|  |
| --- |
| Министерство науки и высшего образования РФ Федеральное государственное автономное образовательное учреждение высшего профессионального образования **«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»** |
| Институт космических и информационных технологий |
| Кафедра вычислительной техники |

**Лабораторная работа №5**

**«**Инструментальные средства анализа кода программ**»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Преподаватель |  |  |  |  |  | Л.С. Артемьев |
|  |  |  |  |  | подпись, дата |  |
| Студент |  | КИ22-07Б, 032214070 |  |  | В.А. Егунько |
|  |  | номер групп, зачетной книжки |  | подпись, дата |  |

Красноярск 2023

**Техническое задание**

1. Исследование программы полученной в ходе выполнения работы 3 с помощью инструментов статического и динамического анализа кода.
2. Оценка полученных решений в соответствии с критериями качества ПО.
3. Улучшение программы – устранение обнаруженных недочётов.

Таблица 1 – Задание по варианту

|  |  |
| --- | --- |
| Вариант | Задание |
| 17 | Меню: название (строка), цена (дробное), время приготовления |

**Ход работы**

Для статического анализа кода я использовал приложение «Cppcheck». «Cppcheck» — статический анализатор кода для языка C/C++, предназначенный для поиска ошибок, которые не обнаруживаются компиляторами. Главной целью приложения является сведение до минимума количества ложных срабатываний при поиске ошибок. Анализатор способен проверять нестандартные участки кода, включающие использование расширений компилятора, встраиваемый ассемблер и т. п.

После анализа моего кода, программа показала следующее:

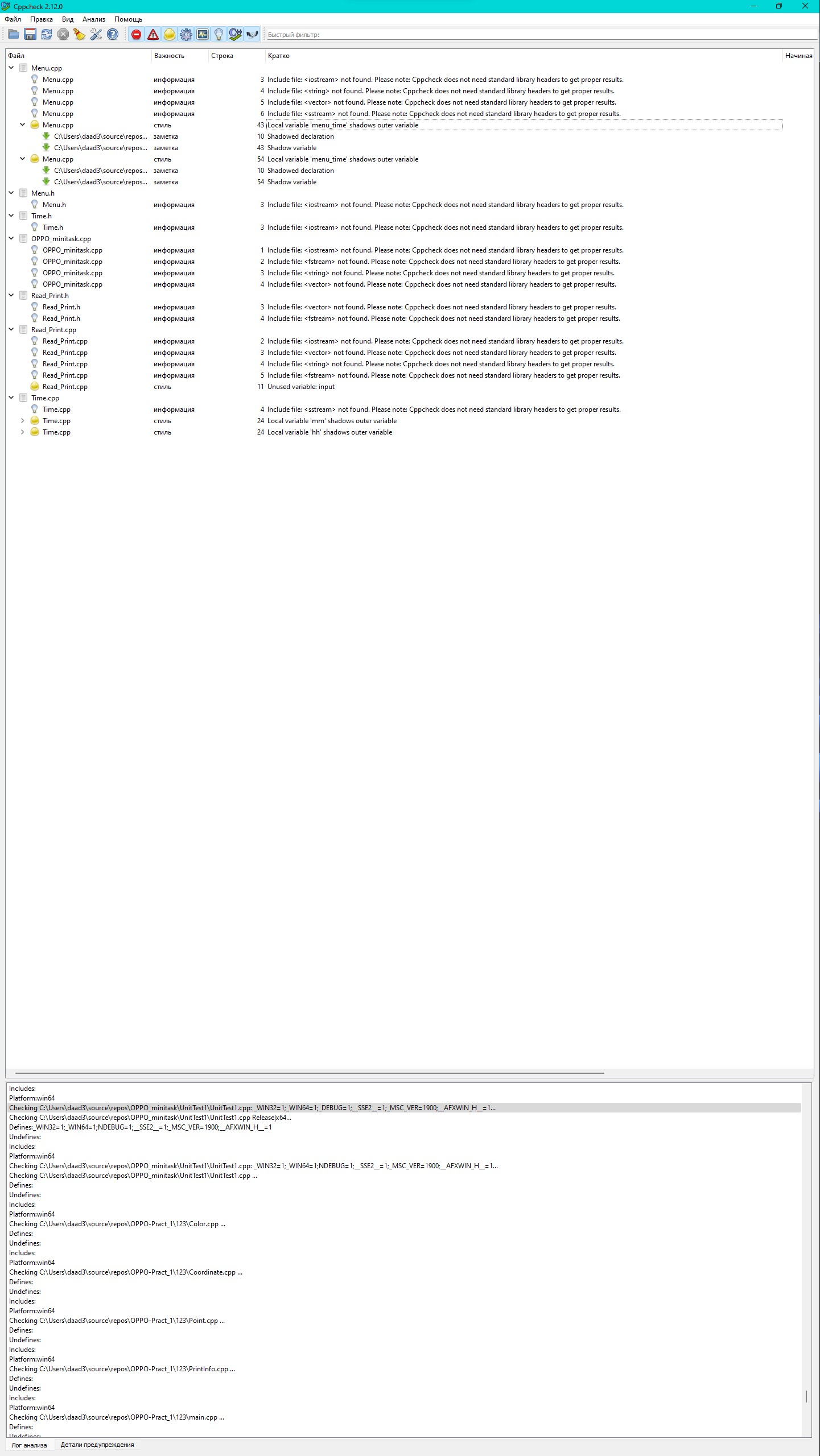


Рисунок 1 – Анализ программы

**Примечания:**

* Во вкладке «Файл» содержатся все открытые пользователем файлы;
* Каждая обнаруженная проблема может быть помечена одним из нескольких уровней важности:

1. Ошибка (error): Проблема, которая, скорее всего, приведет к ошибке в работе программы. Это могут быть серьезные ошибки, такие как утечки памяти или выход за пределы массивов;
2. Предупреждение (warning): Проблема, которая может привести к ошибке, но также может быть ложным срабатыванием. Например, потенциально неверное использование стандартных функций или подозрительные конструкции.;
3. Стиль (style): Это не ошибка и не предупреждение, а скорее вопрос стиля кодирования или оптимизации. Например, неэффективные алгоритмы STL или потенциально неиспользуемый код;
4. Производительность (performance): Проблемы, которые, скорее всего, не приведут к ошибкам, но могут замедлить выполнение программы. Например, использование временных объектов, которые можно было бы избежать;
5. Портативность (portability): Проблемы, которые могут помешать вашей программе работать на других компиляторах или платформах. Например, использование нестандартных функций или поведения, которое может отличаться в разных средах;
6. Информация (information): Это не предупреждения о проблемах, а скорее информация, которая может быть полезна для разработчика. Например, пропущенные определения функций или файлы, которые не были проанализированы.

* Вкладка «Строка» показывает строку в файле, где была обнаружена ошибка;
* Вкладка «Кратко» содержит общую информацию об ошибочности конкретной строки в конкретном файле;

**Устранение полученных недочетов**

1. Сообщение об ошибке **[shadowVariable]** указывает на то, что локальная переменная "затеняет" другую переменную с тем же именем, которая уже определена в более широкой области видимости. Чтобы устранить эти ошибки в файлах «Menu.cpp» и «Time.cpp» необходимо переименовать локальные переменные так, чтобы они не конфликтовали с именами членов класса:

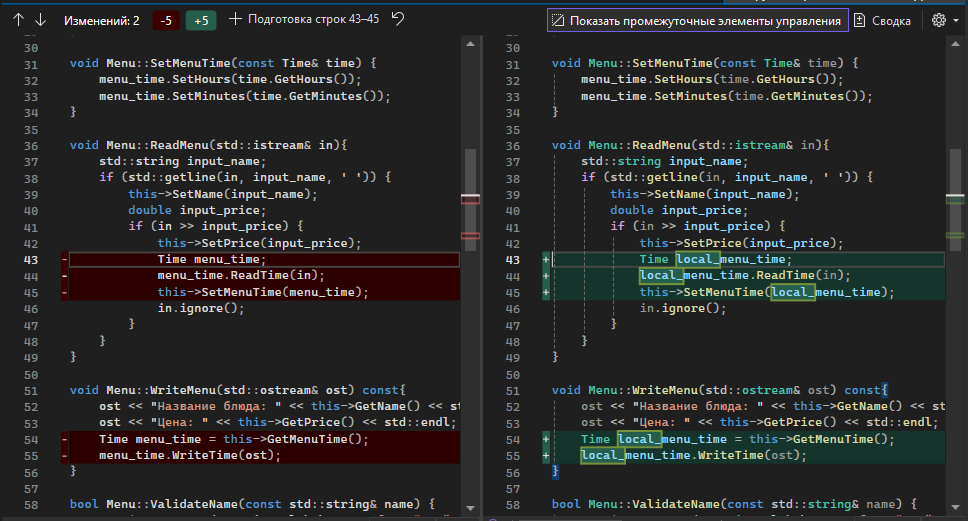


Рисунок 2 – Устранение ошибки в файле «Menu.cpp»

