24|10|2023

Object Oriented Programming Class 01 Homework

📂 01: Const Keyword

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

int main(){

    // x is constant
    const int x = 10;
    // Initialization can be done
    // But we can't reassign a value
    // x = 11; error: assignment of read-only variable 'x'

    cout<< x <<endl;

    return 0;
}
```

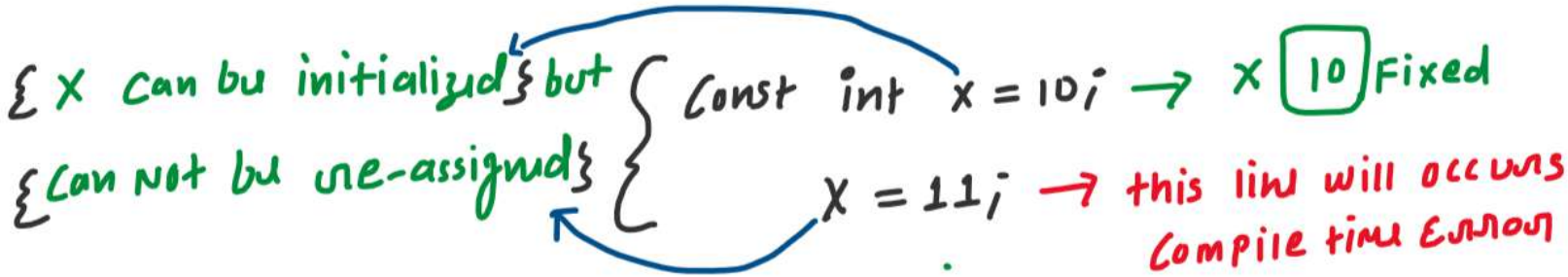
What is the const keyword:
Ans: *it is like a promise*

1 Why the use of the const keyword:

Reason 1: Simplify the code

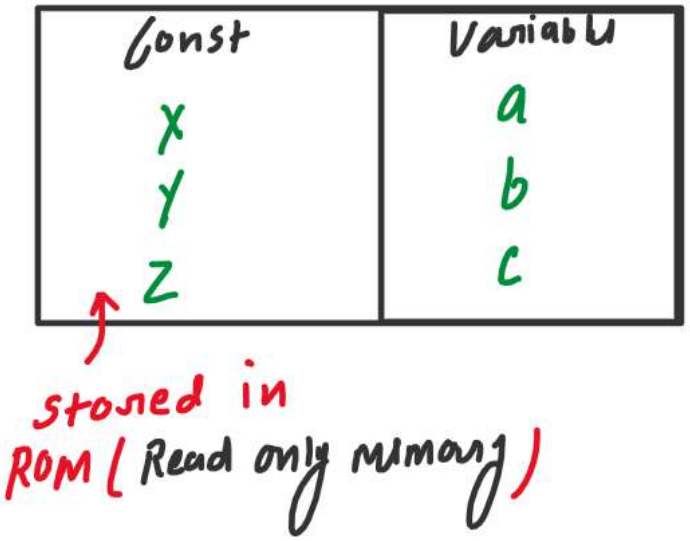
✅ *The const keyword is used to declare that a variable, function, or object is immutable, i.e., its value cannot be changed after initialization.*

✅ *E.g., if you declare a variable as* ***const int x = 10;*** *you cannot modify the value of x later in the program. Any attempt to modify the value will result in a compilation error.*



Reason 2: Optimization of the code

✅ The compiler may be able to store const variables in read-only memory, which can result in faster access time.



2 What is LValue and RValue:

✅ **LValue:** A variable having a memory address means LValues are modifiable

Example: `int x;` and `int y;`

✅ **RValue:** The variable does not have a memory address means RValues are not modifiable

Example: `5;` and `int *a = &b;` where a is alias of b

3 Const with pointer

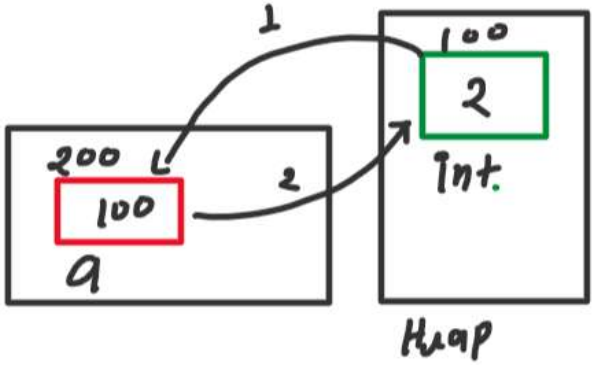
Case 1: Pointer is nonconstant and data is constant

Syntax

