ISE TOOL PROJECT 2025

Introduction

Our aim to to create a Code Review Checklist to help developers create better quality code which is maintainable. We are showing refactoring suggestions to improve code quality by detecting code smells. We are also finding the code complexity and displaying the heatmap. Additionally we are finding code emotions and showing dynamic emoji-based feedback.

Dependencies

Make sure Python and Lizard are installed

1. This extension uses Lizard for complexity analysis: pip install lizard

2. To download python: Pip3 install lizard

Complexity & Heatmap feature won't run on any system if the lizard path dependency is not resolved.

You can download the latest version of Python from the official website: python.org/downloads

Using the Extension

1. Search to find our Extension on VS Code Extensions:

Code Review Helper or Code Review IIT

1. Click on INSTALL Extension

2. To run and use extension on a Particular File:

Ctrl + Shift + P

Features

1. CodeEmotion

This feature provides an interactive and visual way to enhance the C/C++ code analysis by adding emoji decorations based on specific code patterns. It helps highlight common issues, patterns, and optimizations with fun and informative emojis!

Code Emotion Features

1. Code Pattern Emojis: Adds emojis to indicate patterns such as too many if statements, nested loops, large switch statements, and more.

2. Missing Semicolons Detection: Flags missing semicolons at the end of statements in C/C++ code.

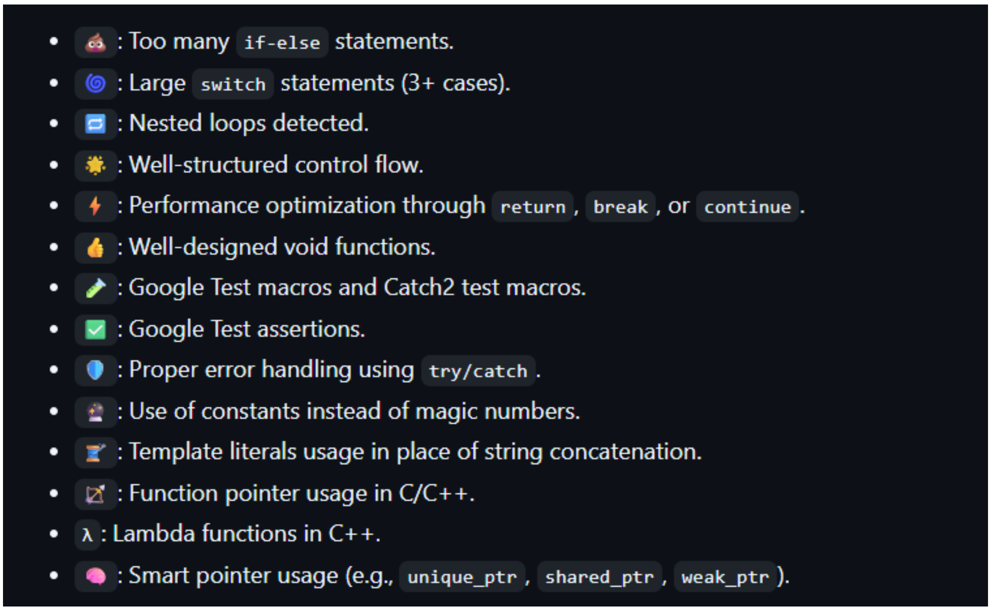
3. Trailing Whitespace Detection: Flags trailing whitespaces in your code to help maintain cleaner formatting.

4. Well-Structured Code: Highlights well-structured control flows like if, for, and while statements, and encourages the use of modern C++ constructs like unique_ptr and shared_ptr.

5. Clear Comments: Adds emojis to single-line comments to remind developers to maintain and update comments when code changes.

Emoji Patterns

The extension detects and decorates your code based on various patterns:



- 💩 : Too many `if-else` statements.
- 🌀 : Large `switch` statements (3+ cases).
- 🔁 : Nested loops detected.
- ☀️ : Well-structured control flow.
- ⚡ : Performance optimization through `return`, `break`, or `continue`.
- 👍 : Well-designed void functions.
- 🧪 : Google Test macros and Catch2 test macros.
- ✅ : Google Test assertions.
- 🛡️ : Proper error handling using `try/catch`.
- 🔮 : Use of constants instead of magic numbers.
- 🧵 : Template literals usage in place of string concatenation.
- 🏹 : Function pointer usage in C/C++.
- λ : Lambda functions in C++.
- 🧠 : Smart pointer usage (e.g., `unique_ptr`, `shared_ptr`, `weak_ptr`).

