41. First Missing Positive

C++ solution:

```
class Solution {
public:
    int firstMissingPositive(vector<int>& nums) {
        int n = nums.size();

        vector<int> exist(n + 1, 0);

        for (int num : nums)
        {
            if (num > 0 && num <= n)
            {
                exist[num] = 1;
            }
        }

        for (int i = 1; i <= n; ++i)
        {
            if (exist[i] == 0)
            {
                return i;
            }
        }

        return n + 1;
    }
};
```

42. Trapping Rain Water

C++ solution:

```
class Solution {
public:
    int trap(vector<int>& height) {

        int n = height.size();

        if (n < 3)
        {
            return 0;
        }

        int max_height = 0, max_index = 0;
        for (int i = 0; i < n; ++i)
        {
            if (height[i] > max_height)
            {
                max_height = height[i];
                max_index = i;
            }
        }

        int left = -1, right = -1, count = 0;
        for (int i = 0; i <= max_index; ++i)
        {
            if (left < 0)
            {
                if (height[i] > 0)
                {
                    left = i;
                }
            }
            else if (height[i] >= height[left])
            {
                count += sum(height, left, i);
                left = i;
            }
        }

        for (int i = n - 1; i >= max_index; --i)
        {
            if (right < 0)
            {
                if (height[i] > 0)
                right = i;
            }
            else if (height[i] >= height[right])
            {
                count += sum(height, i, right);
                right = i;
            }
        }

        return count;
    }

    int sum(vector<int>& height, int left, int right)
    {
        int max_height = min(height[left], height[right]);
        int result = 0;
        for (int i = left + 1; i < right; ++i)
        {
            result += (max_height - height[i]);
        }

        return result;
    }
};
```

43. Multiply Strings

C++ solution:

```
class Solution {
public:
    string multiply(string num1, string num2) {

        if (num1 == "0" || num2 == "0")
        {
            return "0";
        }

        int n1 = num1.length();
        int n2 = num2.length();
        string result = "0", current;
        int n = 0;

        for (int i = n2 - 1; i >= 0; --i)
        {
            current = multiply_string_char(num1, num2[i]);

            for (int j = 0; j < n; ++j)
            {
                current.insert(current.end(), '0');
            }
            ++n;

            result = add(result, current);
        }

        return result;
    }

    string multiply_string_char(string num1, char num2)
    {
        int n1 = num1.length();
        int product = 0;
        int extra = 0;
        char ch;
        string result;

        for (int i1 = n1 - 1; i1 >= 0; --i1)
        {
            product = (num1[i1] - '0') * (num2 - '0') + extra;

            ch = product % 10 + '0';
            extra = product / 10;

            result.insert(result.begin(), ch);
        }

        if (extra > 0)
        {
            result.insert(result.begin(), extra + '0');
        }

        return result;
    }

    string add(string num1, string num2) {

        int n1 = num1.length();
        int n2 = num2.length();
        int sum = 0;
        int extra = 0;
        char ch;
        string result;
        int i1 = n1 - 1, i2 = n2 - 1;

        for (; i1 >= 0 && i2 >= 0; --i1, --i2)
        {
            sum = num1[i1] - '0' + num2[i2] - '0' + extra;

            ch = sum % 10 + '0';
            extra = sum / 10;

            result.insert(result.begin(), ch);
        }

        for (; i1 >= 0; --i1)
        {
            sum = num1[i1] - '0' + extra;

            ch = sum % 10 + '0';
            extra = sum / 10;
            result.insert(result.begin(), ch);
        }

        for (; i2 >= 0; --i2)
        {
            sum = num2[i2] - '0' + extra;

            ch = sum % 10 + '0';
            extra = sum / 10;

            result.insert(result.begin(), ch);
        }

        if (extra == 1)
        {
            result.insert(result.begin(), '1');
        }

        return result;
    }
};
```

[44. Wildcard Matching]

C++ solution:

```
class Solution {
public:
    bool isMatch(string s, string p) {
        int ns = s.length();
        int np = p.length();

        bool matches[ns + 1][np + 1];
        memset(matches, false, sizeof(matches));
        matches[0][0] = true;

        for (int i = 0; i < np; ++i)
        {
            if (p[i] == '*')
            {
                matches[0][i + 1] = true;
            }
            else
            {
                break;
            }
        }

        for (int is = 0; is < ns; ++is)
        {
            for (int ip = 0; ip < np; ++ip)
            {
                if (s[is] == p[ip] || p[ip] == '?')
                {
                    matches[is + 1][ip + 1] = matches[is][ip];
                }
                else if (p[ip] == '*')
                {
                    matches[is + 1][ip + 1] = matches[is + 1][ip] || matches[is][ip +
1];
                }
            }
        }

        return matches[ns][np];
    }
};
```

