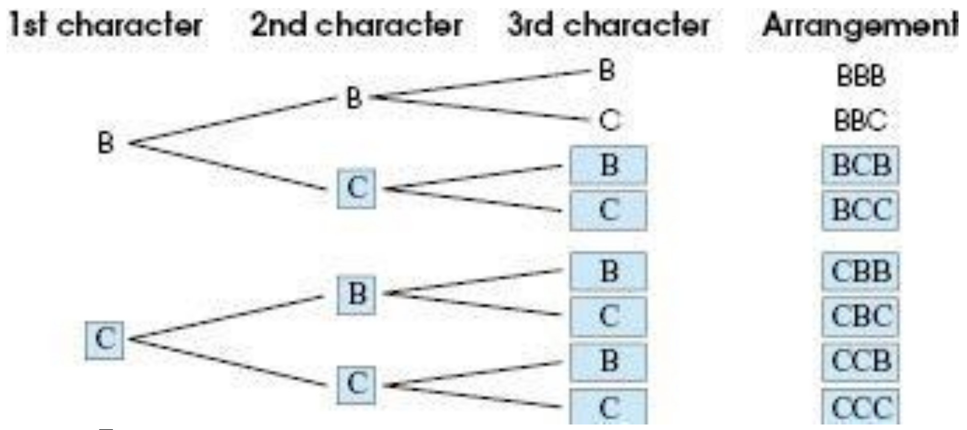
Combinatorics

- The study of **Counting.** Counting usually results in **big values**. Most of problems ask to use % to keep answer small (or BigIntegers)
- In many cases, we can find a **formula**
- Other cases, we need to use **DP** technique.
- There are Some **Counting Principles** to learn
  - E.g. Product Rule, Sum Rule and Inclusion-Exclusion
- Also, some popular **formulas** and sequences
- Please, **read** a discrete mathematics book

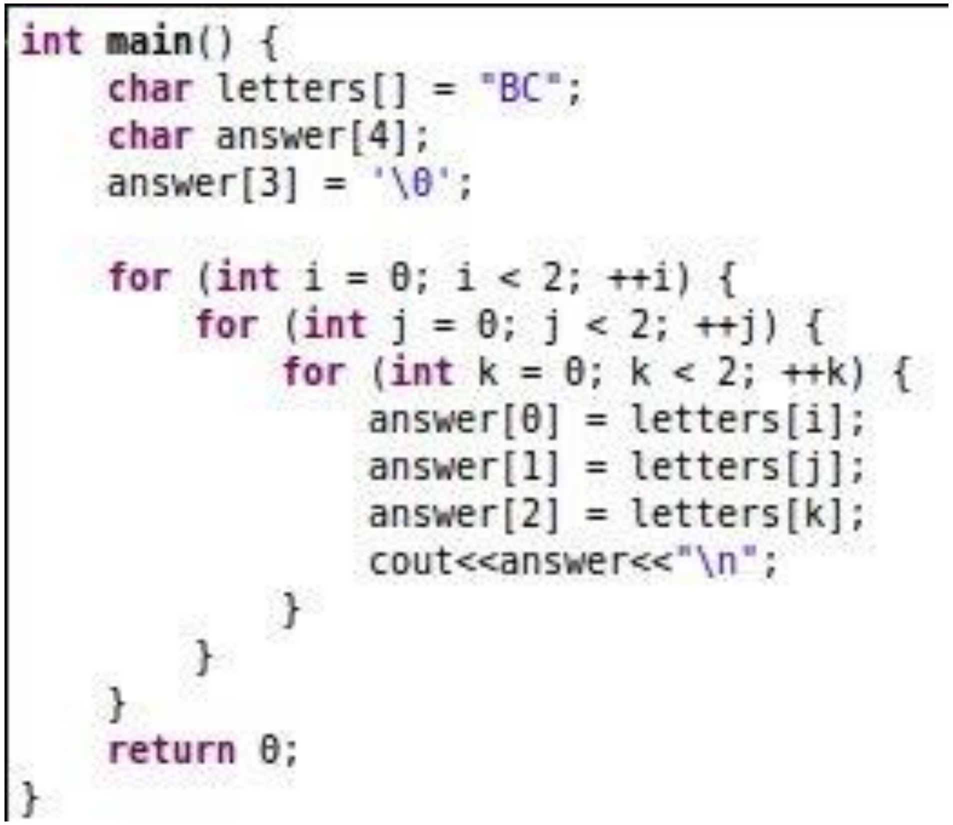
Product rule

- How many words of 3 letters of only B, C?



