1.
Inheritance class B : public(A)
ISA relation; base class: general derived class: more specific
eg: Animal & dog;
Question: Zoo & dog? Get back later

A::print() B::print() A::print()

A::print() B::print() B::print()

Question: Is that function overriding or overloading?
Overriding. Same function for base and derived class.
Overloading is when you have 2 functions with the same name but different arguments. Essentially, they are different functions.

Question: What is polymorphism?
Using a base class pointer pointing to a derived class object.
Why? Zoo has data member Animal[] animals;

Non virtual: call the function of the declared type
Virtual: call the function of the object's actual type

Question: What's missing from the class? Why a destructor has to be virtual?
We need to destruct the actual object.

Note: During the compile time: only the declared type can be seen.
Virtual functions: the functions to call is decided during run time.

Note: If a base class function is virtual, all the derived classes(direct/indirect) are virtual no matter you write the virtual keyword or not. However it is a convention that you always write virtual for all derived classes.

2.
ABBABB
Print1 creates a new copy of the object;

Compile error: Print1 is ilegal.
However Print2 and Print3 both work fine, and that is what we want to do.

When a class contains a pure virtual function, it is a abstract base class. We can't declare an object of the class, but we can still call its virtual functions. BC during run time the actually function of the object will be called.

3.
Construction order

construct base class(Does initialization list tell you which constructor to use?) -> initialize data member in the order they appear in class(1, initialization list? 2. default constructor?) -> go to the body
Destruction order
Reverse
body of destructor -> destruct each data member reverse order -> destruct base class
Note: draw pictures help a lot
A(10), A() ,B(), A(5), B(5),C(), ~C(), ~(B),~(A), ~(B), ~(A), ~(A)
Note: The output should always be symmetric.

Relation to Induction:
Prove: $2^0 + 2^1 + 2^2 + \ldots + 2^k = 2^{k+1} - 1$
Base Case:
k=0, $2^0 = 2^1 - 1$
Inductive Hypothesis:
If $2^0 + 2^1 + 2^2 + \ldots + 2^k = 2^{k+1} - 1$ is true,
then we can prove that $2^0 + 2^1 + 2^2 + \ldots + 2^{k+1} = 2^{k+2} - 1$
Inductive Step:
$2^0 + 2^1 + 2^2 + \ldots + 2^{k+1} = (2^0 + 2^1 + 2^2 + \ldots + 2^k) + 2^{k+1} = 2^{k+1} - 1 + 2^{k+1} = 2^{k+2} - 1$

Think about what is the subproblem: recursive step
Think about the base case

Splitting the problem into the last and the rest:
Factorial

```
int fact(int n)
{
    if (n == 0) return 1;
    return n * fact(n-1);
}
```

Splitting the problem into the first and the rest:

```
void printArrayInOrder(const double a[], int n)
{
   if (n == 0)
         return;
   cout << a[0] << endl;
   printArrayInOrder(a + 1, n-1);
}
```

Switch the 2 statements

Splitting the problem in halves: Mergesort. example in class

```
int expon(int a, int b)
{
    if (b == 0) return 1;
    return a * expon(a, b-1);
}

int expon_fast(int a, int b)
{
    if (b == 0) return 1;
    if (b % 2 == 0){
        int m = expon_fast(a, b/2);
        return m * m;
    }else{
        return a * expon_fast(a, b-1);
    }
}
```

Recursion help you to come up with a solution

Fibonacci:
fib(n) = fib(n-1) + fib(n-2)
Time consuming
Iteration or Memorization

Step:
Go either 1 or 2 steps. How many ways to go n steps?
step(n) = step(n-1) + step(n-2)

```
int solveParade(int n)
{
    if(n == 1) return 2;
    if(n == 2) return 3;
    return solveParade(n-1) + solveParade(n-2)
}
P(n) = F(n)+B(n)
F(n) = P(n-1)
B(n) = F(n-1)
```