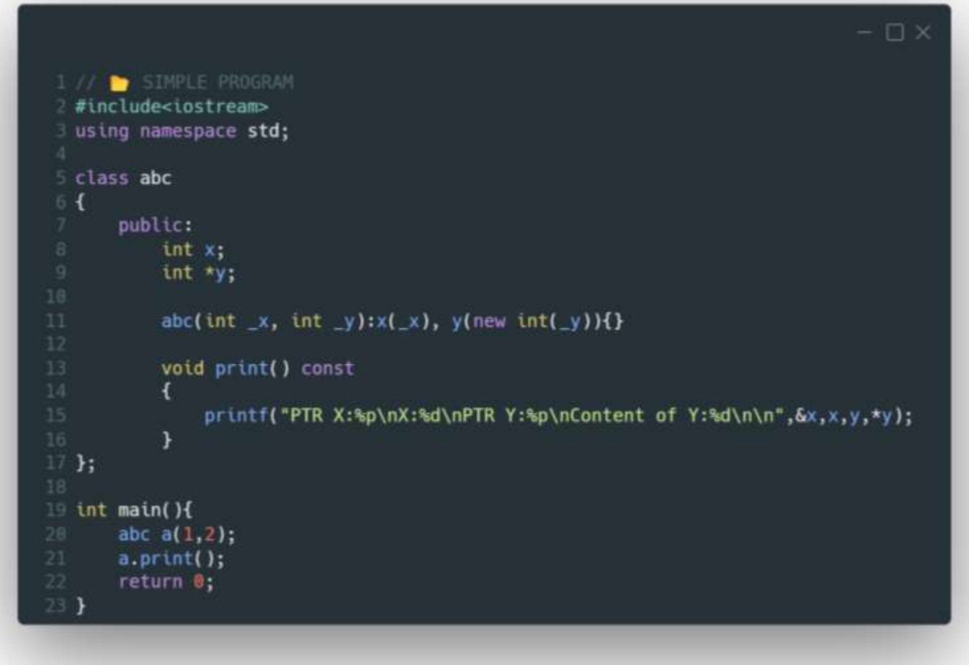
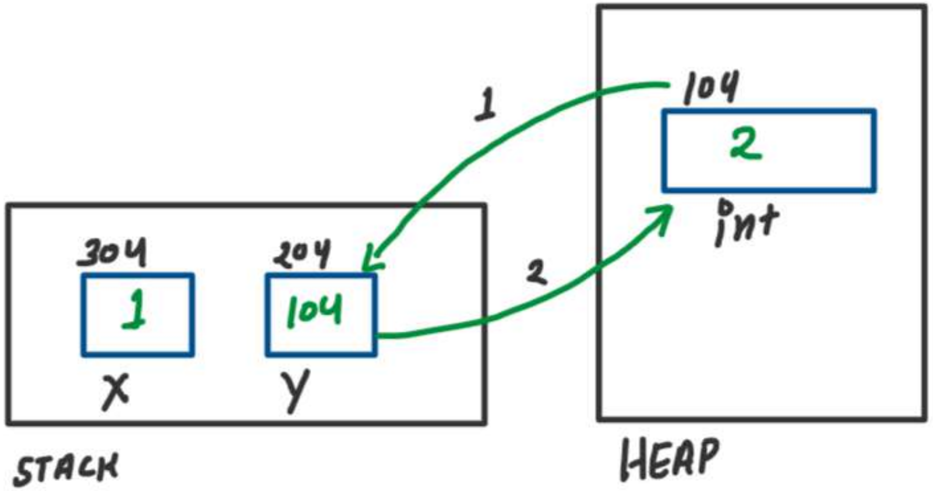
26/10/2023