```
const int *q = new int(2);
```

Where, pointer = q
data = 2

integer Data ko Constant bana diya hai



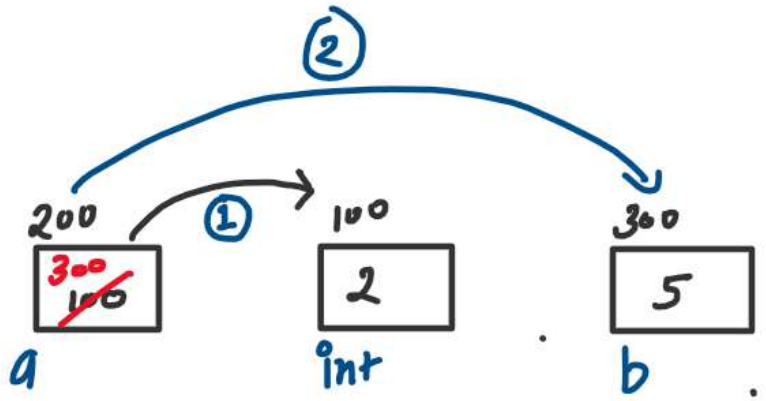
(100) Address par integer ki value (2) ko update nahi kar sakte hai.

```
*q = 5;  → GOT ERROR
```

int const *a = new int(2);

↳ SAME AS LINE 8

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

int main(){

    // 3 Const with pointer
    // Case 1: Pointer is nonconstant and data is constant
    const int *a = new int(2);
    cout<< *a <<endl;
    // *a = 5; error: assignment of read-only location '* a'
    // Cant change the content of pointer

    int b = 5;
    a = &b; // Pointer itself can be reassigned
    cout<< *a <<endl;

    return 0;
}
```



cout << *a << endl;

= *(100)

= 2

5 int

C.T. ERROR

cout << *a << endl;

= *(300)

= 5

Output: 2
5

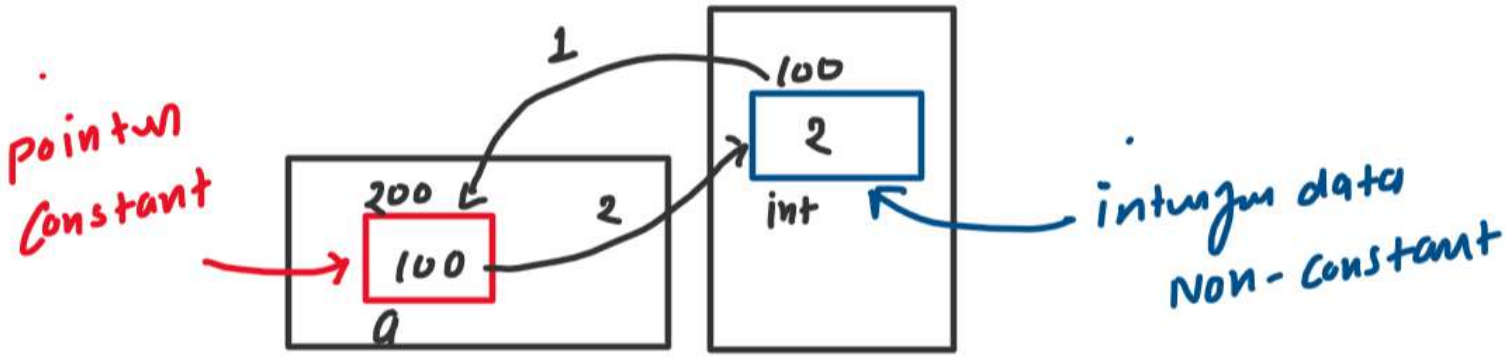
Case 2: *Pointer is constant and data is nonconstant*

