Практика СП:

Билет 1

**#include <windows.h>**

**#include <iostream>**

**#include <string>**

**void PrintProcessInfo(DWORD processId) {**

**HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, FALSE, processId);**

**if (hProcess) {**

**TCHAR processName[MAX\_PATH];**

**if (GetModuleFileNameEx(hProcess, NULL, processName, sizeof(processName) / sizeof(TCHAR))) {**

**std::wcout << L"Process ID: " << processId << L"\n";**

**std::wcout << L"Process Name: " << processName << L"\n";**

**}**

**CloseHandle(hProcess);**

**} else {**

**std::cerr << "Failed to open process for querying information.\n";**

**}**

**}**

**int main() {**

**STARTUPINFO si;**

**PROCESS\_INFORMATION pi;**

**ZeroMemory(&si, sizeof(si));**

**si.cb = sizeof(si);**

**ZeroMemory(&pi, sizeof(pi));**

**// Запуск WordPad**

**if (!CreateProcess(**

**L"C:\\Program Files\\Windows NT\\Accessories\\wordpad.exe", // Путь к WordPad**

**NULL, // Командная строка**

**NULL, // Процесс по умолчанию**

**NULL, // Поток по умолчанию**

**FALSE, // Не наследовать дескрипторы**

**0, // Флаги создания**

**NULL, // Использовать переменные окружения**

**NULL, // Использовать текущий каталог**

**&si, // Структура STARTUPINFO**

**&pi) // Структура PROCESS\_INFORMATION**

**) {**

**std::cerr << "CreateProcess failed (" << GetLastError() << ").\n";**

**return 1;**

**}**

**// Вывод информации о процессе**

**PrintProcessInfo(pi.dwProcessId);**

**// Ожидание ввода от пользователя для завершения процесса**

**std::cout << "Press Enter to close the process...\n";**

**std::cin.get();**

**// Завершение процесса**

**TerminateProcess(pi.hProcess, 0);**

**CloseHandle(pi.hProcess);**

**CloseHandle(pi.hThread);**

**std::cout << "Process closed.\n";**

**return 0;**

**}**

**Билет 2**

**#include <iostream>**

**#include <sstream>**

**#include <string>**

**int main() {**

**std::string input;**

**std::cout << "Введите строку: ";**

**std::getline(std::cin, input);**

**std::istringstream iss(input);**

**std::string word;**

**int count = 0;**

**while (iss >> word) {**

**// Проверяем, является ли слово числом**

**if (std::all\_of(word.begin(), word.end(), ::isdigit)) {**

**count++;**

**}**

**}**

**std::cout << "Количество чисел в строке: " << count << std::endl;**

**return 0;**

**}**

**#include <windows.h>**

**#include <iostream>**

**int main() {**

**STARTUPINFO si;**

**PROCESS\_INFORMATION pi;**

**ZeroMemory(&si, sizeof(si));**

**si.cb = sizeof(si);**

**ZeroMemory(&pi, sizeof(pi));**

**// Запуск дочернего процесса**

**if (!CreateProcess(**

**L"child.exe", // Путь к дочернему процессу**

**NULL, // Командная строка**

**NULL, // Процесс по умолчанию**

**NULL, // Поток по умолчанию**

**FALSE, // Не наследовать дескрипторы**

**0, // Флаги создания**

**NULL, // Использовать переменные окружения**

**NULL, // Использовать текущий каталог**

**&si, // Структура STARTUPINFO**

**&pi) // Структура PROCESS\_INFORMATION**

**) {**

**std::cerr << "CreateProcess failed (" << GetLastError() << ").\n";**

**return 1;**

**}**

**// Ожидание завершения дочернего процесса**

**WaitForSingleObject(pi.hProcess, INFINITE);**

**// Закрытие дескрипторов**

**CloseHandle(pi.hProcess);**

**CloseHandle(pi.hThread);**

**return 0;**

**}**

**Билет 3**

**#include <iostream>**

**#include <fstream>**

**#include <thread>**

**#include <chrono>**

**#include <ctime>**

**#include <mutex>**

**std::mutex fileMutex; // Мьютекс для синхронизации доступа к файлу**

**void writeToFile(int threadNumber) {**

**// Получаем текущее время**

**auto now = std::chrono::system\_clock::now();**

**std::time\_t now\_c = std::chrono::system\_clock::to\_time\_t(now);**

**// Форматируем время**

**std::string timeStr = std::ctime(&now\_c);**

**timeStr.pop\_back(); // Убираем символ новой строки**

**// Записываем в файл**

**{**

**std::lock\_guard<std::mutex> guard(fileMutex); // Защита критической секции**

**std::ofstream outFile("output.txt", std::ios::app);**

**if (outFile.is\_open()) {**

**outFile << "Thread " << threadNumber << ": " << timeStr << std::endl;**

**outFile.close();**

**} else {**

**std::cerr << "Не удалось открыть файл для записи." << std::endl;**

**}**

**}**

**}**

**int main() {**

**const int numThreads = 3;**

**std::thread threads[numThreads];**

**// Запускаем потоки**

**for (int i = 0; i < numThreads; ++i) {**

**threads[i] = std::thread(writeToFile, i + 1);**

**}**

**// Ждем завершения всех потоков**

**for (int i = 0; i < numThreads; ++i) {**

**threads[i].join();**

**}**

**std::cout << "Запись завершена." << std::endl;**

**return 0;**

**}**

**Билет 4**

**#include <iostream>**

**#include <string>**

**#include <vector>**

**#include <regex>**

**void WordParse(const std::string& inputString) {**

**// Регулярное выражение для поиска слов (последовательности букв)**

**std::regex wordRegex("[a-zA-Z]+");**

**std::sregex\_iterator wordsBegin = std::sregex\_iterator(inputString.begin(), inputString.end(), wordRegex);**

**std::sregex\_iterator wordsEnd = std::sregex\_iterator();**

**// Вектор для хранения найденных слов**

**std::vector<std::string> words;**

**// Перебираем найденные слова и добавляем их в вектор**

**for (std::sregex\_iterator i = wordsBegin; i != wordsEnd; ++i) {**

**words.push\_back(i->str());**

**}**

**// Выводим количество слов**

**std::cout << "Количество слов: " << words.size() << std::endl;**

**// Выводим найденные слова**

**std::cout << "Найденные слова:" << std::endl;**

**for (const auto& word : words) {**

**std::cout << word << std::endl;**

**}**

**}**

**int main() {**

**std::string inputString;**

**// Запрашиваем ввод строки у пользователя**

**std::cout << "Введите строку: ";**

**std::getline(std::cin, inputString);**

**// Вызываем функцию для обработки строки**

**WordParse(inputString);**

**return 0;**

**}**

**Билет 5**

**#include <iostream>**

**#include <thread>**

**#include <vector>**

**#include <cmath>**

**void solveBiquadratic(int threadId, double a, double b, double c) {**

**// Вычисляем дискриминант**

**double D = b \* b - 4 \* a \* c;**

**// Массив для хранения корней**

**std::vector<double> roots;**

**if (D > 0) {**

**// Два различных корня**

**double y1 = (-b + std::sqrt(D)) / (2 \* a);**

**double y2 = (-b - std::sqrt(D)) / (2 \* a);**

**// Проверяем, являются ли корни y1 и y2 положительными**

**if (y1 >= 0) {**

**roots.push\_back(std::sqrt(y1)); // x1 = sqrt(y1)**

**roots.push\_back(-std::sqrt(y1)); // x2 = -sqrt(y1)**

**}**

**if (y2 >= 0) {**

**roots.push\_back(std::sqrt(y2)); // x3 = sqrt(y2)**

**roots.push\_back(-std::sqrt(y2)); // x4 = -sqrt(y2)**

**}**

**} else if (D == 0) {**

**// Один корень**

**double y = -b / (2 \* a);**

**if (y >= 0) {**

**roots.push\_back(std::sqrt(y)); // x1 = sqrt(y)**

**roots.push\_back(-std::sqrt(y)); // x2 = -sqrt(y)**

**}**

**}**

**// Выводим результаты**

**std::cout << "Thread " << threadId << ": ";**

**if (roots.empty()) {**

**std::cout << "No real roots." << std::endl;**

**} else {**

**std::cout << "Roots: ";**

**for (const auto& root : roots) {**

**std::cout << root << " ";**

**}**

**std::cout << std::endl;**

**}**

**}**

**int main() {**

**int n;**

**double a, b, c;**

**// Запрашиваем у пользователя количество потоков и коэффициенты**

**std::cout << "Enter the number of threads (n): ";**

**std::cin >> n;**

**std::cout << "Enter coefficients a, b, c (a != 0): ";**

**std::cin >> a >> b >> c;**

**// Проверка, что a не равно 0**

**if (a == 0) {**

**std::cerr << "Coefficient a must not be zero." << std::endl;**

**return 1;**

**}**

**std::vector<std::thread> threads;**

**// Запускаем n потоков**

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

**threads.emplace\_back(solveBiquadratic, i + 1, a, b, c);**

**}**

**// Ждем завершения всех потоков**

**for (auto& thread : threads) {**

**thread.join();**

**}**

**return 0;**

**}**

**Билет 6**

**Заголовочный файл password\_checker.h**

**#ifndef PASSWORD\_CHECKER\_H**

**#define PASSWORD\_CHECKER\_H**

**#include <string>**

**// Функция для проверки пароля**

**bool isPasswordValid(const std::string& password, int minLength, const std::string& requiredChars);**

**#endif // PASSWORD\_CHECKER\_H**

**РЕАЛИЗАИЯ БИБЛИОТЕКИ  
#include "password\_checker.h"**

