### Week 1: Basics & Implementation

**Topics:** - Input/Output, Loops, Conditionals - Arrays, Strings, Basic Math - Simple sorting

**Weekly Tips:** - Focus on writing clean, readable code. - Always test edge cases (0, 1, negative numbers, large numbers). - Use online judge IDE or local compiler to verify behavior.

### Week 2: Ad-hoc & Simulation

**Topics:** - Simulation - Ad-hoc logic problems - Greedy basics

**Weekly Tips:** - Think step by step, simulate processes on paper first. - Carefully read problem constraints to optimize loops. - Greedy approach works if problem guarantees local optimality leads to global optimality.

### Week 3: Sorting & Searching

**Topics:** - Sorting algorithms: QuickSort, MergeSort, STL sort - Binary Search & Ternary Search - Two-pointer technique

**Weekly Tips:** - Always check if STL sort suffices before implementing manually. - Binary search can be applied to sorted arrays or answer space. - Two-pointer technique is useful for finding pairs, sums, or sliding windows.

### Week 4: Strings & Pattern Matching

**Topics:** - String searching: KMP, Rabin-Karp - Palindromes & substrings - Prefix/Suffix techniques

**Weekly Tips:** - Understand failure function in KMP for linear-time matching. - Use rolling hash for fast substring comparison. - Practice manipulating strings efficiently with STL.

### Problem 1: Power Strings

**Link:** [UVa 10298](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=1222) **Difficulty:** Intermediate

**C++ Solution with Explanation Comments:**

#include <iostream>  
#include <string>  
using namespace std;  
  
int main() {  
 string s;  
 while (cin >> s && s != ".") {  
 int n = s.length();  
 int k = 1;  
 for (int i = 1; i <= n; i++) {  
 if (n % i == 0) {  
 string t = s.substr(0, i);  
 string tmp = "";  
 for (int j = 0; j < n/i; j++) tmp += t;  
 if (tmp == s) { k = n/i; break; }  
 }  
 }  
 cout << k << endl;  
 }  
 return 0;  
}

**Explanation Comments:** - Check substring repetition to find smallest repeated unit. - Use substr to generate candidate substring. - Loop through all divisors of string length. - Demonstrates basic string manipulation and matching.

### Problem 2: Extend to Palindrome

**Link:** [UVa 11475](https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=2456) **Difficulty:** Intermediate

**C++ Solution with Explanation Comments:**

#include <iostream>  
#include <string>  
using namespace std;  
  
bool isPalindrome(const string& s) {  
 int i = 0, j = s.length()-1;  
 while (i < j) {  
 if (s[i] != s[j]) return false;  
 i++; j--;  
 }  
 return true;  
}  
  
int main() {  
 string s;  
 while (cin >> s) {  
 string tmp = s;  
 for (int i = 0; i < s.length(); i++) {  
 if (isPalindrome(tmp)) break;  
 tmp = s.substr(0, s.length() + i + 1); // Append characters to make palindrome  
 }  
 cout << tmp.length() << endl;  
 }  
 return 0;  
}

**Explanation Comments:** - Check palindrome by comparing characters from both ends. - Extend string minimally to make palindrome. - Illustrates substring operations and iterative checking.

### Problem 3: String Matching

**Link:** [Kattis String Matching](https://open.kattis.com/problems/stringmatching) **Difficulty:** Intermediate

**C++ Solution with Explanation Comments:**

#include <iostream>  
#include <vector>  
#include <string>  
using namespace std;  
  
vector<int> KMP(const string& text, const string& pattern) {  
 int n = text.size(), m = pattern.size();  
 vector<int> lps(m, 0), result;  
 int len = 0, i = 1;  
 // Preprocess pattern  
 while (i < m) {  
 if (pattern[i] == pattern[len]) { lps[i++] = ++len; }  
 else if (len != 0) len = lps[len-1];  
 else lps[i++] = 0;  
 }  
 // Search  
 i = 0; int j = 0;  
 while (i < n) {  
 if (text[i] == pattern[j]) { i++; j++; }  
 if (j == m) { result.push\_back(i-j); j = lps[j-1]; }  
 else if (i < n && text[i] != pattern[j]) {  
 if (j != 0) j = lps[j-1];  
 else i++;  
 }  
 }  
 return result;  
}  
  
int main() {  
 string text, pattern;  
 cin >> text >> pattern;  
 vector<int> positions = KMP(text, pattern);  
 for (int pos : positions) cout << pos << " ";  
 cout << endl;  
 return 0;  
}

**Explanation Comments:** - Implements KMP for linear-time pattern searching. - lps array stores longest prefix-suffix lengths. - Demonstrates preprocessing and matching efficiently.

### Problem 4: Palindrome Free Strings

**Link:** [Kattis Palindrome Free Strings](https://open.kattis.com/problems/palindromefree) **Difficulty:** Intermediate

**C++ Solution with Explanation Comments:**

#include <iostream>  
#include <string>  
using namespace std;  
  
bool isPalindrome(const string& s) {  
 int n = s.length();  
 for (int i = 0; i < n/2; i++) {  
 if (s[i] != s[n-1-i]) return false;  
 }  
 return true;  
}  
  
int main() {  
 int n; cin >> n;  
 string s = "a";  
 while (s.length() < n) {  
 char next = 'a';  
 s += next;  
 while (s.length() >= 2 && isPalindrome(s.substr(s.length()-2))) {  
 next++;  
 s[s.length()-1] = next;  
 }  
 }  
 cout << s << endl;  
 return 0;  
}

**Explanation Comments:** - Construct string while avoiding palindromes of length >=2. - Uses iterative checking and character increment. - Teaches problem-solving using string manipulation and constraints.

**End of Week 4** - Focus on understanding KMP and substring operations. - Practice detecting and avoiding palindromes. - Use rolling hash or LPS arrays to speed up pattern matching in large strings.