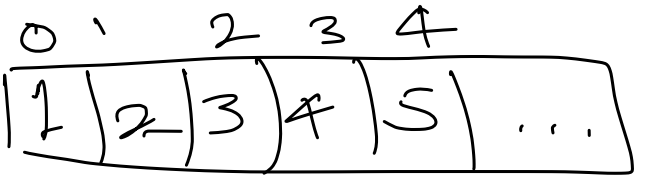
12345 ⇒



count = 5

Dynamic Memory Allocation

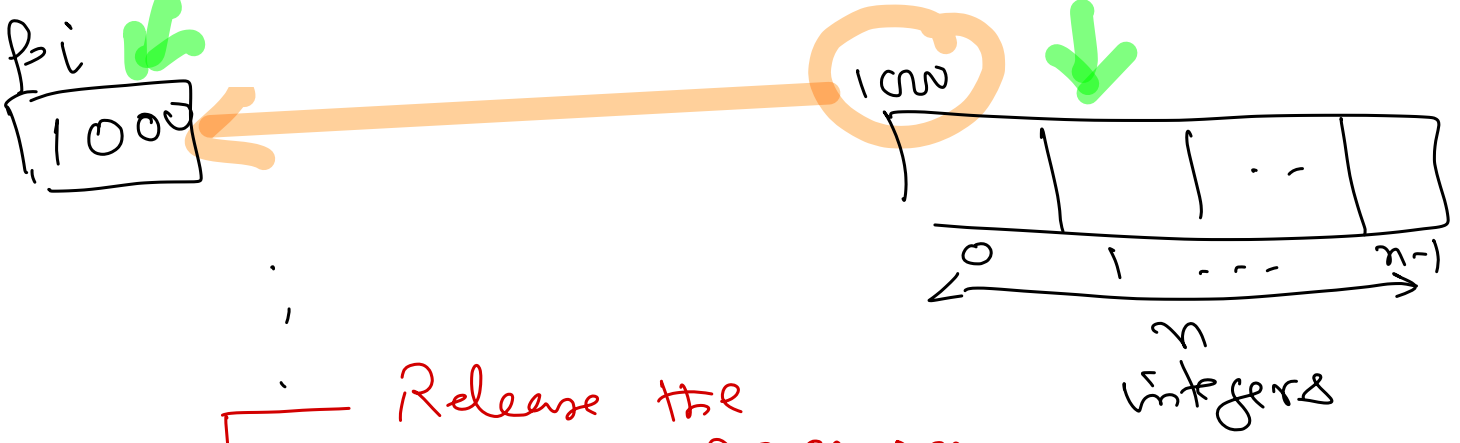
```
int n;
std::cin >> n;
```

malloc() — allocates memory in heap. ← number of bytes to allocate

free()

```
#include <stdlib.h>
```

```
int *pi = (int *) malloc (n * sizeof(int));
```



Release the memory that was allocated by malloc.

```
free (pi);
```

```
void *  malloc (int size);
```