**#include <cctype>**

**bool isPasswordValid(const std::string& password, int minLength, const std::string& requiredChars) {**

**// Проверка длины пароля**

**if (password.length() < minLength) {**

**return false;**

**}**

**// Проверка наличия хотя бы одного из требуемых символов**

**for (char c : requiredChars) {**

**if (password.find(c) != std::string::npos) {**

**return true; // Найден хотя бы один требуемый символ**

**}**

**}**

**return false; // Не найден ни один требуемый символ**

**}**

**КОМПИЛЯЧИЯ**

**g++ -shared -o password\_checker.dll password\_checker.cpp**

**Билет 7**

**#include <iostream>**

**#include <fstream>**

**#include <vector>**

**void writeBishopMoves(int x, int y) {**

**std::ofstream outFile("output.txt");**

**if (!outFile.is\_open()) {**

**std::cerr << "Не удалось открыть файл для записи." << std::endl;**

**return;**

**}**

**// Вектор для хранения возможных ходов**

**std::vector<std::pair<int, int>> moves;**

**// Проверяем все 4 диагонали**

**for (int i = 1; i < 8; ++i) {**

**// Вверх-вправо**

**if (x + i <= 8 && y + i <= 8) {**

**moves.emplace\_back(x + i, y + i);**

**}**

**// Вверх-влево**

**if (x - i >= 1 && y + i <= 8) {**

**moves.emplace\_back(x - i, y + i);**

**}**

**// Вниз-вправо**

**if (x + i <= 8 && y - i >= 1) {**

**moves.emplace\_back(x + i, y - i);**

**}**

**// Вниз-влево**

**if (x - i >= 1 && y - i >= 1) {**

**moves.emplace\_back(x - i, y - i);**

**}**

**}**

**// Записываем возможные ходы в файл**

**outFile << "Возможные ходы слона с позиции (" << x << ", " << y << "):" << std::endl;**

**for (const auto& move : moves) {**

**outFile << "(" << move.first << ", " << move.second << ")" << std::endl;**

**}**

**outFile.close();**

**}**

**int main() {**

**int x, y;**

**// Запрашиваем у пользователя координаты слона**

**std::cout << "Введите координаты слона (x y) от 1 до 8: ";**

**std::cin >> x >> y;**

**// Проверка корректности ввода**

**if (x < 1 || x > 8 || y < 1 || y > 8) {**

**std::cerr << "Координаты должны быть в диапазоне от 1 до 8." << std::endl;**

**return 1;**

**}**

**// Вызываем функцию для записи возможных ходов в файл**

**writeBishopMoves(x, y);**

**std::cout << "Возможные ходы записаны в файл output.txt." << std::endl;**

**return 0;**

**}**

**Билет 8**

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**// Функция для вычисления площади треугольника по координатам его вершин**

**double area(double x1, double y1, double x2, double y2, double x3, double y3) {**

**return abs((x1 \* (y2 - y3) + x2 \* (y3 - y1) + x3 \* (y1 - y2)) / 2.0);**

**}**

**// Функция для проверки, попадает ли точка в треугольник**

**bool isPointInTriangle(double x, double y) {**

**// Координаты вершин треугольника**

**double x1 = -1, y1 = -1;**

**double x2 = 1, y2 = -1;**

**double x3 = 1, y3 = 1;**

**// Площадь всего треугольника**

**double A = area(x1, y1, x2, y2, x3, y3);**

**// Площади треугольников, образованных точкой и вершинами**

**double A1 = area(x, y, x2, y2, x3, y3);**

**double A2 = area(x1, y1, x, y, x3, y3);**

**double A3 = area(x1, y1, x2, y2, x, y);**

**// Если сумма площадей A1, A2 и A3 равна площади A, то точка внутри или на границе треугольника**

**return A == A1 + A2 + A3;**

**}**

**int main() {**

**double x, y;**

**cout << "Введите координаты точки (x, y):" << endl;**

**cout << "x: ";**

**cin >> x;**

**cout << "y: ";**

**cin >> y;**

**if (isPointInTriangle(x, y)) {**

**cout << "Точка попадает в заштрихованную область треугольника." << endl;**

**} else {**

**cout << "Точка не попадает в заштрихованную область треугольника." << endl;**

**}**

**return 0;**

**}**

**Билет 9**

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**// Функция для проверки, находится ли точка в заштрихованной области**