Syntax

```
int *const a = new int(2);
```

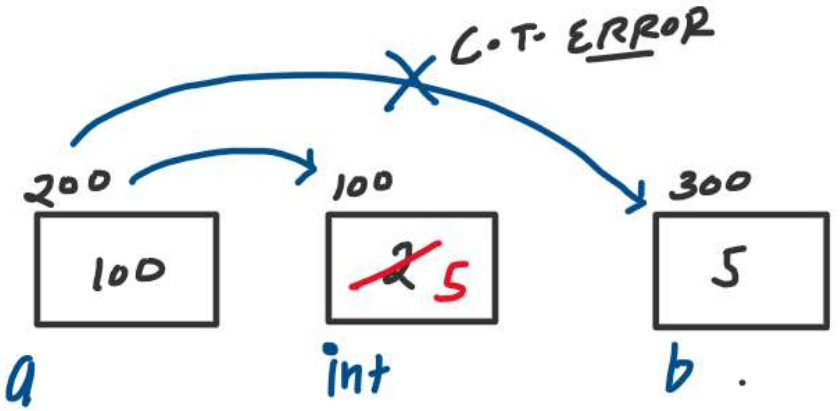
int — Data

*const a — Pointer



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

int main(){
    // 3 Const with pointer
    // Case 2: Pointer is constant and data is nonconstant
    int *const a = new int(2);
    cout<< *a <<endl;
    *a = 5; // we can change the content of pointer
    cout<< *a <<endl;

    int b = 5; ✗
    // a = &b; error: assignment of read-only variable 'a'
    // Pointer itself cant be reassigned

    return 0;
}
```

Output: 2
5



Cout

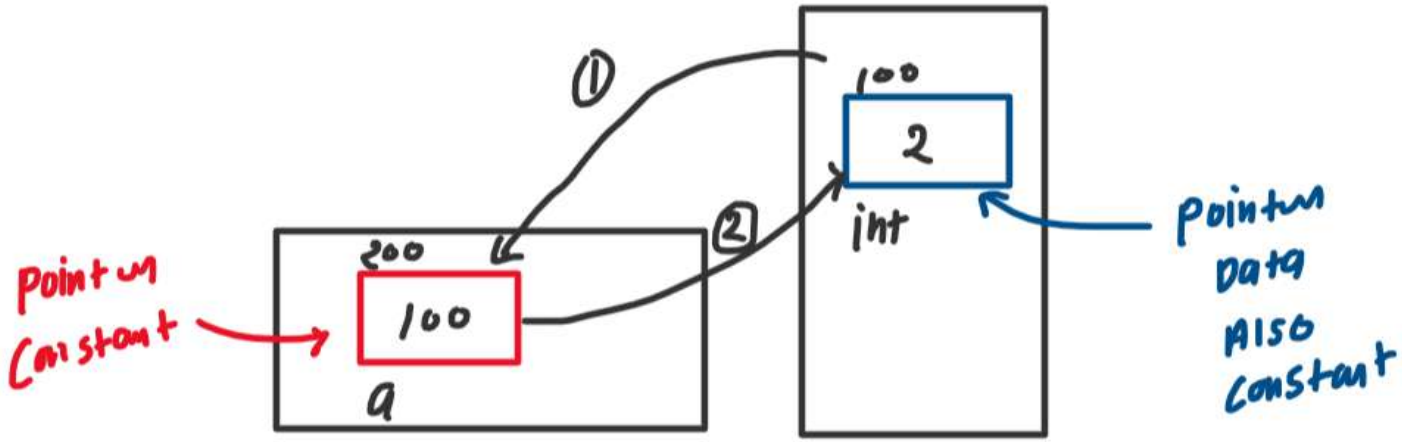
*a = *(100) = 2

*a = *(100) = 5

Case 3: *Both are constant pointers and data*

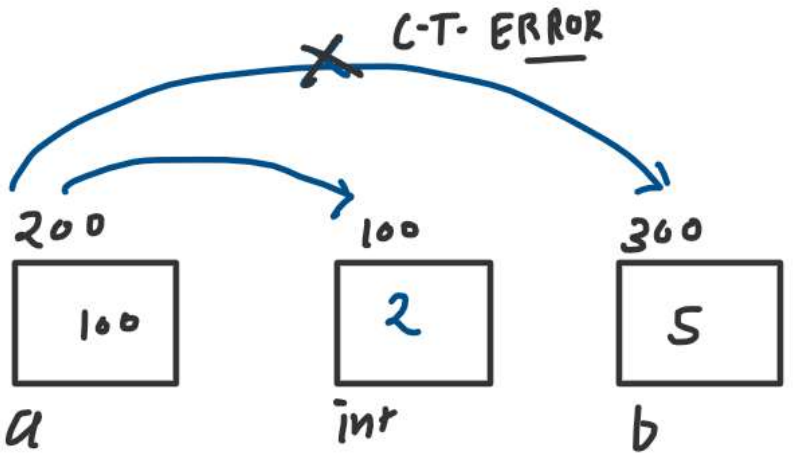
Syntax