void pointer → stores an address

Can not do any pointer arithmetic

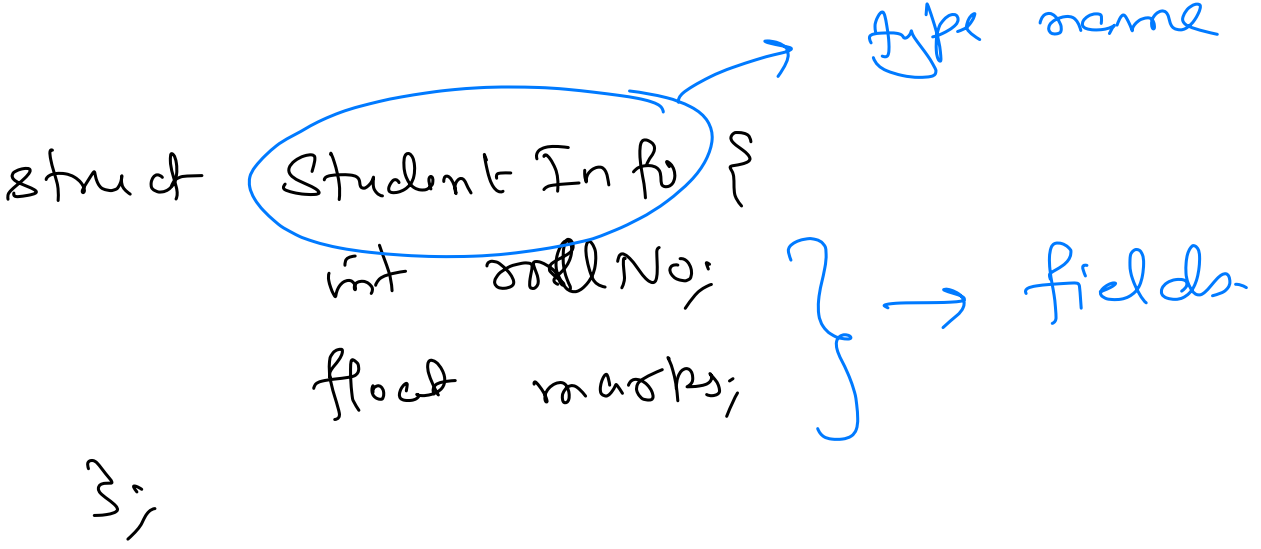
Can not dereference a void pointer

```
int * pi = (int *) malloc ( .... );
if (pi == NULL) {
std::cout<< "failed to allocate mem";
    return;0;
}

free (pi);
```

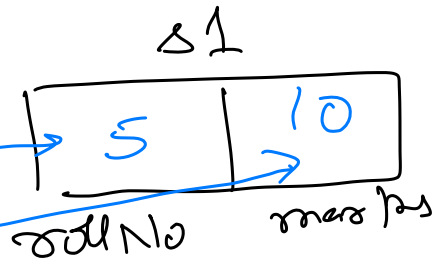
returns address if memory was allocated else returns NULL

creates a user defined type ⟸ struct → we want to combine multiple value together that are related to each other.

int rollNo; }
float marks; } Info of a student



```
struct StudentInfo s1;
```




```
s1.rollNo = 5;
s1.marks = 10;
```

```
struct StudentInfo s2;
```

```
s2 = s1;
```