- Or easily: 2 x 2 x 2 = 8
- General rule: |S1| * |S2| * |S3| … * |Sn|
- 2 ties, 5 jackets, 4 jeans, 2 shoes: Clothings?
  - 2 x 5 x 4 x 2 = 80 dressing styles

Product rule: Code



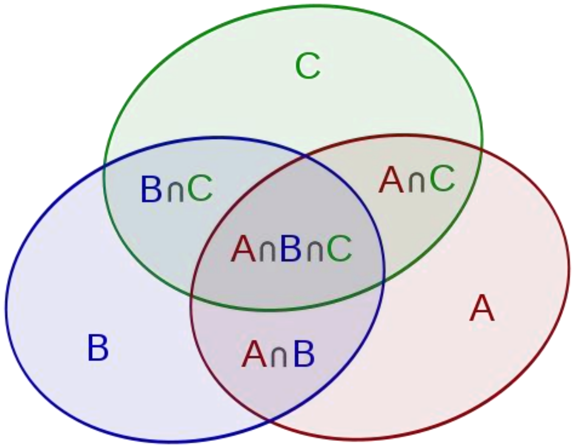
```
int main() {
    char letters[] = "BC";
    char answer[4];
    answer[3] = '\0';

    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 2; ++j) {
            for (int k = 0; k < 2; ++k) {
                answer[0] = letters[i];
                answer[1] = letters[j];
                answer[2] = letters[k];
                cout<<answer<<"\n";
            }
        }
    }
    return 0;
}
```

Sum rule

- words: {bbb, bbc, bcb, bcc, cbb, cbc, ccb, ccc}
- How many words either start with bb or c?
  - 2 + 4 [Notice, no **intersection** between that]
- $|A \cup B| = |A| + |B|$ => [A and B are disjoint]
  - $|A \cup B \cup C \cup D \ldots| = |A| + |B| + |C| + |D| + \ldots$
- How many words either start with cb or c?
  - 2 + 4? Wrong there is overlap: 2 + 4 - 2(intersection) = 4
- How many words either start with cb or end with bc?
  - {cbb, **cbc**} + {bbc, **cbc**} - {**cbc**} = 2+4-1 = 5

Inclusion-Exclusion Principle

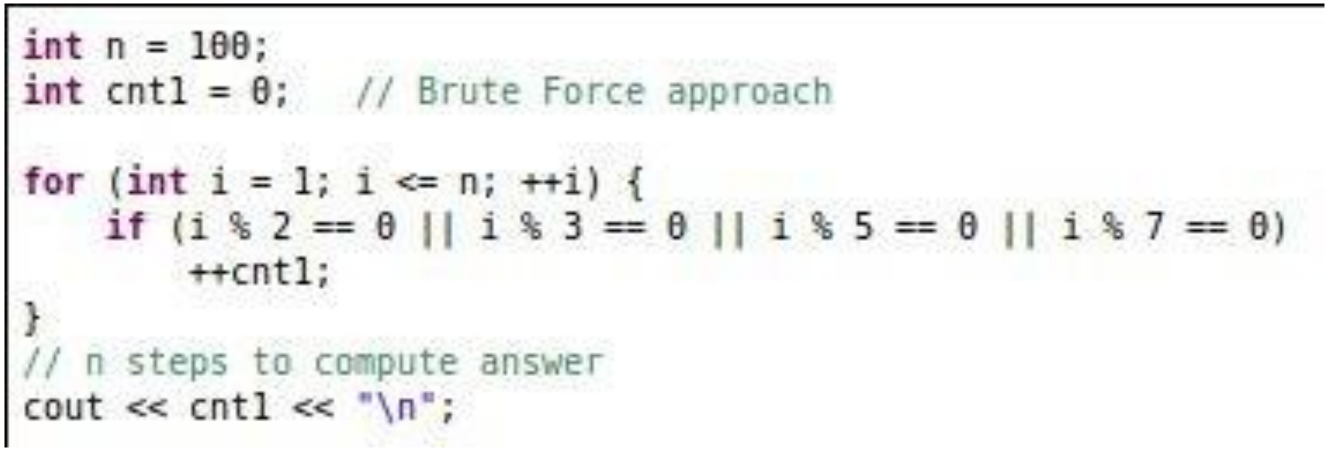
- Most of counting involves **duplicate** counting issue [count item more than once].
- IE principle is a generic sum rule to solve that
- $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|.$
  - 2^3 -1 = 7 subsets (exponential)
- General Computations
  - Enumerate all subsets
  - Compute each one intersection
  - If odd subset add (include) it
  - If even subset subtract (exclude) it



