1

11. Find the Longest Common Prefix

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
string longestCommonPrefix(vector<string>& strs) {
    if (strs.empty()) return "";
    string prefix = strs[0];
    for (int i = 1; i < strs.size(); i++) {
        int j = 0;
        while (j < strs[i].length() && j < prefix.length() &&
strs[i][j] == prefix[j]) {
            j++;
        }
        prefix = prefix.substr(0, j);
        if (prefix == "") return "";
    }
    return prefix;
}
int main() {
    vector<string> strs = {"flower", "flow", "flight"};
    cout << "Longest Common Prefix: " <<
longestCommonPrefix(strs) << endl;
    return 0;
}
```

2

13. Check if a String is a Valid Parentheses String

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;
bool isValid(string s) {
    stack<char> st;
    for (char c : s) {
        if (c == '(' || c == '{' || c == '[') {
            st.push(c);
        } else {
            if (st.empty()) return false;
            char top = st.top();
            st.pop();
            if ((c == ')' && top != '(') || (c == '}' && top != '{') ||
(c == ']' && top != '[')) {
                return false;
            }
        }
    }
    return st.empty();
}
int main() {
    string s = "{[()]}";
    if (isValid(s)) cout << "Valid Parentheses" << endl;
    else cout << "Invalid Parentheses" << endl;
    return 0;
}
```

3

14. Find the Longest Substring Without Repeating Characters

```
#include <iostream>
#include <unordered_map>
#include <string>
using namespace std;
int longestSubstringWithoutRepeating(string s) {
    unordered_map<char, int> charIndex;
    int maxLength = 0, start = 0;
    for (int end = 0; end < s.length(); end++) {
        if (charIndex.find(s[end]) != charIndex.end()) {
            start = max(start, charIndex[s[end]] + 1);
        }
        charIndex[s[end]] = end;
        maxLength = max(maxLength, end - start + 1);
    }
    return maxLength;
}
int main() {
    string s = "abcabcbb";
    cout << "Longest Substring Without Repeating
Characters: " << longestSubstringWithoutRepeating(s)
<< endl;
    return 0;
}
```

4

15. Palindrome Partitioning

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
bool isPalindrome(string s, int start, int end) {
    while (start < end) {
        if (s[start] != s[end]) return false;
        start++;
        end--;
    }
    return true;
}
void partitionHelper(string s, int start, vector<string>&
current, vector<vector<string>>& result) {
    if (start == s.length()) {
        result.push_back(current);
        return;
    }
    for (int end = start; end < s.length(); end++) {
        if (isPalindrome(s, start, end)) {
            current.push_back(s.substr(start, end - start +
1));
            partitionHelper(s, end + 1, current, result);
            current.pop_back();
        }
    }
}
vector<vector<string>> partition(string s) {
    vector<vector<string>> result;
    vector<string> current;
    partitionHelper(s, 0, current, result);
    return result;
}
int main() {
    string s = "aab";
    vector<vector<string>> result = partition(s);
    for (const auto& partition : result) {
        for (const auto& word : partition) {
            cout << word << " ";
        }
        cout << endl;
    }
    return 0;
}
```

5

17. Find All Permutations of a String

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
void permute(string s, int l, int r, vector<string>& result) {
    if (l == r) {
        result.push_back(s);
    } else {
        for (int i = l; i <= r; i++) {
            swap(s[l], s[i]);
            permute(s, l + 1, r, result);
            swap(s[l], s[i]); // backtrack
        }
    }
}
vector<string> getPermutations(string s) {
    vector<string> result;
    permute(s, 0, s.length() - 1, result);
    return result;
}
int main() {
    string s = "abc";
    vector<string> result = getPermutations(s);
    for (const string& perm : result) {
        cout << perm << endl;
    }
    return 0;
}
```

6

Longest Palindromic Substring

```
#include <iostream>
#include <string>
using namespace std;
string expandFromCenter(string s, int left, int right) {
    while (left >= 0 && right < s.length() && s[left] ==
s[right]) {
        left--;
        right++;
    }
    return s.substr(left + 1, right - left - 1);
}
string longestPalindrome(string s) {
    if (s.length() < 1) return "";
    string longest;
    for (int i = 0; i < s.length(); i++) {
        string odd = expandFromCenter(s, i, i);
        string even = expandFromCenter(s, i, i + 1);
        if (odd.length() > longest.length()) longest = odd;
        if (even.length() > longest.length()) longest = even;
    }
    return longest;
}
int main() {
    string s = "babad";
    cout << "Longest Palindromic Substring: " <<
longestPalindrome(s) << endl;
    return 0;
}
```