Object Oriented Programming Class 02 Homework

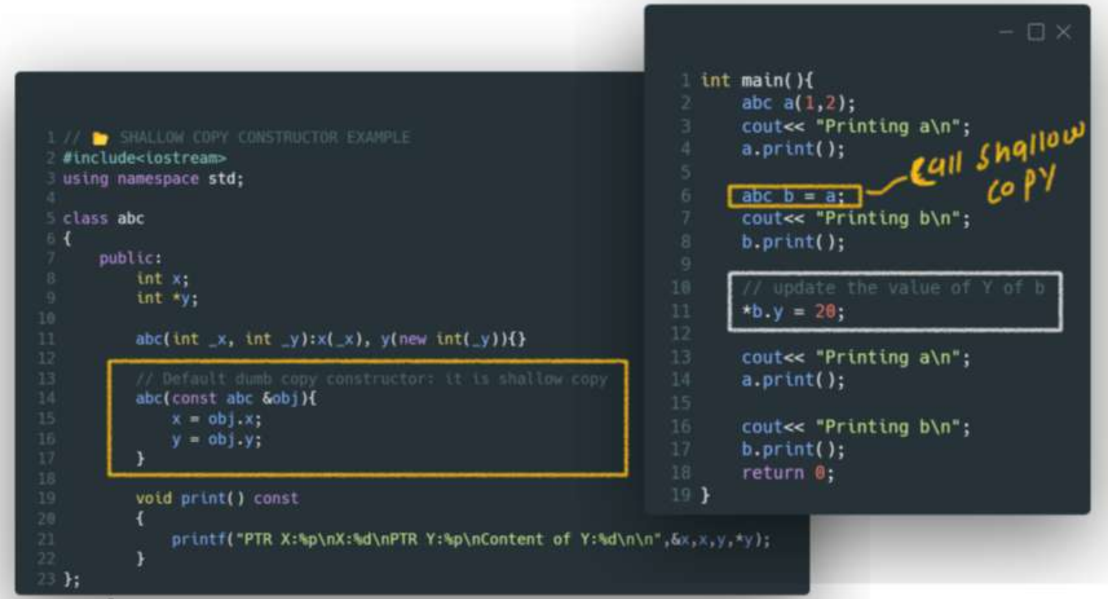
1: Shallow Vs Deep Copy Constructor



```cpp
// SIMPLE PROGRAM
#include<iostream>
using namespace std;

class abc
{
    public:
        int x;
        int *y;

        abc(int _x, int _y):x(_x), y(new int(_y)){}

        void print() const
        {
            printf("PTR X:%p\nX:%d\nPTR Y:%p\nContent of Y:%d\n\n",&x,x,y,*y);
        }
};

int main(){
    abc a(1,2);
    a.print();
    return 0;
}
```



Output
(&x) PTR X : 304 (y) PTR Y : 104
X : 1 (*y) Contunt of Y : 2



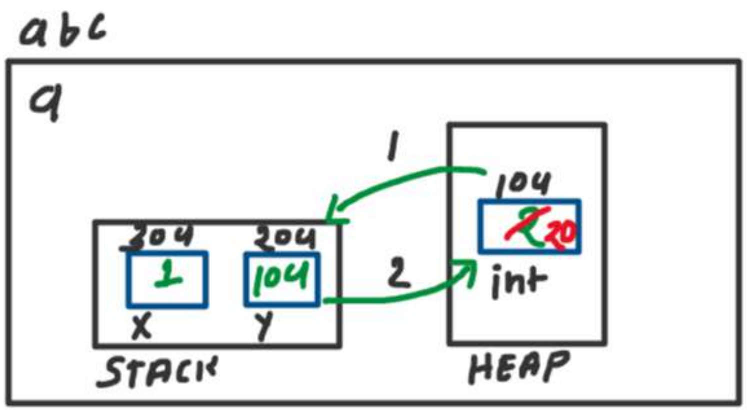
```cpp
// 📁 SHALLOW COPY CONSTRUCTOR EXAMPLE
#include<iostream>
using namespace std;

class abc
{
    public:
        int x;
        int *y;

        abc(int _x, int _y):x(_x), y(new int(_y)){}

        // Default dumb copy constructor: it is shallow copy
        abc(const abc &obj){
            x = obj.x;
            y = obj.y;
        }

        void print() const
        {
            printf("PTR X:%p\nX:%d\nPTR Y:%p\nContent of Y:%d\n\n",&x,x,y,*y);
        }
};
```

```cpp
int main(){
    abc a(1,2);
    cout<< "Printing a\n";
    a.print();

    abc b = a;
    cout<< "Printing b\n";
    b.print();

    // update the value of Y of b
    *b.y = 20;

    cout<< "Printing a\n";
    a.print();

    cout<< "Printing b\n";
    b.print();
    return 0;
}
```