Inclusion-Exclusion Principle

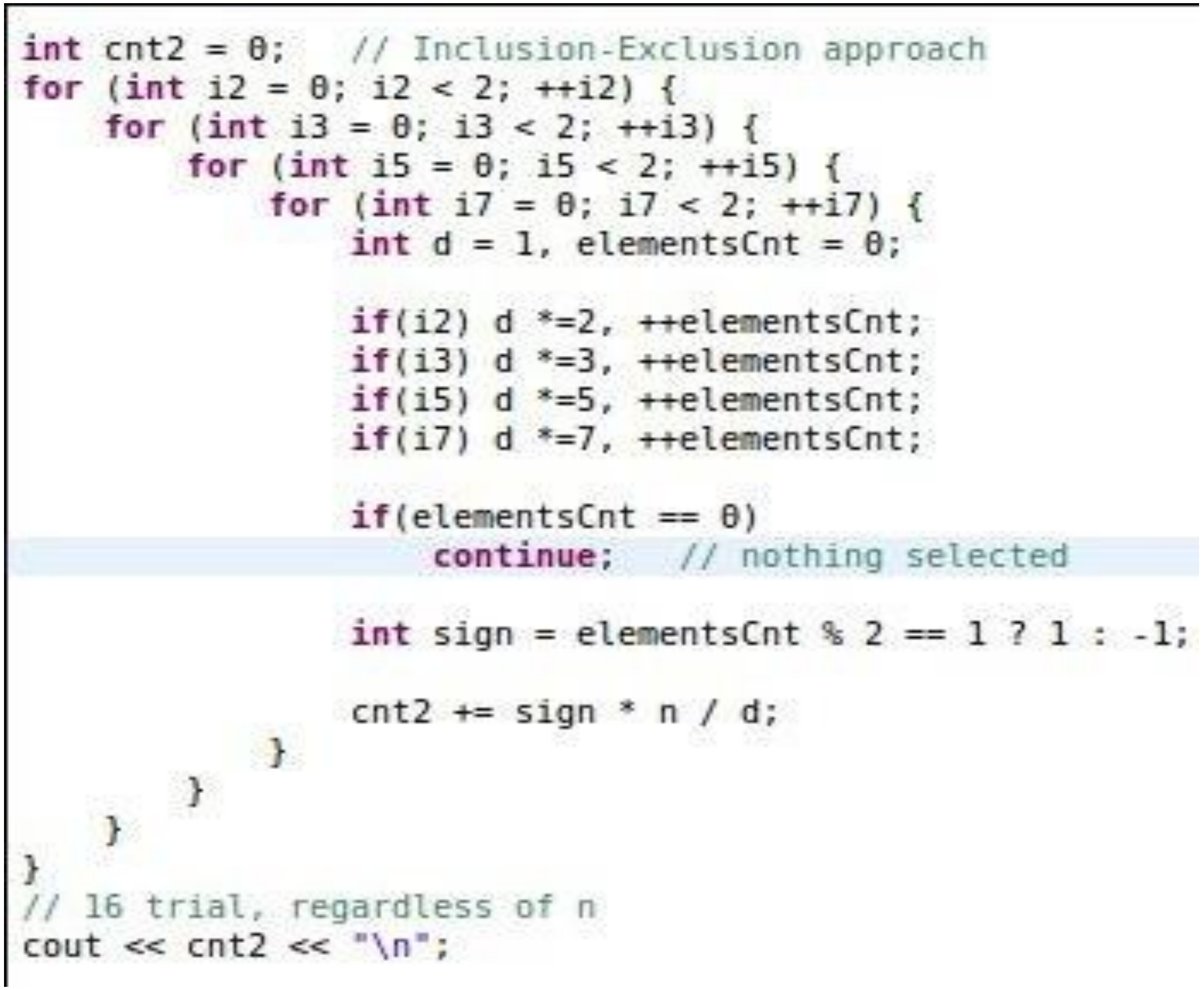
- How many integers in {1,2 ...,100} are divisible by 2, 3, 5 **or** 7?
  - How many divisible by 2? 100 / 2 = 50
  - How many divisible by 3? 100 / 3 = 33
  - How many divisible by 2, 3? 100 / (2*3) = 16
  - How many divisible by 2, 3, 7? 100 / 42 = 2 => **{42, 84}**
- Answer: compute 2^4 -1 terms = 15 terms
  - F(2)+F(3)+F(5)+F(7)
  - -F(2, 3)-F(2, 5)-F(2, 7)-F(3, 5)-F(3, 7)-F(5, 7)
  - +F(2, 3, 5)+F(2, 3, 7)+F(2, 5, 7)+F(3, 5, 7)
  - -F(2, 3, 5, 7)

