**Leetcode Note by Yang**

[**Section 1. Bit Manipulation** 1](#_Toc2184663)

[**136. Single number I (E)** 1](#_Toc2184664)

[**137. Single number II (M)** 1](#_Toc2184665)

[**190. Reverse bits (E)** 3](#_Toc2184666)

[**191. Number of 1 bits (E)** 4](#_Toc2184667)

[**201. Bit wise AND of number range (M)** 5](#_Toc2184668)

[**231. Power of two (E)** 6](#_Toc2184669)

[**260. Single Number III (M)** 7](#_Toc2184670)

[**268. Missing number (M)** 8](#_Toc2184671)

[**318. Maximum product of word lengths (M)** 9](#_Toc2184672)

[**Section 2. Array** 11](#_Toc2184673)

[**015. 3 sum (M)** 11](#_Toc2184674)

[**016. 3 sum closest (M)** 12](#_Toc2184675)

[**018. 4 sum (M)** 14](#_Toc2184676)

[**026. Remove duplicates from sorted array (E)** 15](#_Toc2184677)

[**027. Remove element (E)** 16](#_Toc2184678)

[**031. Next permutation (M)** 17](#_Toc2184679)

[**041. First missing positive (H)** 18](#_Toc2184680)

[**048. Rotate image (M)** 19](#_Toc2184681)

[**Section 3. String** 21](#_Toc2184682)

[**005. Longest palindromic substring (M)** 21](#_Toc2184683)

[**006. ZigZag conversion (E)** 22](#_Toc2184684)

[**008. String to integer (E)** 24](#_Toc2184685)

[**014. Longest common prefix (E)** 26](#_Toc2184686)

[**028. Implement strStr() (E)** 27](#_Toc2184687)

[**038. Count and say (E)** 28](#_Toc2184688)

[**043. Multiply strings (M)** 29](#_Toc2184689)

[**058. Length of last word (E)** 30](#_Toc2184690)

[**067. Add binary (E)** 31](#_Toc2184691)

[**068. Text justification (H)** 32](#_Toc2184692)

[**125. Valid palindrome (E)** 32](#_Toc2184693)

[**165. Compare version numbers (E)** 33](#_Toc2184694)

[**242. Valid anagram (E)** 35](#_Toc2184695)

[**383. Ransom note (E)** 36](#_Toc2184696)

[**405. Convert a number to hexadecimal (E)** 37](#_Toc2184697)

[**408. Valid word abbreviation (E)** 38](#_Toc2184698)

[**415. Add strings (E)** 39](#_Toc2184699)

[**434. Number of segments in a string (E)** 40](#_Toc2184700)

[**443. String compression (E)** 41](#_Toc2184701)

[**Section 4. Linked List** 43](#_Toc2184702)

[**002. Add two numbers (M)** 43](#_Toc2184703)

[**021. Merge two sorted lists (E)** 44](#_Toc2184704)

[**024. Swap nodes in pairs (E)** 46](#_Toc2184705)

[**083. Remove duplicate from sorted list (E)** 47](#_Toc2184706)

[**Section 5. Stack** 50](#_Toc2184707)

[**020. Valid Parentheses (E)** 50](#_Toc2184708)

[**Section 6. Math** 51](#_Toc2184709)

[**007. Reverse integer (E)** 51](#_Toc2184710)

[**009. Palindrome number (E)** 51](#_Toc2184711)

[**012. Integer to roman (E)** 53](#_Toc2184712)

[**013. Roman to integer (E)** 54](#_Toc2184713)

**Section 1. Bit Manipulation**

**136. Single number I (E)**

**Problem**: Given a non-empty array of integers, every element appears twice except for one. Find that single one.

**Example:** Input: [4, 1, 2, 1, 2]

Output: 4

**Analysis:** Solution1 used C++ xor operation.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <numeric>
4. #include <functional>
6. **class** Solution
7. {
8. **public**:
9. **int** singleNumber(std::vector<**int**>& nums)
10. {
11. **return** accumulate(nums.cbegin(), nums.cend(),
12. 0, std::bit\_xor<**int**>());
13. }
14. };
16. **class** Solution1
17. {
18. **public**:
19. **int** singleNumber(std::vector<**int**>& nums)
20. {
21. **int** result = nums[0];
22. **for** (**int** i = 1; i < nums.size(); i++)
23. {
24. result ^= nums[i];
25. }
26. **return** result;
27. }
28. };
30. **int** main()
31. {
32. std::vector<**int**> v{ 4, 1, 2, 1, 2};
33. Solution1 a;
34. **int** result = a.singleNumber(v);
35. std::cout << result << std::endl;
36. std::cin.get();
37. **return** 0;
38. }

**137. Single number II (M)**

**Problem:** Given a non-empty array of integers, every element appears three times except for one, which appears exactly once. Find that single one.

**Example:** Input: [2, 2, 3, 2]

Output: 3

**Note:** Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

**Analysis:** Assume *int* is a 32-bit binary variable. Since all the other elements appear to three times, it is reasonable to use 1-bit binary variable (B) *Ones* to define that element appears one time; use 1-bit binary variable (A) *Twos* to define that element appears two times. A is upper bit and B is lower bit. Then a state transition table can be summarized as followed:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A (*Twos*) | B (*Ones*) | C (*Input*) | D (*Next\_Twos*) | E *(Next\_Ones*) |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |

This is the flow chart:

C=0

C=1

C=1

C=1

C=0

C=0

Karnaugh map for D (*Twos*): Karnaugh map for E (*Ones*):

|  |  |  |  |
| --- | --- | --- | --- |
| C \ AB | 00 | 01 | 10 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| C \ AB | 00 | 01 | 10 |
| 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** singleNumber(vector<**int**>& nums)
10. {
11. **int** ones = 0, twos = 0;
12. **for** (**size\_t** i = 0; i < nums.size(); i++)
13. {
14. **int** ones\_next = ((~twos)&(~ones)&(nums[i])) | ((~twos)&(ones)&(~nums[i]));
15. **int** twos\_next = ((~twos)&(ones)&(nums[i])) | ((twos)&(~ones)&(~nums[i]));
17. ones = ones\_next;
18. twos = twos\_next;
19. }
20. **return** ones;
21. }
22. };
24. **int** main()
25. {
26. vector<**int**> v{ 2, 2, 3, 2 };
27. Solution a;
28. **int** result = a.singleNumber(v);
29. cout << result << endl;
30. cin.get();
31. **return** 1;
32. }

**190. Reverse bits (E)**

**Problem:** Reverse bits of a given 32 bits unsigned integer

**Example:** Input: 00000010100101000001111010011100

Output: 00111001011110000010100101000000

**Analysis:** first,get one lowest (right) bit from input. If it is 1, left shift one bit result and plus 1; if it is zero, left shift one bit result. Then right shift one bit input and repeat the process. Following is the flow chat:

Lowest bit == 1

No

Yes

Input (uint\_32t)

result left shift 1 bit

result + 1

result left shift 1 bit

result left shift 1 bit

**Solution:**

1. #include <stdint.h>
2. #include <iostream>
3. #include <bitset>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. uint32\_t reverseBits(uint32\_t input)
10. {
11. **size\_t** count = 32;
12. uint32\_t result = 0;
13. **for** (**size\_t** i = 0; i < count; i++)
14. {
15. **if** (input & 1 == 1)
16. {
17. result = result << 1;
18. result++;
19. }
20. **else**
21. {
22. result = result << 1;
23. }
24. input = input >> 1;
25. }
26. **return** result;
27. }
28. };
30. **int** main()
31. {
32. Solution a;
33. uint32\_t input = 55;
34. uint32\_t result = a.reverseBits(input);
35. cout << "Input is " << bitset<32>(input) << endl;
36. cout << "Output is " << bitset<32>(result) << endl;
37. cin.get();
38. **return** 1;
39. }

**191. Number of 1 bits (E)**

**Problem:** Write a function that takes an unsigned integer and return the number of '1' bits it has (also known as the Hamming weight).

**Example:** Input: 00000000000000000000000000001011

Output: 3 **Analysis:** Determine the lowest bit of input: if it is 1, sum plus 1; if it is 0, nothing. Input right shift 1 bit. Then repeat this process for all bit of input.

Improvement would be consider n&(n-1). This operation will remove the rightmost 1 in n. This will reduce the iteration times from 32 to sum.