**bool isPointInShadedRegion(double x, double y) {**

**// Проверяем, находится ли точка в квадрате с вершинами (-1, -1) и (0, 0)**

**if (x >= -1 && x <= 0 && y >= -1 && y <= 0) {**

**return true; // Точка находится в квадрате**

**}**

**// Проверяем, находится ли точка внутри четверти круга**

**// (центр в (0,0), радиус 1, с x >= 0 и y >= 0)**

**if (x >= 0 && y >= 0) {**

**if (x \* x + y \* y <= 1) {**

**return true; // Точка находится в четверти круга**

**}**

**}**

**// Если точка не находится ни в квадрате, ни в четверти круга, возвращаем false**

**return false;**

**}**

**int main() {**

**double x, y;**

**cout << "Введите координату x: ";**

**cin >> x;**

**cout << "Введите координату y: ";**

**cin >> y;**

**if (isPointInShadedRegion(x, y)) {**

**cout << "Точка находится в заштрихованной области." << endl;**

**} else {**

**cout << "Точка находится вне заштрихованной области." << endl;**

**}**

**return 0;**

**}**

**Билет 10**

**----------------------------**

**Билет 11**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**int main() {**

**int A, B;**

**// Ввод размеров матрицы**

**cout << "Введите количество строк (A): ";**

**cin >> A;**

**cout << "Введите количество столбцов (B): ";**

**cin >> B;**

**// Создание матрицы**

**vector<vector<int>> matrix(A, vector<int>(B));**

**// Ввод элементов матрицы**

**cout << "Введите элементы матрицы:" << endl;**

**for (int i = 0; i < A; ++i) {**

**for (int j = 0; j < B; ++j) {**

**cin >> matrix[i][j];**

**}**

**}**

**// Вывод строк с четными номерами, где первый элемент больше последнего**

**cout << "Строки с четными номерами, где первый элемент больше последнего:" << endl;**

**for (int i = 0; i < A; ++i) {**

**// Проверяем, является ли номер строки четным (i + 1 - номер строки)**

**if ((i + 1) % 2 == 0) {**

**if (matrix[i][0] > matrix[i][B - 1]) {**

**for (int j = 0; j < B; ++j) {**

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

**}**

**cout << endl; // Переход на новую строку после вывода строки матрицы**

**}**

**}**

**}**

**return 0;**

**}**

**Билет 12**

**#include <iostream>**

**#include <fstream>**

**#include <string>**

**#include <sstream>**

**using namespace std;**

**// Функция для удаления лишних пробелов из строки**

**string DeleteSpace(const string& S) {**

**stringstream ss(S);**

**string word;**

**string result;**

**// Считываем слова из строки и добавляем их в результат с одним пробелом между ними**

**while (ss >> word) {**

**if (!result.empty()) {**

**result += " "; // Добавляем пробел перед следующим словом, если результат не пуст**

**}**

**result += word; // Добавляем текущее слово**

**}**

**return result; // Возвращаем строку без лишних пробелов**

**}**

**int main() {**

**string file1, file2;**

**// Запрос имен файлов у пользователя**

**cout << "Введите имя первого файла: ";**

**cin >> file1;**

**cout << "Введите имя второго файла: ";**

**cin >> file2;**

**// Обработка первого файла**

**ifstream inputFile1(file1);**

**ofstream outputFile1("check\_" + file1);**

**if (!inputFile1.is\_open() || !outputFile1.is\_open()) {**

**cerr << "Ошибка при открытии файлов." << endl;**

**return 1;**

**}**

**string line;**

**while (getline(inputFile1, line)) {**

**outputFile1 << DeleteSpace(line) << endl; // Удаляем пробелы и записываем в новый файл**

**}**

**inputFile1.close();**

**outputFile1.close();**

**// Обработка второго файла**

**ifstream inputFile2(file2);**

**ofstream outputFile2("check\_" + file2);**

**if (!inputFile2.is\_open() || !outputFile2.is\_open()) {**

**cerr << "Ошибка при открытии файлов." << endl;**

**return 1;**

**}**

**while (getline(inputFile2, line)) {**

**outputFile2 << DeleteSpace(line) << endl; // Удаляем пробелы и записываем в новый файл**

**}**

**inputFile2.close();**

**outputFile2.close();**

**cout << "Обработка завершена. Исправленные файлы созданы с префиксом 'check\_'." << endl;**

**return 0;**

**}**

**Билет 13**

**#include <iostream>**

**#include <string>**

**#include <vector>**

**#include <cstdlib>**

**#include <ctime>**

**#include <algorithm>**

**using namespace std;**

**// Функция для генерации одного пароля**

**string generatePassword(int length, const string& requiredChars, int requiredCount) {**

**const string allChars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";**

**string password;**

**// Добавляем обязательные символы**

**for (int i = 