Inclusion-Exclusion Principle: Code



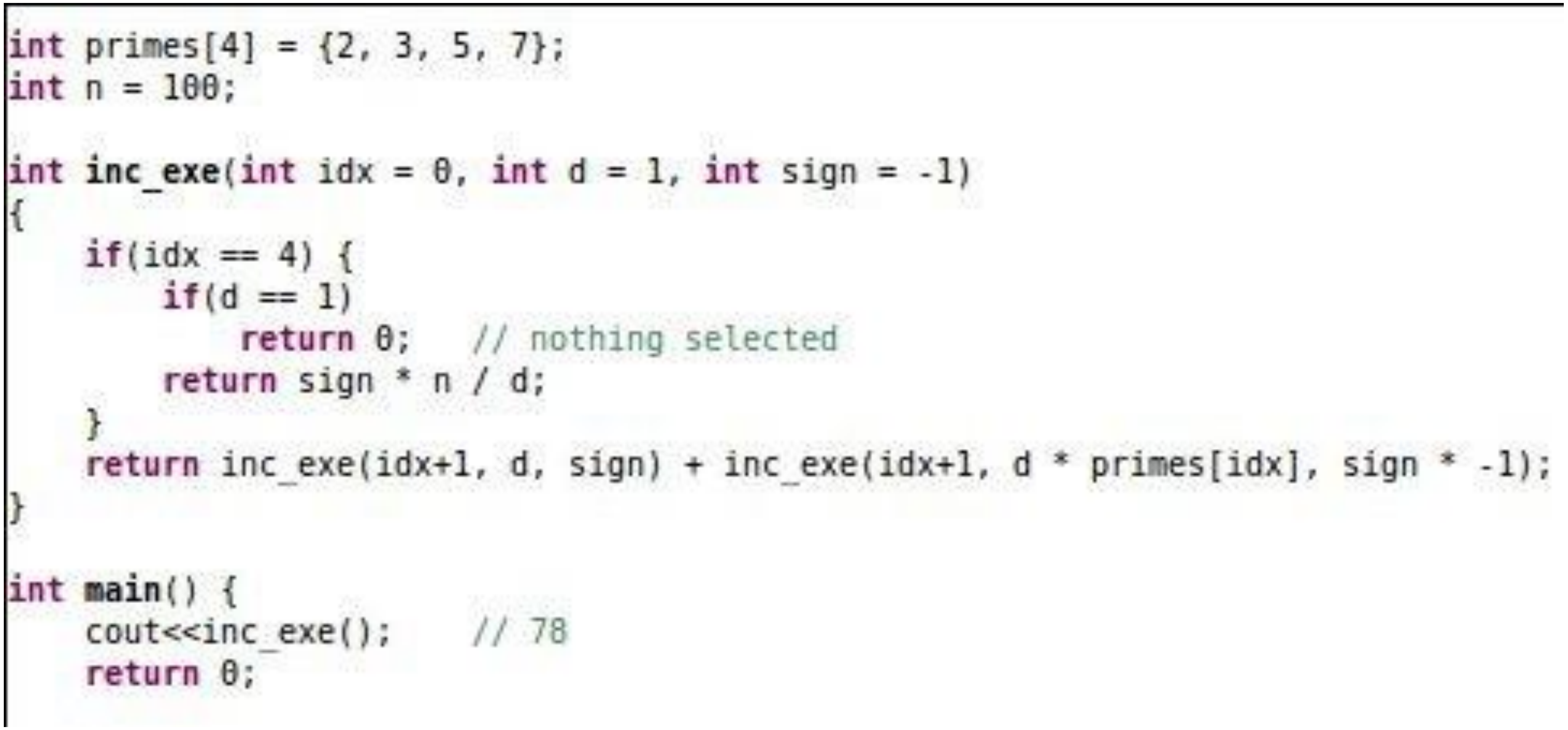
```
int n = 100;
int cnt1 = 0;   // Brute Force approach

for (int i = 1; i <= n; ++i) {
    if (i % 2 == 0 || i % 3 == 0 || i % 5 == 0 || i % 7 == 0)
        ++cnt1;
}
// n steps to compute answer
cout << cnt1 << "\n";
```