**Solution:**

1. #include <stdint.h>
2. #include <iostream>
3. #include <bitset>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. uint32\_t numberof1Bits(uint32\_t input)
11. {
12. uint32\_t result = 0;
13. **while** (input != 0)
14. {
15. input = input & (input - 1);
16. result++;
17. }
18. **return** result;
19. }
20. };
22. **void** main()
23. {
24. Solution a;
25. uint32\_t input = 55;
26. uint32\_t result = a.numberof1Bits(input);
28. cout << "Input is " << bitset<32>(input) << endl;
29. cout << "Number of 1 bits is: " << result << endl;
30. cin.get();
31. }

**201. Bit wise AND of number range (M)**

**Problem:** Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.

**Example:** Input: [5,7]

Output: 4

**Analysis:** similar to 191…

**Solution:**

1. #include <iostream>
2. #include <stdint.h>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. uint32\_t bitWiseAndofNumberRange(uint32\_t m, uint32\_t n)
10. {
11. **while** (m < n)
12. {
13. n &= n - 1;
14. }
15. **return** n;
16. }
17. };
19. **void** main()
20. {
21. Solution a;
22. uint32\_t m = 5, n = 7;
23. uint32\_t result = a.bitWiseAndofNumberRange(m, n);
24. cout << "Result is: " << result << endl;
25. cin.get();
26. }

**231. Power of two (E)**

**Problem:** Given an integer, write a function to determine if it is a power of two.

**Example:** Input: 1

Output: true

Explanation: 20 = 1

**Analysis:** Similar to 191…

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **bool** isPowerofTwo(**int** input)
9. {
10. **return** (input > 0) && ((input&(input - 1)) == 0);
11. }
12. };
14. **int** main()
15. {
16. Solution a;
17. **int** input = 4;
18. **bool** result = a.isPowerofTwo(input);
19. cout << input << " is power of two: " << boolalpha << result << endl;
20. cin.get();
21. **return** 1;
22. }

**260. Single Number III (M)**

**Problem:** Given an array of numbers nums, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once.

**Example:** Input: [1,2,1,3,2,5]

Output: [3,5]

**Note:** 1. The order of the result is not important. So in the above example, [5, 3] is also correct.

2. Your algorithm should run in linear runtime complexity. Could you implement it using only constant space complexity?

**Analysis:**

1. First, xor for all the elements;
2. Get the last bit where 1 occurs by “x & ~(x-1)”. Because –(x-1) = ~(x-1)+1 ⬄ -x = ~(x-1), the last bit can also calculated by x & -x;
3. Get one subset where the number has the bit and get the other subset where the number doesn’t have the bit
4. Xor the two subsets to get the two numbers

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<**int**> singleNumber(vector<**int**>& input)
10. {
11. // store xor result
12. **int** temp = 0;
13. **for** (**int** i = 0; i < input.size(); i++)
14. {
15. temp ^= input[i];
16. }
18. // get last different bit position
19. **int** diff = temp &(~(temp - 1));
21. vector<**int**> result(2, 0);
22. **for** (**int** j = 0; j < input.size(); j++)
23. {
24. **if** (diff&input[j])
25. result[0] ^= input[j];
26. **else**
27. result[1] ^= input[j];
28. }
29. **return** result;
30. }
31. };
33. **int** main()
34. {
35. Solution a;
36. vector <**int**> input{1, 2, 1, 3, 2, 5};
37. vector<**int**> result = a.singleNumber(input);
38. cout << "The results are: " << result[0] << ", " << result[1] << endl;
39. cin.get();
40. **return** 0;
41. }

**268. Missing number (M)**

**Problem:** Given an array containing n distinct numbers taken from 0, 1, 2, .., n, find the one that is missing from the array.

**Example:** Input: [3, 0, 1]

Output: 2

**Note:** Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

**Analysis:**

Solution first calculate the sum from 0-n; then subtract the sum of all elements from the result; the left would be the missing number.

Solution1 performs xor operation for all elements and 0 – n numbers; then the only one left would be the missing number.

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** missingNumber(vector<**int**>& input)
10. {
11. **int** sum = 0;
12. **for** (**int** i = 0; i < input.size(); i++)
13. {
14. sum += input[i];
15. }
16. **return** 0.5\*input.size()\*(input.size() + 1) - sum;
17. }
18. };
20. **class** Solution1
21. {
22. **public**:
23. **int** missingNumber(vector<**int**>& input)
24. {
25. **int** result = 0;
26. **for** (**int** i = 0; i < input.size(); i++)
27. {
28. result ^= (i + 1) ^ input[i];
29. }
30. **return** result;
31. }
32. };
34. **int** main()
35. {
36. Solution1 a;
37. vector<**int**> input{ 1, 0, 3 };
38. **int** result = a.missingNumber(input);
39. cout << "The missing number is: " << result << endl;
40. cin.get();
41. **return** 0;
42. }

**318. Maximum product of word lengths (M)**

**Problem:** Given a string array words, find the maximum value of length(word[i]) \* length(word[j]) where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.

**Example:** Input: ["abcw","baz","foo","bar","xtfn","abcdef"]

Output: 16

Explanation: The two words can be "abcw", "xtfn".

**Analysis:** This problem defines only small letter from a to z. 32 bit int data type can be used to represent each word. The lower 26 of 32 bits of int are used. For example:

abcd -> 0000 0000 0000 0000 0000 0000 0000 1111

acxy -> 0000 0001 1000 0000 0000 0000 0000 0101

1. Claim a int vector to store each word;
2. Use AND operation to determine each two int pair are same (not same -> 0);
3. Compare and return the maximum value;

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <string>
4. #include <algorithm>
6. **using** **namespace** std;
8. **class** Solution
9. {
10. **public**:
11. **int** maxLength(vector<string>& input)
12. {
13. vector<**int**> temp(input.size(), 0);
14. **int** result = 0;
15. **for** (**int** i = 0; i < input.size(); i++)
16. {
17. **for** (**int** j = 0; j < input[i].size(); j++)
18. {
19. temp[i] |= (1 << (input[i][j] - 'a'));
20. //cout << input[i][j];
21. }
23. **for** (**int** k = 0; k < i; k++)
24. {
25. **if** ((temp[i] & temp[k]) == 0) // (temp[i] & temp[k])!!!
26. {
27. result = max(result, **static\_cast**<**int**>(input[i].size()\*input[k].size()));
28. }
29. }
30. }
31. **return** result;
32. }
33. };
35. **int** main()
36. {
37. Solution a;
38. vector<string> input{ "abcw", "baz", "foo", "bar", "xtfn", "abcdef" };
39. **int** result = a.maxLength(input);
40. cout << "The result is: " << result << endl;
41. cin.get();
42. **return** 0;
43. }

**Caution:**

* In line 25, since “==” has higher precedence than “&”, brackets are must!
* To use std::max(), include header <algorithm>

**Section 2. Array**

**015. 3 sum (M)**

**Problem:** Given an array nums of n integers, are there elements a, b, c in nums such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

**Example:** Given array nums = [-1, 0, 1, 2, -1, -4],

A solution set is:

[

[-1, 0, 1],

[-1, -1, 2]

]

**Note:** The solution set must not contain duplicate triplets.

**Analysis:**

1. Sort nums in increasing order;
2. Start the loop from 0 – nums.size()-2;
3. If it is a positive number, break; because this is an order sequence;
4. Add condition, if repetition, continue;
5. Define a target = 0 – nums[i], then find another two elements whose sum equals to the target;
6. Define two “pointers”, one at the next position after target, and the last position after target;

* If this two numbers add up to the target, push back; also need to avoid repetition;
* If this two numbers’ sum smaller than target, left pointer +1
* If this two number’s sum larger than target, right pointer -1

1. Return the result.

**Solution:**

1. #include <vector>
2. #include <iostream>
3. #include <algorithm>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. vector<vector<**int**>> threeSum(vector<**int**>& input)
11. {
12. vector<vector<**int**>> res{};
13. sort(input.begin(), input.end());
15. **for** (**int** i = 0; i < input.size() - 2; i++)
16. {
17. **if** (input[i] > 0) **break**;
19. **if** (i > 0 && (input[i] == input[i - 1])) **continue**;
21. **int** target = 0 - input[i];
22. **int** m = i + 1, n = input.size() - 1;
23. **while** (m < n)
24. {
25. **if** (input[m] + input[n] == target)
26. {
27. res.push\_back({ input[i], input[m], input[n] });
28. **while** ((m < n) && (input[m] == input[m + 1])) m++;
29. **while** ((m < n) && (input[n] == input[n - 1])) n--;
30. m++;
31. n--;
32. }
33. **else** **if** (input[m] + input[n] < target)
34. {
35. m++;
36. }
37. **else**
38. n--;
39. }
40. }
41. **return** res;
42. }
43. };
45. **int** main()
46. {
47. Solution a;
48. vector<**int**> input{ -1, 0, 1, 2, -1, -4 };
49. vector<vector<**int**>> res = a.threeSum(input);
50. cout << "The result is: " << endl;
51. **for** (**int** i = 0; i < res.size(); i++)
52. {
53. **for** (**int** j = 0; j < res[i].size(); j++)
54. {
55. cout << res[i][j] << ", ";
56. }
57. cout << endl;
58. }
59. cin.get();
60. **return** 0;
61. }

