**BROKEN LETTERS**

#include <iostream>

#include <string>

#include <set>

#include <sstream>

using namespace std;

int main() {

string input, brokenLetters;

cout << "Enter the text: ";

getline(cin, input);

cout << "Enter the broken letters: ";

cin >> brokenLetters;

set<char> brokenSet;

for(char c : brokenLetters) {

brokenSet.insert(c);

}

istringstream iss(input);

string word;

int count = 0;

while (iss >> word) {

bool canType = true;

for (char c: word) {

if(brokenSet.find(c) != brokenSet.end()) {

canType = false;

break;

}

}

if(canType) {

count++;

}

}

cout << "Number of words you can type: " << count;

return 0;

}

**FLIPPING AN IMAGE**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

int n;

cout << "Enter the size of the matrix N: ";

cin >> n;

vector<vector<int>> matrix(n, vector<int>(n));

cout << "Enter the elements of the matrix (0 or 1 only):\n";

int i,j;

for(i=0; i<n; ++i) {

for(j=0; j<n; ++j) {

cin >> matrix[i][j];

}

}

for (int i = 0; i < n; ++i) {

reverse(matrix[i].begin(), matrix[i].end());

}

for(i=0; i<n; i++) {

for(j=0; j<n; ++j) {

matrix[i][j] = 1 - matrix[i][j];

}

}

cout << "Flipped and Inverted Image:\n";

for(i=0; i<n; ++i) {

for(j=0; j<n; ++j) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

return 0;

}

#include <iostream>

#include <string>

#include <vector>

void printCombinations(char digit) {

const std::vector<std::string> combinations = {

"abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"

};

int index = digit - '2';

if (index >= 0 && index < combinations.size()) {

std::cout << digit << ": " << combinations[index] << std::endl;

} else {

std::cout << digit << ": Invalid digit" << std::endl;

}

}

void printCombinationsWithNext(char firstDigit, char secondDigit, std::string& output) {

const std::vector<std::string> combinations = {

"abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"

};

int firstIndex = firstDigit - '2';

int secondIndex = secondDigit - '2';

bool flag = false;

output += "[";

if (firstIndex >= 0 && firstIndex < combinations.size() && secondIndex >= 0 && secondIndex < combinations.size()) {

for (char i : combinations[firstIndex]) {

for (char j : combinations[secondIndex]) {

if (flag) {

output += ", ";

}

output += i;

output += j;

flag = true;

}

}

} else {

output += "Invalid digits";

}

output += "]";

}

int main() {

std::string phoneNumber;

std::string output;

std::cout << "Enter a phone number: ";

std::cin >> phoneNumber;

// Iterate over each digit and print its combinations

for (size\_t i = 0; i + 1 < phoneNumber.length(); i++) {

printCombinationsWithNext(phoneNumber[i], phoneNumber[i + 1], output);

}

std::cout << output << std::endl;

return 0;

}

#include <iostream>

#include <string>

#include <unordered\_map>

#include <cctype>

// Define Morse code table using an unordered\_map for efficient lookup

std::unordered\_map<char, std::string> morse\_table = {

{'A', ".-"}, {'B', "-..."}, {'C', "-.-."}, {'D', "-.."}, {'E', "."}, {'F', "..-."},

{'G', "--."}, {'H', "...."}, {'I', ".."}, {'J', ".---"}, {'K', "-.-"}, {'L', ".-.."},

{'M', "--"}, {'N', "-."}, {'O', "---"}, {'P', ".--."}, {'Q', "--.-"}, {'R', ".-."},

{'S', "..."}, {'T', "-"}, {'U', "..-"}, {'V', "...-"}, {'W', ".--"}, {'X', "-..-"},

{'Y', "-.--"}, {'Z', "--.."}, {'0', "-----"}, {'1', ".----"}, {'2', "..---"},

{'3', "...--"}, {'4', "....-"}, {'5', "....."}, {'6', "-...."}, {'7', "--..."},

{'8', "---.."}, {'9', "----."}, {' ', " "}, // Space

{'.', ".-.-.-"}, {',', "--..--"}, {'?', "..--.."}, {'!', "-.-.--"}, {'/', "-..-."},

{'(', "-.--."}, {')', "-.--.-"}, {'&', ".-..."}, {':', "---..."}, {';', "-.-.-."},

{'=', "-...-"}, {'+', ".-.-."}, {'-', "-....-"}, {'\_', "..--.-"}, {'"', ".-..-."},

{'$', "...-..-"}, {'@', ".--.-."}

};

// Function to convert string to Morse code

void string\_to\_morse(const std::string& str) {

for (char c : str) {

auto morse = morse\_table.find(toupper(c)); // Convert to upper and find in map

if (morse != morse\_table.end()) {

std::cout << morse->second << " "; // Output the Morse code

} else {

std::cout << ""; // Handle characters not in the Morse table

}

}

}

int main() {

std::string input;

std::cout << "Enter a string: ";

std::getline(std::cin, input); // Read full line from standard input

std::cout << "Morse Code: ";

string\_to\_morse(input);

std::cout << "\n";

return 0;