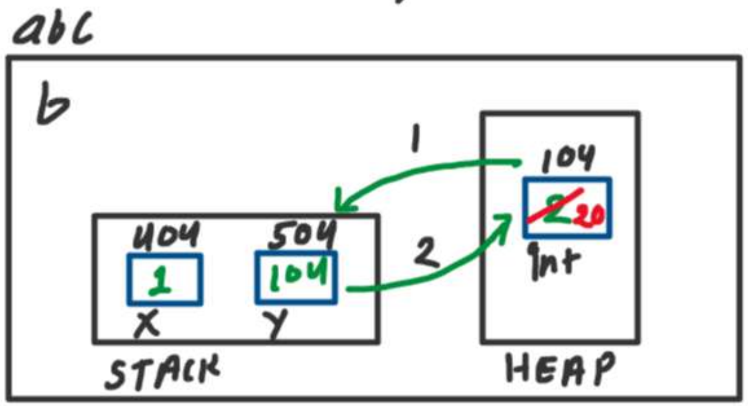
call shallow copy

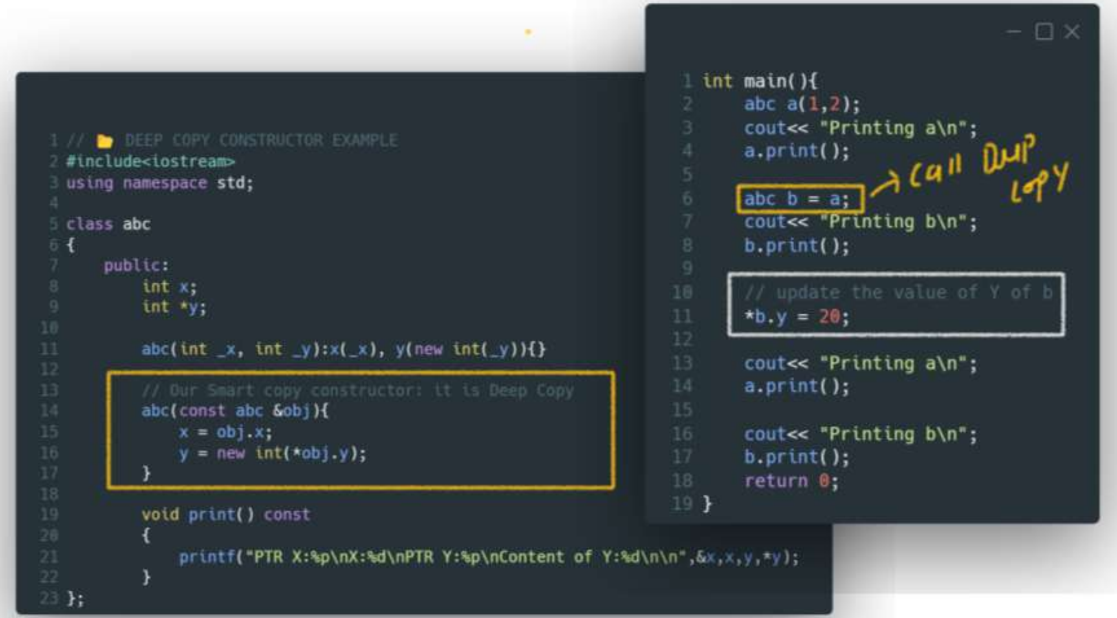
output

```
Printing a          Printing b          Printing a          Printing b
PTR X:304           PTR X:404           PTR X:304           PTR X:404
X:1                 X:1                 X:1                 X:1
PTR Y:104           PTR Y:104           PTR Y:104           PTR Y:104
Content of Y:2      Content of Y:2      Content of Y:20     Content of Y:20
```