Inclusion-Exclusion Principle: Code



```
int cnt2 = 0;    // Inclusion-Exclusion approach
for (int i2 = 0; i2 < 2; ++i2) {
    for (int i3 = 0; i3 < 2; ++i3) {
        for (int i5 = 0; i5 < 2; ++i5) {
            for (int i7 = 0; i7 < 2; ++i7) {
                int d = 1, elementsCnt = 0;

                if(i2) d *=2, ++elementsCnt;
                if(i3) d *=3, ++elementsCnt;
                if(i5) d *=5, ++elementsCnt;
                if(i7) d *=7, ++elementsCnt;

                if(elementsCnt == 0)
                    continue;    // nothing selected

                int sign = elementsCnt % 2 == 1 ? 1 : -1;

                cnt2 += sign * n / d;
            }
        }
    }
}
// 16 trial, regardless of n
cout << cnt2 << "\n";
```

Inclusion-Exclusion Principle: Code

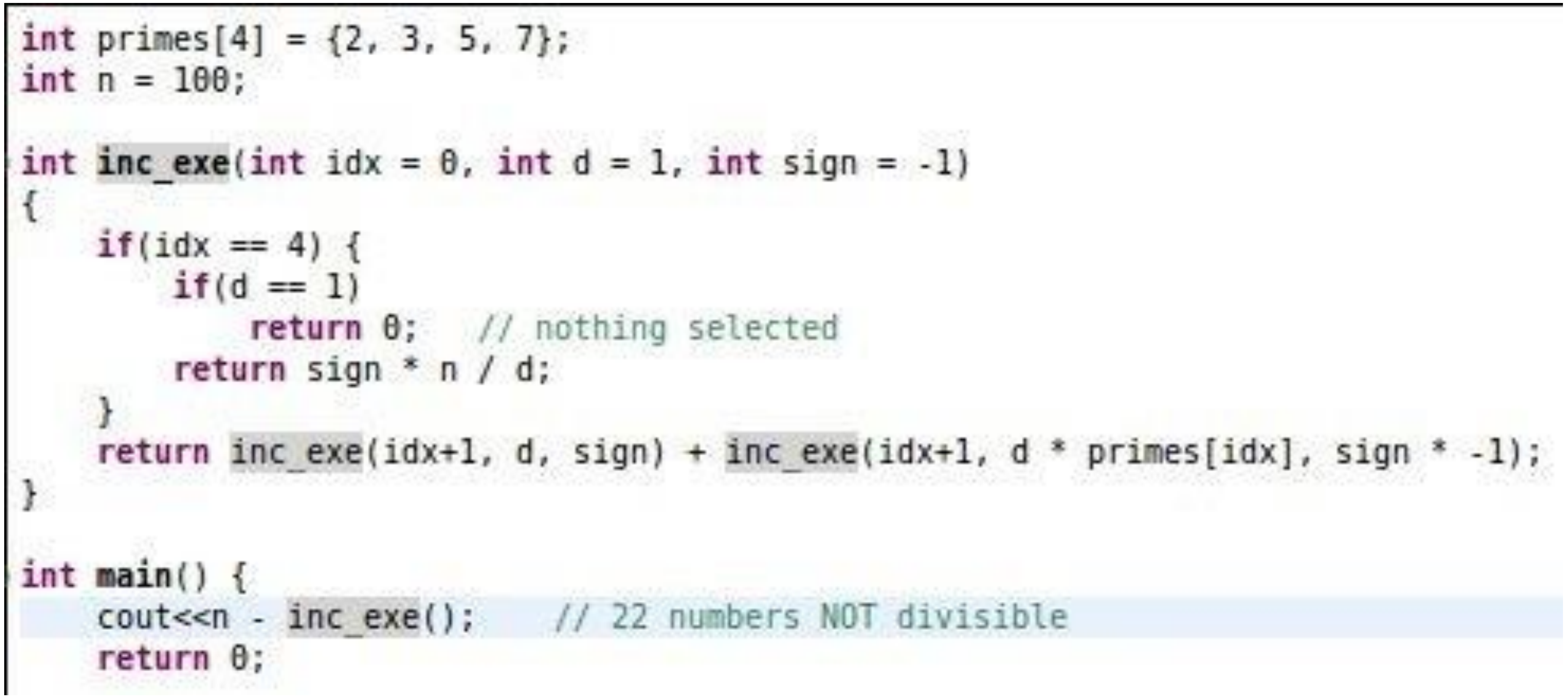


```
int primes[4] = {2, 3, 5, 7};
int n = 100;

int inc_exe(int idx = 0, int d = 1, int sign = -1)
{
    if(idx == 4) {
        if(d == 1)
            return 0;   // nothing selected
        return sign * n / d;
    }
    return inc_exe(idx+1, d, sign) + inc_exe(idx+1, d * primes[idx], sign * -1);
}

int main() {
    cout<<inc_exe();    // 78
    return 0;
```