}

#include <iostream>

#include <string>

void stringToBinary(const std::string& str) {

for (char c : str) {

// Convert each character to its ASCII value

int asciiValue = static\_cast<int>(c);

// Convert ASCII value to binary representation

for (int j = 7; j >= 0; j--) {

int bit = (asciiValue >> j) & 1;

std::cout << bit;

}

std::cout << " "; // Separate each byte by space

}

std::cout << "\n";

}

int main() {

std::string str;

std::cout << "Enter a string: ";

std::getline(std::cin, str);

std::cout << "Binary representation: ";

stringToBinary(str);

return 0;

}

#include <iostream>

#include <vector>

#include <string>

std::string longestCommonPrefix(const std::vector<std::string>& strs) {

if (strs.empty()) {

return "";

}

// Set the first string as the initial common prefix

std::string prefix = strs[0];

// Iterate through the remaining strings to find the common prefix

for (size\_t i = 1; i < strs.size(); ++i) {

size\_t j = 0;

// Compare characters until reaching the end of one of the strings or finding a mismatch

while (j < prefix.size() && j < strs[i].size() && prefix[j] == strs[i][j]) {

j++;

}

// Update the prefix to the common part

prefix.resize(j);

}

return prefix;

}

int main() {

int strsSize;

std::cout << "Enter the number of strings: ";

std::cin >> strsSize;

std::cin.ignore(); // Consume the newline character remaining in the input buffer

std::vector<std::string> strs(strsSize);

for (int i = 0; i < strsSize; ++i) {

std::cout << "Enter string " << i + 1 << ": ";

std::getline(std::cin, strs[i]);

}

std::string prefix = longestCommonPrefix(strs);

std::cout << "Longest common prefix: " << prefix << std::endl;

return 0;

}

#include <iostream>

#include <string>

bool isVowel(char c) {

// Check if the character is a vowel (case-insensitive)

switch (tolower(c)) {

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

return true;

default:

return false;

}

}

void reverseVowels(std::string& s) {

// Initialize two pointers for the beginning and end of the string

int left = 0, right = s.length() - 1;

while (left < right) {

// Find the next vowel from the left

while (left < right && !isVowel(s[left])) {

left++;

}

// Find the next vowel from the right

while (left < right && !isVowel(s[right])) {

right--;

}

// Swap the vowels

if (left < right) {

std::swap(s[left], s[right]);

left++;

right--;

}

}

}

int main() {

std::string s;

std::cout << "Enter a string: ";

std::getline(std::cin, s);

// Reverse the vowels

reverseVowels(s);

std::cout << "Output: " << s << std::endl;

return 0;

}

String Operation

#include <iostream>

#include <cstring>

#include <sstream>

#include <iomanip>

using namespace std;

int main() {

string input;

cout << "Enter the operation: ";

getline(cin, input);

istringstream iss(input);

double num1, num2;

char op;

if (!(iss >> num1 >> op >> num2)) {

cout << "Invalid input format!" << endl;

return 1;

}

double result;

switch(op) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

result = num1 / num2;

break;

default:

break;

}

cout << "Result: " << fixed << setprecision(2) << result << endl;

return 0;

}

**MATRIX DIAGONAL SUM**

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n;

cout << "Enter the size of the matrix N: ";

cin >> n;

vector<vector<int>> matrix(n, vector<int>(n));

cout << "Enter the elements of the matrix\n";

int i,j;

for(i=0; i<n; ++i) {

for(j=0; j<n; ++j) {

cin >> matrix[i][j];

}

}

int sum = 0;

for(i=0; i<n; ++i) {

sum += matrix[i][i];

if(i != n-i-1) {

sum += matrix[i][n-i-1];

}

}

cout << "Sum of the diagonals is: " << sum;

return 0;

}

**TIME DIFFERENCE**

#include <iostream>

#include <iomanip>

#include <string>

using namespace std;

struct Time {

int hour;

int minute;

int second;

};

Time parseTime() {

Time t;

int hour, minute, second;

string period;

cin >> hour >> minute >> second >> period;

if (period == "PM" && hour != 12) {

hour += 12;

} else if (period == "AM" && hour == 12) {

hour = 0;

}

t.hour = hour;

t.minute = minute;

t.second = second;

return t;

}

int main() {

Time startTime, endTime, diffTime;

cout << "Enter the first time (hh mm ss AM/PM): ";

startTime = parseTime();

cout << "Enter the second time (hh mm ss AM/PM): ";

endTime = parseTime();

int startSeconds = startTime.hour \* 3600 + startTime.minute \* 60 + startTime.second;

int endSeconds = endTime.hour \* 3600 + endTime.minute \* 60 + endTime.second;

int diffSeconds = endSeconds - startSeconds;

diffTime.hour = diffSeconds / 3600;

diffTime.minute = (diffSeconds % 3600) / 60;

diffTime.second = diffSeconds % 60;

cout << "Time Difference: "

<< diffTime.hour << " hours, "

<< diffTime.minute << " minutes,"

<< " and " << diffTime.second << " seconds" << endl;

return 0;

}