SHALLOW COPY





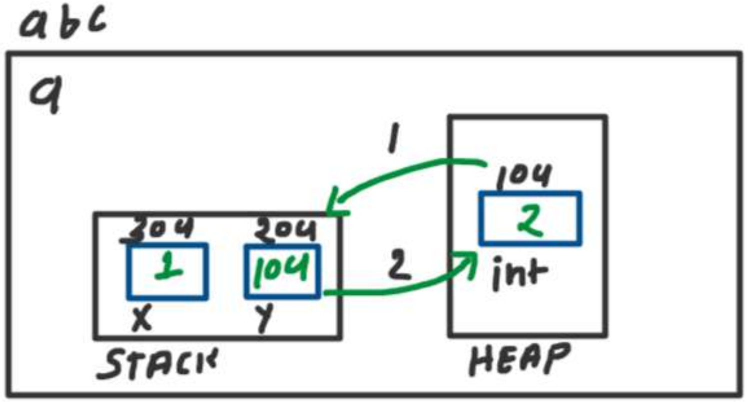
```cpp
// 📁 DEEP COPY CONSTRUCTOR EXAMPLE
#include<iostream>
using namespace std;

class abc
{
    public:
        int x;
        int *y;

        abc(int _x, int _y):x(_x), y(new int(_y)){}

        // Our Smart copy constructor: it is Deep Copy
        abc(const abc &obj){
            x = obj.x;
            y = new int(*obj.y);
        }

        void print() const
        {
            printf("PTR X:%p\nX:%d\nPTR Y:%p\nContent of Y:%d\n\n",&x,x,y,*y);
        }
};
```

```cpp
int main(){
    abc a(1,2);
    cout<< "Printing a\n";
    a.print();

    abc b = a;
    cout<< "Printing b\n";
    b.print();

    // update the value of Y of b
    *b.y = 20;

    cout<< "Printing a\n";
    a.print();

    cout<< "Printing b\n";
    b.print();
    return 0;
}
```