Supported Languages

This feature is designed specifically for C and C++ files.

1. C

2. C++

2. Cyclomatic Complexity Heatmap

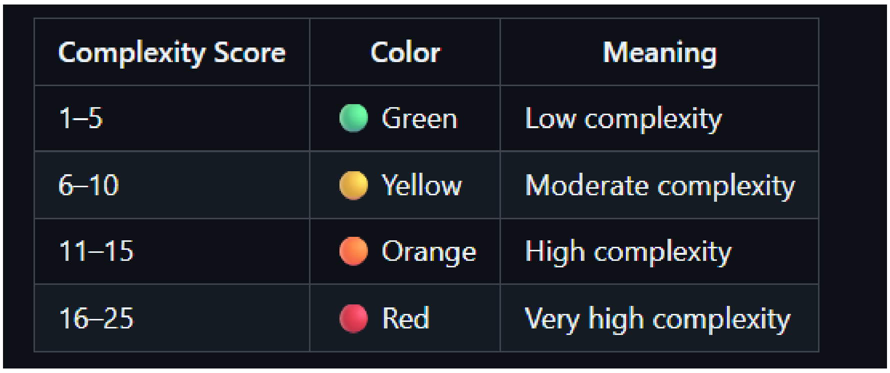
This feature helps developers analyze code complexity and improve code quality with a visual heatmap overlay.

Complexity Features

1. Cyclomatic Complexity Analysis using Lizard
2. Color-coded Heatmap overlay on functions based on complexity
3. Toggle Heatmap ON/OFF dynamically via command
4. Function-level Analysis Panel with webview showing detailed complexity info

Heatmap Colors

The heatmap uses a gradient from green (low complexity) to red (high complexity):



Complexity Score	Color	Meaning
1–5	🟢 Green	Low complexity
6–10	🟡 Yellow	Moderate complexity
11–15	🟠 Orange	High complexity
16–25	🔴 Red	Very high complexity

How Heatmap Toggle Works

When toggled ON, the extension:

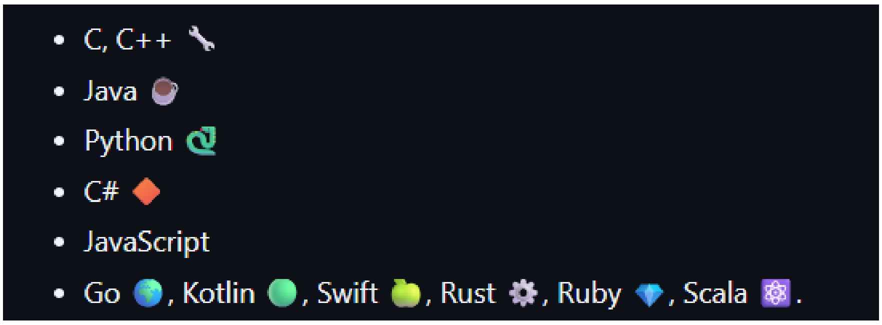
1. Clears any previous decorations
2. Runs Lizard analysis again
3. Applies new heatmap overlays based on updated complexity

When toggled OFF, it clears the decorations without deleting the stored analysis.

When switching between files, the heatmap auto-applies if it was visible previously.

Supported Languages

Works with major languages supported by Lizard, including:



- C, C++ 🔧
- Java ☕
- Python 🐍
- C# 🔶
- JavaScript
- Go 🌍, Kotlin 🟢, Swift 🍏, Rust ⚙️, Ruby 💎, Scala ⚛️.

3. Refactoring Suggestions

This feature helps developers improve the maintainability and readability of their C/C++ code by automatically detecting common code smells using regex-based pattern matching and suggesting actionable refactoring hints.

Refactor Features

1. Code Smell Detection: Scans code for patterns like long methods, deep nesting, large parameter lists, and magic numbers.

2. Regex-Powered Analysis: Uses regular expressions to extract and match patterns that indicate refactoring opportunities.

3. Side Panel Suggestions: Displays suggestions in a collapsible sidebar with issue count, explanation, and recommended actions.

4. One-click Refresh: The suggestions update dynamically via a Refresh button for real-time analysis after each change.

5. Integration with Webview: All refactoring suggestions are shown in a rich, styled panel within the editor using VS Code Webview.

