Or S ~~ext~~ implements X, Y, Z

* This is the minimum requirement, S might implement many more interfaces.

ABC is a genetic class that contains generic type of objects S.

2019-20

Overloaded functions

B 5 (a) (i) ~~They~~ have the same signature except for the param. list.

i.e. ~~(ii)~~ return type ~~might~~ will be the same, just number ~~&~~ or types of arguments or both is redefined.

(ii) Function body

(iii) Library functions can be overloaded too (saw one in Stack overflow)

(iv) Constructors can be overload, but not destructors

(v) using:

```
void f1(int &x);   ] ---> involving by
void f1(int x);    ]      int x=10;
                          f1(x);
```

can cause ambiguity

(vi) Overloaded functions have different addresses. (for obvious reasons)

5(b)

```
int& f()
{ return x;
}
```

here
int a = f(); is the same as
(works as) a = x;
working as r_value.

again, ~~int~~ f() = a ; is also possible,
~~f()~~ ~~int~~ it is like x = a; working as l-value

5(c) Bitwise copy ; copy constructor is called

Bitwise copy → includes address/ ref. copy too which Sometimes causes Various problems

copy constructor is called
→ ~~whe~~ at the declaration statement
→ When object is passed by value to a function as an argument
→ When a fnc. returns an object & received by a same class's obj.

6(a) class Frame body should be given a forward declaration or the body MUST be given at the beginning.

In the Frame class:
adding the following two functions:

```
void setWidth(int x){
    width = x;
}
```

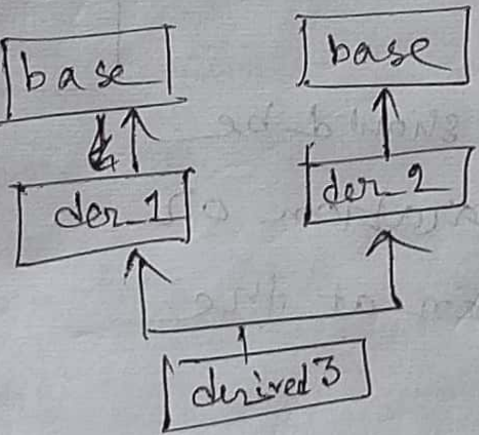
```
void setLength (int x){
      length = x;
}
```

In the ~~ptr~~ Photo class: adding the following operator overloading def^n.

```
   frame operator+ (int x){
      frame f;
        f. setWidth(this->getWidth()+x);
        f.setLength (this->getLength()+x);
      return f;
   }
```

Code: github.com/Ibtida01/TF_Codes_1_2/tree/main

6(b) ~~Yes~~ Yes, it is possible,



Now derived 3 has 2 copies (more than 2 is also possible) of base class

```
class base{
public: int x;
};
class der_1 : public base {
```

```
public:
    int y;
};

class der_2:public base
{ public:
      int z;
};
class derived3:public der_1, publ der_2
{
  public:
     int t;
};
int main(){
  derived3 obj;
  obj.x = 100; //will be a ambiguous assignment
}
```

To avoid this: using

```
(i) class der_1: virtual public base{...};
  & class der_1 : virtual public base{...};

Or, (ii) obj.der_1::x = 100; //OK
         obj.der_2::x = 200; //OK
```

7 ⓐ [Kindly go through the slides please]

7 ⓑ (i): 1st parameter is a stream, we don't have access to the stream objects, so they are external entities to the class. So they must be either friends of the concerning class or a public method.

(ii) cout can be used, but naming conventions will be violated. [I know a little about the answer to the current Q]

8 ⓐ fnc. signatures to be added in the Fraction class:

```
public:
  friend Fraction& Operator++(Fraction& fr);
  void setX(int x);
  void setY(int y);
```

Definitions:

```
Fraction& operator++(Fraction&fr){
    fr.setX(~~x+1);~~ fr.getX()+1);
    fr.setY(fr.getY()+1);
    return fr;
}

void Fraction::setX(int x){
   this->x = x;
}
```

```
void Fraction:: setY(int y){
    this->y=y;
}
```

8(b) → Out of syllabus.

(c) If we want to count the number of objects of the class present in the current execution time, we can have a static member veriable, like this(↓)

```
class hello{
  static int count=0;
  hello( ){
     ++count;
  }
  ~hello(){
    ~~tu~~ --count;
   }
};
```