**016. 3 sum closest (M)**

**Problem:** Given an array nums of n integers and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

**Example:** Given array nums = [-1, 2, 1, -4], and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

**Analysis:** Similar to 015…

**Solution:**

1. #include <iostream>
2. #include <algorithm>
3. #include <vector>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **int** threeSumClosest(vector<**int**> input, **int** target)
11. {
12. sort(input.begin(), input.end());
14. **int** min\_diff = numeric\_limits<**int**>::max();
15. **int** res = numeric\_limits<**int**>::max();
16. **for** (**int** i = 0; i < input.size() - 2; i++)
17. {
18. **if** ((i > 0) && (input[i] == input[i - 1])) **continue**;
20. **int** c = target - input[i];
21. **int** a = i + 1, b = input.size() - 1;
22. **while** (a < b)
23. {
24. **int** sum = input[a] + input[b];
25. **if** (sum == c)
26. {
27. **return** target;
28. }
29. **else** **if** (sum < c)
30. {
31. ++a;
32. }
33. **else**
34. {
35. --b;
36. }
38. **if** (abs(sum - c) < min\_diff)
39. {
40. min\_diff = abs(sum - c);
41. res = sum + input[i];
42. }
43. }
44. }
46. **return** res;
47. }
48. };
50. **int** main()
51. {
52. vector<**int**> input{ -1, 2, 1, -4 };
53. **int** target = 1;
54. Solution a;
55. **int** res = a.threeSumClosest(input, target);
56. cout << res << endl;
57. cin.get();
58. **return** 0;
59. }

**Caution:**

* Include <algorithm>, which will be used by sort(), numeric\_limits<int>::max() and abs()

**018. 4 sum (M)**

**Problem:** Given an array nums of n integers and an integer target, are there elements a, b, c, and d in nums such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

**Example:** Given array nums = [1, 0, -1, 0, -2, 2], and target = 0.

A solution set is: [ [-1, 0, 0, 1],

[-2, -1, 1, 2],

[-2, 0, 0, 2] ]

**Note:** The solution set must not contain duplicate quadruplets.

**Analysis:** similar to 015…

**Solution:**

1. #include <iostream>
2. #include <algorithm>
3. #include <vector>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. vector<vector<**int**>> fourSum(vector<**int**>& input, **int** target)
11. {
12. sort(input.begin(), input.end());
14. vector<vector<**int**>> res{};
15. **for** (**int** i = 0; i < input.size() - 3; i++)
16. {
17. **for** (**int** j = i + 1; j < input.size() - 2; j++)
18. {
19. **if** (input[j] == input[j - 1]) **continue**;
20. **int** a = j + 1, b = input.size() - 1;
21. **while** (a < b)
22. {
23. **int** sum = input[i] + input[j] + input[a] + input[b];
24. **if** (sum == target)
25. {
26. res.push\_back({ input[i], input[j], input[a], input[b] });
27. **while** ((a < b) && (input[a] == input[a + 1])) a++;
28. **while** ((a < b) && (input[b] == input[b - 1])) b--;
29. a++;
30. b--;
31. }
32. **else** **if** (sum < target)
33. a++;
34. **else**
35. b--;
36. }
37. }
38. }
39. **return** res;
40. }
41. };
43. **int** main()
44. {
45. Solution a;
46. vector<**int**> input{ 1, 0, -1, 0, -2, 2 };
47. **int** target = 0;
48. vector<vector<**int**>> res{};
49. res = a.fourSum(input, target);
50. **for** (**int** i = 0; i < res.size(); i++)
51. {
52. **for** (**int** j = 0; j < res[i].size(); j++)
53. {
54. cout << res[i][j] << ", ";
55. }
56. cout << endl;
57. }
58. cin.get();
59. **return** 0;
60. }

**Caution:**

* i loops from 0 – input.size()-3; j loops from i+1 – input.size()-2.

**026. Remove duplicates from sorted array (E)**

**Problem:** Given a sorted array nums, remove the duplicates in-place such that each element appear only once and return the new length.

**Example:** Given nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively.

It doesn't matter what you leave beyond the returned length.

**Note:** Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

**Analysis:** easy…

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** removeDuplicates(vector<**int**>& input)
10. {
11. **int** sum = 0;
12. **for** (**int** i = 0; i < input.size(); i++)
13. {
14. **if** ((i > 0) && (input[i] == input[i - 1])) **continue**;
15. sum++;
16. }
17. **return** sum;
18. }
19. };
21. **int** main()
22. {
23. vector<**int**> input{ 0,0,1,1,1,2,2,3,3,4 };
24. **int** res = 0;
25. Solution a;
26. res = a.removeDuplicates(input);
27. cout << res << endl;
28. cin.get();
29. **return** 0;
30. }

**027. Remove element (E)**

**Problem:** Given an array nums and a value val, remove all instances of that value in-place and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

**Example:** Given nums = [3,2,2,3], val = 3,

Your function should return length = 2, with the first two elements of nums being 2.

**Analysis:** Easy…

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **int** removeElement(vector<**int**>& input, **int** target)
11. {
12. sort(input.begin(), input.end());
14. **int** res = 0;
15. **for** (**int** i = 0; i < input.size(); i++)
16. {
17. **if** (input[i] == target) **continue**;
18. res++;
19. }
20. **return** res;
21. }
22. };
24. **int** main()
25. {
26. Solution a;
27. vector<**int**> input{ 0,1,2,2,3,0,4,2 };
28. **int** target = 2;
29. **int** res = a.removeElement(input, target);
30. cout << res << endl;
31. cin.get();
32. **return** 0;
33. }

**031. Next permutation (M)**

**Problem**: Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place and use only constant extra memory.

**Example:** 1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

**Analysis:**

* Solution: use c++ std::next\_permutation
* Solution1: given an array (325421); first, from the lowest bit, find the first element (2) that descending; then switch this element with the one that is smallest one that is larger than this element in the right hand side (345221); finally, order the right hand side element in ascending order (341225).

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **void** nextPermutation(vector<**int**>& input)
11. {
12. next\_permutation(input.begin(), input.end());
13. }
14. };
16. **class** Solution1
17. {
18. **public**:
19. **void** nextPermutation(vector<**int**>& input)
20. {
21. **int** keyIndex = input.size() - 1;
22. **while** ((keyIndex > 0) && (input[keyIndex] <= input[keyIndex - 1]))
23. {
24. keyIndex--;
25. }
27. **if** (keyIndex == 0)
28. {
29. sort(input.begin(), input.end());
30. }
31. **else**
32. {
33. **int** minNum = input[keyIndex - 1];
34. **for** (**int** i = input.size() - 1; i > keyIndex - 1; i--)
35. {
36. **if** (input[i] > minNum)
37. {
38. **int** temp;
39. temp = input[i];
40. input[i] = input[keyIndex - 1];
41. input[keyIndex - 1] = temp;
42. **break**;
43. }
44. }
45. }
46. sort(input.begin() + keyIndex, input.end());
47. }
48. };
50. **int** main()
51. {
52. vector<**int**> input{ 3,2,5,4,2,1 };
53. Solution1 a;
54. a.nextPermutation(input);
55. **for** (**int** i = 0; i < input.size(); i++)
56. cout << input[i] << ", ";
57. cin.get();
58. **return** 0;
59. }

**041. First missing positive (H)**

**Problem:** Given an unsorted integer array, find the smallest missing positive integer.

**Example:** Input: [1,2,0]

Output: 3

**Note:** Your algorithm should run in O(n) time and uses constant extra space.

**Analysis:**

1. sort the array in ascending order;
2. find the first positive element position i (if not, result is the last element potion);
3. start the loop from i to end of array;
4. if there is repeat element, continue;
5. if A[i] – res != 1, return res+1 (res initial value set as 0);
6. else res=A[i];
7. finish the loop and return res+1.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **int** firstMissing(vector<**int**> input)
11. {
12. sort(input.begin(), input.end());
14. **int** res = 0;
15. **int** i = 0;
16. **while** ((i < input.size()) && (input[i] <= 0)) i++;
18. **for** (; i < input.size(); i++)
19. {
20. **if** ((i > 0) && (input[i] == input[i - 1])) **continue**;
22. **if** (input[i] - res != 1) **return** res + 1;
23. **else** res = input[i];
24. }
25. **return** res + 1;
26. }
27. };
29. **int** main()
30. {
31. vector<**int**> input{ 7,8,9,11,12 };
32. Solution a;
33. **int** res = a.firstMissing(input);
34. cout << res << endl;
35. cin.get();
36. **return** 1;
37. }