Inclusion-Exclusion Principle

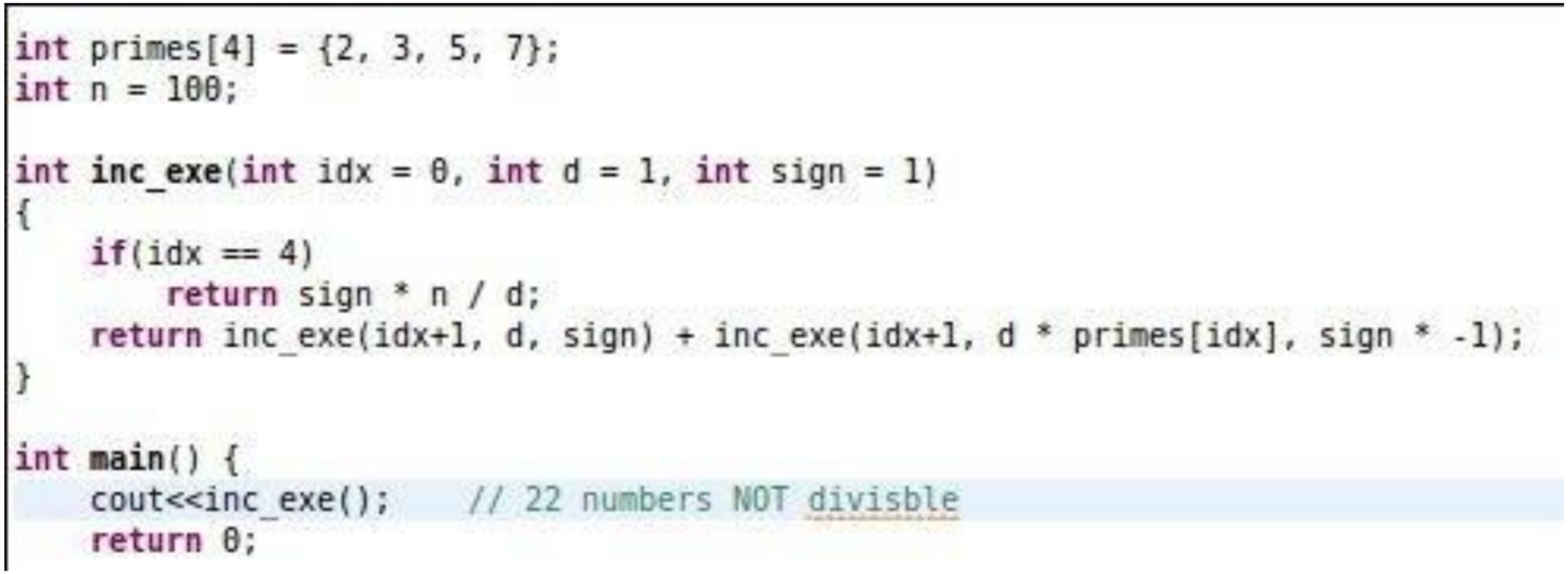
- How many integers in {1,2 ...,100} are **NOT** divisible by 2, 3, 5 **or** 7?
- We can change F function to be F(some numbers): How many NOT divisible
- Generally, the problem or its negate may be easier to tackle. **Complement** thinking is a better approach: E.g. 100 - SumDivisible
- Homework: How many integers in {1,2 ..., 100} are divisible by 2, 3, 8 **or** 10?

Inclusion-Exclusion Principle: Code



```cpp
int primes[4] = {2, 3, 5, 7};
int n = 100;

int inc_exe(int idx = 0, int d = 1, int sign = -1)
{
    if(idx == 4) {
        if(d == 1)
            return 0;   // nothing selected
        return sign * n / d;
    }
    return inc_exe(idx+1, d, sign) + inc_exe(idx+1, d * primes[idx], sign * -1);
}

int main() {
    cout<<n - inc_exe();    // 22 numbers NOT divisible
    return 0;
```

Inclusion-Exclusion Principle: Code



```
int primes[4] = {2, 3, 5, 7};
int n = 100;

int inc_exe(int idx = 0, int d = 1, int sign = 1)
{
    if(idx == 4)
        return sign * n / d;
    return inc_exe(idx+1, d, sign) + inc_exe(idx+1, d * primes[idx], sign * -1);
}

