[C++] 코딩테스트에서 알고리즘 구현에 자주 사용되는 함수 50선!

작성 : Zeromini

디스코드 채널(취업폭격기 제로미니 IT 취업 공부방) : https://discord.gg/yDWrPjqv

1. `std::vector`: 동적 배열을 나타냅니다.

```
std::vector<int> v = {1, 2, 3, 4, 5};
```

2. `std::array`: 고정 크기 배열을 나타냅니다.

```
std::array<int, 5> a = {1, 2, 3, 4, 5};
```

3. `std::string`: 문자열을 나타냅니다.

```
std::string s = "Hello, World!";
```

4. `std::map`: 키-값 쌍을 저장하는 자료 구조를 나타냅니다.

```
std::map<std::string, int> m;
m["apple"] = 1;
```

5. `std::pair`: 두 요소를 묶어서 하나의 쌍으로 만듭니다.

```
std::pair<int, std::string> p = {1, "apple"};
```

6. `std::set`: 중복 없는 요소의 집합을 나타냅니다.

```
std::set<int> s = {1, 2, 3, 4, 5};
```

7. `std::unordered_map`: 키-값 쌍을 저장하는 해시 테이블을 나타냅니다.

```
std::unordered_map<std::string, int> m;
m["apple"] = 1;
```

8. `std::unordered_set`: 중복 없는 요소의 집합을 나타내는 해시 테이블을 나타냅니다.

```
std::unordered_set<int> s = {1, 2, 3, 4, 5};
```

9. `std::queue`: 큐를 나타냅니다.

```
std::queue<int> q;
q.push(1);
```

10. `std::priority_queue` : 우선순위 큐를 나타냅니다.

```
std::priority_queue<int> pq;
pq.push(1);
```

11. `std::stack` : 스택을 나타냅니다.

```
std::stack<int> st;
st.push(1);
```

12. `std::list` : 양방향 연결 리스트를 나타냅니다.

```
std::list<int> l = {1, 2, 3, 4, 5};
```

13. `std::deque` : 양쪽 끝에서 삽입 및 삭제가 가능한 리스트를 나타냅니다.

```
std::deque<int> dq = {1, 2, 3, 4, 5};
```

14. `std::cin` : 표준 입력을 받습니다.

```
int x;
std::cin >> x;
```

15. `std::cout` : 표준 출력을 합니다.

```
int x = 1;
std::cout << x << std::endl;
```

16. `std::endl` : 개행(줄 바꿈)을 나타냅니다.

```
std::cout << "Hello, World!" << std::endl;
```

17. `std::sort()`: 정렬 함수입니다.

```
std::vector<int> v = {5, 2, 3, 1, 4};
std::sort(v.begin(), v.end());
```

18. `std::reverse()`: 순서를 반대로 뒤집는 함수입니다.

```
std::vector<int> v = {1, 2, 3, 4, 5};
std::reverse(v.begin(), v.end());
```

19. `std::min_element()`: 최소값을 찾는 함수입니다.

```
std::vector<int> v = {5, 2, 3, 1, 4};
auto min_value = *std::min_element(v.begin(), v.end());
```

20. `std::max_element()`: 최대값을 찾는 함수입니다.

```
std::vector<int> v = {5, 2, 3, 1, 4};
auto max_value = *std::max_element(v.begin(), v.end());
```

21. `std::lower_bound()`: 이진 탐색으로 특정 값 이상이 처음 나타나는 위치를 찾는 함수입니다.

```
std::vector<int> v = {1, 2, 4, 4, 5, 6};
auto lower = std::lower_bound(v.begin(), v.end(), 4);
```

22. `std::upper_bound()`: 이진 탐색으로 특정 값보다 큰 값이 처음 나타나는 위치를 찾는 함수입니다.

```
std::vector<int> v = {1, 2, 4, 4, 5, 6};
auto upper = std::upper_bound(v.begin(), v.end(), 4);
```

23. `std::unique()`: 범위 내에서 중복된 원소를 제거합니다.

```
std::vector<int> v = {1, 1, 2, 2, 3, 3};
auto last = std::unique(v.begin(), v.end());
v.erase(last, v.end());
```

24. `std::find()`: 범위 내에서 특정 값이 처음 등장하는 위치를 찾습니다.

```
std::vector<int> v = {1, 2, 3, 4, 5};
auto result = std::find(v.begin(), v.end(), 3);
if (result != v.end()) {
    std::cout << "Found" << std::endl;
} else {
    std::cout << "Not Found" << std::endl;
}
```

25. `std::next_permutation()`: 주어진 범위에 대해 다음 순열을 생성합니다.

```
std::vector<int> v = {1, 2, 3};
do {
    for (int i : v) {
        std::cout << i <<
```

26. `std::accumulate()`: 범위 내의 모든 요소들의 합을 계산합니다.

```
std::vector<int> v = {1, 2, 3, 4, 5};
int sum = std::accumulate(v.begin(), v.end(), 0);  // sum == 15
```

27. `std::fill()`: 특정 범위에 대해 값을 채웁니다.

```
std::vector<int> v(5);
std::fill(v.begin(), v.end(), 1);  // v: {1, 1, 1, 1, 1}
```