0; i < requiredCount; ++i) {**

**password += requiredChars[rand() % requiredChars.length()];**

**}**

**// Заполняем оставшуюся длину пароля случайными символами**

**for (int i = requiredCount; i < length; ++i) {**

**password += allChars[rand() % allChars.length()];**

**}**

**// Перемешиваем символы в пароле**

**random\_shuffle(password.begin(), password.end());**

**return password;**

**}**

**// Функция для генерации N паролей**

**vector<string> generatePasswords(int N, int length, const string& requiredChars, int requiredCount) {**

**vector<string> passwords;**

**for (int i = 0; i < N; ++i) {**

**passwords.push\_back(generatePassword(length, requiredChars, requiredCount));**

**}**

**return passwords;**

**}**

**int main() {**

**srand(static\_cast<unsigned int>(time(0))); // Инициализация генератора случайных чисел**

**int N, L, K;**

**string S;**

**// Запрос параметров у пользователя**

**cout << "Введите количество паролей (N): ";**

**cin >> N;**

**cout << "Введите длину пароля (L): ";**

**cin >> L;**

**cout << "Введите строку с обязательными символами (S): ";**

**cin >> S;**

**cout << "Введите количество обязательных символов (K): ";**

**cin >> K;**

**// Проверка на корректность входных данных**

**if (K > L) {**

**cout << "Количество обязательных символов не может превышать длину пароля." << endl;**

**return 1;**

**}**

**if (K > S.length()) {**

**cout << "Количество обязательных символов не может превышать количество символов в строке S." << endl;**

**return 1;**

**}**

**// Генерация паролей**

**vector<string> passwords = generatePasswords(N, L, S, K);**

**// Вывод сгенерированных паролей**

**cout << "Сгенерированные пароли:" << endl;**

**for (const string& password : passwords) {**

**cout << password << endl;**

**}**

**return 0;**

**}**

**Билет 14**

**#include <iostream>**

**#include <map>**

**#include <string>**

**using namespace std;**

**int main() {**

**int N;**

**cout << "Введите количество пар синонимов (N): ";**

**cin >> N;**

**map<string, string> synonyms; // Словарь для хранения пар синонимов**

**cout << "Введите " << N << " пар синонимов (через пробел):" << endl;**

**for (int i = 0; i < N; ++i) {**

**string word1, word2;**

**cin >> word1 >> word2;**

**synonyms[word1] = word2; // Добавляем пару синонимов в словарь**

**synonyms[word2] = word1; // Добавляем обратную пару**

**}**

**string query;**

**cout << "Введите слово для поиска синонима: ";**

**cin >> query;**

**// Поиск синонима**

**auto it = synonyms.find(query);**

**if (it != synonyms.end()) {**

**cout << "Синоним для слова '" << query << "': " << it->second << endl;**

**} else {**

**cout << "Синоним не найден." << endl;**

**}**

**return 0;**

**}**

**Билет 15**

**#include <iostream>**

**#include <vector>**

**#include <thread>**

**using namespace std;**

**// Функция для проверки, является ли число простым**

**bool isPrime(int num) {**

**if (num <= 1) return false;**

**for (int i = 2; i \* i <= num; ++i) {**

**if (num % i == 0) return false;**

**}**

**return true;**

**}**

**// Функция для поиска простых чисел в заданном диапазоне**

**void findPrimes(int start, int end, vector<int>& primes) {**

**for (int i = start; i <= end; ++i) {**

**if (isPrime(i)) {**

**primes.push\_back(i);**

**}**

**}**

**}**

**int main() {**

**int A, B;**

**cout << "Введите диапазон (A и B, где B < 1000000):" << endl;**

**cout << "A: ";**

**cin >> A;**

**cout << "B: ";**

**cin >> B;**

**if (B >= 1000000) {**

**cout << "B должно быть меньше 1000000." << endl;**

**return 1;**

**}**

**vector<int> primes; // Вектор для хранения простых чисел**

**// Определяем количество потоков**

**const int numThreads = 4; // Количество потоков**

**vector<thread> threads;**

**int range = (B - A + 1) / numThreads; // Размер диапазона для каждого потока**

**// Вектор для хранения результатов от каждого потока**

**vector<vector<int>> threadPrimes(numThreads);**

**// Запускаем потоки**

**for (int i = 0; i < numThreads; ++i) {**

**int start = A + i \* range;**

**int end = (i == numThreads - 1) ? B : (start + range - 1);**

**threads.emplace\_back(findPrimes, start, end, ref(threadPrimes[i]));**

**}**

**// Ожидаем завершения всех потоков**

**for (auto& t : threads) {**

**t.join();**

**}**

**// Объединяем результаты из всех потоков в один вектор**

**for (const auto& vec : threadPrimes) {**

**primes.insert(primes.end(), vec.begin(), vec.end());**