```
const int *const a = new int(2);
```



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

int main(){
    // 3 Const with pointer
    // Case 3: Both are constant pointers and data
    const int *const a = new int(2);
    cout<< *a <<endl;
    // *a = 5; error: assignment of read-only location '*(const int*)a'
    // we cant change the content of pointer

    int b = 5;
    // a = &b; error: assignment of read-only variable 'a'
    // Pointer itself cant be reassigned

    return 0;
}
```

output: 2



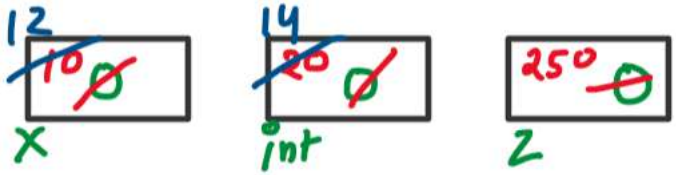
Cout *a = *(100) = 2

Const with class

- You can declare a function as const, which means that it does not modify the state of the object it is called on.
- In short, Constant function never changes the data member variable

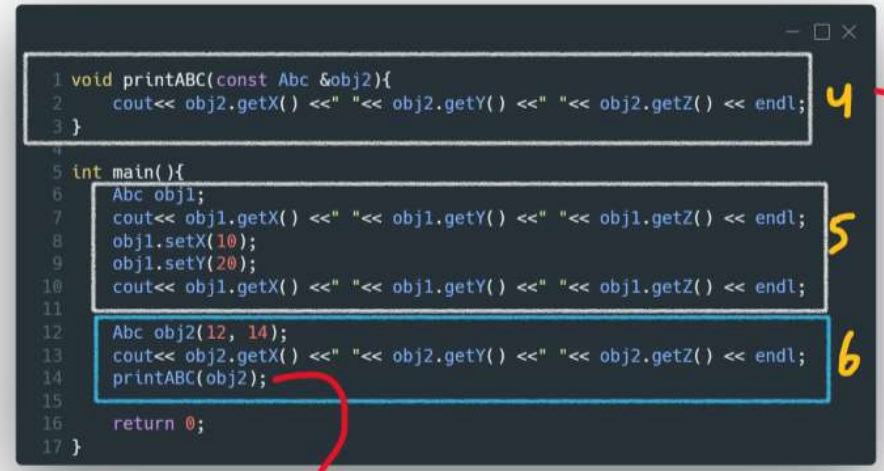
```
class Abc
{
private:
    int x;
    int *y;
    mutable int z;

public:
    Abc(){
        x = 0;
        y = new int(0);
        z = 0;
    }

    Abc(int _x, int _y){
        x = _x;
        y = new int(_y);
    }

    int getX() const
    {
        // x = 10;
        z = 250;
        return x;
    }

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

    int getY() const
    {
        return *y;
    }

    void setY(int _val){
        *y = _val;
    }

    int getZ() const
    {
        return z;
    }
};
```