**048. Rotate image (M)**

**Problem:** You are given an n x n 2D matrix representing an image. Rotate the image by 90 degrees (clockwise).

Example: Given input matrix =

[ [1,2,3],

[4,5,6],

[7,8,9]],

rotate the input matrix in-place such that it becomes:

[

[7,4,1],

[8,5,2],

[9,6,3]

]

**Note:** You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

**Analysis:**

* i loops from 0 – input.size()/2
* j loops from i – input[i].size-1-i

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<vector<**int**>> rotateImage(vector<vector<**int**>>& input)
10. {
11. **int** n = input.size() - 1;
13. **for** (**int** i = 0; i < input.size() / 2; i++)
14. {
15. **for** (**int** j = i; j < input[i].size() - 1 - i; j++)
16. {
17. **int** temp = input[i][j];
18. input[i][j] = input[n - j][i];
19. input[n - j][i] = input[n - i][n - j];
20. input[n - i][n - j] = input[j][n - i];
21. input[j][n - i] = temp;
22. }
23. }
24. }
25. };

**Section 3. String**

**005. Longest palindromic substring (M)**

**Problem:** Given a string s, find the longest palindromic substring in s. You may assume that the maximum length of s is 1000.

**Example:** Input: "babad"

Output: "bab"

Note: "aba" is also a valid answer.

Analysis: Manacher’s anglrithms

1. inset “#” to each side of each element (bob -> #b#o#b#, noon -> #n#o#o#n#) so that it will surely have odd number of elements
2. insert “$” to beginning of the string (#b#o#b# -> $#b#o#b#)
3. the longest palindromic substring’s length is radius -1 ($#b#o#b# -> length = 4-1 =3 -> bob)
4. the longest palindromic substring’s position is (middle position – radius)/2 ($#b#o#b# -> position = (4-4)/2 =0 -> bob)
5. assume there is a palindromic substring x that can reach the most right side; define mx as the right side position of x; define id as the middle point of x; define p[i] is the radius of the palindromic substring which t[i] character as center
6. p[i] = mx > i ? min(p[2 \* id - i], mx - i) : 1;
7. if mx > i, then p[i] = min(p[2 \* id - i], mx - i);

* mx - i > P[j], palindromic substring centered at S[j] must be included by S[id], since i and j are symmetric, palindromic substring centered at S[i] also must be include by S[id] and p[i]=p[j];



* mx - i <= P[j], palindromic substring centered at S[j] is not necessarily included by S[id], since i and j are symmetric, the green portion in the following would be same, this means palindromic substring centered at S[i] will expand at least to mx position, then p[i]-mx-i. However, for element after mx, need to find out if it is palindromic…



1. mx <= i, have to assume P[i]=1 and then find out palindromic situation…

**Solution:**

1. #include <vector>
2. #include <string>
3. #include <iostream>
4. #include <algorithm>
6. **using** **namespace** std;
8. **class** Solution
9. {
10. **public**:
11. string longestPalindromicSubstring(string s)
12. {
13. string t = "$#";
14. **for** (**int** i = 0; i < s.size(); i++)
15. {
16. t += s[i];
17. t += "#";
18. }
20. vector<**int**> p(t.size(), 0);
21. **int** mx = 0, id = 0, resLen = 0, resCenter = 0;
22. **for** (**int** i = 1; i < t.size(); i++)
23. {
24. p[i] = mx > i ? min(p[2 \* id - i], mx - i) : 1;
26. **while** (t[i + p[i]] == t[i - p[i]]) ++p[i];
27. **if** (mx < i + p[i])
28. {
29. mx = i + p[i];
30. id = i;
31. }
32. **if** (resLen < p[i])
33. {
34. resLen = p[i];
35. resCenter = i;
36. }
37. }
39. **return** s.substr((resCenter - resLen) / 2, resLen - 1);
40. }
41. };
43. **int** main()
44. {
45. string s = "12212";
46. Solution a;
47. cout << a.longestPalindromicSubstring(s) << endl;
48. cin.get();
49. **return** 0;
50. }

**006. ZigZag conversion (E)**

**Problem:** The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

P A H N

A P L S I I G

Y I R

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows:

string convert(string s, int numRows);

**Example:** s = "PAYPALISHIRING", numRows = 4

Output: "PINALSIGYAHRPI"

Explanation:

P I N

A L S I G

Y A H R

P I

**Analysis:**

1. every row repeat in a pattern of 2\*nRow-2;
2. other than the first and last row, the rows in middle will repeat just on more time, this distant of this element to the first position of row is 2\*nRow -2 – 2\*i;
3. first loop i is from 0 – nRow, with step 1; second loop j is from i – input.size();
4. push back i+2\*nRow-2
5. if it is one of the middle rows, push back another j + 2\*nRow-2-2\*i

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string zigzagConvert(string s, **int** numRows)
10. {
11. string res;
12. **if** (numRows < 2) **return** s;
13. **for** (**int** i = 0; i < numRows; i++)
14. {
15. **for** (**int** j = i; j < s.size(); j += 2 \* (numRows - 1))
16. {
17. res.push\_back(s[j]);
19. **if** ((i > 0) && (i < numRows - 1))
20. {
21. **if** (j + 2 \* (numRows - i - 1) < s.length())
22. {
23. res.push\_back(s[j + 2 \* (numRows - i - 1)]);
24. }
25. }
26. }
27. }
28. **return** res;
29. }
30. };
32. **int** main()
33. {
34. Solution a;
35. cout << a.zigzagConvert("PAYPALISHIRING", 3) << endl;
36. cin.get();
37. **return** 0;
38. }

**008. String to integer (E)**

**Problem:** Implement atoi (ASCII to integer) which converts a string to an integer.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned.

**Example:** Input: "42" Output: 42

Input: " -42" Output: -42

Input: "words and 987" Output: 0

Input: "4193 with words" Output: 4193

Input: "-91283472332" Output: -2147483648

**Note:**

* Only the space character ' ' is considered as whitespace character.
* Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: [−231, 231 − 1]. If the numerical value is out of the range of representable values, INT\_MAX (231 − 1) or INT\_MIN (−231) is returned.

**Analysis:**

1. if string s is empty, return 0;
2. define ans as 0, sign as 1, i as 0;
3. remove whilespace ‘ ’ or ‘\t’;
4. if i==s.length(), return 0;
5. determine sign
6. if ans > (numeric\_limits<int>::max()-(s[i]-‘0’))/10 return max or min
7. ans \*= 10, ans+=s[i] – ‘0’
8. determine sign and return ans

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** string2integer(string s)
10. {
11. **if** (s.empty()) **return** 0;
13. **int** ans = 0;
14. **int** sign = 1;
15. **int** i = 0;
17. **while** ((s[i] == ' ') || (s[i] == '\t')) i++;
19. **if** (i == s.length()) **return** 0;
21. **if** (s[i] == '+') i++;
22. **else** **if** (s[i] == '-')
23. {
24. sign = -1;
25. i++;
26. }
28. **for** (; (i < s.length()) && (isdigit(s[i])); i++)
29. {
30. **if** ((ans) > (numeric\_limits<**int**>::max()- (s[i] - '0'))/10)
31. {
32. **return** sign > 0 ? numeric\_limits<**int**>::max() : numeric\_limits<**int**>::min();
33. }
34. ans \*= 10;
35. ans += s[i] - '0';
36. }
37. ans \*= sign;
38. **return** ans;
39. }
40. };
42. **int** main()
43. {
44. Solution a;
45. cout << numeric\_limits<**int**>::max() << " " << numeric\_limits<**int**>::min() << endl;
46. cout << a.string2integer("-91283472332") << endl;
47. cin.get();
48. **return** 0;
49. }

**014. Longest common prefix (E)**

**Problem:** Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

**Example:** Input: ["flower","flow","flight"]

Output: "fl"

**Note:** All given inputs are in lowercase letters a-z.

**Analysis:** seems easy…but…

1. If input is empty, return “”;
2. first loop i is from 0 to input[0].size();
3. second loop j is from 0 to input.size();
4. if i>=input[j].size || input[j][i]!=input[0][i], return input[0].substr(0, i)
5. finish loops; and if nothing is return, return input[0]

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <vector>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. string longestCommonPrefix(vector<string> input)
11. {
12. **if** (input.empty()) **return** "";
14. **for** (**int** i = 0; i < input[0].size(); i++)
15. {
16. **for** (**int** j = 0; j < input.size(); j++)
17. {
18. **if** ((i >= input[j].size()) || (input[j][i] != input[0][i]))
19. {
20. **return** input[0].substr(0, i);
21. }
22. }
23. }
24. //return input[0];
25. }
26. };
28. **int** main()
29. {
30. Solution a;
31. vector<string> input{ "flower", "flowerere", "flowererereight" };
32. cout << a.longestCommonPrefix(input) << endl;
33. cin.get();
34. **return** 0;
35. }

**028. Implement strStr() (E)**

**Problem:** Return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

**Example:** Input: haystack = "hello", needle = "ll"

Output: 2

**Note:** What should we return when needle is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return 0 when needle is an empty string. This is consistent to C's strstr() and Java's indexOf().