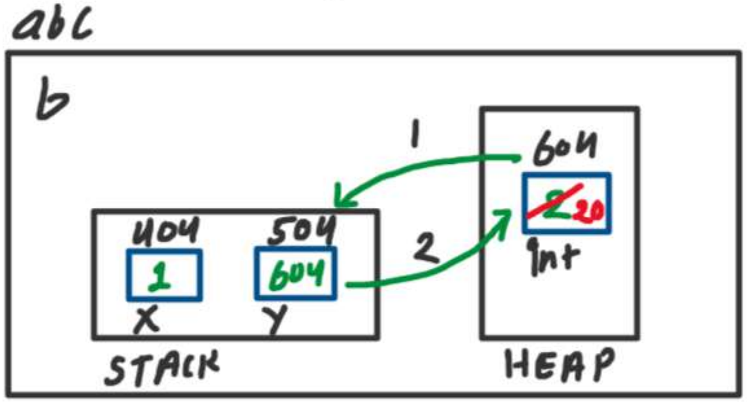
→ Call Deep Copy

Output

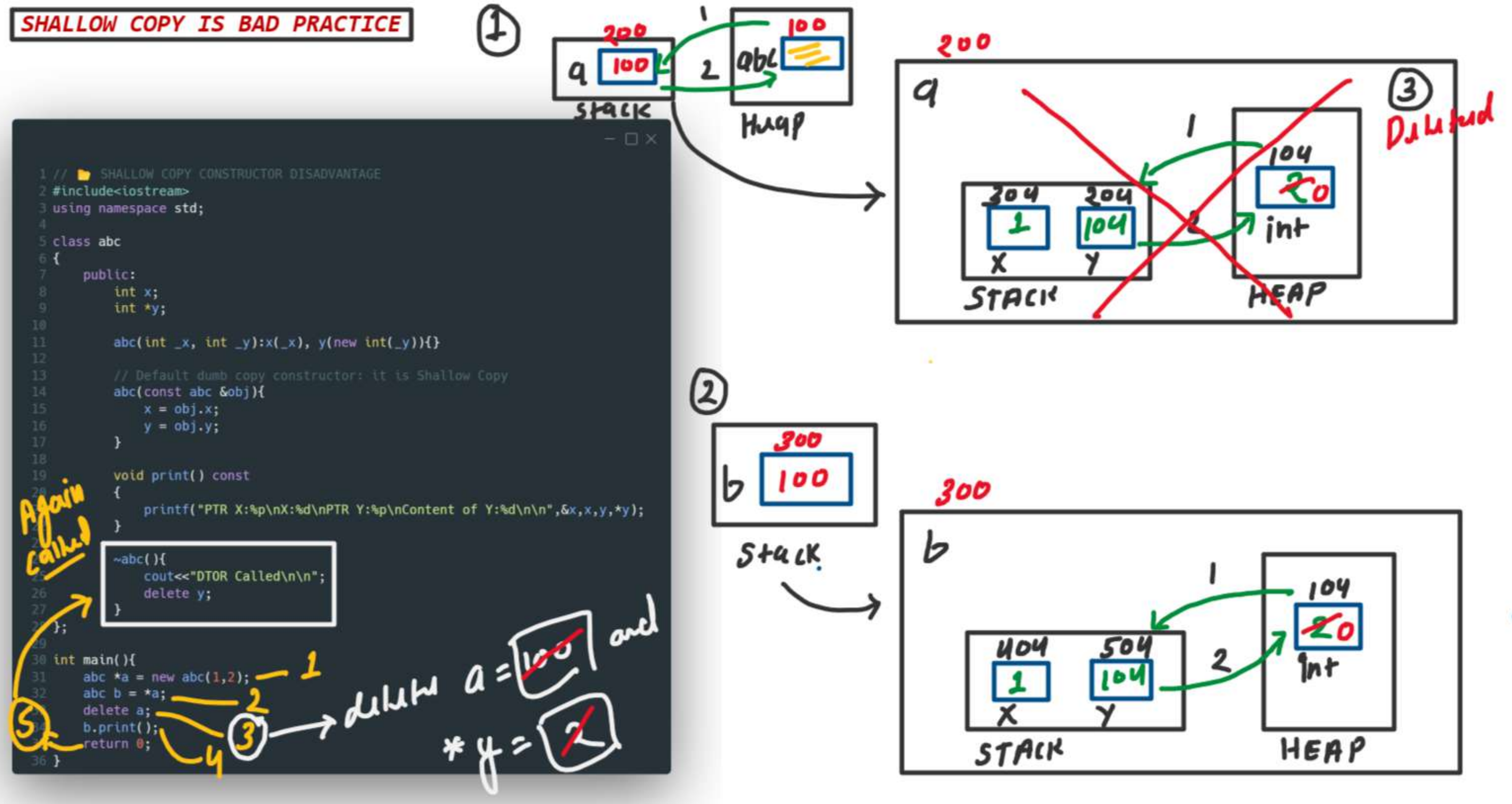
```
Printing a          Printing b          Printing a          Printing b
PTR X:304           PTR X:404           PTR X:304           PTR X:404
X:1                 X:1                 X:1                 X:1
PTR Y:104           PTR Y:604           PTR Y:104           PTR Y:604
Content of Y:2      Content of Y:2      Content of Y:2      Content of Y:20
```



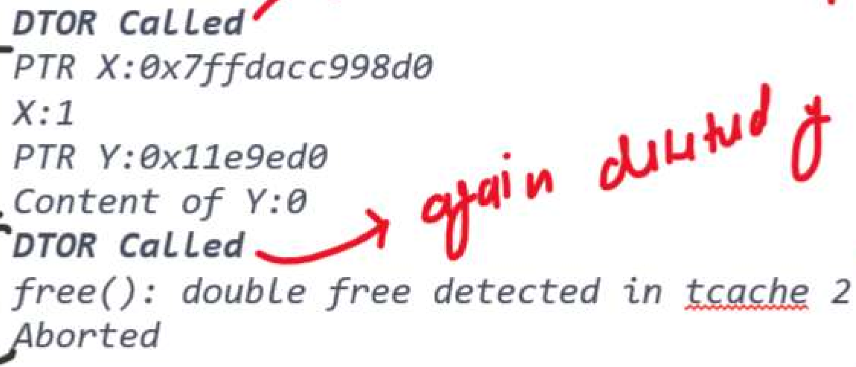
DEEP COPY



SHALLOW COPY IS BAD PRACTICE



```cpp
// 📁 SHALLOW COPY CONSTRUCTOR DISADVANTAGE
#include<iostream>
using namespace std;

class abc
{
    public:
        int x;
        int *y;

        abc(int _x, int _y):x(_x), y(new int(_y)){}

        // Default dumb copy constructor: it is Shallow Copy
        abc(const abc &obj){
            x = obj.x;
            y = obj.y;
        }

        void print() const
        {
            printf("PTR X:%p\nX:%d\nPTR Y:%p\nContent of Y:%d\n\n",&x,x,y,*y);
        }

        ~abc(){
            cout<<"DTOR Called\n\n";
            delete y;
        }
};

int main(){
    abc *a = new abc(1,2);
    abc b = *a;
    delete a;
    b.print();
    return 0;
}
```