**}**

**// Выводим найденные простые числа**

**cout << "Простые числа в диапазоне от " << A << " до " << B << ":" << endl;**

**for (int prime : primes) {**

**cout << prime << " ";**

**}**

**cout << endl;**

**return 0;**

**}**

**Билет 16**

**#include <iostream>**

**#include <curl/curl.h>**

**#include <fstream>**

**using namespace std;**

**// Функция для записи данных в файл**

**size\_t WriteCallback(void\* contents, size\_t size, size\_t nmemb, void\* userp) {**

**ofstream\* out = static\_cast<ofstream\*>(userp);**

**size\_t totalSize = size \* nmemb;**

**out->write(static\_cast<char\*>(contents), totalSize);**

**return totalSize;**

**}**

**// Функция для проверки доступности URL и сохранения изображения**

**void downloadImage(const string& url, const string& filename) {**

**CURL\* curl;**

**CURLcode res;**

**ofstream outFile(filename, ios::binary);**

**if (!outFile) {**

**cerr << "Не удалось открыть файл для записи: " << filename << endl;**

**return;**

**}**

**curl = curl\_easy\_init();**

**if (curl) {**

**curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());**

**curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);**

**curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &outFile);**

**curl\_easy\_setopt(curl, CURLOPT\_FOLLOWLOCATION, 1L); // Следовать за перенаправлениями**

**// Выполняем запрос**

**res = curl\_easy\_perform(curl);**

**// Проверяем на ошибки**

**if (res != CURLE\_OK) {**

**cerr << "Ошибка при загрузке изображения: " << curl\_easy\_strerror(res) << endl;**

**} else {**

**cout << "Изображение успешно загружено и сохранено в " << filename << endl;**

**}**

**// Освобождаем ресурсы**

**curl\_easy\_cleanup(curl);**

**}**

**outFile.close();**

**}**

**int main() {**

**string url;**

**string filename;**

**cout << "Введите URL изображения: ";**

**cin >> url;**

**cout << "Введите имя файла для сохранения (например, image.jpg): ";**

**cin >> filename;**

**downloadImage(url, filename);**

**return 0;**

**}**

**Билет 17**

**#include <iostream>**

**#include <windows.h>**

**#include <string>**

**using namespace std;**

**void sendTextToClipboard(const string& text) {**

**// Открываем буфер обмена**

**if (OpenClipboard(NULL)) {**

**EmptyClipboard(); // Очищаем буфер обмена**

**// Создаем глобальный текстовый буфер**

**HGLOBAL hGlob = GlobalAlloc(GMEM\_MOVEABLE, text.size() + 1);**

**memcpy(GlobalLock(hGlob), text.c\_str(), text.size() + 1);**

**GlobalUnlock(hGlob);**

**// Устанавливаем текст в буфер обмена**

**SetClipboardData(CF\_TEXT, hGlob);**

**CloseClipboard();**

**} else {**

**cerr << "Не удалось открыть буфер обмена." << endl;**

**}**

**}**

**int main() {**

**string text;**

**cout << "Введите текст для передачи: ";**

**getline(cin, text);**

**sendTextToClipboard(text);**

**cout << "Текст передан в буфер обмена." << endl;**

**return 0;**

**}** **#include <iostream>**

**#include <windows.h>**

**#include <string>**

**using namespace std;**

**string getTextFromClipboard() {**

**string result;**

**// Открываем буфер обмена**

**if (OpenClipboard(NULL)) {**

**// Получаем дескриптор данных**

**HGLOBAL hGlob = GetClipboardData(CF\_TEXT);**

**if (hGlob) {**

**char\* data = static\_cast<char\*>(GlobalLock(hGlob));**

**if (data) {**

**result = data; // Копируем данные в строку**

**GlobalUnlock(hGlob);**

**}**

**}**

**CloseClipboard();**

**} else {**

**cerr << "Не удалось открыть буфер обмена." << endl;**

**}**

**return result;**

**}**

**int main() {**

**string receivedText = getTextFromClipboard();**

**if (!receivedText.empty()) {**

**cout << "Полученный текст: " << receivedText << endl;**

**} else {**

**cout << "Буфер обмена пуст или не содержит текст." << endl;**

**}**

**return 0;**

**}**

**Билет 18**

**#include <windows.h>**

**#include <iostream>**

**#include <fstream>**

**#include <vector>**

**#include <cstdlib>**

**#include <ctime>**

**#include <string>**

**using namespace std;**

**#define SERVICE\_NAME L"MyRandomNumberService"**

**// Прототипы функций**

**void WriteRandomNumbers(int N);**

**void DeleteNumberFromFile();**

**void ServiceMain(DWORD argc, LPTSTR\* argv);**

**void ServiceCtrlHandler(DWORD ctrlCode);**

**// Глобальные переменные**

**SERVICE\_STATUS\_HANDLE g\_ServiceStatusHandle = NULL;**

**SERVICE\_STATUS g\_ServiceStatus = { 0 };**

**void WriteRandomNumbers(int N) {**

**ofstream