Analysis: easy…

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** strStr(string haystack, string needle)
10. {
11. **int** m = haystack.size(), n = needle.size();
13. **if** (n == 0) **return** 0;
15. **for** (**int** i = 0; i < m; i++)
16. {
17. **if** (i + n > m) **break**;
18. **for** (**int** j = 0; j < n; j++)
19. {
20. **if** (haystack[i + j] != needle[j]) **break**;
22. **else** **if** ((j = n - 1) && (haystack[i + j] == needle[j])) **return** i;
23. }
24. }
25. **return** -1;
26. }
27. };
29. **int** main()
30. {
31. Solution a;
32. cout << a.strStr("aaaaa", "bba") << endl;
33. cin.get();
34. **return** 0;
35. }

**038. Count and say (E)**

**Problem:** The count-and-say sequence is the sequence of integers with the first five terms as following:

1. 1

2. 11

3. 21

4. 1211

**Example:** Input: 4

Output: "1211"

**Analysis:** easy…

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string countAndSay(**int** n)
10. {
11. **if** (n < 1) **return** "";
13. string str = "1";
14. **for** (**int** i = 1; i < n; i++)
15. {
16. string tmp = "";
17. **int** count = 0;
18. **char** current = str[0];
19. **int** index = 0;
20. **while** (index < str.size())
21. {
22. **if** (str[index] == current)
23. {
24. count++;
25. }
26. **else**
27. {
28. tmp += to\_string(count) + current;
29. current = str[index];
30. count = 1;
31. }
32. index++;
33. }
34. tmp += to\_string(count) + current;
35. str = tmp;
36. }
37. **return** str;
38. }
39. };
41. **int** main()
42. {
43. Solution a;
44. **for** (**int** i = 1; i < 13; i++)
45. {
46. cout << "i = " << i << ",\t" << "seq = " << a.countAndSay(i) << endl;
47. }
48. cin.get();
49. **return** 0;
51. }

**043. Multiply strings (M)**

**Problem:** Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2, also represented as a string.

**Example:** Input: num1 = "2", num2 = "3"

Output: "6"

**Note:**

* The length of both num1 and num2 is < 110.
* Both num1 and num2 contain only digits 0-9.
* **Both num1 and num2 do not contain any leading zero, except the number 0 itself**.
* You must not use any built-in BigInteger library or convert the inputs to integer directly

**Analysis:**

1. determine if num1 or num2 are null, if so, return “”;
2. reverse both num1 and num2 so that lower bit starting from the right;
3. fisrt loop j starts from 0 to num2.size()
4. initialize int carry = 0 and int val = num2[j]-‘0’
5. second loop I starts from 0 to num1.size();
6. calculate carry, res[i+j] and update carry
7. complete second loop and deal with the situation thet if highest carry !=0
8. complete first loop and reverse res
9. remove left hand redundant ‘0’

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string multiplyStrings(string num1, string num2)
10. {
11. **if** (num1.empty() || num2.empty()) **return** "";
13. reverse(num1.begin(), num1.end());
14. reverse(num2.begin(), num2.end());
16. string res(num1.size() + num2.size(), '0');
18. **for** (**int** j = 0; j < num2.size(); j++)
19. {
20. **int** val = num2[j] - '0';
21. **int** carry = 0;
22. **for** (**int** i = 0; i < num1.size(); i++)
23. {
24. carry += (num1[i] - '0') \* val + (res[i + j] - '0');
25. res[i + j] = carry % 10 + '0';
26. carry /= 10;
27. }
28. **if** (carry != 0) res[num1.size() + j] = carry + '0';
29. }
31. reverse(res.begin(), res.end());
33. **int** count = 0;
34. **while** ((count < res.size() - 1) && (res[count] == '0')) count++;
35. res.erase(0, count);
36. **return** res;
37. }
38. };
40. **int** main()
41. {
42. string num1 = "123", num2 = "456";
43. Solution a;
44. cout << num1 << " x " << num2 << " = " << a.multiplyStrings(num2, num1) << endl;
45. cin.get();
46. **return** 0;
47. }

**058. Length of last word (E)**

**Problem:** Given a string s consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string. If the last word does not exist, return 0.

**Example:** Input: "Hello World"

Output: 5

**Note:** A word is defined as a character sequence consists of non-space characters only.

Analysis: east…

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** lengthOfLastWord(string s)
10. {
11. **int** i = s.size() - 1;
12. **while** ((i >= 0) && (s[i] == ' ')) --i;
14. **int** length = 0;
15. **while** ((i >= 0) && (s[i] != ' '))
16. {
17. length++;
18. i--;
19. }
20. **return** length;
21. }
22. };
24. **int** main()
25. {
26. Solution a;
27. cout << a.lengthOfLastWord("Hello world    ") << endl;
28. cin.get();
29. **return** 0;
30. }

**067. Add binary (E)**

**Problem:** Given two binary strings, return their sum (also a binary string).

The input strings are both non-empty and contains only characters 1 or 0.

**Example:** Input: a = "11", b = "1"

Output: "100"

**Analysis:** easy…

**Solution:**

1. #include <string>
2. #include <algorithm>
3. #include <iostream>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. string addBinary(string bin1, string bin2)
11. {
12. reverse(bin1.begin(), bin1.end());
13. reverse(bin2.begin(), bin2.end());
15. string sum;
16. **int** n = max(bin1.size(), bin2.size());
17. **int** carry = 0;
18. **for** (**int** i = 0; i < n; i++)
19. {
20. **if** (i < bin1.size()) carry += bin1[i] - '0';
21. **if** (i < bin2.size()) carry += bin2[i] - '0';
22. sum.push\_back(carry % 2 + '0');
23. carry /= 2;
24. }
25. **if** (carry) sum.push\_back('1');
26. reverse(sum.begin(), sum.end());
27. **return** sum;
28. }
29. };
31. **int** main()
32. {
33. Solution a;
34. string bin1 = "1010", bin2 = "1011";
35. cout << a.addBinary(bin1, bin2) << endl;
36. cin.get();
37. **return** 0;
38. }

**068. Text justification (H)**

**Problem:** Given an array of words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

**Example:** Input: words = ["This", "is", "an", "example", "of", "text", "justification."] maxWidth = 16

Output: [ "This is an",

"example of text",

"justification. "]

**Analysis:** I gave up… from now on, I will change strategy which starting from easy to medium to hard ones…

**125. Valid palindrome (E)**

**Problem:** Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

**Example:** Input: "A man, a plan, a canal: Panama"

Output: true

**Analysis:** isalnum()

tolower()

**Solution:**

1. #include <iostream>
2. #include <string>
3. //#include <cctype>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **bool** isPalindrome(string str)
11. {
12. **if** (str.empty()) **return** **true**;
14. **int** left = 0, right = str.size() - 1;
15. **while** (left < right)
16. {
17. **if** (!isalnum(str[left])) left++;
18. **else** **if** (!isalnum(str[right])) right--;
19. **else** **if** (tolower(str[left]) != tolower(str[right])) **return** **false**;
20. **else** {
21. left++;
22. right--;
23. }
24. }
25. **return** **true**;
26. }
27. };
29. **int** main()
30. {
31. Solution a;
32. cout << a.isPalindrome("A man, a plan, a canal: Panama") << endl;
33. system("pause");
34. **return** 1;
35. }

**165. Compare version numbers (E)**

**Problem:** Compare two version numbers version1 and version2. If version1 > version2 return 1; if version1 < version2 return -1; otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the . characters.

The . character does not represent a decimal point and is used to separate number sequences.

For instance, 2.5 is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.

You may assume the default revision number for each level of a version number to be 0. For example, version number 3.4 has a revision number of 3 and 4 for its first and second level revision number. Its third and fourth level revision number are both 0.

