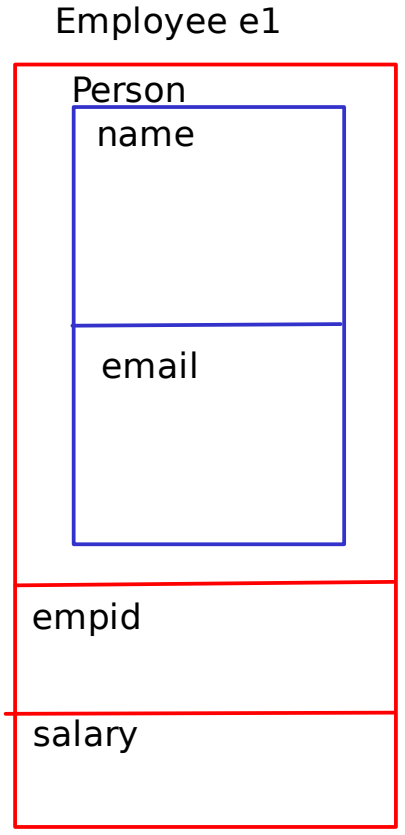
is-a RelationShip
Inheritance

Apple is-a Fruit
Employee is-a Person

```
class Person // Parent/Base
{
string name;
string email;

}

class Employee : Person // child /Derived
{
int empid;
double salary;

}
```

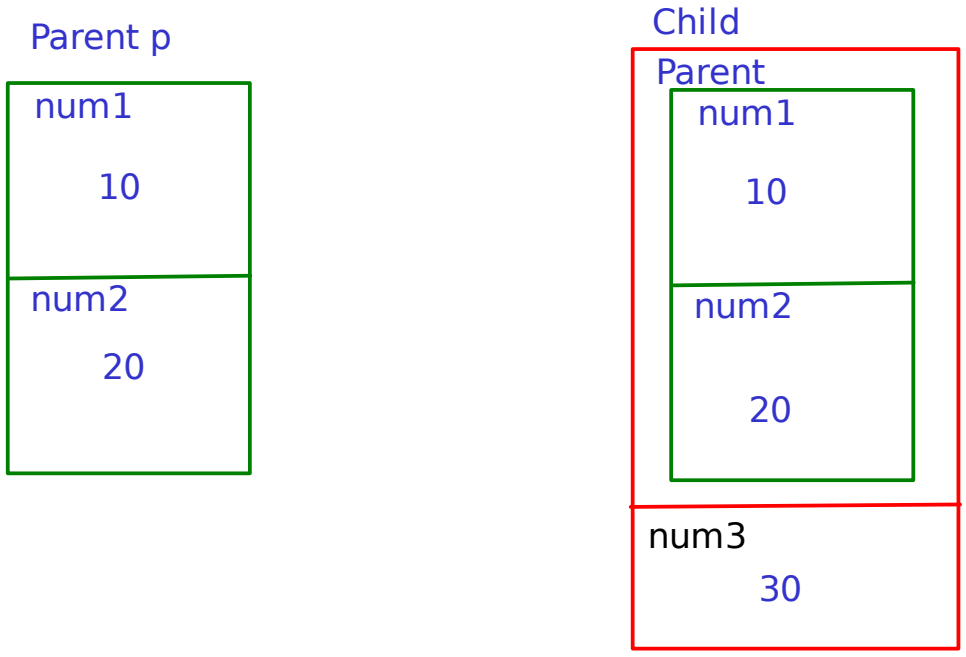


new Car();



```
class Parent{
int num1;
int num2;
}

class Child:Parent{
int num3;
}
```