1. (Bad Practice) mutable is used for debugging purpose only



2. getX() is constant function x variable ki value ko change nhi kar skta hai (Compile Time Error)

3. getX() is constant function z variable ki value ko change kar skta hai kyunki z mutable variable hai



```cpp
void printABC(const Abc &obj2){
    cout<< obj2.getX() <<" "<< obj2.getY() <<" "<< obj2.getZ() << endl;
}

int main(){
    Abc obj1;
    cout<< obj1.getX() <<" "<< obj1.getY() <<" "<< obj1.getZ() << endl;
    obj1.setX(10);
    obj1.setY(20);
    cout<< obj1.getX() <<" "<< obj1.getY() <<" "<< obj1.getZ() << endl;

    Abc obj2(12, 14);
    cout<< obj2.getX() <<" "<< obj2.getY() <<" "<< obj2.getZ() << endl;
    printABC(obj2);

    return 0;
}
```

4 5 6

(4) 📂Jab class ke object ko as an constant argument pass karte hai to us function me only constant function ko hi use kiya ja skta hai like

📂**printABC()** takes constant **Abc** object to yanha constant function use kiye ja skte hai like **getX()**, **getY()** and **getZ()**

Yanha Abc ka object **obj1** printABC() function me pass kiya ja rha hai

Output

x int z

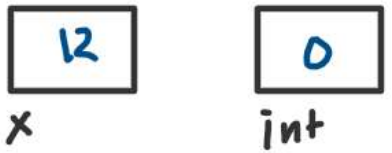
0	0	250
10	20	250
12	14	250
12	14	250

02: Default Argument

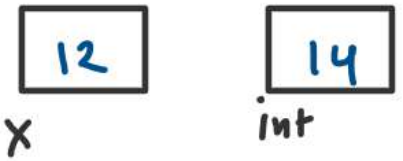
```cpp
#include<iostream>
using namespace std;

class Abc
{
private:
    int x = 0;
    int *y;

public:
    Abc(int _x, int _y = 0){
        x = _x;
        y = new int(_y);
    }

    int getX() const
    {
        return x;
    }
    int getY() const
    {
        return *y;
    }
};

void printABC(const Abc &obj){
    cout<< obj.getX() <<" "<< obj.getY() <<endl;
}

int main(){
    Abc obj1(12);
    printABC(obj1);

    Abc obj2(12, 14);
    printABC(obj2);

    return 0;
}
```

(1) **Note:** *Default argument always right most side me likha jata hai*

(2)



(3)