int main() {
    cout<<inc_exe();    // 22 numbers NOT divisble
    return 0;
```

The Division Rule

- A food table with 3 chairs. Given 3 persons, in how many ways we can seat them?
  - 1 2 3, 1 3 2, 2 1 3, 2 3 1, 3 1 2, 3 2 1 => 6 ways
  - Wrong! 123 same as 231 same as 312 [by making 1 shift]
  - So given 1 seating, we can generate 3 similar seatings
  - so answer is 6 / 3 = 2 .. or generally n! / n = n-1!
- Division rule: solution = m / d, where each d elements of m are same (e.g. symmetric)
- In an 8x8 chess, how many ways to put rock?
  - Product rule: 8 rows x 8 cols = 64 ways

The Division Rule

- In an 8x8 chess, how many ways to put 2 rocks, with no shared rows or columns?
  - First piece has 64 choices.. then 1 row & 1 col are blocked
  - So we have 7x7= 49 choices for 2nd rock. Total 64*49
  - Wrong! part of your solution {(0,0), (1,1)}, {(1,1), (0,0)}
  - **Symmetry** of each 2 rocks. Answer: 64 * 49 / 2
- When **generating** the actual results, **symmetric** relationships gives faster code
  - Generate the main part (major processing time)
  - Use that to generate the symmetric answer
  - See USACO problem: Checker Challenge