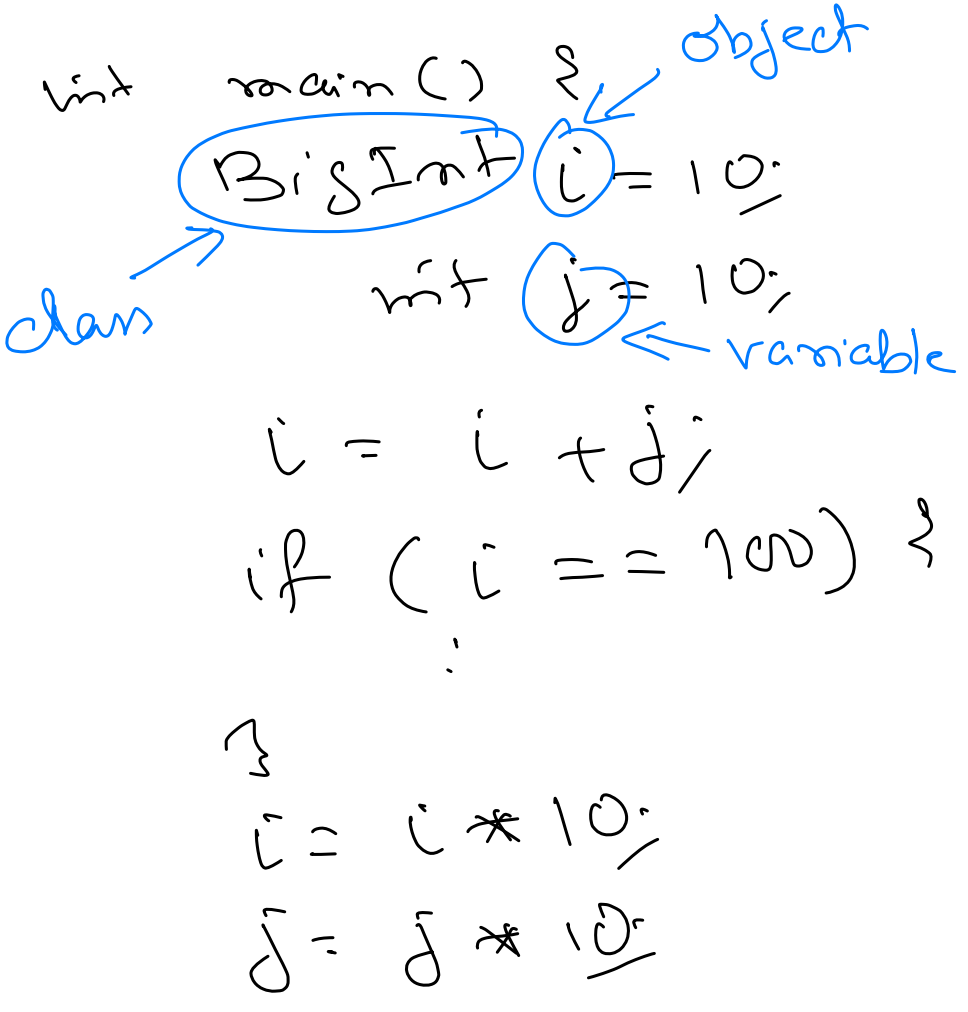
```
if (s1 == s2) {
}
```

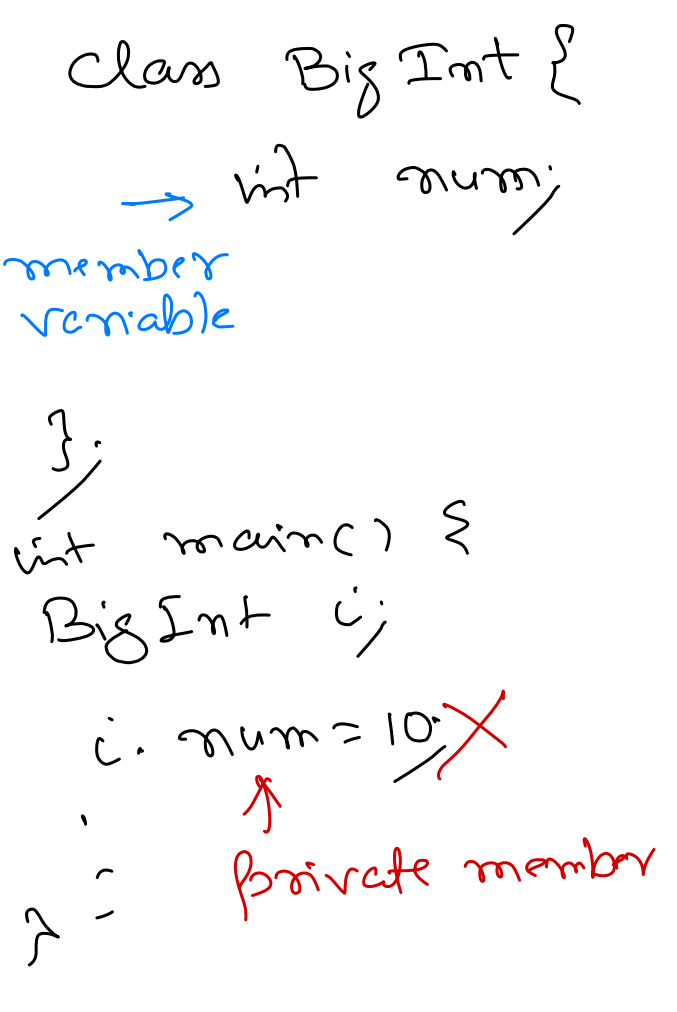
class → keyword using which we define a user defined type.