outputFile("output.txt", ios::app);**

**if (outputFile.is\_open()) {**

**for (int i = 0; i < N; ++i) {**

**int randomNumber = rand() % 26; // Генерация случайного числа от 0 до 25**

**outputFile << randomNumber << endl;**

**}**

**outputFile.close();**

**} else {**

**cerr << "Не удалось открыть файл output.txt для записи." << endl;**

**}**

**}**

**void DeleteNumberFromFile() {**

**ifstream inputFile("input.txt");**

**ofstream tempFile("temp.txt");**

**int number;**

**if (inputFile.is\_open() && tempFile.is\_open()) {**

**if (inputFile >> number) {**

**// Читаем число и не записываем его в временный файл**

**}**

**string line;**

**while (getline(inputFile, line)) {**

**tempFile << line << endl; // Копируем остальные строки**

**}**

**inputFile.close();**

**tempFile.close();**

**remove("input.txt");**

**rename("temp.txt", "input.txt");**

**} else {**

**cerr << "Не удалось открыть файл input.txt." << endl;**

**}**

**}**

**void ServiceMain(DWORD argc, LPTSTR\* argv) {**

**g\_ServiceStatus.dwServiceType = SERVICE\_WIN32;**

**g\_ServiceStatus.dwCurrentState = SERVICE\_START\_PENDING;**

**g\_ServiceStatus.dwControlsAccepted = SERVICE\_ACCEPT\_STOP;**

**g\_ServiceStatus.dwWin32ExitCode = 0;**

**g\_ServiceStatus.dwCheckPoint = 0;**

**g\_ServiceStatus.dwWaitHint = 0;**

**g\_ServiceStatusHandle = RegisterServiceCtrlHandler(SERVICE\_NAME, ServiceCtrlHandler);**

**if (g\_ServiceStatusHandle == 0) {**

**return;**

**}**

**g\_ServiceStatus.dwCurrentState = SERVICE\_RUNNING;**

**SetServiceStatus(g\_ServiceStatusHandle, &g\_ServiceStatus);**

**srand(static\_cast<unsigned int>(time(NULL))); // Инициализация генератора случайных чисел**

**while (g\_ServiceStatus.dwCurrentState == SERVICE\_RUNNING) {**

**ifstream inputFile("input.txt");**

**int N;**

**if (inputFile >> N) {**

**WriteRandomNumbers(N);**

**DeleteNumberFromFile();**

**}**

**inputFile.close();**

**Sleep(60000); // Ожидание 60 секунд**

**}**

**g\_ServiceStatus.dwCurrentState = SERVICE\_STOPPED;**

**SetServiceStatus(g\_ServiceStatusHandle, &g\_ServiceStatus);**

**}**

**void ServiceCtrlHandler(DWORD ctrlCode) {**

**switch (ctrlCode) {**

**case SERVICE\_CONTROL\_STOP:**

**g\_ServiceStatus.dwCurrentState = SERVICE\_STOP\_PENDING;**

**SetServiceStatus(g\_ServiceStatusHandle, &g\_ServiceStatus);**

**g\_ServiceStatus.dwCurrentState = SERVICE\_STOPPED;**

**SetServiceStatus(g\_ServiceStatusHandle, &g\_ServiceStatus);**

**break;**

**default:**

**break;**

**}**

**}**

**int main() {**

**SERVICE\_TABLE\_ENTRY serviceTable[] = {**

**{ SERVICE\_NAME, (LPSERVICE\_MAIN\_FUNCTION)ServiceMain },**

**{ NULL, NULL }**

**};**

**if (StartServiceCtrlDispatcher(serviceTable) == 0) {**

**cerr << "Не удалось запустить диспетчер служб." << endl;**

**return 1;**

**}**

**return 0;**

**}**

**Билет 19**

**using System;**

**using System.IO;**

**using System.IO.Compression;**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**Console.WriteLine("Выберите действие:");**

**Console.WriteLine("1. Архивировать каталог");**

**Console.WriteLine("2. Разархивировать ZIP-архив");**

**Console.Write("Введите номер действия: ");**

**int choice = int.Parse(Console.ReadLine());**

**if (choice == 1)**

**{**

**Console.Write("Введите путь к каталогу для архивации: ");**

**string directoryPath = Console.ReadLine();**

**Console.Write("Введите имя ZIP-архива (например, archive.zip): ");**

**string zipFilePath = Console.ReadLine();**

**try**

**{**

**ZipFile.CreateFromDirectory(directoryPath, zipFilePath);**

**Console.WriteLine($"Каталог успешно архивирован в {zipFilePath}");**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Ошибка при архивировании: {ex.Message}");**

**}**

**}**

**else if (choice == 2)**

**{**

**Console.Write("Введите путь к ZIP-архиву: ");**

**string zipFilePath = Console.ReadLine();**

**Console.Write("Введите путь для извлечения: ");**

**string extractPath = Console.ReadLine();**

**try**

**{**

**ZipFile.ExtractToDirectory(zipFilePath, extractPath);**

**Console.WriteLine($"ZIP-архив успешно разархивирован в {extractPath}");**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Ошибка при разархивировании: {ex.Message}");**

**}**

**}**

**else**

**{**

**Console.WriteLine("Неверный выбор.");**

**}**

**}**

**}**

**Билет 20**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**// Функция для проверки, является ли число степенью пятерки**