**Example:** Input: version1 = "7.5.2.4", version2 = "7.5.3"

Output: -1

Input: version1 = "1.01", version2 = "1.001"

Output: 0

**Analysis:** easy…

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** compareVersion(string str1, string str2)
10. {
11. **int** m = str1.size(), n = str2.size();
13. **for** (**int** i = 0, j = 0; i < m || j < n; i++, j++)
14. {
15. **int** v1 = 0, v2 = 0;
16. **while** ((i < m) && (str1[i] != '.'))
17. {
18. v1 = v1 \* 10 + (str1[i] - '0');
19. i++;
20. }
21. **while** ((j < n) && (str2[j] != '.'))
22. {
23. v2 = v2 \* 10 + (str2[j] - '0');
24. j++;
25. }
26. **if** (v1 != v2)
27. **return** v1 > v2 ? 1 : -1;
28. }
29. **return** 0;
30. }
31. };
33. **int** main()
34. {
35. Solution a;
36. cout << a.compareVersion("1.01", "1.001") << endl;
37. system("pause");
38. **return** 1;
39. }

**242. Valid anagram (E)**

**Problem:** Given two strings s and t , write a function to determine if t is an anagram of s.

**Note:** You may assume the string contains only lowercase alphabets.

**Follow up:** What if the inputs contain unicode characters? How would you adapt your solution to such case?

L“” ⬄ wstring

**Example:** Input: s = "anagram", t = "nagaram"

Output: true

**Solution:**

1. #include <string>
2. #include <unordered\_map>
3. #include <iostream>
4. #include <algorithm> // sort()
6. **using** **namespace** std;
8. **class** Solution
9. {
10. **public**:
11. **bool** validAnagram(wstring str1, wstring str2)
12. {
13. sort(str1.begin(), str1.end());
14. sort(str2.begin(), str2.end());
15. **return** str1 == str2;
16. }
17. };
19. **class** Solution2
20. {
21. **public**:
22. **bool** validAnagram(string str1, string str2)
23. {
24. **if** (str1.size() != str2.size()) **return** **false**;
26. unordered\_map<**char**, **int**> count;
27. **for** (**int** i = 0; i < str1.size(); i++)
28. {
29. count[str1[i]]++;
30. }
32. **for** (**int** j = 0; j < str2.size(); j++)
33. {
34. count[str2[j]]--;
35. **if** (count[str2[j]] < 0) **return** **false**;
36. }
37. **return** **true**;
38. }
39. };
41. **int** main()
42. {
43. Solution a;
44. //cout << a.validAnagram("anagrama", "nagaram") << endl;
45. cout << a.validAnagram(L"你好", L"好你") << endl;
46. system("pause");
47. **return** 1;
48. }

**383. Ransom note (E)**

**Problem:** Given an arbitrary ransom note string and another string containing letters from all the magazines, write a function that will return true if the ransom note can be constructed from the magazines ; otherwise, it will return false.

Each letter in the magazine string can only be used once in your ransom note.

**Example:** canConstruct("aa", "aab") -> true

canConstruct("aa", "ab") -> false

**Analysis:** similar to 242, using hash map

**Solution:**

1. #include <string>
2. #include <unordered\_map>
3. #include <iostream>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **bool** canConstruct(string str1, string str2)
11. {
12. **int** m = str1.size(), n = str2.size();
13. unordered\_map<**char**, **int**> count;
14. **for** (**int** j = 0; j < n; j++)
15. {
16. ++count[str2[j]];
17. }
18. **for** (**int** i = 0; i < m; i++)
19. {
20. --count[str1[i]];
21. **if** (count[str1[i]] < 0) **return** **false**;
22. }
23. **return** **true**;
24. }
25. };
27. **int** main()
28. {
29. Solution a;
30. cout << a.canConstruct("aa", "cbtdyayca") << endl;
31. system("pause");
32. **return** 1;
34. }

**405. Convert a number to hexadecimal (E)**

**Problem:** Given an integer, write an algorithm to convert it to hexadecimal. For negative integer, two’s complement method is used.

**Example:** Input:26

Output: "1a"

Input: -1

Output: "ffffffff"

**Note:**

* All letters in hexadecimal (a-f) must be in lowercase.
* The hexadecimal string must not contain extra leading 0s. If the number is zero, it is represented by a single zero character '0'; otherwise, the first character in the hexadecimal string will not be the zero character.
* The given number is guaranteed to fit within the range of a **32-bit signed integer**.
* You must not use any method provided by the library which converts/formats the number to hex directly.

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string int2hex(**int** num)
10. {
11. string res = "";
12. **if** (!num) **return** "0";
14. **while** (num && res.size() != **sizeof**(**int**) \* 2)
15. {
16. **int** hex = num & 15;
17. **if** (hex > 10)
18. {
19. res.push\_back(hex - 10 + 'a');
20. }
21. **else**
22. {
23. res.push\_back(hex + '0');
24. }
25. num >>= 4;
26. cout << num << endl;
27. }
28. reverse(res.begin(), res.end());
29. **return** res;
30. }
31. };
33. **int** main()
34. {
35. Solution a;
36. cout << a.int2hex(111) << endl;
37. system("pause");
38. **return** 1;
39. }

**408. Valid word abbreviation (E)**

**Problem:** Given a non-empty string s and an abbreviation abbr, return whether the string matches with the given abbreviation.

A string such as "word" contains only the following valid abbreviations:

["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1", "1o2", "2r1", "3d", "w3", "4"]

Notice that only the above abbreviations are valid abbreviations of the string "word". Any other string is not a valid abbreviation of "word".

**Example:** Given s = "internationalization", abbr = "i12iz4n":

Return true.

**Note:** Assume s contains only lowercase letters and abbr contains only lowercase letters and digits.

Analysis: don’t forget two situations!

* abbr ended with numbers
* ‘0’ in abbr

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **bool** validAbbr(string word, string abbr)
10. {
11. **int** j = 0;
12. **int** temp = 0;
13. **for** (**int** i = 0; i < abbr.size(); i++)
14. {
15. **if** ((abbr[i] >= '0') && (abbr[i] <= '9'))
16. {
17. **if** ((abbr[i] == '0') && (temp == 0)) **return** **false**;
18. temp = temp \* 10 + (abbr[i] - '0');
19. }
20. **else** **if** (abbr[i] >= 'a'&&abbr[i] <= 'z')
21. {
22. j += temp;
23. **if** (abbr[i] != word[j]) **return** **false**;
24. **else**
25. {
26. j++;
27. temp = 0;
28. }
29. }
31. }
33. **if** (j + temp == word.size()) **return** **true**;
34. **else** **return** **false**;
35. }
36. };
38. **int** main()
39. {
40. Solution a;
41. cout << a.validAbbr("internationalizationaaa", "i12iz4n3") << endl;
42. system("pause");
43. **return** 1;
44. }

**415. Add strings (E)**

**Problem:** Given two non-negative integers num1 and num2 represented as string, return the sum of num1 and num2.

Note:

* The length of both num1 and num2 is < 5100.
* Both num1 and num2 contains only digits 0-9.
* Both num1 and num2 does not contain any leading zero.
* You must not use any built-in BigInteger library or convert the inputs to integer directly.

**Analysis:** similar to 067…

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string addStrings(string str1, string str2)
10. {
11. reverse(str1.begin(), str1.end());
12. reverse(str2.begin(), str2.end());
14. **int** m = str1.size() >= str2.size() ? str1.size() : str2.size();
15. **int** carry = 0;
16. string res;
17. **for** (**int** i = 0; i < m; i++)
18. {
19. **if** (i < str1.size()) carry += (str1[i] - '0');
20. **if** (i < str2.size()) carry += (str2[i] - '0');
21. res.push\_back(carry % 10 + '0');
22. carry /= 10;
23. }
24. **if** (carry) res.push\_back(carry + '0');
25. reverse(res.begin(), res.end());
26. **return** res;
27. }
28. };
30. **int** main()
31. {
32. Solution a;
33. cout << a.addStrings("99", "99") << endl;
34. system("pause");
35. **return** 1;
36. }

**434. Number of segments in a string (E)**

**Problem:** Count the number of segments in a string, where a segment is defined to be a contiguous sequence of non-space characters.

Please note that the string does not contain any non-printable characters.