3

```
DTOR Called
```

Y Heap se 1 time Delete Hua

4

```
PTR X:0x7ffdacc998d0
X:1
PTR Y:0x11e9ed0
Content of Y:0
```

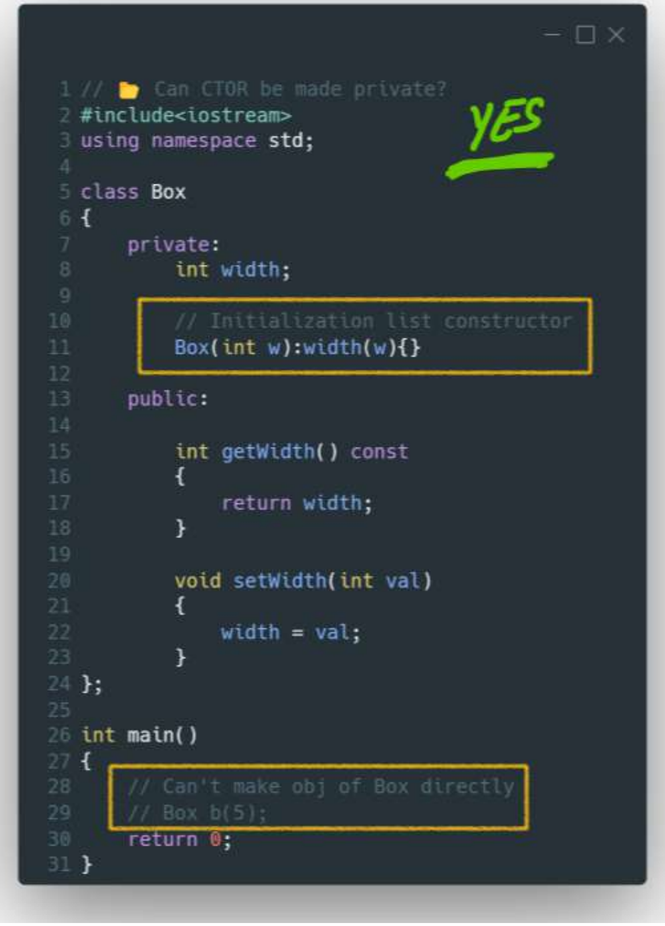
5

```
DTOR Called
free(): double free detected in tcache 2
Aborted
```

again deleted y ko delete hua

Dis-Advantage of shallow copy

📂 2: Can CTOR be made private?



```cpp
// 📂 Can CTOR be made private?
#include<iostream>
using namespace std;

class Box
{
    private:
        int width;

        // Initialization list constructor
        Box(int w):width(w){}

    public:

        int getWidth() const
        {
            return width;
        }

        void setWidth(int val)
        {
            width = val;
        }
};

int main()
{
    // Can't make obj of Box directly
    // Box b(5);
    return 0;
}
```

