**Вариант 1 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

Программа предназначена для вычисления среднего арифметического значений, которые загружаются из текстового файла. Результат вычислений записывается в другой файл. Однако программа содержит первичную и вторичную ошибку. Задача студента — найти ошибки, объяснить их влияние и предложить исправления.

Объяснение ошибок:

**Первичная ошибка**: Неправильное условие в цикле.

* + В строке: for (size\_t i = 0; i <= numbers.size(); i++)

Цикл выполняется до i <= numbers.size(), что вызывает обращение к индексу за пределами массива на последней итерации.

**Последствие:** Ошибка времени выполнения (segmentation fault).

**Вторичная ошибка: Неправильное вычисление среднего значения.**

В строке:

return sum / numbers.size();

Выполняется целочисленное деление, что приводит к потере дробной части результата.

**Последствие:** Программа возвращает неверный результат.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <fstream>  #include <sstream>  #include <vector>  class FileProcessor {  private:  std::string inputFile;  std::string outputFile;  public:  void setFiles(const std::string& inFile, const std::string& outFile) {  inputFile = inFile;  outputFile = outFile;  }  double calculateAverageFromFile() {  std::ifstream inFile(inputFile);  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл для чтения.");  }  std::vector<int> numbers;  std::string line;  while (std::getline(inFile, line)) {  std::stringstream ss(line);  int num;  while (ss >> num) {  numbers.push\_back(num);  }  }  int sum = 0;  for (size\_t i = 0; i <= numbers.size(); i++) { // Ошибка: неправильное условие цикла  sum += numbers[i];  }  return sum / numbers.size(); // Ошибка: целочисленное деление  }  void writeResultToFile(double result) {  std::ofstream outFile(outputFile);  if (!outFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл для записи.");  }  outFile << "Среднее арифметическое: " << result << std::endl;  }  };  int main() {  try {  FileProcessor processor;  processor.setFiles("input.txt", "output.txt");  double average = processor.calculateAverageFromFile();  processor.writeResultToFile(average);  std::cout << "Результат сохранён в файл." << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } | #include <iostream>  #include <fstream>  #include <sstream>  #include <vector>  class FileProcessor {  private:  std::string inputFile;  std::string outputFile;  public:  void setFiles(const std::string& inFile, const std::string& outFile) {  inputFile = inFile;  outputFile = outFile;  }  double calculateAverageFromFile() {  std::ifstream inFile(inputFile);  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл для чтения.");  }  std::vector<int> numbers;  std::string line;  while (std::getline(inFile, line)) {  std::stringstream ss(line);  int num;  while (ss >> num) {  numbers.push\_back(num);  }  }  if (numbers.empty()) {  throw std::runtime\_error("Файл не содержит чисел для обработки.");  }  int sum = 0;  for (size\_t i = 0; i < numbers.size(); i++) { // Исправлено условие  sum += numbers[i];  }  return static\_cast<double>(sum) / numbers.size(); // Исправлено для корректного деления  }  void writeResultToFile(double result) {  std::ofstream outFile(outputFile);  if (!outFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл для записи.");  }  outFile << "Среднее арифметическое: " << result << std::endl;  }  };  int main() {  try {  FileProcessor processor;  processor.setFiles("input.txt", "output.txt");  double average = processor.calculateAverageFromFile();  processor.writeResultToFile(average);  std::cout << "Результат сохранён в файл." << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 2 МДК 04.02**

**Задание2**

**Описание:** Программа предназначена для вычисления количества дней между двумя датами, которые вводятся из текстового файла. Программа содержит ошибки:

**Первичная ошибка:** Неверная проверка на допустимость формата даты.

**Вторичная ошибка:** Ошибка в расчёте разницы дней из-за неправильного учёта високосных лет.

**Объяснение ошибок:**

1. **Первичная ошибка: Неполная проверка формата даты.**

Код проверяет только наличие разделителей '/' между компонентами даты.

**Последствие:** Неверный формат, например, "32/13/2022" или "abc/12/2022", не будет корректно обрабатываться.

1. **Вторичная ошибка: Неправильный расчёт разницы дней.**

Логика вычисления количества дней не учитывает високосные годы.

**Последствие:** Результат будет неверным для дат, включающих високосные годы.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <fstream>  #include <sstream>  #include <stdexcept>  #include <string>  #include <cmath>  class Date {  public:  int day, month, year;  Date(int d, int m, int y) : day(d), month(m), year(y) {}  static Date parse(const std::string& dateStr) {  std::stringstream ss(dateStr);  int d, m, y;  char delim1, delim2;  ss >> d >> delim1 >> m >> delim2 >> y;  if (delim1 != '/' || delim2 != '/' || ss.fail()) { // Ошибка: проверка только на формат  throw std::runtime\_error("Некорректный формат даты.");  }  return Date(d, m, y);  }  };  class DateProcessor {  public:  static int daysBetween(const Date& start, const Date& end) {  int days1 = calculateDays(start.year, start.month, start.day);  int days2 = calculateDays(end.year, end.month, end.day);  return abs(days2 - days1); // Ошибка: високосные года не учитываются  }  private:  static int calculateDays(int year, int month, int day) {  static const int daysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };  int totalDays = year \* 365 + day;  for (int i = 0; i < month - 1; ++i) {  totalDays += daysInMonth[i];  }  return totalDays;  }  };  int main() {  try {  std::ifstream inFile("dates.txt");  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл.");  }  std::string dateStr1, dateStr2;  std::getline(inFile, dateStr1);  std::getline(inFile, dateStr2);  Date date1 = Date::parse(dateStr1);  Date date2 = Date::parse(dateStr2);  int days = DateProcessor::daysBetween(date1, date2);  std::cout << "Количество дней между датами: " << days << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0; } | #include <iostream>  #include <fstream>  #include <sstream>  #include <stdexcept>  #include <string>  #include <cmath>  class Date {  public:  int day, month, year;  Date(int d, int m, int y) : day(d), month(m), year(y) {  if (!isValid()) {  throw std::runtime\_error("Некорректная дата: " + std::to\_string(d) + "/" + std::to\_string(m) + "/" + std::to\_string(y));  }  }  static Date parse(const std::string& dateStr) {  std::stringstream ss(dateStr);  int d, m, y;  char delim1, delim2;  ss >> d >> delim1 >> m >> delim2 >> y;  if (delim1 != '/' || delim2 != '/' || ss.fail()) {  throw std::runtime\_error("Некорректный формат даты.");  }  return Date(d, m, y);  }  private:  bool isValid() {  if (month < 1 || month > 12 || day < 1) return false;  static const int daysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };  int maxDays = daysInMonth[month - 1];  if (month == 2 && isLeapYear(year)) {  maxDays = 29;  }  return day <= maxDays;  }  static bool isLeapYear(int y) {  return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);  }  };  class DateProcessor {  public:  static int daysBetween(const Date& start, const Date& end) {  int days1 = calculateDays(start.year, start.month, start.day);  int days2 = calculateDays(end.year, end.month, end.day);  return abs(days2 - days1);  }  private:  static int calculateDays(int year, int month, int day) {  static const int daysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };  int totalDays = year \* 365 + day;  for (int i = 0; i < month - 1; ++i) {  totalDays += daysInMonth[i];  }  totalDays += countLeapYears(year, month);  return totalDays;  }  static int countLeapYears(int year, int month) {  if (month <= 2) year--;  return year / 4 - year / 100 + year / 400;  }  };  int main() {  try {  std::ifstream inFile("dates.txt");  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл.");  }  std::string dateStr1, dateStr2;  std::getline(inFile, dateStr1);  std::getline(inFile, dateStr2);  Date date1 = Date::parse(dateStr1);  Date date2 = Date::parse(dateStr2);  int days = DateProcessor::daysBetween(date1, date2);  std::cout << "Количество дней между датами: " << days << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;} |

**Вариант 3 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Первичная ошибка:** В конструкторе класса Rectangle ширина (width) и высота (height) допускают отрицательные значения.

**Последствие:** Площадь и периметр вычисляются некорректно, что может вызвать ошибки в логике программы.

**Вторичная ошибка:** Формула расчёта площади содержит лишний множитель 0.5:

return width \* height \* 0.5;

**Последствие:** Площадь всегда рассчитывается неверно.

**Исправления:**

**Проверка входных данных:**

Добавлена проверка в конструктор:

if (w <= 0 || h <= 0) {

throw std::invalid\_argument("Ширина и высота должны быть положительными числами.");

}

Это предотвращает создание объекта с некорректными значениями.

**Корректная формула площади:** Формула изменена на: return width \* height;

**Улучшение обработки ошибок:**

Добавлено исключение с понятным сообщением: throw std::invalid\_argument("Ширина и высота должны быть положительными числами.");

**Пояснение исправлений:**

**Первичная ошибка:** Отрицательные значения ширины и высоты теперь обрабатываются через исключения.

**Вторичная ошибка:** Формула площади исправлена, чтобы результат был корректным.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  class Rectangle {  private:  double width;  double height;  public:  Rectangle(double w, double h) : width(w), height(h) {}  double calculateArea() const {  return width \* height \* 0.5; // Ошибка: неправильная формула  }  double calculatePerimeter() const {  return 2 \* (width + height);  }  void displayProperties() const {  std::cout << "Ширина: " << width << ", Высота: " << height << std::endl;  std::cout << "Площадь: " << calculateArea() << std::endl;  std::cout << "Периметр: " << calculatePerimeter() << std::endl;  }  };  int main() {  try {  Rectangle rect(-5, 10); // Ошибка: отрицательное значение ширины  rect.displayProperties();  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } | #include <iostream>  #include <stdexcept>  class Rectangle {  private:  double width;  double height;  public:  Rectangle(double w, double h) {  if (w <= 0 || h <= 0) {  throw std::invalid\_argument("Ширина и высота должны быть положительными числами.");  }  width = w;  height = h;  }  double calculateArea() const {  return width \* height; // Исправлено: убран лишний множитель  }  double calculatePerimeter() const {  return 2 \* (width + height);  }  void displayProperties() const {  std::cout << "Ширина: " << width << ", Высота: " << height << std::endl;  std::cout << "Площадь: " << calculateArea() << std::endl;  std::cout << "Периметр: " << calculatePerimeter() << std::endl;  }  };  int main() {  try {  Rectangle rect(5, 10); // Исправлено: положительные значения  rect.displayProperties();  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 4 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа записывает массив строк в файл, а затем считывает содержимое файла и выводит его на экран. Однако в коде присутствуют ошибки:

**Первичная ошибка:** Отсутствует проверка успешного открытия файлов.

**Вторичная ошибка:** Программа некорректно считывает строки из файла из-за неверного использования потоков.

**Объяснение ошибок:**

**Первичная ошибка:** Файл для записи открывается без проверки успешности:

std::ofstream outFile(fileName);

**Последствие:** Если файл не откроется (например, из-за отсутствия прав доступа), программа продолжит выполнение с неинициализированным потоком, что приведёт к потере данных.