**bool isPowerOfFive(int num) {**

**if (num < 1) return false; // Степени пятерки начинаются с 1 (5^0)**

**while (num % 5 == 0) {**

**num /= 5; // Делим число на 5, пока это возможно**

**}**

**return num == 1; // Если в конце мы получили 1, значит, это степень пятерки**

**}**

**int main() {**

**int n;**

**cout << "Введите количество чисел в последовательности (должно быть 7): ";**

**cin >> n;**

**if (n != 7) {**

**cout << "Ошибка: количество чисел должно быть равно 7." << endl;**

**return 1;**

**}**

**vector<int> numbers(7);**

**cout << "Введите 7 натуральных чисел:" << endl;**

**for (int i = 0; i < 7; ++i) {**

**cin >> numbers[i];**

**}**

**int count = 0;**

**for (int num : numbers) {**

**if (isPowerOfFive(num)) {**

**count++;**

**}**

**}**

**cout << "Количество чисел, являющихся степенями пятерки: " << count << endl;**

**return 0;**

**}**

**Билет 21**

**using System;**

**class BankAccount**

**{**

**public double CalculateDeposit(double initialAmount, double monthlyInterestRate, int months)**

**{**

**double totalAmount = initialAmount;**

**for (int i = 0; i < months; i++)**

**{**

**totalAmount += totalAmount \* monthlyInterestRate; // Начисление процентов**

**}**

**return totalAmount;**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**BankAccount account = new BankAccount();**

**Console.Write("Введите начальную сумму вклада: ");**

**double initialAmount = Convert.ToDouble(Console.ReadLine());**

**Console.Write("Введите месячную процентную ставку (в десятичном формате, например, 0.01 для 1%): ");**

**double monthlyInterestRate = Convert.ToDouble(Console.ReadLine());**

**Console.Write("Введите количество месяцев: ");**

**int months = Convert.ToInt32(Console.ReadLine());**

**double finalAmount = account.CalculateDeposit(initialAmount, monthlyInterestRate, months);**

**Console.WriteLine($"Итоговая сумма вклада через {months} месяцев: {finalAmount:F2}");**

**}**

**}**

**Билет 22**

**using System;**

**class Employee**

**{**

**private double monthlySalary; // Оклад сотрудника**

**public Employee(double salary)**

**{**

**monthlySalary = salary;**

**}**

**public double CalculateSalary(int workedDays, int months)**

**{**

**// Рассчитываем заработную плату**

**double dailySalary = monthlySalary / 23; // Оклад делим на 23 рабочих дня**

**double totalSalary = dailySalary \* workedDays \* months; // Общая заработная плата**

**return totalSalary;**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**Console.Write("Введите оклад сотрудника: ");**

**double salary = Convert.ToDouble(Console.ReadLine());**

**Employee employee = new Employee(salary);**

**Console.Write("Введите количество отработанных дней: ");**

**int workedDays = Convert.ToInt32(Console.ReadLine());**

**Console.Write("Введите количество месяцев: ");**

**int months = Convert.ToInt32(Console.ReadLine());**

**double totalSalary = employee.CalculateSalary(workedDays, months);**

**Console.WriteLine($"Общая заработная плата за {months} месяцев: {totalSalary:F2}");**

**}**

**}**

**Билет 23**

**using System;**

**class Program**

**{**

**// Метод для генерации коэффициентов**

**static (double a, double b, double c) GenerateCoefficients()**

**{**

**Random random = new Random();**

**double a = random.Next(1, 10); // Генерируем a от 1 до 9**

**double b = random.Next(-10, 10); // Генерируем b от -10 до 9**

**double c = random.Next(-10, 10); // Генерируем c от -10 до 9**

**return (a, b, c);**

**}**

**// Метод для решения квадратного уравнения**

**static void SolveQuadraticEquation(double a, double b, double c)**

**{**

**double discriminant = b \* b - 4 \* a \* c;**

**Console.WriteLine($"Квадратное уравнение: {a}x^2 + {b}x + {c} = 0");**

**if (discriminant > 0)**

**{**

**double root1 = (-b + Math.Sqrt(discriminant)) / (2 \* a);**

**double root2 = (-b - Math.Sqrt(discriminant)) / (2 \* a);**

**Console.WriteLine($"Корни: x1 = {root1}, x2 = {root2}");**

**}**

**else if (discriminant == 0)**

**{**

**double root = -b / (2 \* a);**

**Console.WriteLine($"Двойной корень: x = {root}");**

**}**

**else**

**{**

**Console.WriteLine("Нет действительных корней.");**

**}**

**}**

**static void Main(string[] args)**

**{**

**// Генерируем коэффициенты**

**var (a, b, c) = GenerateCoefficients();**

**// Решаем квадратное уравнение**

**SolveQuadraticEquation(a, b, c);**

**}**

**}**