OOP

- data abstraction
- data hiding.
- data encapsulation
  - data
  - operation

are combined in a single entity. ⇒ class


int main() {
BigInt i = 10;
int j = 10;
object
class
variable
i = i + j;
if (i == 100) {
}
i = i * 10;
j = j * 10;


class BigInt {
int num;
member variable
};
int main() {
BigInt i;
i.num = 10;
private member
}

private public protected → access specifiers to class members.

```
class BigInt {
    public:
        int num;
};
int main() {
    BigInt i;
    i.num = 10;
}
```

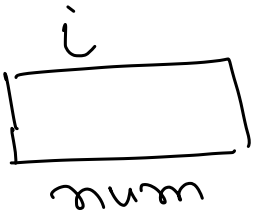
```
class BigInt {
    int num;
public:
    void setNum(int n) {
        num = n;
    }
    int getNum() {
        return num;
    }
};
```

member variable

member function

int main() {

BigInt i;

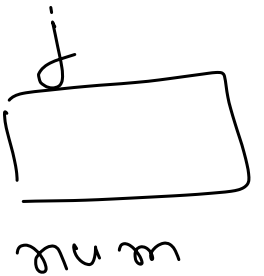


i.num = 10; ✗ → as num is private

i.setNum(10); ✓

BigInt j = 10; ✗

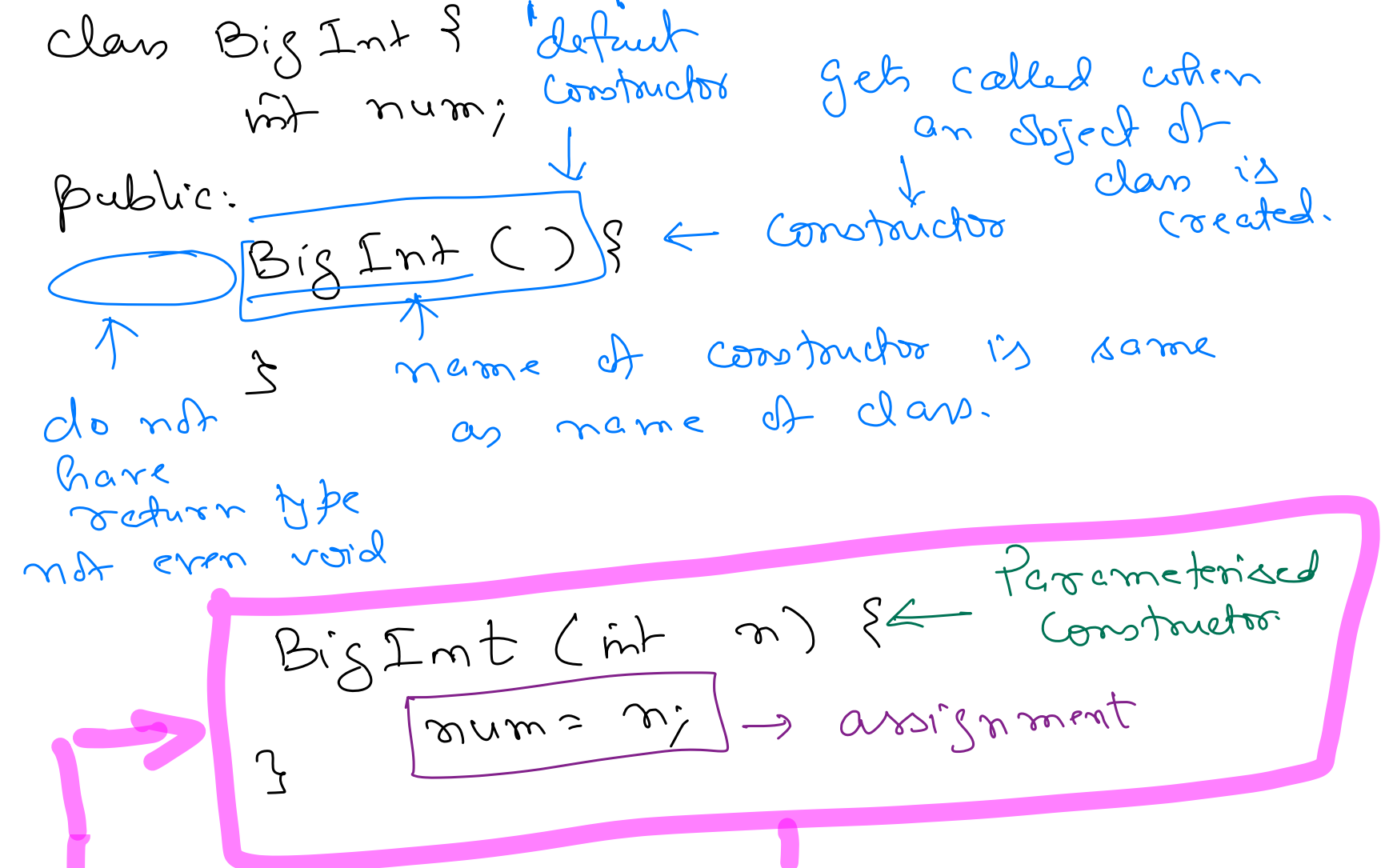
num is private



Parameterised Constructor

Gets called when we create an object without initializing.

Constructor that do not take any argument.


class BigInt {
int num;
public:
BigInt ( ) {
}
default constructor
gets called when an object of class is created.
Constructor
name of constructor is same as name of class.