**Вторичная ошибка:** Использование оператора >> для чтения строк:

while (inFile >> line)

**Последствие:** Чтение прекращается на первом пробеле, и строки выводятся неполностью.

Исправленный код:

**Исправления: Проверка открытия файлов:** Добавлена проверка потока:

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

throw std::runtime\_error("Не удалось открыть файл для записи.");

}

Аналогичная проверка добавлена для чтения файла:

if (!inFile.is\_open()) {

throw std::runtime\_error("Не удалось открыть файл для чтения.");

}

**Корректное чтение строк:** Вместо оператора >> используется метод std::getline:

while (std::getline(inFile, line)) {

std::cout << line << std::endl;

}

**Пояснение исправлений:**

**Первичная ошибка:** Добавлена проверка открытия файлов с соответствующим исключением, что предотвращает некорректное выполнение программы.

**Вторичная ошибка:** Метод std::getline позволяет корректно считывать строки, включая пробелы, обеспечивая ожидаемый вывод.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <fstream>  #include <vector>  #include <string>  class FileHandler {  private:  std::string fileName;  public:  FileHandler(const std::string& name) : fileName(name) {}  void writeToFile(const std::vector<std::string>& data) {  std::ofstream outFile(fileName);  outFile << "Данные записаны:\n"; // Ошибка: без проверки открытия файла  for (const auto& line : data) {  outFile << line << std::endl;  }  }  void readFromFile() const {  std::ifstream inFile(fileName);  std::string line;  while (inFile >> line) { // Ошибка: считывание только до первого пробела  std::cout << line << std::endl;  }  }  };  int main() {  try {  FileHandler fileHandler("data.txt");  std::vector<std::string> data = {"Hello, World!", "C++ is powerful.", "File I/O is essential."};  fileHandler.writeToFile(data);  std::cout << "Содержимое файла:" << std::endl;  fileHandler.readFromFile();  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } | #include <iostream>  #include <fstream>  #include <vector>  #include <string>  #include <stdexcept>  class FileHandler {  private:  std::string fileName;  public:  FileHandler(const std::string& name) : fileName(name) {}  void writeToFile(const std::vector<std::string>& data) {  std::ofstream outFile(fileName);  if (!outFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл для записи.");  }  outFile << "Данные записаны:\n";  for (const auto& line : data) {  outFile << line << std::endl;  }  }  void readFromFile() const {  std::ifstream inFile(fileName);  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл для чтения.");  }  std::string line;  while (std::getline(inFile, line)) { // Исправлено: используется getline для корректного чтения строк  std::cout << line << std::endl;  }  }  };  int main() {  try {  FileHandler fileHandler("data.txt");  std::vector<std::string> data = {"Hello, World!", "C++ is powerful.", "File I/O is essential."};  fileHandler.writeToFile(data);  std::cout << "Содержимое файла:" << std::endl;  fileHandler.readFromFile();  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 5 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа предназначена для выполнения базовых арифметических операций: сложения, вычитания, умножения и деления. В коде присутствуют ошибки:

**Первичная ошибка:** Деление на ноль не обрабатывается.

**Вторичная ошибка:** Результат деления всегда округляется до целого числа, даже если ожидается дробное значение.

**Объяснение ошибок:**

**Первичная ошибка:** Деление на ноль не обрабатывается в методе divide.

**Последствие:** При попытке деления на ноль программа завершится аварийно (ошибка времени выполнения).

**Вторичная ошибка:** Метод divide выполняет целочисленное деление:

return a / b;

**Последствие:** Если аргументы метода являются целыми числами, результат всегда округляется до целого, даже если требуется дробное значение.

Исправленный код:

**Исправления:**

**Обработка деления на ноль:** Добавлена проверка аргумента b в методе divide:

if (b == 0) {

throw std::invalid\_argument("Деление на ноль недопустимо.");

}

Это предотвращает аварийное завершение программы.

**Корректное вычисление дробного результата:** Результат деления теперь приводится к типу double:

return static\_cast<double>(a) / b;

Это позволяет вернуть точное значение, включая дробную часть.

**Исправление входных данных:** Значение y изменено на ненулевое:

int x = 10, y = 2;

**Пояснение исправлений:**

**Первичная ошибка:** Проверка аргумента b в методе divide исключает возможность деления на ноль.

**Вторичная ошибка:** Приведение типов исправляет проблему целочисленного деления.

**Дополнительно:** Код стал более надёжным благодаря обработке исключений с осмысленным сообщением об ошибке.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <stdexcept>  class Calculator {  public:  int add(int a, int b) {  return a + b;  }  int subtract(int a, int b) {  return a - b;  }  int multiply(int a, int b) {  return a \* b;  }  int divide(int a, int b) {  return a / b; // Ошибка: деление на ноль не обрабатывается  }  };  int main() {  try {  Calculator calc;  int x = 10, y = 0; // Ошибка: y равен 0  std::cout << "Сложение: " << calc.add(x, y) << std::endl;  std::cout << "Вычитание: " << calc.subtract(x, y) << std::endl;  std::cout << "Умножение: " << calc.multiply(x, y) << std::endl;  std::cout << "Деление: " << calc.divide(x, y) << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } | #include <iostream>  #include <stdexcept>  class Calculator {  public:  int add(int a, int b) {  return a + b;  }  int subtract(int a, int b) {  return a - b;  }  int multiply(int a, int b) {  return a \* b;  }  double divide(int a, int b) {  if (b == 0) {  throw std::invalid\_argument("Деление на ноль недопустимо.");  }  return static\_cast<double>(a) / b; // Исправлено: используется дробное деление  }  };  int main() {  try {  Calculator calc;  int x = 10, y = 2; // Исправлено: y больше нуля  std::cout << "Сложение: " << calc.add(x, y) << std::endl;  std::cout << "Вычитание: " << calc.subtract(x, y) << std::endl;  std::cout << "Умножение: " << calc.multiply(x, y) << std::endl;  std::cout << "Деление: " << calc.divide(x, y) << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 6 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа предназначена для сортировки массива чисел методом пузырька. В коде содержатся следующие ошибки:

**Первичная ошибка:** Неправильная логика сравнения элементов массива.

**Вторичная ошибка:** Игнорируется последний элемент массива в процессе сортировки.

**Объяснение ошибок:**

**Первичная ошибка:** Внутренний цикл выполняется до arr.size() вместо arr.size() - 1 - i.

**Последствие:** Сравнение выходит за границы массива на последней итерации, что вызывает ошибку времени выполнения.

**Вторичная ошибка:** Последний элемент массива фактически не сортируется, так как внутренний цикл не корректно уменьшает количество итераций.

**Последствие:** Сортировка завершится, оставив последний элемент на месте, что приводит к некорректному результату.

**Исправления:**

**Обновление внутреннего цикла:** Исправлено условие цикла, чтобы каждый раз уменьшать количество элементов для сравнения:

for (size\_t j = 0; j < arr.size() - 1 - i; j++) {

Это гарантирует, что последний элемент учитывается и сортировка выполняется корректно.

**Корректная обработка элементов массива:** Сравнение и обмен значений теперь выполняются без выхода за пределы массива:

if (arr[j] > arr[j + 1]) {

std::swap(arr[j], arr[j + 1]);

}

**Повышена надёжность:** Исключения обрабатываются через try-catch, предотвращая аварийное завершение программы.

**Пояснение исправлений:**

**Первичная ошибка:** Условие внутреннего цикла теперь корректно предотвращает выход за границы массива.

**Вторичная ошибка:** Обновление цикла внутренней сортировки позволяет последнему элементу участвовать в процессе сортировки, завершая её корректно.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <vector>  class BubbleSort {  public:  void sort(std::vector<int>& arr) {  for (size\_t i = 0; i < arr.size(); i++) {  for (size\_t j = 0; j < arr.size(); j++) { // Ошибка: должен быть `arr.size() - 1 - i`  if (arr[j] > arr[j + 1]) { // Ошибка: возможен выход за пределы массива  std::swap(arr[j], arr[j + 1]);  }  }  }  }  void display(const std::vector<int>& arr) const {  for (int num : arr) {  std::cout << num << " ";  }  std::cout << std::endl;  }  };  int main() {  BubbleSort sorter;  std::vector<int> numbers = {5, 3, 8, 4, 2};  sorter.sort(numbers); // Ошибка: неправильный алгоритм  std::cout << "Отсортированный массив: ";  sorter.display(numbers);  return 0;  } | #include <iostream>  #include <vector>  #include <stdexcept>  class BubbleSort {  public:  void sort(std::vector<int>& arr) {  for (size\_t i = 0; i < arr.size(); i++) {  for (size\_t j = 0; j < arr.size() - 1 - i; j++) { // Исправлено: цикл сокращается с каждой итерацией  if (arr[j] > arr[j + 1]) {  std::swap(arr[j], arr[j + 1]);  }  }  }  }  void display(const std::vector<int>& arr) const {  for (int num : arr) {  std::cout << num << " ";  }  std::cout << std::endl;  }  };  int main() {  try {  BubbleSort sorter;  std::vector<int> numbers = {5, 3, 8, 4, 2};  sorter.sort(numbers);  std::cout << "Отсортированный массив: ";  sorter.display(numbers);  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 7 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа считывает строки из файла и генерирует отчёт, подсчитывая количество строк. Ошибки в программе:

**Первичная ошибка:** Пустые строки игнорируются при подсчёте.

**Вторичная ошибка:** Общее количество строк считается неверно из-за ошибки в логике подсчёта.

**Объяснение ошибок:**

**Первичная ошибка:** Пустые строки пропускаются из-за проверки:

if (!line.empty())

**Последствие:** Итоговый отчёт не учитывает пустые строки, что приводит к искажению результата.

**Вторичная ошибка:** Итоговый подсчёт строк может быть неверным, так как код не обрабатывает ситуации, когда файл пуст.

Исправленный код:

**Исправления:**

**Корректный подсчёт строк:** Удалена проверка на пустоту строки:

if (!line.empty())

Теперь каждая строка, включая пустую, учитывается.