Output

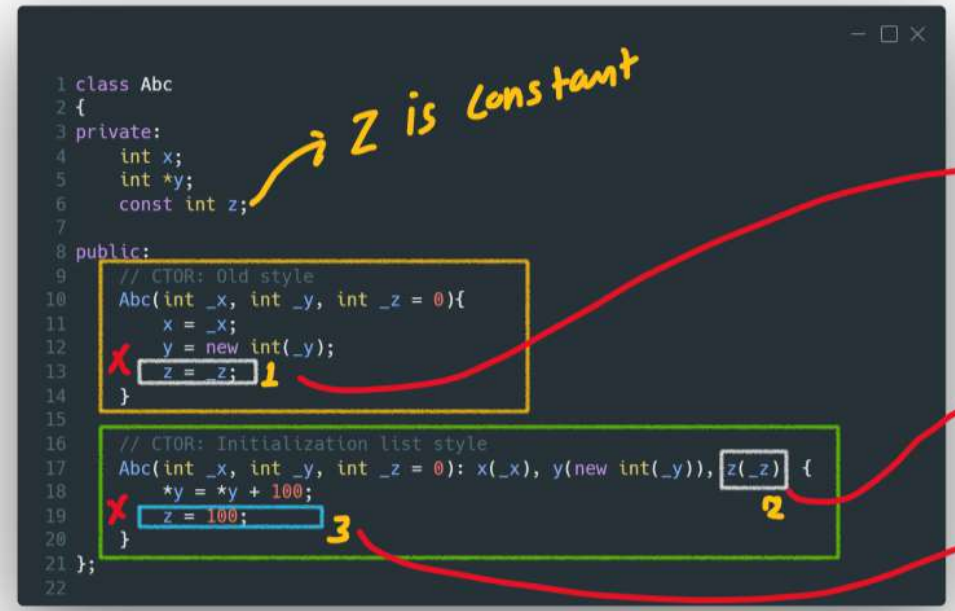
x	int
12	0
12	14

03: Initialization list

(Writing of new way of CTOR)

What is initialization list:
It is a way to write constructor with new style.

Why use of initialization list:
It helps to re-initialization of a constant variable but not re-assignment



① *const variable ko* ***re-initialized*** *nhi kar skte ho* ***(Compile time error)***

② *const variable ko* ***re-initialized*** *kar skte ho*

③ *const variable ko* ***re-assign*** *nhi kar skte ho* ***(Compile time error)***

📂 04: MACROS

What is MACROS:
A MACROS is a piece of code in a program that is replaced by the value of the MACROS.
In compiler, *MACROS are pre-processor directives tool that allow to define constants, functions, or code snippets.*
MACROS does not take memory space.

Why use of MACROS:
Readability of code and reusability of MACROS definition in C++ program.

② Working process of MACROS:

Whenever a MACROS name is encountered by the compiler, it replaces the name with the definition of the MACROS before compilation of the code.

① How to defined MACROS:

MACROS is defined by #define directive.
MACROS definitions need not be terminated by a semi-colon(;)

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

#define PI 3.14
#define MAXX(x,y) (x > y ? x : y)

float circleArea(float r){
    return PI * r * r;
}

float circlePerimeter(float r){
    return 2 * PI * r;
}

int getMaxForA(){
    int a = 10;
    int b = 20;
    int c = MAXX(a,b);
    return c;
}

int getMaxForB(){
    int a = 1000;
    int b = 100;
    int c = MAXX(a,b);
    return c;
}

int main(){
    cout<<"Circle Area: "<<circleArea(2.5)<<endl;
    cout<<"Circle Perimeter: "<<circlePerimeter(7.5)<<endl;
    cout<<"Max for A: "<<getMaxForA()<<endl;
    cout<<"Max for B: "<<getMaxForB()<<endl;

    return 0;
}
```

1

PreProcessing

Before Compilation MACROS NAME REPLACE WITH It's VALUE

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

float circleArea(float r){
    return 3.14 * r * r;
}

float circlePerimeter(float r){
    return 2 * 3.14 * r;
}

int getMaxForA(){
    int a = 10;
    int b = 20;
    int c = a > b ? a : b;
    return c;
}

int getMaxForB(){
    int a = 1000;
    int b = 100;
    int c = a > b ? a : b;
    return c;
}

int main(){
    cout<<"Circle Area: "<<circleArea(2.5)<<endl;
    cout<<"Circle Perimeter: "<<circlePerimeter(7.5)<<endl;
    cout<<"Max for A: "<<getMaxForA()<<endl;
    cout<<"Max for B: "<<getMaxForB()<<endl;

    return 0;
}
```

TEXT → Code → PreProcessing → Compile → MACHINE CODE → EXE FILE (Program)

05: Static Keyword in Class

```cpp
// How does class work:
#include<iostream>
using namespace std;

class Abc{
public:
    int x, y;

    void printABC(){
        cout<< this->x <<" "<< this->y <<endl;
        cout<< x <<" "<< y <<endl;
    }

};

int main(){
    Abc obj1 = {1, 2};
    Abc obj2 = {4, 5};
    obj1.printABC();
    obj2.printABC();
    return 0;
}
```