Detected Code Smells

Here are some of the key patterns identified:

Pattern Detected Explanation

Long functions - Functions with too many lines, suggesting modularization

Magic numbers - Hardcoded numeric values should be replaced with named constants

Deep nesting - If/Else or loops nested more than 2 levelsâ€”recommend simplification

Large parameter lists - Functions with >3 parametersâ€”suggest grouping or using a struct

Repeated code blocks - Duplicate code logic detectedâ€—recommend creating helper functions

Switch without default switch cases missing default handlingâ€”can lead to missed conditions

How It Works

1. On triggering the Analyze Refactor button via the Command Palette or side button, the extension:

2. Extracts all function definitions.

3. Applies multiple regex rules on the function body and surrounding code.

4. Flags issues and sends the list to a Webview Panel.

5. Each suggestion includes:

6. The location (line number)

7. A reason why it's considered a code smell

8. A clear recommendation for improvement

Example Suggestions (Displayed in Webview)

Function `processData()` is 54 lines long. Consider breaking it down.

Magic number `42` found in function `calculateTotal()`. Use named constant.

Nested loop depth is 3 in function `parseResponse()`. Try to simplify.

Supported Languages

This feature is designed specifically for C and C++ files.

1. C

2. C++

Team Member Contributions

Sai Jagadeesh (CS24M101)

1. Dynamic Code Complexity Analysis
2. Dynamic Toggle heatmap
3. Color to Complexity mapping
4. Maintaining heatmap state for multiple files

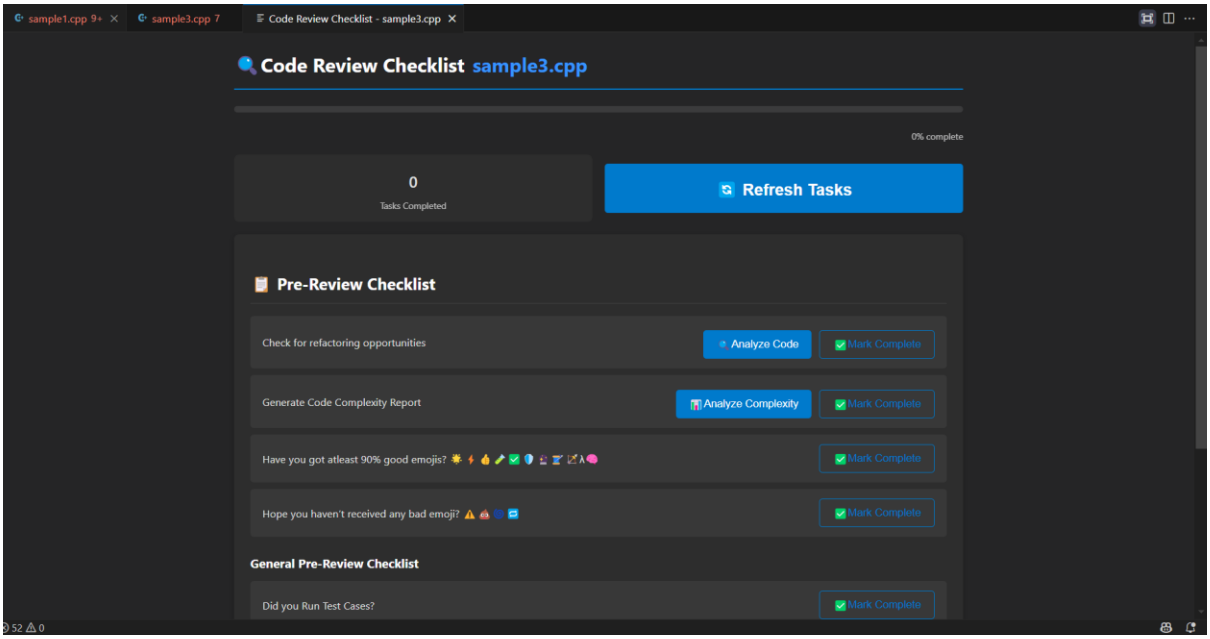
Yashraj Motwani (CS24M104)

1. Dynamic Code Emotions
2. Feature Integration
3. Webview Panel and UI
4. Progress bar and Tasks handling

Tejas Meshram (CS24M108)

1. Refactoring Suggestions using Regex
2. Refresh button for Tasks
3. Add to VS Code Marketplace

ScreenShots:



🔍 Code Review Checklist sample3.cpp

0

Tasks Completed

📋 Pre-Review Checklist

Check for refactoring opportunities

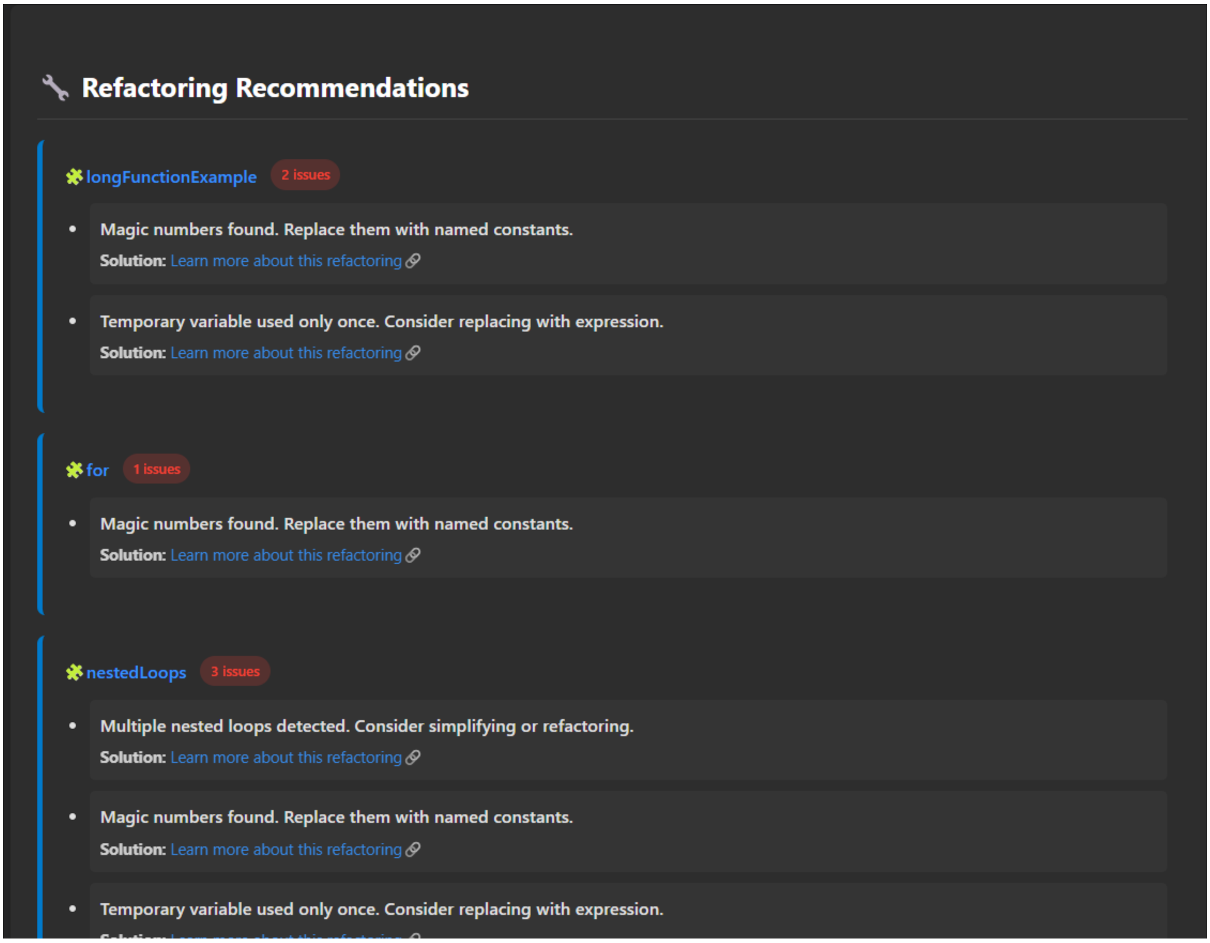
Generate Code Complexity Report

Have you got atleast 90% good emojis? 🌟 ⚡ 👍 🧪 ✅ 🛡️ 🔮 🧵 📈 λ 🧠

Hope you haven't received any bad emoji? ⚠️ 💩 🔵 🔁

General Pre-Review Checklist

Did you Run Test Cases?