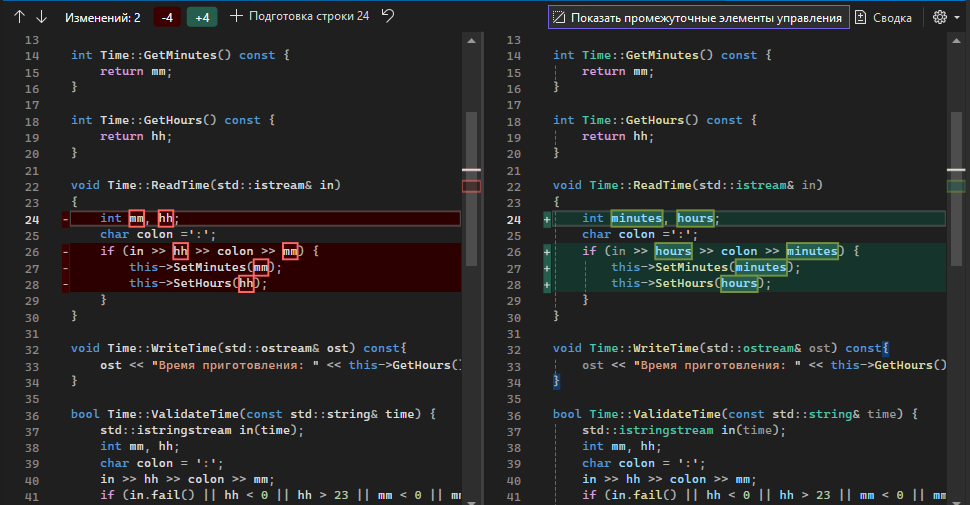


Рисунок 3 – Устранение ошибки в файле «Menu.cpp»

1. Для устранения ошибки **[unusedVariable]**, обнаруженной в файле «Read\_Print.cpp» необходимо удалить неиспользуемую переменную input:

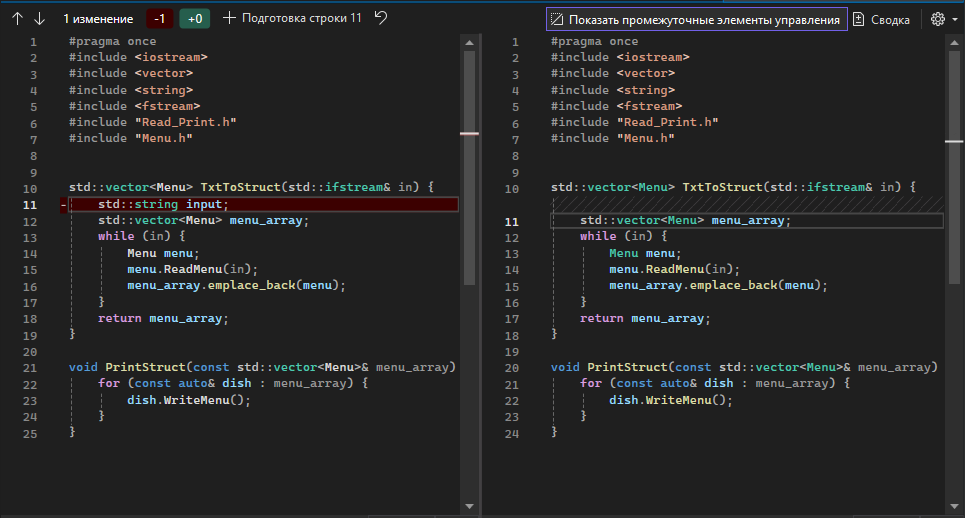


Рисунок 4 – Устранение ошибки

1. Сообщение от «cppcheck» [missingIncludeSystem] означает, что «cppcheck» не смог найти заголовочные файлы стандартных библиотек, которые включаются в коде. Это сообщение - информационное. Программа нуждается в стандартных библиотеках для компиляции и выполнения, поэтому необходимо оставить эти #include.