**Проверка на пустой файл:** Добавлено исключение, выбрасываемое при отсутствии строк:

if (count == 0) {

throw std::runtime\_error("Файл пуст. Нет строк для обработки.");

}

**Пояснение исправлений:**

**Первичная ошибка:** Удаление проверки на пустые строки гарантирует, что все строки из файла, включая пустые, будут учитываться в итоговом отчёте.

**Вторичная ошибка:** Исключение для пустого файла предотвращает создание некорректного отчёта.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <fstream>  #include <string>  class ReportGenerator {  public:  int generateReport(const std::string& fileName) {  std::ifstream inFile(fileName);  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл.");  }  std::string line;  int count = 0;  while (std::getline(inFile, line)) {  if (!line.empty()) { // Ошибка: пустые строки пропускаются  count++;  }  }  return count; // Ошибка: результат подсчёта неверен  }  };  int main() {  try {  ReportGenerator generator;  int totalLines = generator.generateReport("data.txt");  std::cout << "Количество строк в файле: " << totalLines << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } | #include <iostream>  #include <fstream>  #include <string>  class ReportGenerator {  public:  int generateReport(const std::string& fileName) {  std::ifstream inFile(fileName);  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл.");  }  std::string line;  int count = 0;  while (std::getline(inFile, line)) {  count++; // Исправлено: все строки учитываются, включая пустые  }  if (count == 0) {  throw std::runtime\_error("Файл пуст. Нет строк для обработки.");  }  return count;  }  };  int main() {  try {  ReportGenerator generator;  int totalLines = generator.generateReport("data.txt");  std::cout << "Количество строк в файле: " << totalLines << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 8 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа вычисляет факториал числа с использованием рекурсивной функции. В коде допущены следующие ошибки:

**Первичная ошибка:** Программа не обрабатывает отрицательные значения.

**Вторичная ошибка:** Отсутствует корректное условие завершения рекурсии, что приводит к бесконечной рекурсии.

**Объяснение ошибок:**

**Первичная ошибка:** В методе factorial не предусмотрена проверка на отрицательные числа.

**Последствие:** Программа пытается вычислить факториал отрицательного числа, что вызывает некорректное поведение.

**Вторичная ошибка:** Отсутствие условия завершения рекурсии:

return n \* factorial(n - 1);

**Последствие:** Программа зацикливается, вызывая бесконечную рекурсию.

Исправленный код:

**Исправления:**

**Обработка отрицательных чисел:** В методе factorial добавлена проверка на отрицательные значения:

if (n < 0) {

throw std::invalid\_argument("Факториал отрицательных чисел не определён.");

}

**Добавлено условие завершения рекурсии:** При достижении n == 0 или n == 1, метод возвращает 1:

if (n == 0 || n == 1) {

return 1;

}

**Корректное использование входных данных:** Введено положительное значение number:

int number = 5;

**Пояснение исправлений:**

**Первичная ошибка:** Добавление проверки на отрицательные числа предотвращает некорректное поведение программы.

**Вторичная ошибка:** Условие завершения рекурсии обеспечивает правильный возврат результата и предотвращает бесконечный вызов функции.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  class FactorialCalculator {  public:  int factorial(int n) {  return n \* factorial(n - 1); // Ошибка: нет условия завершения рекурсии  }  };  int main() {  try {  FactorialCalculator calc;  int number = -5; // Ошибка: отрицательное число  int result = calc.factorial(number);  std::cout << "Факториал числа " << number << " равен " << result << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } | #include <iostream>  #include <stdexcept>  class FactorialCalculator {  public:  int factorial(int n) {  if (n < 0) {  throw std::invalid\_argument("Факториал отрицательных чисел не определён.");  }  if (n == 0 || n == 1) {  return 1; // Условие завершения рекурсии  }  return n \* factorial(n - 1);  }  };  int main() {  try {  FactorialCalculator calc;  int number = 5; // Исправлено: положительное число  int result = calc.factorial(number);  std::cout << "Факториал числа " << number << " равен " << result << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 9 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа предназначена для копирования данных из одного массива в другой с использованием указателей. В коде содержатся следующие ошибки:

**Первичная ошибка:** Размер целевого массива не проверяется, что может привести к выходу за пределы памяти.

**Вторичная ошибка:** Некорректная работа с указателями приводит к пропуску элементов.

**Объяснение ошибок:**

**Первичная ошибка:** Размер целевого массива меньше, чем требуется, но это не проверяется.

**Последствие:** Выход за пределы памяти может вызвать неопределённое поведение.

**Вторичная ошибка:** Цикл копирования выполняется на одну итерацию больше, чем нужно:

for (int i = 0; i <= size; i++)

**Последствие:** Программа записывает данные за пределы целевого массива.

Исправленный код:

**Исправления:**

**Добавлена проверка размеров массивов:** В методе copyArray добавлена проверка:

if (destSize < srcSize) {

throw std::invalid\_argument("Целевой массив меньше исходного. Копирование невозможно.");

}

**Корректное условие цикла:** Условие цикла изменено на:

for (int i = 0; i < srcSize; i++)

**Целевой массив исправлен:** Размер целевого массива теперь соответствует исходному:

int destination[5];

**Пояснение исправлений:**

**Первичная ошибка:** Проверка размеров предотвращает выход за пределы памяти целевого массива.

**Вторичная ошибка:** Корректный цикл исключает копирование данных за пределы исходного массива.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  class ArrayCopier {  public:  void copyArray(int\* src, int\* dest, int size) {  for (int i = 0; i <= size; i++) { // Ошибка: цикл выполняется на один шаг больше  dest[i] = src[i];  }  }  };  int main() {  int source[5] = {1, 2, 3, 4, 5};  int destination[4]; // Ошибка: размер массива меньше, чем требуется  ArrayCopier copier;  copier.copyArray(source, destination, 5); // Ошибка: размер превышает границу массива  std::cout << "Содержимое целевого массива: ";  for (int i = 0; i < 5; i++) {  std::cout << destination[i] << " ";  }  std::cout << std::endl;  return 0;  } | #include <iostream>  #include <stdexcept>  class ArrayCopier {  public:  void copyArray(int\* src, int\* dest, int srcSize, int destSize) {  if (destSize < srcSize) {  throw std::invalid\_argument("Целевой массив меньше исходного. Копирование невозможно.");  }  for (int i = 0; i < srcSize; i++) { // Исправлено: цикл ограничен srcSize  dest[i] = src[i];  }  }  };  int main() {  try {  int source[5] = {1, 2, 3, 4, 5};  int destination[5]; // Исправлено: размер соответствует исходному массиву  ArrayCopier copier;  copier.copyArray(source, destination, 5, 5);  std::cout << "Содержимое целевого массива: ";  for (int i = 0; i < 5; i++) {  std::cout << destination[i] << " ";  }  std::cout << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 10 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа предназначена для преобразования строки в верхний регистр. В коде содержатся ошибки:

**Первичная ошибка:** Программа не обрабатывает пустую строку, что приводит к сбою.

**Вторичная ошибка:** Не учитываются неалфавитные символы, которые вызывают некорректное поведение.

**Объяснение ошибок:**

**Первичная ошибка:** Если строка пустая, программа пытается обработать её символы, что вызывает неопределённое поведение.

**Последствие:** Программа аварийно завершается при доступе к несуществующему элементу строки.

**Вторичная ошибка:** Метод std::toupper применяется ко всем символам, включая неалфавитные, без проверки.

**Последствие:** Некорректная обработка символов, таких как цифры и знаки препинания.

Исправленный код:

**Исправления:**

**Обработка пустой строки:** Добавлена проверка:

if (input.empty()) {

throw std::invalid\_argument("Строка пуста. Преобразование невозможно.");

}

**Корректная обработка символов:** Алфавитные символы проверяются перед преобразованием:

if (std::isalpha(input[i])) {

input[i] = std::toupper(input[i]);

}

**Исправление цикла:** Условие цикла изменено:

for (size\_t i = 0; i < input.size(); i++)

**Корректные входные данные:** Для проверки использована строка с алфавитными и неалфавитными символами:

std::string text = "Hello, World! 123";

**Пояснение исправлений:**

**Первичная ошибка:** Проверка пустой строки предотвращает доступ к несуществующим элементам.

**Вторичная ошибка:** Условие обработки гарантирует, что только алфавитные символы изменяются, сохраняя остальные символы в исходном виде.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <string>  #include <cctype>  class StringTransformer {  public:  void toUpperCase(std::string& input) {  for (size\_t i = 0; i <= input.size(); i++) { // Ошибка: цикл выходит за пределы строки  input[i] = std::toupper(input[i]); // Ошибка: не проверяются неалфавитные символы  }  }  };  int main() {  std::string text = ""; // Ошибка: пустая строка  StringTransformer transformer;  transformer.toUpperCase(text);  std::cout << "Преобразованная строка: " << text << std::endl;  return 0;  } | #include <iostream>  #include <string>  #include <cctype>  class StringTransformer {  public:  void toUpperCase(std::string& input) {  if (input.empty()) {  throw std::invalid\_argument("Строка пуста. Преобразование невозможно.");  }  for (size\_t i = 0; i < input.size(); i++) { // Исправлено: цикл до input.size()  if (std::isalpha(input[i])) { // Исправлено: проверка на алфавитные символы  input[i] = std::toupper(input[i]);  }  }  }  };  int main() {  try {  std::string text = "Hello, World! 123"; // Исправлено: непустая строка  StringTransformer transformer;  transformer.toUpperCase(text);  std::cout << "Преобразованная строка: " << text << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 11 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа предназначена для вычисления среднего арифметического элементов массива. В коде содержатся ошибки:

**Первичная ошибка:** Программа не обрабатывает массив с нулевым размером, что приводит к делению на ноль.

**Вторичная ошибка:** Программа неверно суммирует элементы массива из-за ошибки в индексации.

**Объяснение ошибок:**

**Первичная ошибка:** При пустом массиве программа пытается выполнить деление на ноль:

return sum / numbers.size();

**Последствие:** Аварийное завершение программы.

**Вторичная ошибка:** Цикл выполняется на одну итерацию больше, чем размер массива:

for (size\_t i = 0; i <= numbers.size(); i++)

**Последствие:** Выход за пределы массива вызывает неопределённое поведение.

Исправленный код:

**Исправления:**

**Проверка на пустой массив:** Добавлена проверка:

if (numbers.empty()) {

throw std::invalid\_argument("Массив пуст. Среднее значение не может быть вычислено.");

}

**Корректное выполнение цикла:** Условие цикла изменено:

for (size\_t i = 0; i < numbers.size(); i++)

**Корректное деление:** Приведение типов обеспечивает вычисление дробного результата:

return static\_cast<double>(sum) / numbers.size();

**Исправлены входные данные:** Использован массив с несколькими элементами:

std::vector<int> numbers = {5, 10, 15};

**Пояснение исправлений:**

**Первичная ошибка:** Проверка на пустой массив исключает попытку деления на ноль.

**Вторичная ошибка:** Исправление цикла предотвращает выход за пределы массива.

**Дополнительно:** Дробное деление гарантирует точный результат.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <vector>  class AverageCalculator {  public:  double calculateAverage(const std::vector<int>& numbers) {  int sum = 0;  for (size\_t i = 0; i <= numbers.size(); i++) { // Ошибка: цикл выходит за пределы массива  sum += numbers[i];  }  return sum / numbers.size(); // Ошибка: деление на ноль при пустом массиве  }  };  int main() {  try {  std::vector<int> numbers = {}; // Ошибка: массив пуст  AverageCalculator calculator;  double avg = calculator.calculateAverage(numbers);  std::cout << "Среднее значение: " << avg << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } | #include <iostream>  #include <vector>  #include <stdexcept>  class AverageCalculator {  public:  double calculateAverage(const std::vector<int>& numbers) {  if (numbers.empty()) {  throw std::invalid\_argument("Массив пуст. Среднее значение не может быть вычислено.");  }  int sum = 0;  for (size\_t i = 0; i < numbers.size(); i++) { // Исправлено: цикл до numbers.size()  sum += numbers[i];  }  return static\_cast<double>(sum) / numbers.size(); // Исправлено: деление с приведением типов  }  };  int main() {  try {  std::vector<int> numbers = {5, 10, 15}; // Исправлено: массив содержит элементы  AverageCalculator calculator;  double avg = calculator.calculateAverage(numbers);  std::cout << "Среднее значение: " << avg << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 12 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа предназначена для подсчёта количества слов в текстовом файле. В коде содержатся ошибки:

**Первичная ошибка:** Файл может не открыться, но это не проверяется.

**Вторичная ошибка:** Слова подсчитываются некорректно, так как разделители (например, знаки препинания) не обрабатываются.

**Объяснение ошибок:**

**Первичная ошибка:** Отсутствует проверка успешного открытия файла:

std::ifstream inFile(fileName);

**Последствие:** Программа завершится с ошибкой, если файл отсутствует.

**Вторичная ошибка:** Подсчёт слов выполняется с использованием >>, который обрабатывает только пробелы как разделители:

while (inFile >> word)

**Последствие:** Знаки препинания остаются частью слов, из-за чего результат некорректен.

Исправленный код:

**Исправления:**

**Добавлена проверка открытия файла:** Введена проверка перед чтением:

if (!inFile.is\_open()) {

throw std::runtime\_error("Не удалось открыть файл.");

}

**Обработка разделителей:** Добавлен метод cleanWord для очистки слов от знаков препинания:

std::string cleanWord(const std::string& word);

**Чтение строки и разбиение на слова:** Использован std::getline для чтения строк, а затем разбиение строки на слова:

while (std::getline(inFile, line)) {

std::stringstream ss(line);

while (ss >> word) { ... }

}

**Пояснение исправлений:**

**Первичная ошибка:** Проверка файла предотвращает запуск программы с отсутствующим или некорректным файлом.

**Вторичная ошибка:** Обработка разделителей позволяет подсчитывать слова корректно, исключая знаки препинания.

**Дополнительно:** Использование методов std::getline и std::stringstream позволяет работать с текстом на уровне строк и слов.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <fstream>  #include <string>  class WordCounter {  public:  int countWords(const std::string& fileName) {  std::ifstream inFile(fileName);  std::string word;  int count = 0;  while (inFile >> word) { // Ошибка: файл может не открыться  count++;  }  return count; // Ошибка: не обрабатываются разделители  }  };  int main() {  WordCounter counter;  int totalWords = counter.countWords("data.txt"); // Ошибка: отсутствует проверка файла  std::cout << "Количество слов в файле: " << totalWords << std::endl;  return 0;  } | #include <iostream>  #include <fstream>  #include <sstream>  #include <string>  #include <cctype>  class WordCounter {  public:  int countWords(const std::string& fileName) {  std::ifstream inFile(fileName);  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл.");  }  std::string line, word;  int count = 0;  while (std::getline(inFile, line)) {  std::stringstream ss(line);  while (ss >> word) {  word = cleanWord(word);  if (!word.empty()) {  count++;  }  }  }  return count;  }  private:  std::string cleanWord(const std::string& word) {  std::string cleaned;  for (char ch : word) {  if (std::isalpha(ch)) { // Оставляем только буквы  cleaned += ch;  }  }  return cleaned;  }  };  int main() {  try {  WordCounter counter;  int totalWords = counter.countWords("data.txt");  std::cout << "Количество слов в файле: " << totalWords << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 13 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа предназначена для чтения списка студентов из файла, вычисления среднего балла и вывода студентов с баллом выше среднего. В коде содержатся ошибки:

**Первичная ошибка:** Программа не проверяет, был ли успешно открыт файл.

**Вторичная ошибка:** Средний балл рассчитывается неправильно из-за некорректного подсчёта общего числа студентов.

**Объяснение ошибок:**

**Первичная ошибка:** Отсутствует проверка на успешное открытие файла:

std::ifstream inFile(fileName);

**Последствие:** Если файл отсутствует или повреждён, программа завершится с ошибкой.

**Вторичная ошибка:** Если список студентов пуст, возникает деление на ноль:

return total / students.size();

Исправленный код:

**Исправления:**

**Проверка открытия файла:** Добавлена проверка:

if (!inFile.is\_open()) {

throw std::runtime\_error("Не удалось открыть файл.");

}

**Обработка пустого списка:** Добавлено исключение для пустого списка:

if (students.empty()) {

throw std::runtime\_error("Список студентов пуст.");

}

**Проверка формата данных:** Исключение выбрасывается при некорректном формате строки:

if (!(iss >> name >> grade)) {

throw std::runtime\_error("Некорректный формат строки: " + line);

}

**Пояснение исправлений:**

**Первичная ошибка:** Проверка файла предотвращает аварийное завершение программы.

**Вторичная ошибка:** Исключение для пустого списка предотвращает деление на ноль.

**Дополнительно:** Проверка данных файла обеспечивает корректное поведение программы.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <fstream>  #include <vector>  #include <string>  #include <sstream>  class StudentProcessor {  public:  void processStudents(const std::string& fileName) {  std::ifstream inFile(fileName);  std::string line;  std::vector<std::pair<std::string, double>> students;  while (std::getline(inFile, line)) { // Ошибка: файл может быть не открыт  std::istringstream iss(line);  std::string name;  double grade;  iss >> name >> grade;  students.push\_back({name, grade});  }  double avg = calculateAverage(students);  std::cout << "Средний балл: " << avg << std::endl;  std::cout << "Студенты с баллом выше среднего:" << std::endl;  for (const auto& student : students) {  if (student.second > avg) {  std::cout << student.first << " - " << student.second << std::endl;  }  }  }  private:  double calculateAverage(const std::vector<std::pair<std::string, double>>& students) {  double total = 0;  for (const auto& student : students) {  total += student.second;  }  return total / students.size(); // Ошибка: деление на ноль при пустом списке  }  };  int main() {  StudentProcessor processor;  processor.processStudents("students.txt"); // Ошибка: файл может отсутствовать  return 0;  } | #include <iostream>  #include <fstream>  #include <vector>  #include <string>  #include <sstream>  #include <stdexcept>  class StudentProcessor {  public:  void processStudents(const std::string& fileName) {  std::ifstream inFile(fileName);  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть файл.");  }  std::string line;  std::vector<std::pair<std::string, double>> students;  while (std::getline(inFile, line)) {  std::istringstream iss(line);  std::string name;  double grade;  if (!(iss >> name >> grade)) {  throw std::runtime\_error("Некорректный формат строки: " + line);  }  students.push\_back({name, grade});  }  if (students.empty()) {  throw std::runtime\_error("Список студентов пуст.");  }  double avg = calculateAverage(students);  std::cout << "Средний балл: " << avg << std::endl;  std::cout << "Студенты с баллом выше среднего:" << std::endl;  for (const auto& student : students) {  if (student.second > avg) {  std::cout << student.first << " - " << student.second << std::endl;  }  }  }  private:  double calculateAverage(const std::vector<std::pair<std::string, double>>& students) {  double total = 0;  for (const auto& student : students) {  total += student.second;  }  return total / students.size();  }  };  int main() {  try {  StudentProcessor processor;  processor.processStudents("students.txt");  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 14 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа реализует игру "Камень, ножницы, бумага" между пользователем и компьютером. В коде содержатся следующие ошибки:

**Первичная ошибка:** Программа не проверяет ввод пользователя на корректность, что приводит к некорректной работе при ошибочном вводе.