🔧 Refactoring Recommendations

🧩 longFunctionExample 2 issues

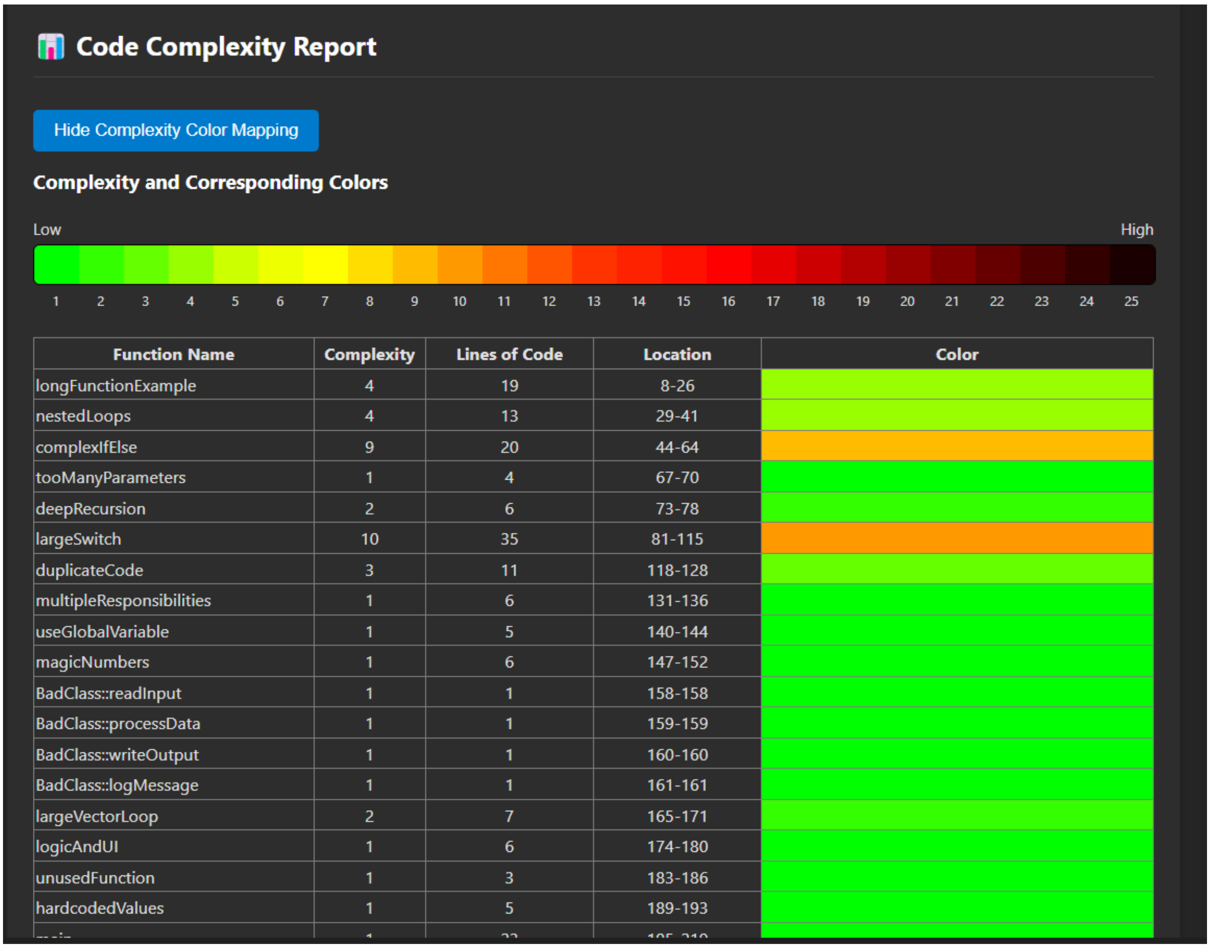
- Magic numbers found. Replace them with named constants.
  Solution: Learn more about this refactoring 🔗
- Temporary variable used only once. Consider replacing with expression.
  Solution: Learn more about this refactoring 🔗

🧩 for 1 issues

- Magic numbers found. Replace them with named constants.
  Solution: Learn more about this refactoring 🔗