**Example:** Input: "Hello, my name is John"

Output: 5

**Analysis:** consider several situations:

* white space ‘\n’ ‘\t’ ‘ ’
* null string
* white space at the end

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** countSegments(string str)
10. {
11. **int** count = 0;
12. **if** (str.empty()) **return** 0;
14. **for** (**int** i = 1; i < str.size(); i++)
15. {
16. **if** (((str[i] == ' ') || (str[i] == '\t')) &&
17. (str[i - 1] != ' ') && str[i - 1] != '\t') count++;
18. }
19. **if** ((str[str.size() - 1] == ' ')||(str[str.size() - 1] == '\t')) **return** count;
20. **else** **return** count - 1;
21. }
22. };
24. **int** main()
25. {
26. Solution a;
27. cout << a.countSegments("Hello, my name is John ") << endl;
28. system("pause");
29. **return** 1;
30. }

**443. String compression (E)**

**Problem:** Given an array of characters, compress it in-place.

The length after compression must always be smaller than or equal to the original array.

Every element of the array should be a character (not int) of length 1.

After you are done modifying the input array in-place, return the new length of the array.

**Example:** Input: ["a","a","b","b","c","c","c"]

Output: Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"]

Explanation: "aa" is replaced by "a2". "bb" is replaced by "b2". "ccc" is replaced by "c3".

Note:

* All characters have an ASCII value in [35, 126].
* 1 <= len(chars) <= 1000.

**Follow up:**

Could you solve it using only O(1) extra space?

**Analysis:**

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** stringCompression(vector<**char**>& input)
10. {
11. **int** n = input.size();
12. **int** j = 0, cur = 0;
13. **for** (**int** i = 0; i < n; i++)
14. {
15. **if** ((i + 1 == n) || (input[i] != input[i+1]))
16. {
17. input[j] = input[i];
18. j++;
19. **if** (i - cur > 0)
20. {
21. auto n = i - cur + 1, cnt = 0;
22. **while** (n > 0)
23. {
24. input[j] = n % 10 + '0';
25. j++;
26. n /= 10;
27. cnt++;
28. }
29. reverse(input.begin() + j - cnt, input.begin() + j);
30. }
31. cur = i + 1;
32. }
33. }
35. **return** j;
36. }
37. };
39. **int** main()
40. {
41. Solution a;
42. vector<**char**> input{ 'a','b','b','b','b','b','b','b','b','b','b','a','a' };
43. //cout << a.stringCompression(input) << endl;
44. **int** n = a.stringCompression(input);
45. **for** (**int** i = 0; i < n; i++)
46. cout << input[i] << ' ';
47. system("pause");
48. **return** 1;
49. }

**Section 4. Linked List**

**002. Add two numbers (M)**

**Problem:** You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

**Example:** Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

Explanation: 342 + 465 = 807.

**Analysis:** learn about linked list, check if a linked list is circular, then perform adding two numbers

**Solution:**

1. // c++ program to check if a linked list is circular
2. #include<iostream>
4. **using** **namespace** std;
6. **struct** Node
7. {
8. **int** data;
9. Node\* next;
10. };
12. Node\* newNode(**int** data)
13. {
14. Node\* temp = **new** Node;
15. temp->data = data;
16. temp->next = NULL;
17. **return** temp;
18. }
20. **class** Solution
21. {
22. **public**:
23. Node\* addTwoNumbers(Node\* node1, Node\* node2)
24. {
25. **int** carry = 0;
26. Node\* head = newNode(0);
27. Node\* res = head;
28. **while** ((node1 != NULL) || (node2 != NULL))
29. {
30. **if** (node1 != NULL)
31. {
32. carry += node1->data;
33. node1 = node1->next;
34. }
35. **if** (node2 != NULL)
36. {
37. carry += node2->data;
38. node2 = node2->next;
39. }
40. res->next = newNode(carry % 10);
41. res = res->next;
42. carry /= 10;
43. //node1 = node1->next;
44. //node2 = node2->next;
45. }
46. **if** (carry) res->next = newNode(carry);
47. **return** head->next;
48. }
50. **bool** isCircular(Node\* head)
51. {
52. **if** (head == NULL) **return** **true**;
54. Node\* node = head->next;
55. **while** ((node != NULL) && (node != head))
56. {
57. node = node->next;
58. }
59. **return** (node == head);
60. }
61. };
63. **int** main()
64. {
65. Node\* head = newNode(1);
66. head->next = newNode(2);
67. head->next = newNode(3);
68. //head->next = head;
69. cout << "This linked list ";
70. Solution a;
71. a.isCircular(head) ? cout << " is circular!" : cout << "is not curcular!" << endl;
73. Node \*node1 = newNode(2);
74. node1->next = newNode(4);
75. node1->next->next = newNode(3);
77. Node \*node2 = newNode(5);
78. node2->next = newNode(6);
79. node2->next->next = newNode(9);
81. Node \*res = a.addTwoNumbers(node1, node2);
82. **while** (res != NULL)
83. {
84. cout << res->data << " ";
85. res = res->next;
86. }
87. system("pause");
88. **return** 1;
89. }

**021. Merge two sorted lists (E)**

**Problem:** Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

**Example:** Input: 1->2->4, 1->3->4

Output: 1->1->2->3->4->4

**Analysis:** key:

1. temp->next = node1 ? node1 : node2;

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **struct** Node
6. {
7. **int** data;
8. Node\* next;
9. };
11. Node\* newNode(**int** data)
12. {
13. Node\* temp = **new** Node;
14. temp->data = data;
15. temp->next = NULL;
16. **return** temp;
17. }
19. **class** Solution
20. {
21. **public**:
22. Node\* mergeLists(Node\* node1, Node\* node2)
23. {
24. Node\* head = newNode(0);
25. Node\* temp = head;
26. **while** ((node1 != NULL) && (node2 != NULL))
27. {
28. **if** (node1->data <= node2->data)
29. {
30. temp->next = node1;
31. node1 = node1->next;
32. }
33. **else**
34. {
35. temp->next = node2;
36. node2 = node2->next;
37. }
38. temp = temp->next;
39. }
40. temp->next = node1 ? node1 : node2;
41. **return** head->next;
42. }
43. };
45. **int** main()
46. {
47. Node\* node1 = newNode(1);
48. node1->next = newNode(2);
49. node1->next->next = newNode(4);
51. Node\* node2 = newNode(1);
52. node2->next = newNode(3);
53. node2->next->next = newNode(4);
55. Solution a;
56. Node\* res = a.mergeLists(node1, node2);
57. **while** (res != NULL)
58. {
59. cout << res->data << " ";
60. res = res->next;
61. }
62. system("pause");
63. **return** 1;
64. }

**024. Swap nodes in pairs (E)**

**Problem:** Given a linked list, swap every two adjacent nodes and return its head. You may not modify the values in the list's nodes, only nodes itself may be changed.

**Example:** Given 1->2->3->4, you should return the list as 2->1->4->3

**Analysis:**

0

dummy (pre)

1

1

3

4

cur



**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **struct** Node
6. {
7. **int** data;
8. Node\* next;
9. };
11. Node\* newNode(**int** data)
12. {
13. Node\* temp = **new** Node;
14. temp->data = data;
15. temp->next = NULL;
16. **return** temp;
17. }
19. **class** Solution
20. {
21. **public**:
22. Node\* swapNodes(Node\* node)
23. {
24. Node\* head = newNode(0);
25. head->next = node;
26. Node\* pre = head;
27. Node\* cur = head->next;
29. **while** ((cur != NULL) && (cur->next != NULL))
30. {
31. pre->next = cur->next;
32. cur->next = cur->next->next;
33. pre->next->next = cur;
35. pre = cur;
36. cur = cur->next;
37. }
38. **return** head->next;
39. }
40. };
42. **int** main()
43. {
44. Node\* node = newNode(1);
45. node->next = newNode(2);
46. node->next->next = newNode(3);
47. node->next->next->next = newNode(4);
48. node->next->next->next->next = newNode(5);
50. Solution a;
51. Node\* res = a.swapNodes(node);
52. **while** (res != NULL)
53. {
54. cout << res->data << " ";
55. res = res->next;
56. }
57. system("pause");
58. **return** 1;
59. }

**083. Remove duplicate from sorted list (E)**

**Problem:** Given a sorted linked list, delete all duplicates such that each element appear only once.

**Example:** Input: 1->1->2->3->3

Output: 1->2->3

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **struct** Node
6. {
7. **int** data;
8. Node\* next;
9. };
11. Node\* newNode(**int** data)
12. {
13. Node\* temp = **new** Node;
14. temp->data = data;
15. temp->next = NULL;
16. **return** temp;
17. }
19. **class** Solution
20. {
21. **public**:
22. Node\* removeDuplicated(Node\* node)
23. {
24. Node\* head = newNode(0);
25. head->next = node;
26. **while** (node != NULL && node->next != NULL)
27. {
28. //cout << node->next->data;
30. **if** (node->data == node->next->data)
31. {
32. Node\* temp = node->next;
33. node->next = node->next->next;
34. **delete** temp;
35. }
36. **else**
37. {
38. node = node->next;
39. }
40. }
42. **return** head->next;
43. }
44. };
46. **int** main()
47. {
48. Node\* node = newNode(1);
49. node->next = newNode(1);
50. node->next->next = newNode(1);
51. node->next->next->next = newNode(3);
52. node->next->next->next->next = newNode(3);
53. Solution a;
54. Node\* res = a.removeDuplicated(node);
55. **while** (res != NULL)
56. {
57. cout << res->data << " ";
58. res = res->next;
59. }
60. system("pause");
61. **return** 1;
62. }