do not have return type not even void
BigInt (int n) {
num = n;
}
Parameterised Constructor
assignment

Not the correct way.

```
BigInt ( int n) : num (n) {
    num = n;
}
```

member initializer list

```
num = n;
   ⇓
this -> num = n;
```

```
BigInt ( int n, int m) : num (n+m)
{
}
```

```
int main() {
    BigInt R( 10 , 20 );
}
```

Destructor → member function that gets called when an object is destroyed (memory is getting freed).

↓ + class Name

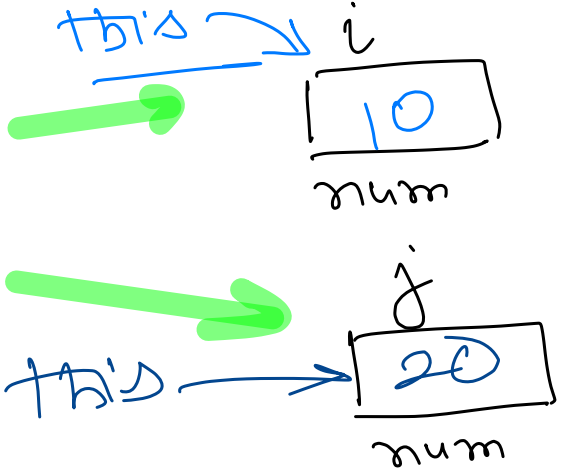
```
~BigInt () {

}
```

destructor never receive argument

int main() {

BigInt i(10);

BigInt j(20);



if (i == j) ⟹ i.operator==(j)

this

obj2

std::cout << "eq";

else

std::cout << "neq";

i.Read(); j.write();

Operator function → a function implemented to define behaviour of a built in operator for user defined type.

Operator overloading ⟹ a built in operator for user defined type.

⇓

implement a member function (it can also be a non-member) with name as operator. → Keyword

```
class BigInt {
    :
public:
    bool operator == ( BigInt obj2) {
        if (num == obj2.num)
            return true;
        return false;
    }
    void Read() {
        std::cin >> num;
```

