Initializer list

```
class A
{
   int x, y, z;

   public:
          A() { }
```

```
A(int a, int b, int c)
{
    X = a;
    Y = b;
    z = c;
}
```

→ ①

```
A(int x, int y, int z)
{
   this -> x = x;
   this -> Y = y;
   this -> z = z;
}
```

→ ②

this pointer

Object.member

pointer -> member

instance variable / current object → (x, y, z)

```
A(int x, int y, int z)
{
   (*this).x = x;
   (*this).y = y;
   (*this).z = z;
}
```

→ ③

```
A( int a , int b, int c) : x(a), y(b), z(c)
{


}
```

initializer list

(4)

```
A( int x , int y, int z) : x(x), y(y), z(z)
{

}
```

(5)

Points to be noted :- (When it is used ?)

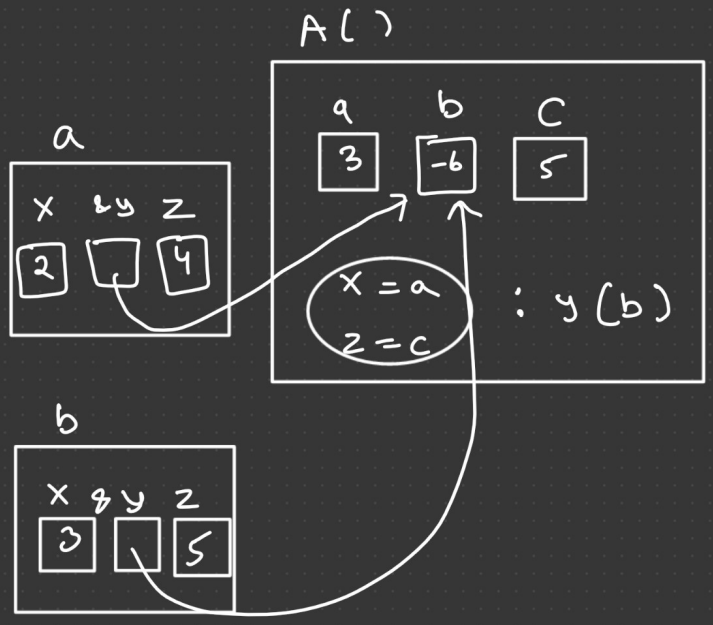
- When constructor's parameter name is same as data member.
- For initialization of non-static const data member.
- For initialization of reference members.
- For initialization of base class.
- For initialization of member object which do not have default constructor.

```
main()
{
→ A a(2,-5,4);
  a.display();
```

2 -5 4

```
A b(3,-6,5)
```

3, -6, 5



Ex :-

```
Class ABC
{
    int x;
  public :
        ABC(int a)
        {
            x = a;
        }
};
```

→ No default constructor.

```
Class A
{
    int x;
    int &y;
    ABC a1;
```

→ It will call default constructor.

It will call parameterized constructor

```
public :
        A(int a, int b) : y(b), a1(3)
        {
            x = a;
            // y = b;  → error
            // a1 = 3; → error
        }
};

int main()
{
    A a(2, 3);
    a.display();

    return 0;
}
```

Soultion is initializer list.

Type Casting

float x = 3.5;

2 + a = c
int + int = int → char

double → float
8 byte → 4 byte

3.5 + 2.0 = 5.5
double double
double → float

1) Implicit Type casting
2) Explicit Type casting.

char a = 97;
int → char

What about user defined ?

2 byte → 4 byte

4 byte → 2 byte
↓
may be data loss

float y = (float) 3.5;
↓
explicit
↓
int *p = (int *)malloc(4);

Student s = 5;

predefined ⟷ predefined

1) predefined → userdefined → Rupee r = 5;

2) Userdefined → predefined → int x = r;

3) Userdefined → Userdefined → Dollar d;
Rupee r = d;

1) Predefined → Userdefined with the help of parameterized Constructor

Rupee a = 5;

```
Rupee (int x)
{
   r = x;
}
```

2) User-defined ⟶ predefined

```
Rupee r;
int x = r;
```

```
class Rupee
{
    int r;

  public:
        operator int()
        {
          return r;
        }
};
```

→ return type is int & Rupee is passed implicitly

3) Userdefined ⟶ Userdefined;

```
Dollar b;
Rupee a = b;
```