28. `std::count()`: 범위 내에서 특정 값의 개수를 셉니다.

```
std::vector<int> v = {1, 1, 2, 2, 3, 3};
int count = std::count(v.begin(), v.end(), 1);  // count == 2
```

29. `std::swap()`: 두 변수의 값을 교환합니다.

```
int a = 1;
int b = 2;
std::swap(a, b);  // a == 2, b == 1
```

30. `std::make_pair()`: 두 요소를 묶어 pair를 만듭니다.

```
auto p = std::make_pair(1, 'a');  // p: (1, 'a')
```

31. `std::tie()`: 여러 변수를 한 번에 할당할 수 있게 합니다.

```
std::pair<int, char> p = std::make_pair(1, 'a');
int a;
char b;
std::tie(a, b) = p;  // a == 1, b == 'a'
```

32. `std::stoi()`: 문자열을 정수로 변환합니다.

```
std::string s = "123";
int num = std::stoi(s);  // num == 123
```

33. `std::to_string()`: 정수를 문자열로 변환합니다.

```
int num = 123;
std::string s = std::to_string(num);  // s == "123"
```

34. `std::max()`: 두 요소 중 더 큰 값을 반환합니다.

```
int a = 1;
int b = 2;
int max_val = std::max(a, b);  // max_val == 2
```

35. `std::min()`: 두 요소 중 더 작은 값을 반환합니다.

```
int a = 1;
int b = 2;
int min_val = std::min(a, b);  // min_val == 1
```

36. `std::pow()`: 거듭제곱을 계산합니다.

```
double base = 2.0;
int exponent = 3;
double result = std::pow(base, exponent);  // result == 8.0
```

37. `std::sqrt()`: 제곱근을 계산합니다.

```
double num = 4.0;
double result = std::sqrt(num);  // result == 2.0
```

38. `std::abs()`: 절대값을 계산합니다.

```
int num = -1;
int abs_val = std::abs(num);  // abs_val == 1
```

39. `std::round()`: 반올림한 값을 반환합니다.

```
double num = 3.6;
double rounded = std::round(num);  // rounded == 4.0
```

40. `std::ceil()`: 올림한 값을 반환합니다.

```
double num = 3.1;
double ceiled = std::ceil(num);  // ceiled == 4.0
```

41. `std::floor()`: 내림한 값을 반환합니다.

```
double num = 3.9;
double floored = std::floor(num);  // floored == 3.0
```

42. `std::stod()`: 문자열을 double로 변환합니다.

```
std::string s = "3.14";
double num = std::stod(s);  // num == 3.14
```

43. `std::regex_match()`, `std::regex_search()`: 정규 표현식을 사용하여 문자열에서 패턴을 찾습니다.

```
std::string s = "Hello, World!";
std::regex e ("(\\\\w+)\\\\W+(\\\\w+)");
bool match = std::regex_match(s, e);  // match == true
```

44. `std::nth_element()`: n번째 위치의 원소를 찾아 정렬합니다.

```
std::vector<int> v = {5, 6, 4, 3, 2, 6, 7, 9, 3};
std::nth_element(v.begin(), v.begin() + v.size()/2, v.end());
```

45. `std::push_heap()`, `std::pop_heap()`: 힙 연산을 수행합니다.

```
std::vector<int> v = {10, 20, 30, 5, 15};
std::make_heap(v.begin(), v.end());
std::cout << "initial max heap : " << v.front() << std::endl;
std::pop_heap(v.begin(), v.end());
v.pop_back();
std::cout << "max heap after pop : " << v.front() << std::endl;
```

46. `std::distance()`: 두 반복자 사이의 거리를 반환합니다.

```
std::vector<int> v = {3, 1, 4, 1, 5};
std::cout << "distance(first, last): " << std::distance(v.begin(), v.end()) <<
 '\\n';
```

47. `std::next()`, `std::prev()`: 반복자를 한 단계 앞으로 또는 뒤로 이동시킵니다.

```
std::vector<int> v = {1, 2, 3, 4, 5};
auto it = v.begin();
it = std::next(it, 2);  // it이 가리키는 값은 3
it = std::prev(it, 1);  // it이 가리키는 값은 2
```

48. `std::iota()`: 특정 범위의 숫자를 채워넣습니다.

```
std::vector<int> v(5);
std::iota(v.begin(), v.end(), 1);  // v: {1, 2, 3, 4, 5}
```

49. `std::set_intersection()`: 두 집합의 교집합을 계산합니다.

```
std::vector<int> v1 = {1, 2, 3, 4, 5};
std::vector<int> v2 = {4, 5, 6, 7, 8};
std::vector<int> v_intersection;
std::set_intersection(v1.begin(), v1.end(), v2.begin(), v2.end(), std::back_insert
er(v_intersection));
// v_intersection: {4, 5}
```

50. `std::set_union()`: 두 집합의 합집합을 계산합니다.

```
std::vector<int> v1 = {1, 2, 3, 4, 5};
std::vector<int> v2 = {4, 5, 6, 7, 8};
std::vector<int> v_union;
std::set_union(v1.begin(), v1.end(), v2.begin(), v2.end(), std::back_inserter(v_un
ion));
// v_union: {1, 2, 3, 4, 5, 6, 7, 8}
```