this -> num

```
}
void write() {
    std::cout << num;
}
```

cout → object of class ostream

cin → object of class istream

iostream.h

↓

all identifiers are declared in global scope

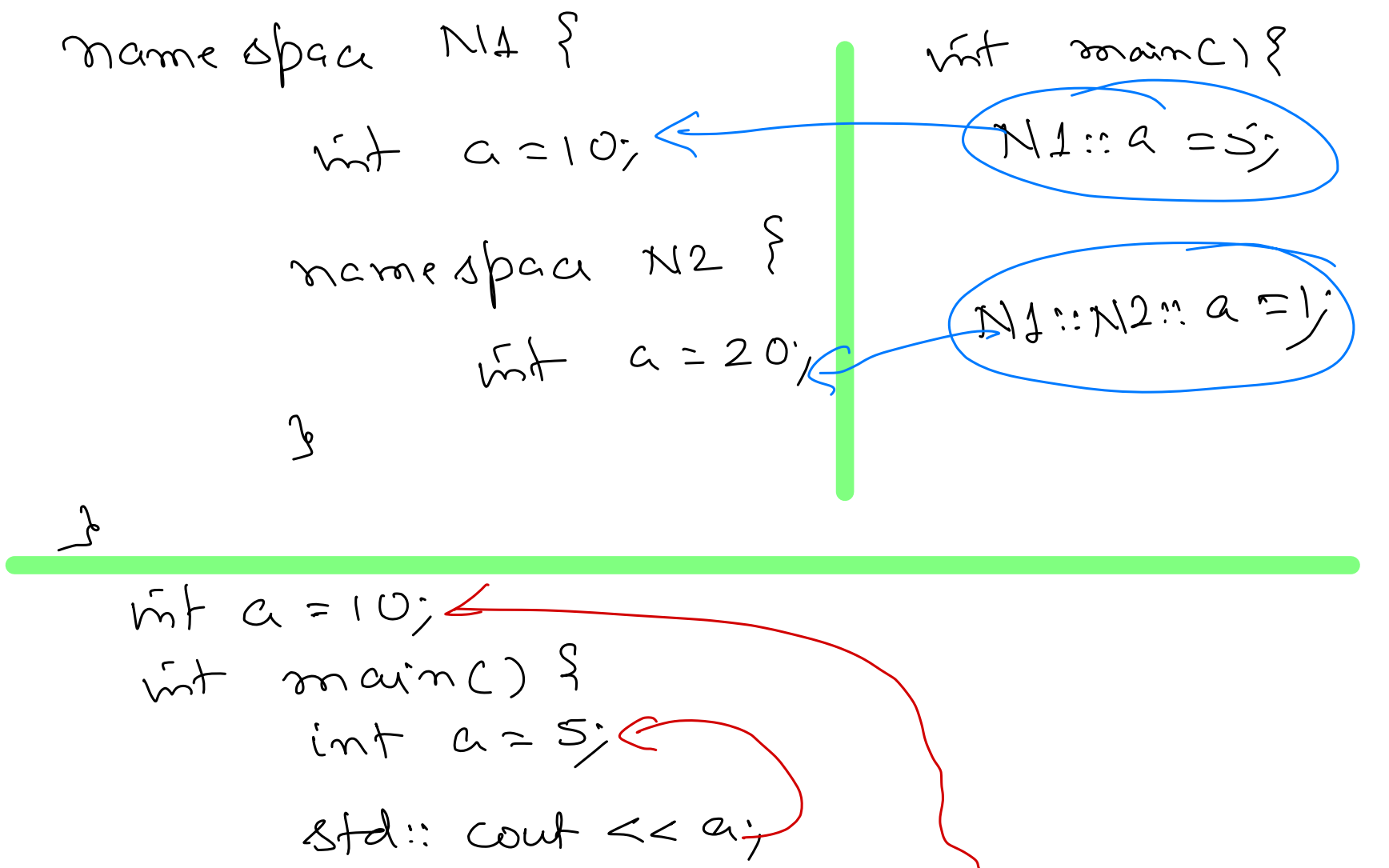
iostream

everything declared in iostream.h but in a namespace - std.

```
namespace N1 {
    int a = 10;
    void F1( ) {
        :
    }
}
namespace N2 {
    int a = 20;
}
```

```
int main( ) {
    N1 :: a = 5;
    N2 :: a = 13;
```

Scope resolution operator


namespace N1 {
int a = 10;
namespace N2 {
int a = 20;
}
}
int main() {
N1::a = 5;
N1::N2::a = 1;
int a = 10;
int main() {
int a = 5;
std:: cout << a;

```
    std :: cout << a;
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main() {

    std :: cout << " ... ";
    :
    :

}
```

⟹ add all identifiers of std namespace to global scope