Fast Power in C++

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

class FastPower {
public:
    static int fastpower(int a, int b) {
        int res = 1;
        while (b > 0) {
            if (b & 1) {
                res = res * a;
            }
            a = a * a;
            b = b >> 1;
        }
        return res;
    }

    static void main() {
        cout << fastpower(3, 5) << endl;
    }
};

int main() {
    FastPower::main();
    return 0;
}
```

Dry Run Table:

Step	b (binary)	b (decimal)	a	res	Operation	Explanation
0	101	5	3	1		Initial values
1	101	5	3	3	res = res * a	LSB is 1 → multiply res by a
2	10	2	9	3	a = a * a, b >>= 1	Square a → $3^2 = 9$, shift b → b = 2
3	10	2	9	3	(skip multiplication)	LSB is 0 → skip multiplying res
4	1	1	81	3	a = a * a, b >>= 1	$a = 9^2 = 81$, b = 1
5	1	1	81	243	res = res * a	LSB is 1 → res = 3 × 81 = 243
6	0	0			Done	Loop ends

✅ Final Output:

```
243
```

243

GCD in C++

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

class GCD {
public:
    static int gcd(int a, int b) {
        if (b == 0) {
            return a;
        } else {
            return gcd(b, a % b);
        }
    }

    static void main() {
        cout << gcd(30, 36) << endl;
    }
};

int main() {
    GCD::main();
    return 0;
}
```

Function: gcd(a, b)

This uses the rule:

gcd(a, b) = gcd(b, a % b)

…until b == 0.

Dry Run Table for gcd(30, 36)

Call Depth	a	b	a % b	Next Call	Returned Value
1	30	36	30	gcd(36, 30)	
2	36	30	6	gcd(30, 6)	
3	30	6	0	gcd(6, 0)	6
← Return				← back to depth 2	6
← Return				← back to depth 1	6

Final Output:

6

6

Prime Factor in C++

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

class PrimeFactors {
public:
   static void main() {
      int n = 26;
      int n2 = 2;

      while (n2 * n2 <= n) {
         while (n % n2 == 0) {
            n = n / n2;
            cout << n2 << " ";
         }
         n2++;
      }

      if (n != 1) {
         cout << n << " ";
      }
   }
};

int main() {
   PrimeFactors::main();
   return 0;
}
```

Print all **prime factors** of n = 26.

🧠 Logic:

- Start with n2 = 2.
- While n2 * n2 <= n, divide n by n2 as long as it's divisible.
- Increment n2 and repeat.
- After the loop, if n != 1, print the remaining prime factor.

📋 Dry Run Table:

Step	n2	n	n % n2 == 0	Action	Output
1	2	26	Yes	n = 26 / 2 = 13	2
2	2	13	No	n2++	
3	3	13	No	n2++	
4	4	13	No	n2++	
5	5	13	No	n2++	
6	6	13	6*6 > 13 → stop		
7	-	13	-	n != 1 → print n	13

🖨 Final Output:

2 13

2 13

Seive in C++

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

class SeiveofErastostenins {
public:
    static void main() {
        vector<bool> myseive = seive(20);
        for (int i = 0; i < myseive.size(); i++) {
            cout << i << " " << (myseive[i] ? "true" : "false")
<< endl;
        }
    }

    static vector<bool> seive(int n) {
        vector<bool> arr(n + 1, true);
        arr[0] = false;
        arr[1] = false;
        for (int i = 2; i <= sqrt(n); i++) {
            if (arr[i]) {
                for (int j = i * i; j <= n; j += i) {
                    arr[j] = false;
                }
            }
        }
        return arr;
    }
};

    int main() {
        SeiveofErastostenins::main();
        return 0;
    }
```

Sieve of Eratosthenes Dry Run for n = 20

🔢 Step 1: Initialize Boolean Vector

vector<bool> arr(n + 1, true); // arr[0..20] all set to true
arr[0] = false;
arr[1] = false;

📋 Initial Table:

i	isPrime
0	false
1	false
2	true
3	true
4	true
5	true
6	true
7	true
8	true
9	true
10	true
11	true
12	true
13	true
14	true
15	true
16	true
17	true
18	true
19	true
20	true

🔁 Step 2: Outer loop — for (int i = 2; i <= sqrt(n); i++)

- sqrt(20) is ~4.47 → so i goes from 2 to 4

➤ i = 2:

arr[2] == true → mark all multiples of 2 from 4 onward as false

Inner loop (j = i*i; j <= n; j += i) → j = 4, 6, 8, 10, 12, 14, 16, 18, 20

❌ Marked False:

4, 6, 8, 10, 12, 14, 16, 18, 20

➤ i = 3:

arr[3] == true → mark all multiples of 3 from 9 onward as false

j = 9, 12, 15, 18

❌ Marked False:

9, 15

(12 and 18 already marked by i = 2)

➤ i = 4:

arr[4] == false → skip

✅ Final Table After Sieve:

i	isPrime
0	false
1	false
2	true
3	true
4	false
5	true
6	false
7	true
8	false
9	false
10	false
11	true
12	false
13	true
14	false
15	false
16	false
17	true
18	false
19	true
20	false

📜 Output Printed by the Code:

0 false

	1 false 2 true 3 true 4 false 5 true 6 false 7 true 8 false 9 false 10 false 11 true 12 false 13 true 14 false 15 false 16 false 17 true 18 false 19 true 20 false ✅ **Prime Numbers ≤ 20:** **2, 3, 5, 7, 11, 13, 17, 19**
0 false 1 false 2 true 3 true 4 false 5 true 6 false 7 true 8 false 9 false 10 false 11 true 12 false 13 true 14 false 15 false 16 false 17 true 18 false 19 true 20 false	

Trailing Zeroes in C++

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

class TrailingZeroes {
public:
    static void main() {
        int res = 1000;
        int n = 7;
        for (int i = 5; i <= n; i = i * 5) {
            res = res + n / i;
        }
        cout << "zeroes: " << res << endl;
    }
};

int main() {
    TrailingZeroes::main();
    return 0;
}
```

Dry Run for `n = 7`

`i`	`n / i`	`res` (cumulative)
5	7 / 5 = 1	0 + 1 = 1
25	7 / 25 = 0	loop ends

✅ Final answer: **1 trailing zero in 7!**

🖨️ Output (after fixing `res = 0`):

```
zeroes: 1
```

zeroes: 1