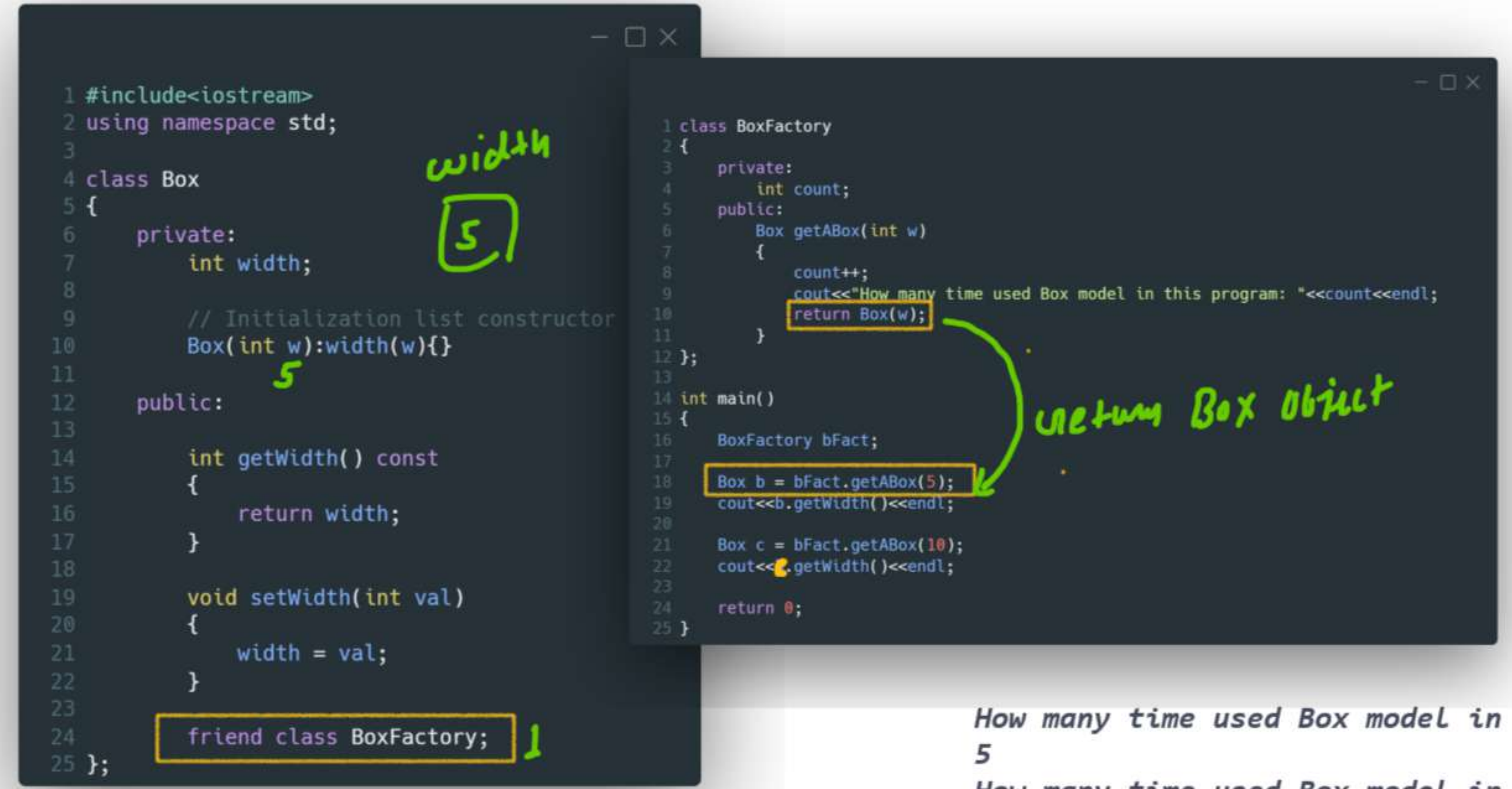
Can CTOR be made private?
ANS: Yes we can make private CTOR implicitly.

Why use of private CTOR?

We can make private CTOR but can not be called in main() method directly.
Means, We can't initialize data members directly in main() method.
Means, We can't create object directly in main() method.
But we can make object of private constructor class using friend class.

Application of private CTOR:
1. Singleton class
2. Count how many object of private CTOR class are used in whole program.

We can make object of private constructor class using friend class



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

class Box
{
    private:
        int width;

        // Initialization list constructor
        Box(int w):width(w){}

    public:

        int getWidth() const
        {
            return width;
        }

        void setWidth(int val)
        {
            width = val;
        }

        friend class BoxFactory;
};
```

```cpp
class BoxFactory
{
    private:
        int count;
    public:
        Box getABox(int w)
        {
            count++;
            cout<<"How many time used Box model in this program: "<<count<<endl;
            return Box(w);
        }
};

int main()
{
    BoxFactory bFact;

    Box b = bFact.getABox(5);
    cout<<b.getWidth()<<endl;

    Box c = bFact.getABox(10);
    cout<<c.getWidth()<<endl;