После устранения всех ошибок запускаем анализ программы и видим, что остались только информационные сообщения, а все ошибки устранены:

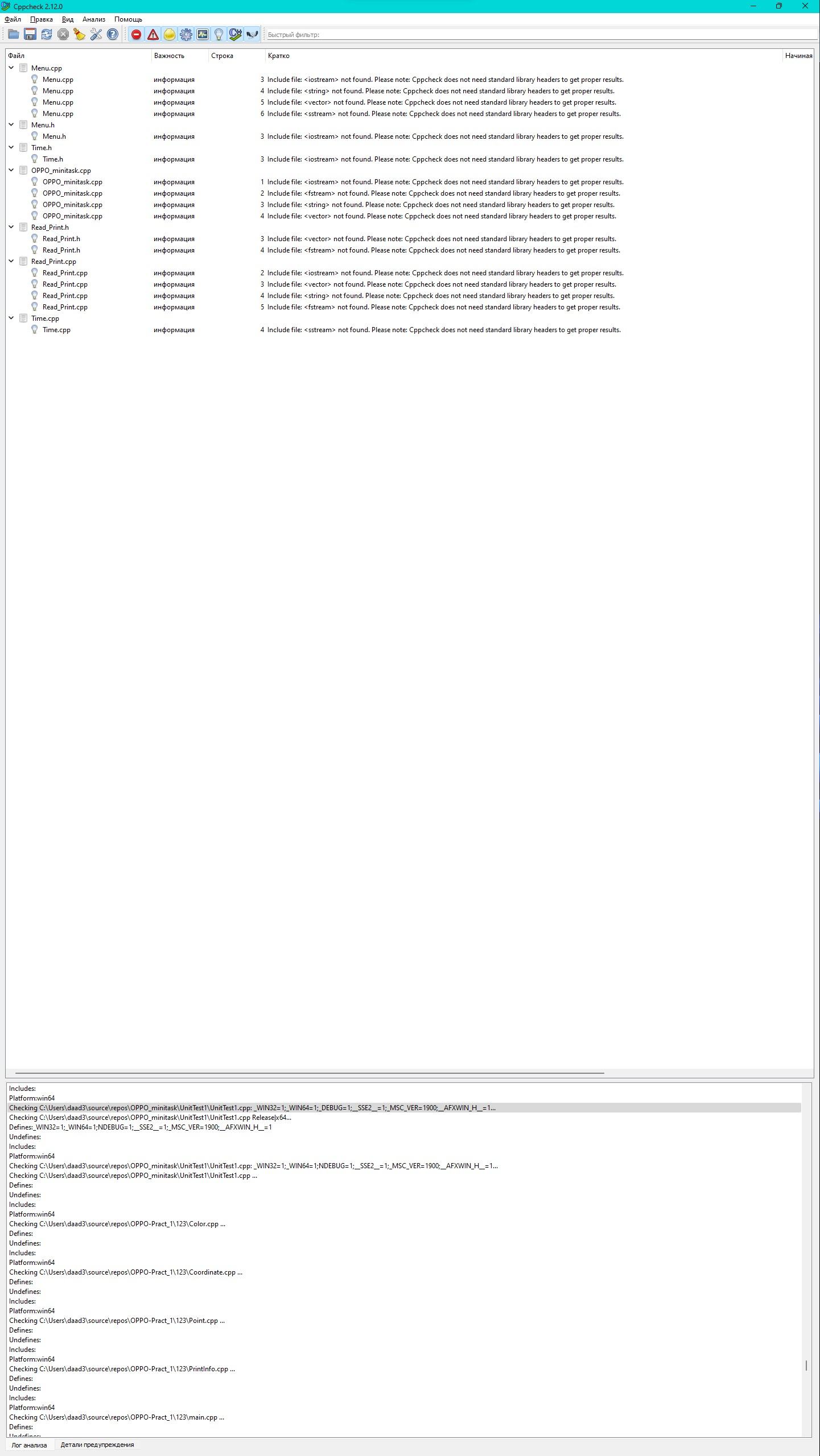


Рисунок 13 – Запуск анализа программы

Для динамического анализа я использовал библиотеку CRT (C RunTime Library). Для её подключения необходимо изменить код файла «Main.cpp» следующим образом:

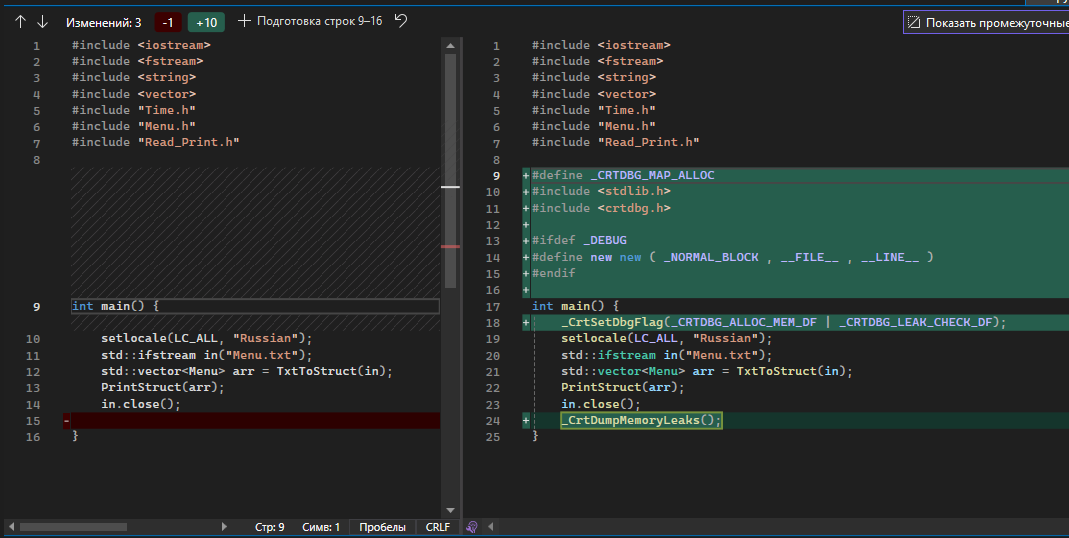


Рисунок 14 – Применение CRT

После запуска программы получим данные об утечке памяти:

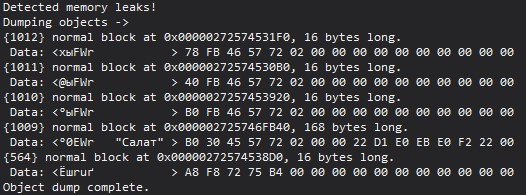


Рисунок 15 – Утечки памяти

**Заключение**

В ходе выполнения практической работы "Инструментальные средства анализа кода программ", была проведена всесторонняя оценка программного продукта, полученного в результате работы 3. Анализ охватывал использование как статических, так и динамических инструментов для проверки исходного кода, что позволило выявить ряд недостатков, влияющих на качество конечного ПО.

Среди основных этапов работы были:

Статический анализ кода:

Применение инструмента статического анализа «cppcheck» выявило синтаксические ошибки, проблемы со стилем кодирования и потенциальные утечки памяти. Это позволило устранить множество недочетов на раннем этапе разработки, улучшив читаемость и поддерживаемость кода.

Динамический анализ кода:

Использование инструментов динамического анализа, обеспечило обнаружение утечек памяти и других ошибок, которые могут возникнуть только во время выполнения программы. Это помогло оптимизировать управление ресурсами и повысить стабильность ПО.

Оценка качества ПО:

После исправления обнаруженных ошибок была проведена оценка программы на соответствие критериям качества, включая надежность, производительность и безопасность. Исправление выявленных проблем позволило повысить общее качество программы и снизить потенциальный риск ошибок в будущем.

Улучшение программы:

На основе результатов анализа были внесены соответствующие изменения в код, включая рефакторинг и оптимизацию. Были устранены все известные недочеты, что повысило уровень стабильности и надежности программного обеспечения.

В результате проведенной работы, программа была значительно улучшена. Статический и динамический анализы помогли выявить и устранить множество проблем, которые могли бы негативно сказаться на работе ПО в продуктивной среде. Подход, включающий использование различных инструментов анализа, показал свою эффективность и должен быть рекомендован для применения в процессе разработки и тестирования программного обеспечения.