**Вторичная ошибка:** Логика определения победителя содержит ошибку, из-за которой результат игры может быть неверным.

**Объяснение ошибок:**

**Первичная ошибка:** Программа не проверяет ввод пользователя, из-за чего некорректное значение (например, 5 или -1) приводит к некорректному поведению:

std::cin >> userChoice;

**Вторичная ошибка:** Логика определения победителя пропускает некоторые случаи, из-за чего победа может быть присуждена неверно:

if ((userChoice == 0 && computerChoice == 1) || ...)

Исправленный код:

**Исправления:**

**Проверка ввода пользователя:** Добавлена проверка для исключения некорректных значений:

if (userChoice < 0 || userChoice > 2) {

std::cerr << "Некорректный ввод. Попробуйте снова." << std::endl;

return;

}

**Функция для преобразования числового выбора в строку:** Добавлена функция choiceToString для отображения выбора в удобочитаемом формате:

std::string choiceToString(int choice);

**Дополнительно:** Программа выводит выбор компьютера и пользователя, что делает её более понятной.

**Пояснение исправлений:**

**Первичная ошибка:** Проверка ввода предотвращает некорректную работу программы и улучшает пользовательский опыт.

**Вторичная ошибка:** Логика победы исправлена для правильной обработки всех возможных комбинаций.

**Дополнительно:** Усовершенствован пользовательский интерфейс для большей наглядности.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <cstdlib>  class RockPaperScissorsGame {  public:  void play() {  srand(static\_cast<unsigned>(time(0)));  int computerChoice = rand() % 3; // 0 - Камень, 1 - Ножницы, 2 - Бумага  int userChoice;  std::cout << "Выберите: 0 - Камень, 1 - Ножницы, 2 - Бумага: ";  std::cin >> userChoice;  if (userChoice == computerChoice) {  std::cout << "Ничья!" << std::endl;  } else if ((userChoice == 0 && computerChoice == 1) ||  (userChoice == 1 && computerChoice == 2) ||  (userChoice == 2 && computerChoice == 0)) {  std::cout << "Вы победили!" << std::endl;  } else {  std::cout << "Компьютер победил!" << std::endl;  }  }  };  int main() {  RockPaperScissorsGame game;  game.play();  return 0;  } | #include <iostream>  #include <cstdlib>  #include <ctime>  #include <stdexcept>  class RockPaperScissorsGame {  public:  void play() {  srand(static\_cast<unsigned>(time(0)));  int computerChoice = rand() % 3; // 0 - Камень, 1 - Ножницы, 2 - Бумага  int userChoice;  std::cout << "Выберите: 0 - Камень, 1 - Ножницы, 2 - Бумага: ";  std::cin >> userChoice;  if (userChoice < 0 || userChoice > 2) { // Исправлено: проверка ввода пользователя  std::cerr << "Некорректный ввод. Попробуйте снова." << std::endl;  return;  }  std::cout << "Компьютер выбрал: " << choiceToString(computerChoice) << std::endl;  std::cout << "Вы выбрали: " << choiceToString(userChoice) << std::endl;  if (userChoice == computerChoice) {  std::cout << "Ничья!" << std::endl;  } else if ((userChoice == 0 && computerChoice == 1) || // Камень бьёт ножницы  (userChoice == 1 && computerChoice == 2) || // Ножницы бьют бумагу  (userChoice == 2 && computerChoice == 0)) { // Бумага бьёт камень  std::cout << "Вы победили!" << std::endl;  } else {  std::cout << "Компьютер победил!" << std::endl;  }  }  private:  std::string choiceToString(int choice) {  switch (choice) {  case 0: return "Камень";  case 1: return "Ножницы";  case 2: return "Бумага";  default: return "Неизвестно";  }  }  };  int main() {  RockPaperScissorsGame game;  game.play();  return 0;  } |

**Вариант 15 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа реализует упрощённую версию игры "Морской бой", где игрок делает один выстрел по игровому полю компьютера. Поле представляет собой матрицу 5x5, на которой случайным образом размещён один корабль. В коде содержатся следующие ошибки:

**Первичная ошибка:** Программа не проверяет ввод координат игрока, что может привести к ошибке выхода за пределы массива.

**Вторичная ошибка:** Корабль всегда размещается в одной и той же позиции, так как генератор случайных чисел не инициализирован.

**Объяснение ошибок:**

**Первичная ошибка:** Программа не проверяет корректность ввода координат, что может привести к выходу за пределы игрового поля:

std::cin >> guessX >> guessY;

**Вторичная ошибка:** Позиция корабля зафиксирована, поэтому игра становится предсказуемой:

int shipX = 3; int shipY = 3;

Исправленный код:

**Исправления:**

**Инициализация генератора случайных чисел:** Корабль теперь размещается случайным образом на поле:

srand(static\_cast<unsigned>(time(0)));

int shipX = rand() % 5; int shipY = rand() % 5;

**Проверка корректности ввода:** Добавлена проверка, чтобы координаты попадали в диапазон от 0 до 4:

if (guessX < 0 || guessX >= 5 || guessY < 0 || guessY >= 5) {

std::cout << "Некорректные координаты. Повторите попытку.\n";

continue;

}

**Цикл для повторных попыток:** Игрок может стрелять до тех пор, пока не попадёт:

while (true) {

...

}

**Пояснение исправлений:**

**Первичная ошибка:** Проверка координат ввода предотвращает ошибки выхода за пределы массива.

**Вторичная ошибка:** Случайная позиция корабля делает каждую игру уникальной.

**Дополнительно:** Игра теперь интерактивная, пользователь может делать несколько попыток.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <cstdlib>  class BattleshipGame {  public:  void play() {  int shipX = 3; // Ошибка: фиксированное положение корабля  int shipY = 3;  int guessX, guessY;  std::cout << "Поле игры: 5x5. Попробуйте поразить корабль.\n";  std::cout << "Введите координаты выстрела (x y): ";  std::cin >> guessX >> guessY;  if (guessX == shipX && guessY == shipY) {  std::cout << "Попадание! Вы потопили корабль.\n";  } else {  std::cout << "Мимо! Корабль находится в другой точке.\n";  }  }  };  int main() {  BattleshipGame game;  game.play();  return 0;  } | #include <iostream>  #include <cstdlib>  #include <ctime>  class BattleshipGame {  public:  void play() {  srand(static\_cast<unsigned>(time(0)));  int shipX = rand() % 5; // Генерация случайной позиции корабля  int shipY = rand() % 5;  int guessX, guessY;  std::cout << "Поле игры: 5x5. Попробуйте поразить корабль.\n";  while (true) {  std::cout << "Введите координаты выстрела (x y, от 0 до 4): ";  std::cin >> guessX >> guessY;  if (guessX < 0 || guessX >= 5 || guessY < 0 || guessY >= 5) {  std::cout << "Некорректные координаты. Повторите попытку.\n";  continue;  }  if (guessX == shipX && guessY == shipY) {  std::cout << "Попадание! Вы потопили корабль.\n";  break;  } else {  std::cout << "Мимо! Попробуйте снова.\n";  }  }  }  };  int main() {  BattleshipGame game;  game.play();  return 0;  } |

**Вариант 16 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа подсчитывает количество уникальных чисел в массиве. В коде содержатся ошибки:

**Первичная ошибка:** Дублирующиеся элементы не удаляются корректно.

**Вторичная ошибка:** Результат подсчёта может быть некорректным из-за ошибки в обработке массива.

**Объяснение ошибок:**

**Первичная ошибка:** Цикл выходит за пределы массива:

for (size\_t i = 0; i <= arr.size(); i++)

**Последствие:** Программа пытается обратиться к несуществующим элементам массива, вызывая неопределённое поведение.

**Вторичная ошибка:** Элементы обрабатываются некорректно, так как std::unordered\_set не удаляет повторяющиеся элементы при выходе за пределы массива.

**Последствие:** Итоговый результат может содержать ошибки.

Исправленный код:

**Исправления:**

**Проверка на пустой массив:** Добавлена проверка перед началом обработки:

if (arr.empty()) {

throw std::invalid\_argument("Массив пуст. Подсчёт невозможен.");

}

**Корректное выполнение цикла:** Цикл ограничен размером массива:

for (size\_t i = 0; i < arr.size(); i++)

**Исправлены входные данные:**

* Использован массив с числами, содержащими дубликаты:

std::vector<int> numbers = {1, 2, 2, 3, 4, 4, 5};

**Пояснение исправлений:**

**Первичная ошибка:** Условие цикла исправлено, чтобы избежать выхода за пределы массива.

**Вторичная ошибка:** std::unordered\_set теперь корректно обрабатывает элементы, так как обращение к массиву находится в пределах допустимого.