Child c1; //12 bytes

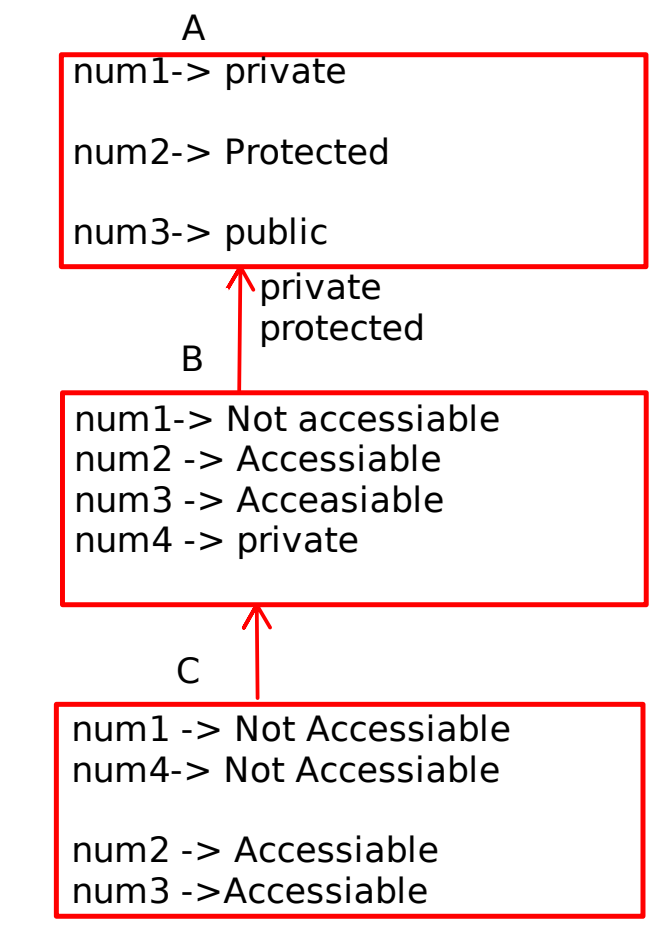


If we use private, public or protected keywords for our data members then they are called as access specifiers

If these same keywords are used at the time of inheritance then we call it as Mode of inheritance

Default Mode of inheritance is Private

```
class Employee{
string name; // Association

}
```




```
Base{
All members of Base class are directly
accessiable within the same class
}

Derived : private Base{
1. we can access protected and public
members  of Base class inside Derived class

2. Inside this class all the visibility of
members of Base class except private
are made as priavate
}

main(){
Derived d;
// using Derived object  we cannot access
any members  of Base class in main

}
```

```
Base{
All members of Base class are directly
accessiable within the same class
}

Derived : protected Base{
1. we can access protected and public
members  of Base class inside Derived class

2. Inside this class the visibility of public
members of Base class are made as protected
}

main(){
Derived d;
// using Derived object  we cannot access
any members  of Base class in main

}
```

```
Base{
All members of Base class are directly
accessiable within the same class
}

Derived : public Base{
1. we can access protected and public
members  of Base class inside Derived class

2. Inside this class the visibility of
members of Base class are not changed
}

main(){
Derived d;
// using Derived object  we can access
only public members of Base class in main

}
```

```
Base{
private:
      int n1;
protected:
      int n2;
public:
      int n3;
}
```

Dervied :public Base{
No change in access spceifiers of base class inside derived class
}

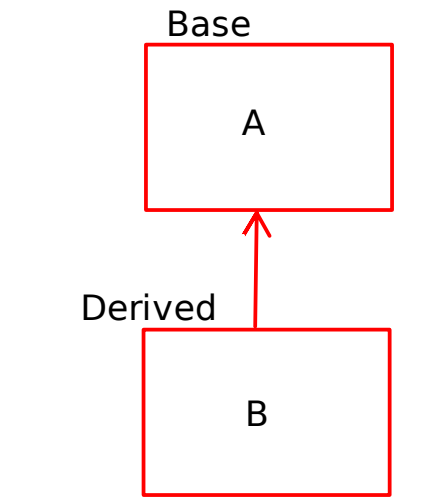
```
Base{
private:
      int n1;
protected:
      int n2;
public:
      int n3;
}
```

Dervied :protected Base{
public memners of base class will become protected inside derived class
}

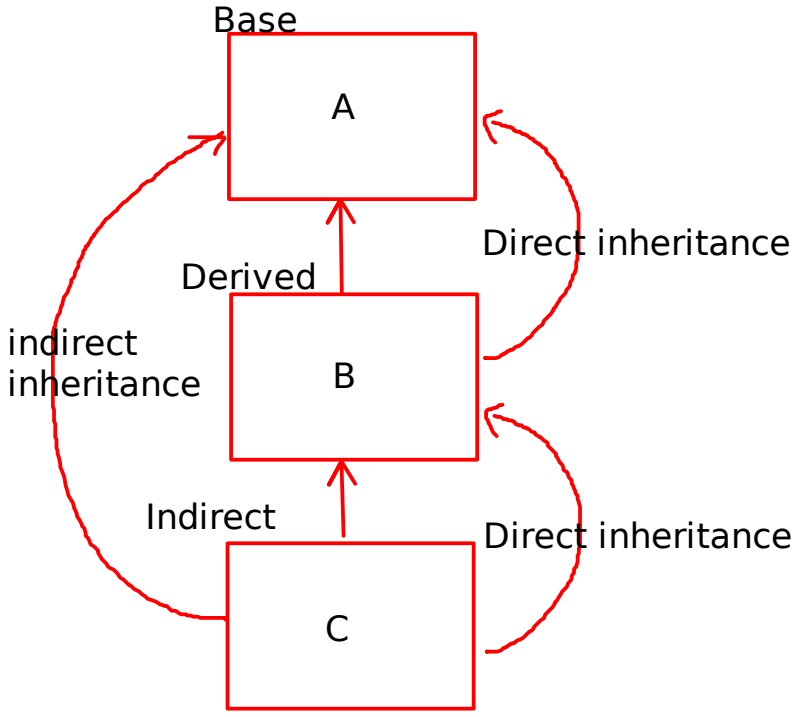
```
Base{
private:
      int n1;
protected:
      int n2;
public:
      int n3;
}
```