Not shared variable for all instance of class now

100 → 1, 104 → 2

200 → 4, 204 → 5

Line no. 10 and 11 are same

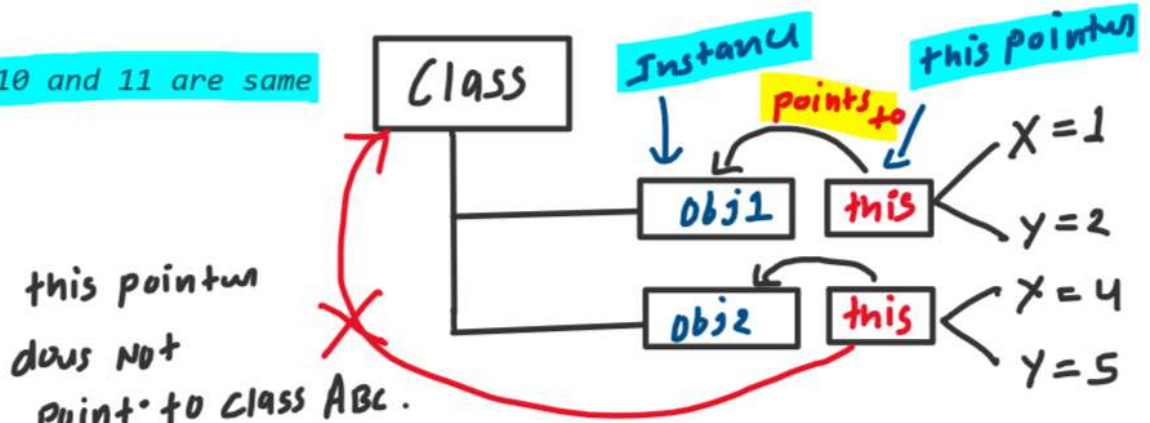
output
1 2
1 2
4 5
4 5

How does class work:

1. In class, this pointer points to an instance of class. but does not point to any class.

2. This pointer is a hidden variable/perimeter of a function in a class.

3. This pointer is used to avoid the conflict between two or more instances/objects of a class.



NOTE { Iss code me x and y ki different copy ban rhi hai different object/instance ke liye

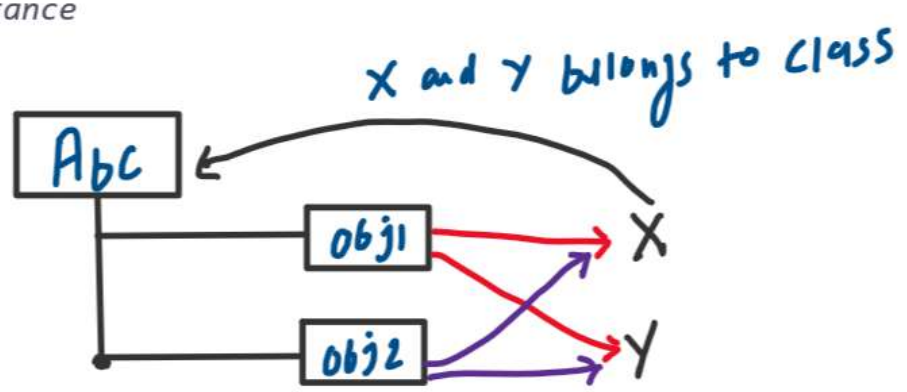
Static data members:
The variable is going to share memory with all of the class instance.