**Дополнительно:** Добавлена проверка на пустой массив для повышения устойчивости программы.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <vector>  #include <unordered\_set>  class UniqueCounter {  public:  int countUnique(const std::vector<int>& arr) {  std::unordered\_set<int> uniqueNumbers;  for (size\_t i = 0; i <= arr.size(); i++) { // Ошибка: цикл выходит за пределы массива  uniqueNumbers.insert(arr[i]); // Ошибка: возможен выход за границы массива  }  return uniqueNumbers.size(); // Возвращает некорректное количество уникальных чисел  }  };  int main() {  std::vector<int> numbers = {1, 2, 2, 3, 4, 4, 5};  UniqueCounter counter;  int uniqueCount = counter.countUnique(numbers);  std::cout << "Количество уникальных чисел: " << uniqueCount << std::endl;  return 0;  } | #include <iostream>  #include <vector>  #include <unordered\_set>  class UniqueCounter {  public:  int countUnique(const std::vector<int>& arr) {  if (arr.empty()) {  throw std::invalid\_argument("Массив пуст. Подсчёт невозможен.");  }  std::unordered\_set<int> uniqueNumbers;  for (size\_t i = 0; i < arr.size(); i++) { // Исправлено: цикл ограничен размером массива  uniqueNumbers.insert(arr[i]);  }  return uniqueNumbers.size();  }  };  int main() {  try {  std::vector<int> numbers = {1, 2, 2, 3, 4, 4, 5}; // Корректные входные данные  UniqueCounter counter;  int uniqueCount = counter.countUnique(numbers);  std::cout << "Количество уникальных чисел: " << uniqueCount << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 17 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа копирует строки, начинающиеся с заглавной буквы, из одного файла в другой. В коде содержатся ошибки:

**Первичная ошибка:** Программа не проверяет, открыт ли входной файл, что приводит к сбою.

**Вторичная ошибка:** Строки проверяются некорректно, из-за чего не все строки копируются.

**Объяснение ошибок:**

**Первичная ошибка:** Входной файл не проверяется на успешное открытие:

std::ifstream inFile(inputFile);

**Последствие:** Программа завершится с ошибкой, если файл отсутствует.

**Вторичная ошибка:** Проверка строки на заглавную букву выполнена неправильно:

if (line[0] >= 'A' && line[0] <= 'Z')

**Последствие:** Строки с русскими заглавными буквами или другими символами не учитываются.

Исправленный код:

**Исправления:**

**Проверка открытия файлов:** Добавлены проверки для входного и выходного файлов:

if (!inFile.is\_open()) {

throw std::runtime\_error("Не удалось открыть входной файл.");

}

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

throw std::runtime\_error("Не удалось открыть выходной файл.");

}

**Корректная проверка заглавной буквы:** Использован метод std::isupper:

if (!line.empty() && std::isupper(static\_cast<unsigned char>(line[0])))

**Обработка пустых строк:** Добавлена проверка на пустую строку:

if (!line.empty())

**Пояснение исправлений:**

**Первичная ошибка:** Проверка входного файла исключает сбои при отсутствии файла.

**Вторичная ошибка:** Корректная проверка заглавной буквы позволяет работать с многоязычными строками.

**Дополнительно:** Обработка пустых строк предотвращает ненужную запись в выходной файл.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <fstream>  #include <cctype>  #include <string>  class FileCopier {  public:  void copyLines(const std::string& inputFile, const std::string& outputFile) {  std::ifstream inFile(inputFile);  std::ofstream outFile(outputFile);  std::string line;  while (std::getline(inFile, line)) { // Ошибка: не проверяется успешное открытие файла  if (line[0] >= 'A' && line[0] <= 'Z') { // Ошибка: некорректная проверка заглавной буквы  outFile << line << std::endl;  }  }  }  };  int main() {  FileCopier copier;  copier.copyLines("input.txt", "output.txt"); // Ошибка: файл может отсутствовать  std::cout << "Копирование завершено." << std::endl;  return 0;  } | #include <iostream>  #include <fstream>  #include <cctype>  #include <string>  class FileCopier {  public:  void copyLines(const std::string& inputFile, const std::string& outputFile) {  std::ifstream inFile(inputFile);  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть входной файл.");  }  std::ofstream outFile(outputFile);  if (!outFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть выходной файл.");  }  std::string line;  while (std::getline(inFile, line)) {  if (!line.empty() && std::isupper(static\_cast<unsigned char>(line[0]))) { // Исправлено: корректная проверка заглавной буквы  outFile << line << std::endl;  }  }  }  };  int main() {  try {  FileCopier copier;  copier.copyLines("input.txt", "output.txt");  std::cout << "Копирование завершено." << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 18 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа сравнивает два массива на идентичность по элементам и их порядку. В коде содержатся ошибки:

**Первичная ошибка:** Программа не проверяет, совпадает ли размер массивов перед сравнением.

**Вторичная ошибка:** Ошибка в логике цикла приводит к некорректному результату при сравнении.

**Объяснение ошибок:**

**Первичная ошибка:** Размеры массивов не проверяются перед сравнением:

for (size\_t i = 0; i <= arr1.size(); i++)

**Последствие:** Программа может обращаться к элементам за пределами одного из массивов.

**Вторичная ошибка:** Цикл выходит за пределы массива из-за некорректного условия:

for (size\_t i = 0; i <= arr1.size(); i++)

**Последствие:** Программа аварийно завершится.

Исправленный код:

**Исправления:**

1. **Проверка размеров массивов:**
   * Добавлена проверка на равенство размеров перед сравнением:

if (arr1.size() != arr2.size()) {

return false;

}

**Корректное выполнение цикла:**

* Цикл ограничен размером первого массива:

for (size\_t i = 0; i < arr1.size(); i++)

**Исправлены входные данные:**

* Массивы были сделаны одинаковыми для проверки:

std::vector<int> array2 = {1, 2, 3, 4};

**Пояснение исправлений:**

**Первичная ошибка:** Проверка размеров массивов исключает доступ к несуществующим элементам.

**Вторичная ошибка:** Исправление цикла гарантирует корректное сравнение элементов в пределах массива.

**Дополнительно:** Программа стала устойчивой к ошибкам и корректно обрабатывает любые массивы.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <vector>  class ArrayComparator {  public:  bool areIdentical(const std::vector<int>& arr1, const std::vector<int>& arr2) {  for (size\_t i = 0; i <= arr1.size(); i++) { // Ошибка: цикл выходит за пределы массива  if (arr1[i] != arr2[i]) {  return false;  }  }  return true; // Ошибка: длина массивов не проверяется  }  };  int main() {  std::vector<int> array1 = {1, 2, 3, 4};  std::vector<int> array2 = {1, 2, 3};  ArrayComparator comparator;  if (comparator.areIdentical(array1, array2)) {  std::cout << "Массивы идентичны." << std::endl;  } else {  std::cout << "Массивы различны." << std::endl;  }  return 0;  } | #include <iostream>  #include <vector>  class ArrayComparator {  public:  bool areIdentical(const std::vector<int>& arr1, const std::vector<int>& arr2) {  if (arr1.size() != arr2.size()) {  return false; // Исправлено: проверка размеров массивов  }  for (size\_t i = 0; i < arr1.size(); i++) { // Исправлено: цикл ограничен arr1.size()  if (arr1[i] != arr2[i]) {  return false;  }  }  return true;  }  };  int main() {  std::vector<int> array1 = {1, 2, 3, 4};  std::vector<int> array2 = {1, 2, 3, 4}; // Исправлено: массивы одинаковы  ArrayComparator comparator;  if (comparator.areIdentical(array1, array2)) {  std::cout << "Массивы идентичны." << std::endl;  } else {  std::cout << "Массивы различны." << std::endl;  }  return 0;  } |

**Вариант 19 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа находит общие элементы между двумя массивами. В коде содержатся ошибки:

**Первичная ошибка:** Программа не проверяет размеры массивов, что может привести к выходу за пределы массива.

**Вторичная ошибка:** Общие элементы добавляются в результат несколько раз, из-за чего пересечение содержит дубли.

**Объяснение ошибок:**

**Первичная ошибка:** Программа не проверяет дублирующиеся элементы:

intersection.push\_back(arr1[i]);

**Последствие:** Общие элементы могут добавляться несколько раз.

**Вторичная ошибка:** Логика не предусматривает проверку размеров массивов перед поиском, что может вызвать неопределённое поведение.

**Исправления:**

**Удаление дублирующихся элементов:** Использован std::unordered\_set для исключения повторений:

std::unordered\_set<int> intersection;

**Универсальность работы с массивами:** Для сравнения используется std::unordered\_set, что упрощает поиск:

if (elements.find(num) != elements.end()) {

intersection.insert(num);

}

**Оптимизация кода:** Вместо вложенных циклов используется структура данных unordered\_set, что снижает сложность.

**Пояснение исправлений:**

**Первичная ошибка:** Использование unordered\_set устраняет дублирование элементов.

**Вторичная ошибка:** Оптимизация алгоритма делает программу быстрее и надёжнее.

**Дополнительно:** Код стал проще, а его выполнение – эффективнее, особенно для больших массивов.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <vector>  class ArrayIntersection {  public:  std::vector<int> findIntersection(const std::vector<int>& arr1, const std::vector<int>& arr2) {  std::vector<int> intersection;  for (size\_t i = 0; i < arr1.size(); i++) {  for (size\_t j = 0; j < arr2.size(); j++) {  if (arr1[i] == arr2[j]) {  intersection.push\_back(arr1[i]); // Ошибка: не проверяются дубли  }  }  }  return intersection;  }  };  int main() {  std::vector<int> array1 = {1, 2, 3, 4};  std::vector<int> array2 = {3, 4, 4, 5};  ArrayIntersection intersectionFinder;  std::vector<int> result = intersectionFinder.findIntersection(array1, array2);  std::cout << "Пересечения массивов: ";  for (int num : result) {  std::cout << num << " ";  }  std::cout << std::endl;  return 0;  } | #include <iostream>  #include <vector>  #include <unordered\_set>  class ArrayIntersection {  public:  std::vector<int> findIntersection(const std::vector<int>& arr1, const std::vector<int>& arr2) {  std::unordered\_set<int> elements(arr1.begin(), arr1.end());  std::unordered\_set<int> intersection;  for (const int& num : arr2) {  if (elements.find(num) != elements.end()) {  intersection.insert(num); // Исправлено: используется set для удаления дублей  }  }  return std::vector<int>(intersection.begin(), intersection.end());  }  };  int main() {  std::vector<int> array1 = {1, 2, 3, 4};  std::vector<int> array2 = {3, 4, 4, 5};  ArrayIntersection intersectionFinder;  std::vector<int> result = intersectionFinder.findIntersection(array1, array2);  std::cout << "Пересечения массивов: ";  for (int num : result) {  std::cout << num << " ";  }  std::cout << std::endl;  return 0;  } |