45. Jump Game II

C++ solution:

```
class Solution {
public:
    int jump(vector<int>& nums) {
        int n = nums.size();
        if (n < 2)
        {
            return 0;
        }

        int steps = 0;
        int min_position = 0;
        int max_position = 0;
        int next_max_position = 0;

        while (max_position < n - 1)
        {
            ++steps;
            for (int i = min_position; i <= max_position; ++i)
            {
                if (i + nums[i] > next_max_position)
                {
                    next_max_position = i + nums[i];
                    if (next_max_position > n - 2)
                    {
                        break;
                    }
                }
            }
            min_position = max_position + 1;
            max_position = next_max_position;
        }

        return steps;
    }
};
```

46. Permutations

C++ solution:

```
class Solution {
public:
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> result;
        permuteFromPosition(nums, 0, result);
        return result;
    }

    void permuteFromPosition(vector<int>& nums, int start_index, vector<vector<int>>&
result) {

        int n = nums.size();

        if (start_index >= n - 1)
        {
            result.insert(result.end(), nums);
            return;
        }

        permuteFromPosition(nums, start_index + 1, result);

        for (int i = start_index + 1; i < n; ++i)
        {
            nums[i] = nums[i] ^ nums[start_index];
            nums[start_index] = nums[i] ^ nums[start_index];
            nums[i] = nums[i] ^ nums[start_index];
            permuteFromPosition(nums, start_index + 1, result);
            nums[i] = nums[i] ^ nums[start_index];
            nums[start_index] = nums[i] ^ nums[start_index];
            nums[i] = nums[i] ^ nums[start_index];
        }
    }
};
```

47. Permutations II

C++ solution:

```
class Solution {
public:
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        vector<vector<int>> result;
        permuteFromPosition(nums, 0, result);
        return result;
    }

    void permuteFromPosition(vector<int>& nums, int start_index, vector<vector<int>>&
result) {

        int n = nums.size();

        if (start_index >= n - 1)
        {
            result.insert(result.end(), nums);
            return;
        }

        permuteFromPosition(nums, start_index + 1, result);

        unordered_set<int> swapped_nums;
        swapped_nums.insert(nums[start_index]);

        for (int i = start_index + 1; i < n; ++i)
        {
            if (swapped_nums.find(nums[i]) != swapped_nums.end())
            {
                continue;
            }
            swapped_nums.insert(nums[i]);
            nums[i] = nums[i] ^ nums[start_index];
            nums[start_index] = nums[i] ^ nums[start_index];
            nums[i] = nums[i] ^ nums[start_index];
            permuteFromPosition(nums, start_index + 1, result);
            nums[i] = nums[i] ^ nums[start_index];
            nums[start_index] = nums[i] ^ nums[start_index];
            nums[i] = nums[i] ^ nums[start_index];
        }
    }
};
```

48. Rotate Image

C++ solution:

```
class Solution {
public:
```

```
void rotate(vector<vector<int>>& matrix) {

    int n = matrix.size();
    bool changed[n][n];
    memset(changed, false, sizeof(changed));

    int temp = 0, r = 0, c = 0, temp_r = 0, temp_c = 0, original_r = 0;

    while (true)
    {
        temp = matrix[r][c];
        temp_r = r;
        temp_c = c;
        while (!changed[r][c])
        {
            changed[r][c] = true;
            original_r = r;
            r = n - c - 1;
            c = original_r;
            if (r == temp_r && c == temp_c)
            {
                matrix[original_r][n - r - 1] = temp;
                break;
            }
            matrix[original_r][n - r - 1] = matrix[r][c];
        }

        while (changed[r][c])
        {
            if (c < n - 1)
            {
                ++c;
            }
            else
            {
                ++r;
                if (r == n)
                {
                    break;
                }
                c = 0;
            }
        }

        if (r == n)
        {
            break;
        }
        }
    }
};
```

49. Group Anagrams

C++ solution:

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        map<multiset<char>, vector<string>> set_strs;

        for (string s : strs)
        {
            multiset<char> set_chars;
            for (char c : s)
            {
                set_chars.insert(c);
            }

            set_strs[set_chars].insert(set_strs[set_chars].end(), s);
        }

        vector<vector<string>> result;
        for (auto iter : set_strs)
        {
            result.insert(result.end(), iter.second);
        }

        return result;
    }
};
```

50. Pow(x, n)

C++ solution:

```
class Solution {
public:
    double myPow(double x, int n) {

        double result = 1.0;

        if (n == 0 || x == 1)
        {
            return result;
        }
        if (x == -1)
        {
            return (n % 2 == 0) ? 1 : -1;
        }

        double original_x = x;

        while (n > 0)
        {
            if (n & 1)
            {
                result *= x;

                if (abs(result) < 0.000005)
                {
                    return 0;
                }
            }
            x *= x;
            n >>= 1;
        }

        if (n == INT_MIN)
        {
            original_x *= original_x;
            n >>= 1;
        }

        int _n = -n;

        while (_n > 0)
        {
            if (_n & 1)
            {
                result /= original_x;

                if (abs(result) < 0.000005)
                {
                    return 0;
                }
            }
            original_x *= original_x;
            _n >>= 1;
        }

        return result;
    }
```

```
};
```