7

21. Wildcard Matching

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
bool isMatch(string s, string p) {
    int m = s.length(), n = p.length();
    vector<vector<bool>> dp(m + 1, vector<bool>(n + 1,
false));
    dp[0][0] = true;
    for (int i = 1; i <= n; i++) {
        if (p[i - 1] == '*') dp[0][i] = dp[0][i - 1];
    }
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (p[j - 1] == s[i - 1] || p[j - 1] == '?') {
                dp[i][j] = dp[i - 1][j - 1];
            } else if (p[j - 1] == '*') {
                dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
            }
        }
    }
    return dp[m][n];
}
int main() {
    string s = "adceb", p = "*a*b";
    if (isMatch(s, p)) {
        cout << "Pattern matches!" << endl;
    } else {
        cout << "Pattern doesn't match!" << endl;
    }
    return 0;
}
```

8

23. Minimum Window Substring

```
#include <iostream>
#include <string>
#include <unordered_map>
#include <climits>
using namespace std;
string minWindow(string S, string T) {
    unordered_map<char, int> charCountT, charCountS;
    4 (char c : T) charCountT[c]++;
    int lft = 0, right = 0, minLen = INT_MAX, start = 0;
    int required = charCountT.size(), formed = 0;
    while (right < S.length()) {
        char rightChar = S[right];
        charCountS[rightChar]++;
        if (charCountT.count(rightChar) &&
charCountS[rightChar] == charCountT[rightChar]) {
            formed++;
        }
        while (lft <= right && formed == required) {
            if (right - lft + 1 < minLen) {
                minLen = right - lft + 1;
                start = lft;
            }
            char
 leftChar = S[lft];
            charCountS[leftChar]--;
            if (charCountT.count(leftChar) &&
charCountS[leftChar] < charCountT[leftChar]) {
                formed--;
            }
            lft++;
        }
        right++;
    }
    return minLen == INT_MAX ? "" : S.substr(start,
minLen);
}
int main() {
    string S = "ADOBECODEBANC", T = "ABC";
    cout << "Minimum Window Substring: " <<
minWindow(S, T) << endl;
    return 0;
}
```

24. Z-Algorithm (Pattern Matching) 9

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
vector<int> ZAlgorithm(string s) {
    int n = s.length();
    vector<int> Z(n, 0);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        if (i <= r) {
            Z[i] = min(r - i + 1, Z[i - l]);
        }
        while (i + Z[i] < n && s[Z[i]] == s[i + Z[i]]) {
            Z[i]++;
        }
        if (i + Z[i] - 1 > r) {
            l = i;
            r = i + Z[i] - 1;
        }
    }
    return Z;
}
int main() {
    string s = "aabxaabxca";
    vector<int> Z = ZAlgorithm(s);
    for (int z : Z) {
        cout << z << " ";
    }
    cout << endl;
    return 0;
}
```

27. Longest Common Prefix (LCP) 10

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
string longestCommonPrefix(vector<string>& strs) {
    if (strs.empty()) return "";
    string prefix = strs[0];
    for (int i = 1; i < strs.size(); i++) {
        int j = 0;
        while (j < prefix.length() && j < strs[i].length() &&
prefix[j] == strs[i][j]) {
            j++;
        }
        prefix = prefix.substr(0, j);
    }
    return prefix;
}
int main() {
    vector<string> strs = {"flower", "flow", "flight"};
    cout << "Longest Common Prefix: " <<
longestCommonPrefix(strs) << endl;
    return 0;
}
```

29. Find All Occurrences of a Substring 11

```
#include <iostream>
#include <string>
using namespace std;
void findAllOccurrences(string text, string pattern) {
    int n = text.length();
    int m = pattern.length();
    for (int i = 0; i <= n - m; i++) {
        if (text.substr(i, m) == pattern) {
            cout << "Pattern found at index " << i << endl;
        }
    }
}
int main() {
    string text = "ABABABAB";
    string pattern = "AB";
    findAllOccurrences(text, pattern);
    return 0;
}
```