    return 0;
}
```

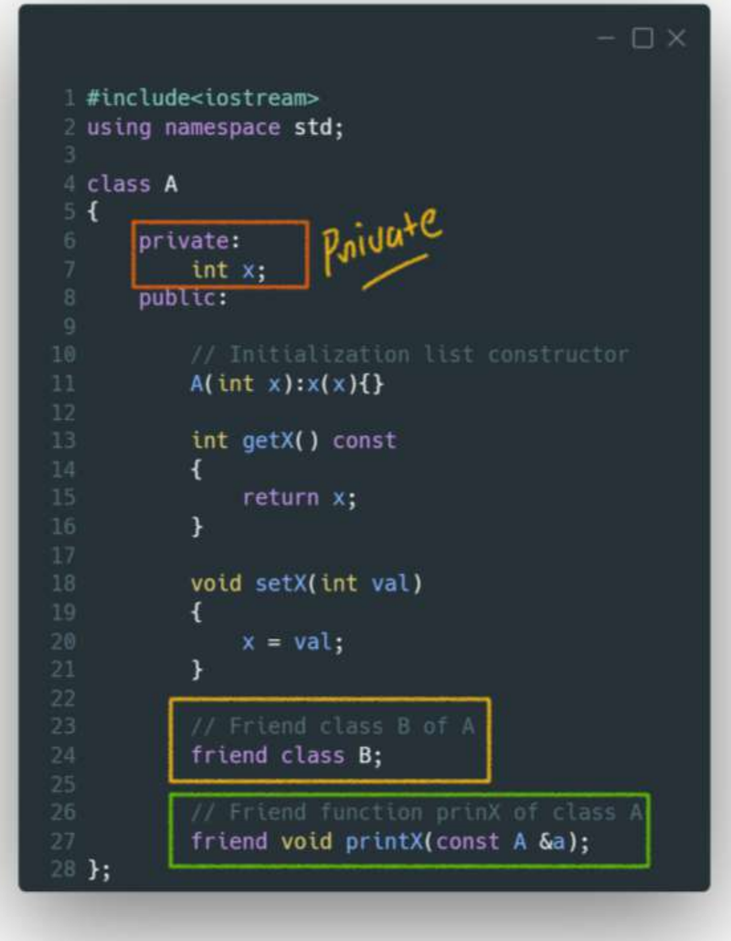
```
How many time used Box model in this program: 1
5
How many time used Box model in this program: 2
10
```

📂 3: Friend Keyword

1. friend is a keyword in C++ that is used to share the information of a class that was previously hidden.

2. For example, the private members of a class are hidden from every other class and cannot be modified except through getters or setters.

Similarly, the protected members are hidden from all classes other than its children classes.

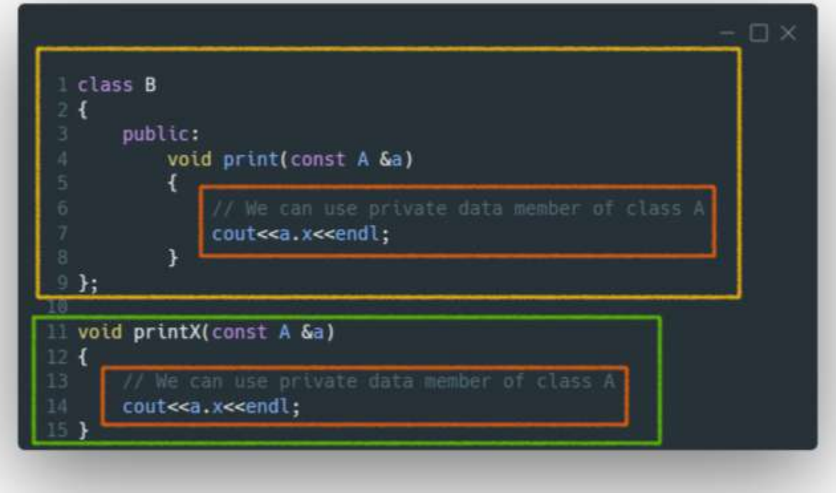


```
#include<iostream>
using namespace std;

class A
{
    private:
        int x;
    public:

        // Initialization list constructor
        A(int x):x(x){}

        int getX() const
        {
            return x;
        }

        void setX(int val)
        {
            x = val;
        }

        // Friend class B of A
        friend class B;

        // Friend function prinX of class A
        friend void printX(const A &a);
};
```

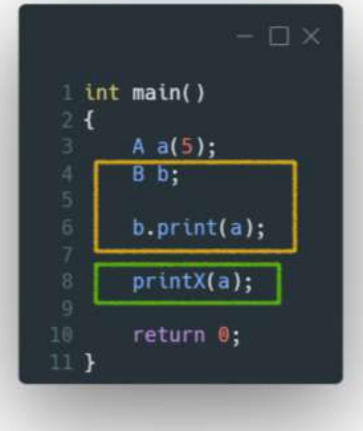
Private

Output

5
5



```
class B
{
    public:
        void print(const A &a)
        {
            // We can use private data member of class A
            cout<<a.x<<endl;
        }
};

void printX(const A &a)
{
    // We can use private data member of class A
    cout<<a.x<<endl;
}
```



```
int main()
{
    A a(5);
    B b;

    b.print(a);

    printX(a);

    return 0;
}
```