**Вариант 20 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа ищет минимальное значение в двумерной матрице. В коде содержатся ошибки:

**Первичная ошибка:** Матрица не проверяется на наличие строк, что может привести к сбою.

**Вторичная ошибка:** Ошибка в сравнении приводит к тому, что минимальное значение определяется некорректно.

**Объяснение ошибок:**

**Первичная ошибка:** Матрица не проверяется на наличие строк или столбцов:

for (size\_t i = 0; i <= matrix.size(); i++)

**Последствие:** Программа может обратиться к несуществующему элементу, что вызовет сбой.

**Вторичная ошибка:** Ошибка в логике циклов: проверяются индексы, выходящие за пределы матрицы:

for (size\_t j = 0; j <= matrix[i].size(); j++)

**Последствие:** Минимальное значение может быть рассчитано неверно.

**Исправления:**

**Проверка на пустую матрицу:** Добавлена проверка перед началом обработки:

if (matrix.empty() || matrix[0].empty()) {

throw std::invalid\_argument("Матрица пуста. Поиск минимума невозможен.");

}

**Корректное выполнение циклов:** Циклы ограничены реальными размерами строк и столбцов:

for (size\_t i = 0; i < matrix.size(); i++) {

for (size\_t j = 0; j < matrix[i].size(); j++) {

**Исправлены входные данные:** Матрица содержит данные для проверки:

std::vector<std::vector<int>> matrix = {{3, 5, 1}, {8, 2, 9}, {4, 6, 7}};

**Пояснение исправлений:**

**Первичная ошибка:** Проверка матрицы исключает возможные сбои из-за пустого набора данных.

**Вторичная ошибка:** Корректное ограничение циклов предотвращает выход за пределы матрицы.

**Дополнительно:** Код стал надёжным и способен корректно обрабатывать любые матрицы.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <vector>  #include <limits>  class MatrixProcessor {  public:  int findMinimum(const std::vector<std::vector<int>>& matrix) {  int minVal = std::numeric\_limits<int>::max();  for (size\_t i = 0; i <= matrix.size(); i++) { // Ошибка: цикл выходит за пределы строк  for (size\_t j = 0; j <= matrix[i].size(); j++) { // Ошибка: некорректное обращение к столбцам  if (matrix[i][j] < minVal) {  minVal = matrix[i][j];  }  }  }  return minVal;  }  };  int main() {  std::vector<std::vector<int>> matrix = {{3, 5, 1}, {8, 2, 9}, {4, 6, 7}};  MatrixProcessor processor;  int minValue = processor.findMinimum(matrix);  std::cout << "Минимальное значение в матрице: " << minValue << std::endl;  return 0;  } | #include <iostream>  #include <vector>  #include <limits>  #include <stdexcept>  class MatrixProcessor {  public:  int findMinimum(const std::vector<std::vector<int>>& matrix) {  if (matrix.empty() || matrix[0].empty()) {  throw std::invalid\_argument("Матрица пуста. Поиск минимума невозможен.");  }  int minVal = std::numeric\_limits<int>::max();  for (size\_t i = 0; i < matrix.size(); i++) {  for (size\_t j = 0; j < matrix[i].size(); j++) {  if (matrix[i][j] < minVal) {  minVal = matrix[i][j];  }  }  }  return minVal;  }  };  int main() {  try {  std::vector<std::vector<int>> matrix = {{3, 5, 1}, {8, 2, 9}, {4, 6, 7}};  MatrixProcessor processor;  int minValue = processor.findMinimum(matrix);  std::cout << "Минимальное значение в матрице: " << minValue << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 21 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа транспонирует двумерную матрицу. В коде содержатся ошибки:

**Первичная ошибка:** Программа не проверяет, является ли матрица пустой, что может привести к сбою.

**Вторичная ошибка:** Алгоритм транспонирования нарушает порядок элементов из-за неправильного доступа к индексам.

**Объяснение ошибок:**

**Первичная ошибка:** Программа не проверяет, пуста ли матрица, что приводит к неопределённому поведению:

std::vector<std::vector<int>> result(matrix[0].size(), std::vector<int>(matrix.size()));

**Вторичная ошибка:** Циклы выхода за пределы индекса приводят к сбоям:

for (size\_t i = 0; i <= matrix.size(); i++) { ... }

Исправленный код:

**Исправления:**

**Проверка пустой матрицы:** Добавлена проверка, исключающая работу с пустым набором данных:

if (matrix.empty() || matrix[0].empty()) {

throw std::invalid\_argument("Матрица пуста. Транспонирование невозможно.");

}

**Корректные циклы:** Оба цикла исправлены для работы в пределах реальных размеров матрицы:

for (size\_t i = 0; i < matrix.size(); i++) {

for (size\_t j = 0; j < matrix[i].size(); j++) {

**Исправлены входные данные:** Использована квадратная матрица с корректными значениями:

std::vector<std::vector<int>> matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

**Пояснение исправлений:**

**Первичная ошибка:** Проверка входных данных предотвращает аварийное завершение программы.

**Вторичная ошибка:** Исправленные циклы обеспечивают корректное транспонирование матрицы.

**Дополнительно:** Программа обрабатывает как квадратные, так и прямоугольные матрицы.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <vector>  class MatrixProcessor {  public:  std::vector<std::vector<int>> transpose(const std::vector<std::vector<int>>& matrix) {  std::vector<std::vector<int>> result(matrix[0].size(), std::vector<int>(matrix.size()));  for (size\_t i = 0; i <= matrix.size(); i++) { // Ошибка: цикл выходит за пределы строк  for (size\_t j = 0; j <= matrix[i].size(); j++) { // Ошибка: доступ к неверным индексам  result[j][i] = matrix[i][j];  }  }  return result;  }  };  int main() {  std::vector<std::vector<int>> matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};  MatrixProcessor processor;  std::vector<std::vector<int>> transposed = processor.transpose(matrix);  std::cout << "Транспонированная матрица:" << std::endl;  for (const auto& row : transposed) {  for (int val : row) {  std::cout << val << " ";  }  std::cout << std::endl;  }  return 0;  } | #include <iostream>  #include <vector>  #include <stdexcept>  class MatrixProcessor {  public:  std::vector<std::vector<int>> transpose(const std::vector<std::vector<int>>& matrix) {  if (matrix.empty() || matrix[0].empty()) {  throw std::invalid\_argument("Матрица пуста. Транспонирование невозможно.");  }  std::vector<std::vector<int>> result(matrix[0].size(), std::vector<int>(matrix.size()));  for (size\_t i = 0; i < matrix.size(); i++) { // Исправлено: цикл до matrix.size()  for (size\_t j = 0; j < matrix[i].size(); j++) { // Исправлено: цикл до matrix[i].size()  result[j][i] = matrix[i][j];  }  }  return result;  }  };  int main() {  try {  std::vector<std::vector<int>> matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};  MatrixProcessor processor;  std::vector<std::vector<int>> transposed = processor.transpose(matrix);  std::cout << "Транспонированная матрица:" << std::endl;  for (const auto& row : transposed) {  for (int val : row) {  std::cout << val << " ";  }  std::cout << std::endl;  }  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 22 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа проверяет, является ли квадратная матрица симметричной относительно главной диагонали. В коде содержатся ошибки:

**Первичная ошибка:** Матрица не проверяется на квадратность перед выполнением проверки.

**Вторичная ошибка:** Логика сравнения элементов нарушена из-за ошибки в индексации.

**Объяснение ошибок:**

**Первичная ошибка:** Матрица не проверяется на квадратность:

for (size\_t i = 0; i < matrix.size(); i++) { ... }

**Последствие:** Программа может попытаться обратиться к несуществующим элементам в неквадратной матрице.

**Вторичная ошибка:** Условие цикла выходит за пределы столбцов:

for (size\_t j = 0; j <= matrix.size(); j++)

**Последствие:** Программа завершится с ошибкой.

Исправленный код:

**Исправления:**

**Проверка квадратности матрицы:** Добавлена проверка перед началом обработки:

if (matrix.empty() || matrix.size() != matrix[0].size()) {

throw std::invalid\_argument("Матрица должна быть квадратной.");

}

**Исправление цикла:** Внутренний цикл ограничен до текущей строки:

for (size\_t j = 0; j < i; j++)

**Исправлены индексы:** Сравниваются только элементы относительно главной диагонали:

if (matrix[i][j] != matrix[j][i])

**Пояснение исправлений:**

**Первичная ошибка:** Проверка квадратности предотвращает сбои при работе с некорректными входными данными.

**Вторичная ошибка:** Уменьшение количества проверок улучшает производительность и предотвращает ненужные итерации.

**Дополнительно:** Код стал устойчивым к ошибкам и корректно проверяет симметричность.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <vector>  class MatrixChecker {  public:  bool isSymmetric(const std::vector<std::vector<int>>& matrix) {  for (size\_t i = 0; i < matrix.size(); i++) {  for (size\_t j = 0; j <= matrix.size(); j++) { // Ошибка: цикл выходит за пределы  if (matrix[i][j] != matrix[j][i]) { // Ошибка: проверяются неверные индексы  return false;  }  }  }  return true;  }  };  int main() {  std::vector<std::vector<int>> matrix = {{1, 2, 3}, {2, 1, 4}, {3, 4, 1}};  MatrixChecker checker;  if (checker.isSymmetric(matrix)) {  std::cout << "Матрица симметрична." << std::endl;  } else {  std::cout << "Матрица не симметрична." << std::endl;  }  return 0;  } | #include <iostream>  #include <vector>  #include <stdexcept>  class MatrixChecker {  public:  bool isSymmetric(const std::vector<std::vector<int>>& matrix) {  if (matrix.empty() || matrix.size() != matrix[0].size()) {  throw std::invalid\_argument("Матрица должна быть квадратной.");  }  for (size\_t i = 0; i < matrix.size(); i++) {  for (size\_t j = 0; j < i; j++) { // Исправлено: сравниваем только нижний треугольник  if (matrix[i][j] != matrix[j][i]) {  return false;  }  }  }  return true;  }  };  int main() {  try {  std::vector<std::vector<int>> matrix = {{1, 2, 3}, {2, 1, 4}, {3, 4, 1}};  MatrixChecker checker;  if (checker.isSymmetric(matrix)) {  std::cout << "Матрица симметрична." << std::endl;  } else {  std::cout << "Матрица не симметрична." << std::endl;  }  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 23 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа удаляет заданную строку из текстового файла и записывает результат в новый файл. В коде содержатся ошибки:

**Первичная ошибка:** Программа не проверяет, открыт ли входной файл.

**Вторичная ошибка:** Программа удаляет не только совпадающие строки, но и похожие, из-за ошибки сравнения.

**Объяснение ошибок:**

**Первичная ошибка:** Отсутствует проверка успешного открытия файлов:

std::ifstream inFile(inputFile);

std::ofstream outFile(outputFile);

**Последствие:** Программа может завершиться с ошибкой, если файл отсутствует.

**Вторичная ошибка:** Условие сравнения некорректно:

if (line.find(lineToRemove) != std::string::npos)

**Последствие:** Удаляются строки, содержащие подстроку, а не точное совпадение.

Исправленный код:

**Исправления:**

**Проверка открытия файлов:** Добавлена проверка для входного и выходного файлов:

if (!inFile.is\_open()) {

throw std::runtime\_error("Не удалось открыть входной файл.");

}

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

throw std::runtime\_error("Не удалось открыть выходной файл.");

}

**Точное сравнение строк:**

* Изменено условие, чтобы удалялись только строки, полностью совпадающие с указанной:

if (line != lineToRemove)

**Устойчивость к ошибкам:** Программа выбрасывает исключения при ошибках с файлами, что делает её более надёжной.

**Пояснение исправлений:**

**Первичная ошибка:** Проверка файлов исключает аварийное завершение программы.

**Вторичная ошибка:** Точное сравнение строк предотвращает удаление строк, содержащих только часть указанной строки.

**Дополнительно:** Улучшенная обработка делает программу более точной и надёжной для реальных сценариев.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <fstream>  #include <string>  class FileEditor {  public:  void removeLine(const std::string& inputFile, const std::string& outputFile, const std::string& lineToRemove) {  std::ifstream inFile(inputFile);  std::ofstream outFile(outputFile);  std::string line;  while (std::getline(inFile, line)) { // Ошибка: не проверяется успешное открытие файлов  if (line.find(lineToRemove) != std::string::npos) { // Ошибка: сравнение некорректно  continue;  }  outFile << line << std::endl;  }  }  };  int main() {  FileEditor editor;  editor.removeLine("input.txt", "output.txt", "Удаляемая строка"); // Ошибка: возможен сбой при отсутствии файла  std::cout << "Операция завершена." << std::endl;  return 0;  } | #include <iostream>  #include <fstream>  #include <string>  #include <stdexcept>  class FileEditor {  public:  void removeLine(const std::string& inputFile, const std::string& outputFile, const std::string& lineToRemove) {  std::ifstream inFile(inputFile);  if (!inFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть входной файл.");  }  std::ofstream outFile(outputFile);  if (!outFile.is\_open()) {  throw std::runtime\_error("Не удалось открыть выходной файл.");  }  std::string line;  while (std::getline(inFile, line)) {  if (line != lineToRemove) { // Исправлено: точное сравнение строк  outFile << line << std::endl;  }  }  }  };  int main() {  try {  FileEditor editor;  editor.removeLine("input.txt", "output.txt", "Удаляемая строка");  std::cout << "Операция завершена." << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |

**Вариант 24 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа проверяет, является ли строка палиндромом (читается одинаково в обе стороны). В коде содержатся ошибки:

**Первичная ошибка:** Программа не обрабатывает строку с пробелами и знаками препинания.

**Вторичная ошибка:** Логика проверки выходит за пределы строки из-за ошибки в индексации.

**Объяснение ошибок:**

**Первичная ошибка:** Программа не обрабатывает пробелы и знаки препинания, что приводит к неверной проверке:

if (str[i] != str[n - i - 1])

**Вторичная ошибка:** Цикл сравнения выходит за пределы строки из-за неверного условия:

for (size\_t i = 0; i <= n / 2; i++)

Исправленный код:

**Исправления:**

**Очистка строки от лишних символов:** Добавлен метод cleanString, который удаляет пробелы и знаки препинания, а также приводит символы к нижнему регистру:

std::string cleanString(const std::string& str);

**Корректное условие цикла:** Цикл ограничен половиной длины строки:

for (size\_t i = 0; i < n / 2; i++)

**Исправлены входные данные:** Использована строка с пробелами и знаками препинания:

std::string input = "A man, a plan, a canal, Panama";

**Пояснение исправлений:**

**Первичная ошибка:** Удаление неалфавитных символов и приведение строки к единому регистру обеспечивают корректную проверку.

**Вторичная ошибка:** Условие цикла исправлено для работы в пределах строки.

**Дополнительно:** Программа стала универсальной для строк любой сложности.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  #include <string>  class PalindromeChecker {  public:  bool isPalindrome(const std::string& str) {  size\_t n = str.size();  for (size\_t i = 0; i <= n / 2; i++) { // Ошибка: цикл выходит за пределы  if (str[i] != str[n - i - 1]) {  return false;  }  }  return true;  }  };  int main() {  std::string input = "A man, a plan, a canal, Panama"; // Ошибка: строка с пробелами и знаками препинания  PalindromeChecker checker;  if (checker.isPalindrome(input)) {  std::cout << "\"" << input << "\" является палиндромом." << std::endl;  } else {  std::cout << "\"" << input << "\" не является палиндромом." << std::endl;  }  return 0;  } | #include <iostream>  #include <string>  #include <cctype>  class PalindromeChecker {  public:  bool isPalindrome(const std::string& str) {  std::string cleaned = cleanString(str);  size\_t n = cleaned.size();  for (size\_t i = 0; i < n / 2; i++) { // Исправлено: цикл до n / 2  if (cleaned[i] != cleaned[n - i - 1]) {  return false;  }  }  return true;  }  private:  std::string cleanString(const std::string& str) {  std::string cleaned;  for (char ch : str) {  if (std::isalnum(static\_cast<unsigned char>(ch))) {  cleaned += std::tolower(static\_cast<unsigned char>(ch));  }  }  return cleaned;  }  };  int main() {  std::string input = "A man, a plan, a canal, Panama";  PalindromeChecker checker;  if (checker.isPalindrome(input)) {  std::cout << "\"" << input << "\" является палиндромом." << std::endl;  } else {  std::cout << "\"" << input << "\" не является палиндромом." << std::endl;  }  return 0;  } |

**Вариант 25 МДК 04.02**

**Задание2**

Дана программа, ваша задача найти ошибки, объяснить их влияние на результат работы программы и предложить исправления.

**Описание:** Программа вычисляет факториал введённого числа. В коде содержатся ошибки:

**Первичная ошибка:** Программа не проверяет, является ли введённое число отрицательным.

**Вторичная ошибка:** Переполнение данных при вычислении факториала для больших чисел приводит к неверным результатам.

**Объяснение ошибок:**

**Первичная ошибка:** Программа не проверяет, является ли число отрицательным, и выполняет расчёт:

int factorial = calculator.calculate(number);

**Последствие:** Факторил отрицательного числа не имеет математического смысла.

**Вторичная ошибка:** Переполнение целочисленного значения происходит при вычислении факториала больших чисел:

result \*= i;

Исправленный код:

**Исправления:**

**Обработка отрицательных чисел:** Добавлено исключение для отрицательных значений:

if (n < 0) {

throw std::invalid\_argument("Факториал отрицательного числа не определён.");

}

**Обработка переполнения:** Добавлена проверка на переполнение перед умножением:

if (result > std::numeric\_limits<long long>::max() / i) {

throw std::overflow\_error("Переполнение при вычислении факториала.");

}

**Использование типа long long:** Заменён тип результата для поддержки больших значений:

long long result = 1;

**Пояснение исправлений:**

**Первичная ошибка:** Исключение для отрицательных чисел предотвращает некорректные вычисления.

**Вторичная ошибка:** Проверка на переполнение делает программу устойчивой при больших входных данных.

**Дополнительно:** Использование типа long long позволяет работать с факториалами до 20 без переполнения.

|  |  |
| --- | --- |
| **Код:** | **Исправленный код:** |
| #include <iostream>  class FactorialCalculator {  public:  int calculate(int n) {  int result = 1;  for (int i = 1; i <= n; i++) {  result \*= i; // Ошибка: переполнение для больших n  }  return result; // Ошибка: нет проверки отрицательных значений  }  };  int main() {  int number = -5; // Ошибка: ввод отрицательного числа  FactorialCalculator calculator;  int factorial = calculator.calculate(number);  std::cout << "Факториал числа " << number << " равен " << factorial << "." << std::endl;  return 0;  } | #include <iostream>  #include <stdexcept>  #include <limits>  class FactorialCalculator {  public:  long long calculate(int n) {  if (n < 0) {  throw std::invalid\_argument("Факториал отрицательного числа не определён.");  }  long long result = 1;  for (int i = 1; i <= n; i++) {  if (result > std::numeric\_limits<long long>::max() / i) {  throw std::overflow\_error("Переполнение при вычислении факториала.");  }  result \*= i;  }  return result;  }  };  int main() {  try {  int number = 20; // Корректное положительное число  FactorialCalculator calculator;  long long factorial = calculator.calculate(number);  std::cout << "Факториал числа " << number << " равен " << factorial << "." << std::endl;  } catch (const std::exception& e) {  std::cerr << "Ошибка: " << e.what() << std::endl;  }  return 0;  } |