Dervied : Base{
All visibility of members of base class except private will become private
}

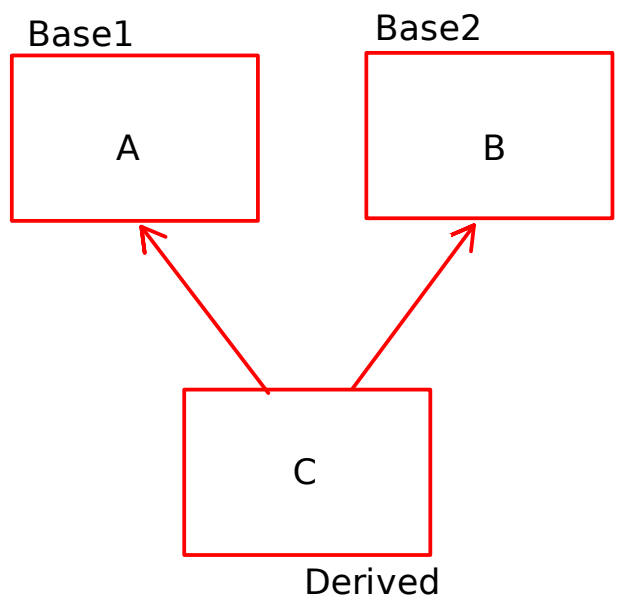
Types of Inheritance



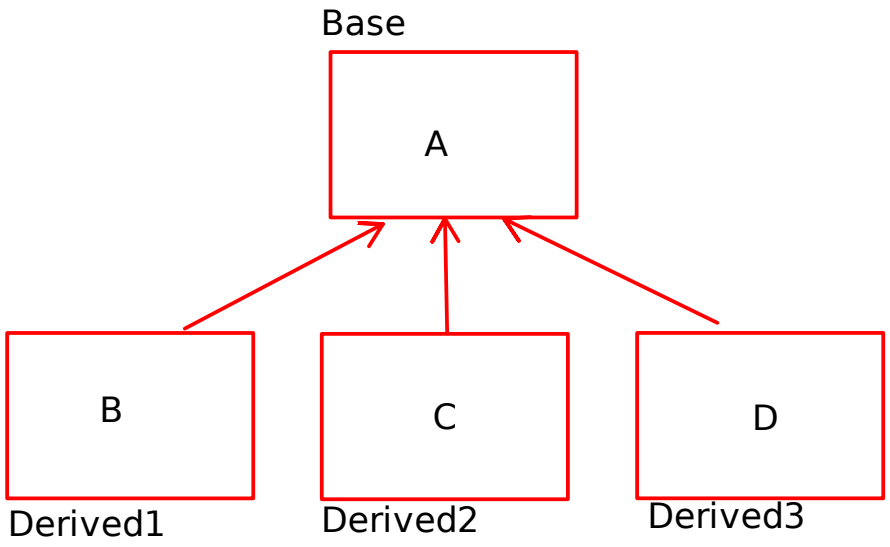
1. Single Inheritance



2. Multilevel Inheritance



3. Multiple Inheritance



4. Hirerachical Inheritance

5. Hybrid Inheritance

- Mix any 2 types of inheritance we call it as hybrid inheritance

```
single inheritance
class Base{
}
class Derived : public Base{
}
```

```
Multilevel inheritance
class Base{
}

class Derived : public Base{
}

classs Indirect : public Derived{
}
```

```
Multiple Inheritance
class Base1{
}

class Base2{
}

class Derived : public Base1, public Base2{
}
```

```
class Base{
}
class Derived:public Base{
}
class Derived2:public Base{
}
class Derived3:Public Derived2{
}
```

```
Hirerachical Inheritance
class Base{
}

class Derived1:public Base{
}

class Derived2:public Base{
}
```