Why use of static data members?
If we want to spread out the single copy of variable of class to all instance/object of class.

```cpp
// Static data members
#include<iostream>
using namespace std;

class Abc{
public:
    static int x, y; // Shared variable for all instance of class now

    void printABC(){
        cout<< x <<" "<< y <<endl;
    }

};

// Initialization of static data members x and y of class Abc
int Abc::x;
int Abc::y;

int main(){
    Abc obj1;
    obj1.x = 1;
    obj1.y = 2;
    obj1.printABC();

    Abc obj2;
    obj2.x = 4;
    obj2.y = 5;
    obj2.printABC();
    obj1.printABC();

    return 0;
}
```

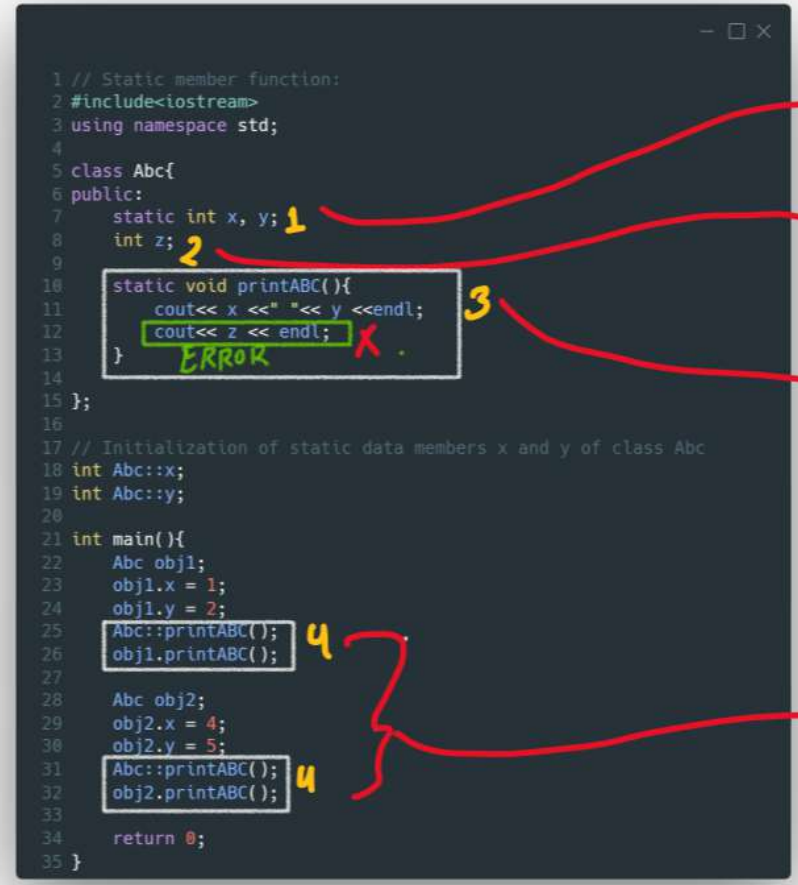
100 104
4 5
x y

1. *Iska mtlb x and y will never be different for different instance of a class*
2. *Iska mtlb x and y belongs to a class but does not belong to a particular instance of a class jabki both are same for all instance*



Output

1	2
4	5
4	5



```cpp
// Static member function:
#include<iostream>
using namespace std;

class Abc{
public:
    static int x, y;
    int z;

    static void printABC(){
        cout<< x <<" "<< y <<endl;
        cout<< z << endl;
    }
};

// Initialization of static data members x and y of class Abc
int Abc::x;
int Abc::y;

int main(){
    Abc obj1;
    obj1.x = 1;
    obj1.y = 2;
    Abc::printABC();
    obj1.printABC();

    Abc obj2;
    obj2.x = 4;
    obj2.y = 5;
    Abc::printABC();
    obj2.printABC();

    return 0;
}
```

1 2 3 4 4 ERROR

① ***Static data member variable:*** *Shared variable for all instance of class now*

② ***Non-static data member variable:*** *iss variable ko printABC() method use nhi kar skta hai*

③ ***Static member method:*** *belongs to a class but does not belong a particular object.*

Iska mtlb yeh hai ki pritABC() method ke pass hidden argument "this pointer" nhi hoga. Or wo kisi bhi non-static data member variable ka bhi use nhi kar skta hai.

④ ***Both are same things***

Output

1 2
1 2
4 5
4 5