**Section 5. Stack**

**020. Valid Parentheses (E)**

**Problem:** Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Note that an empty string is also considered valid.

**Example:** Input: "{[]}"

Output: true

Solution:

**Section 6. Math**

**007. Reverse integer (E)**

**Problem:** Given a 32-bit signed integer, reverse digits of an integer.

**Example:** Input: -123

Output: -321

**Note:** Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: [−231, 231 − 1]. For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **int** reverseInteger(**int** x)
9. {
10. **int** res = 0;
11. **while** (x)
12. {
13. auto prev = res;
14. res \*= 10;
15. res += x % 10;
16. **if** (res / 10 != prev)
17. {
18. res = 0;
19. **break**;
20. }
21. x /= 10;
22. }
23. **return** res;
24. }
25. };
27. **int** main()
28. {
29. Solution a;
30. cout << a.reverseInteger(123) << endl; // 2^31-1=2147483647
31. system("pause");
32. **return** 1;
33. }

**009. Palindrome number (E)**

**Problem:** Determine whether an integer is a palindrome. An integer is a palindrome when it reads the same backward as forward.

**Example:** Input: 121 Output: true

Input: -121 Output: false

**Follow up:** could you solve it without converting the integer to a string?

**Solution:**

1. #include <iostream>
2. #include <string>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **bool** palindromeNumber(**int** x)
10. {
11. **if** (x < 0) **return** **false**;
13. string num = to\_string(x);
14. **int** i = 0, j = num.size() - 1;
16. **while** (i < j)
17. {
18. **if** (num[i] != num[j]) **return** **false**;
19. i++;
20. j--;
21. }
22. **return** **true**;
23. }
24. };
26. **class** Solution2
27. {
28. **public**:
29. **bool** palindromeNumber(**int** x)
30. {
31. **if** (x < 0) **return** **false**;
33. **int** temp = x, reverse = 0;
34. **while** (temp)
35. {
36. **if** (isOverflow(reverse, temp % 10)) **return** **false**;
37. **else**
38. {
39. reverse = reverse \* 10 + temp % 10;
40. temp /= 10;
41. }
42. }
43. **return** reverse == x;
44. }
46. **private**:
47. **bool** isOverflow(**int** q, **int** r)
48. {
49. **static** **const** **int** q\_max = numeric\_limits<**int**>::max() / 10;
50. **static** **const** **int** r\_max = numeric\_limits<**int**>::max() % 10;
51. **return**(q > q\_max) || ((q == q\_max) && (r > r\_max));
52. }
53. };
55. **int** main()
56. {
57. Solution2 a;
58. cout << a.palindromeNumber(121) << endl; // -121
59. system("pause");
60. **return** 1;
61. }

**012. Integer to roman (E)**

**Problem:** Given an integer, convert it to a roman numeral. Input is guaranteed to be within the range from 1 to 3999.

**Example:** Input: 1994

Output: "MCMXCIV"

Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

**Analysis:** similar to 013…

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <string>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. string integerToRoman(**int** x)
11. {
12. vector<string> rom = { "M", "CM", "D", "CD", "C",
13. "XC", "L", "XL", "X", "IX", "V", "IV", "I" };
15. vector<**int**> num = { 1000, 900, 500, 400, 100,
16. 90, 50, 40, 10, 9, 5, 4, 1 };
18. string res;
19. **int** i = 0;
20. **while** (x > 0)
21. {
22. **int** times = x / num[i];
23. **while** (times)
24. {
25. x -= num[i];
26. res.append(rom[i]);
27. times--;
28. }
29. i++;
30. }
31. **return** res;
32. }
33. };
35. **int** main()
36. {
37. Solution a;
38. cout << a.integerToRoman(1994) << endl;
39. system("pause");
40. **return** 1;
41. }

**013. Roman to integer (E)**

**Problem:** Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol Value

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

For example, two is written as II in Roman numeral, just two one's added together. Twelve is written as, XII, which is simply X + II. The number twenty seven is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

I can be placed before V (5) and X (10) to make 4 and 9.

X can be placed before L (50) and C (100) to make 40 and 90.

C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer. Input is guaranteed to be within the range from 1 to 3999.

**Example:** Input: "MCMXCIV"

Output: 1994

Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

**Analysis:**

* hash map
* subtract one more lower digit…

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <unordered\_map>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **int** romanToInt(string str)
11. {
12. unordered\_map<**char**, **int**> rom\_num = {
13. {'I', 1},
14. {'V', 5},
15. {'X', 10},
16. {'L', 50},
17. {'C', 100},
18. {'D', 500},
19. {'M', 1000}
20. };
21. **int** res = 0;
22. **for** (**int** i = 0; i < str.size(); i++)
23. {
24. **if** ((i > 0) && (rom\_num[str[i]] > rom\_num[str[i - 1]]))
25. {
26. res += rom\_num[str[i]] - 2 \* rom\_num[str[i - 1]];
27. }
28. **else**
29. {
30. res += rom\_num[str[i]];
31. }
32. }
33. **return** res;
34. }
35. };
37. **int** main()
38. {
39. Solution a;
40. cout << a.romanToInt("MCMXCIV") << endl;
41. system("pause");
42. **return** 0;
43. }

**168. Excel sheet column title (E)**

**Problem:** Given a positive integer, return its corresponding column title as appear in an Excel sheet.

**Example:** Input: 701

Output: "ZY"

Notice: first subtract 1 then…

**Solution:**

1. #include <iostream>
2. #include <string>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string excelTitle(**int** x)
10. {
11. string str;
12. **while** (x)
13. {
14. str.push\_back((x - 1) % 26 + 'A');
15. x = (x - 1) / 26;
16. }
17. reverse(str.begin(), str.end());
18. **return** str;
19. }
20. };
22. **int** main()
23. {
24. Solution a;
25. cout << a.excelTitle(701) << endl;
26. system("pause");
27. **return** 1;
28. }

**171. Excel sheet column number (E)**

**Problem:** Given a column title as appear in an Excel sheet, return its corresponding column number.

**Example:** Input: "AB" Output: 28

Input: "ZY" Output: 701

**Solution:**

1. #include <iostream>
2. #include <string>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** excelToNumber(string str)
10. {
11. **int** result = 0;
12. **for** (**int** i = str.size()-1; i >= 0; i--)
13. {
14. result += (str[i] - 'A' + 1) \* pow(26, str.size() -1 - i);
15. }
16. **return** result;
17. }
18. };
20. **int** main()
21. {
22. Solution a;
23. cout << a.excelToNumber("AB") << endl;
24. system("pause");
25. **return** 1;
26. }

**172. Factorial trailing zeros (E)**

**Problem:** Given an integer n, return the number of trailing zeroes in n!.

**Example:** Input: 5

Output: 1

Explanation: 5! = 120, one trailing zero

**Note:** Your solution should be in logarithmic time complexity.

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **int** fractorialTrailingZeros(**int** x)
9. {
10. **int** num = 1;
11. **for** (**int** i = x; i >= 1; i--)
12. {
13. num \*= i;
14. }
15. **int** res = 0;
16. **while** (num % 10 == 0)
17. {
18. num /= 10;
19. res++;
20. }
21. **return** res;
22. }
23. };
25. **class** Solution2
26. {
27. **public**:
28. **int** fractorialTrailingZeros(**int** x)
29. {
30. **int** res = 0;
31. **while** (x)
32. {
33. res += x / 5;
34. x /= 5;
35. }
36. **return** res;
37. }
38. };
40. **int** main()
41. {
42. Solution2 a;
43. cout << a.fractorialTrailingZeros(10) << endl;
44. system("pause");
45. **return** 1;
46. }

**223. Rectangle area (M)**

**Problem:** Find the total area covered by two rectilinear rectangles in a 2D plane. Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.

**Example:** Input: A = -3, B = 0, C = 3, D = 4, E = 0, F = -1, G = 9, H = 2

Output: 45

**Note:** Assume that the total area is never beyond the maximum possible value of int.

**Solution:**