🧩 nestedLoops 3 issues

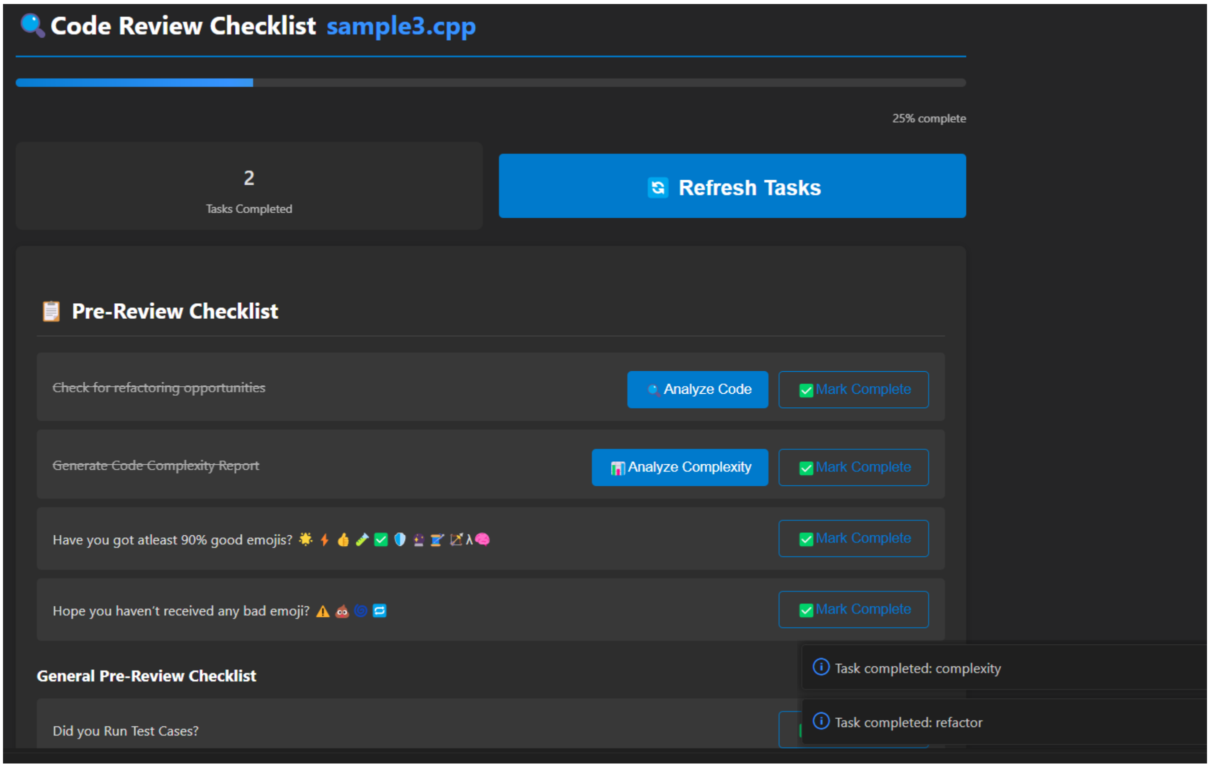
- Multiple nested loops detected. Consider simplifying or refactoring.
  Solution: Learn more about this refactoring 🔗
- Magic numbers found. Replace them with named constants.
  Solution: Learn more about this refactoring 🔗
- Temporary variable used only once. Consider replacing with expression.
  Solution: Learn more about this refactoring 🔗



Code Complexity Report

Complexity and Corresponding Colors

Function Name	Complexity	Lines of Code	Location	Color
longFunctionExample	4	19	8-26	🟩
nestedLoops	4	13	29-41	🟩
complexIfElse	9	20	44-64	🟧
tooManyParameters	1	4	67-70	🟩
deepRecursion	2	6	73-78	🟩
largeSwitch	10	35	81-115	🟧
duplicateCode	3	11	118-128	🟩
multipleResponsibilities	1	6	131-136	🟩
useGlobalVariable	1	5	140-144	🟩
magicNumbers	1	6	147-152	🟩
BadClass::readInput	1	1	158-158	🟩
BadClass::processData	1	1	159-159	🟩
BadClass::writeOutput	1	1	160-160	🟩
BadClass::logMessage	1	1	161-161	🟩
largeVectorLoop	2	7	165-171	🟩
logicAndUI	1	6	174-180	🟩
unusedFunction	1	3	183-186	🟩
hardcodedValues	1	5	189-193	🟩
main	1	22	195-210	🟩



🔍 Code Review Checklist sample3.cpp

Pre-Review Checklist

~~Check for refactoring opportunities~~

~~Generate Code Complexity Report~~

Have you got atleast 90% good emojis? 🌟 ⚡ 👍 🧪 ✅ 🛡️ 🔮 🧵 🏹 λ 🧠

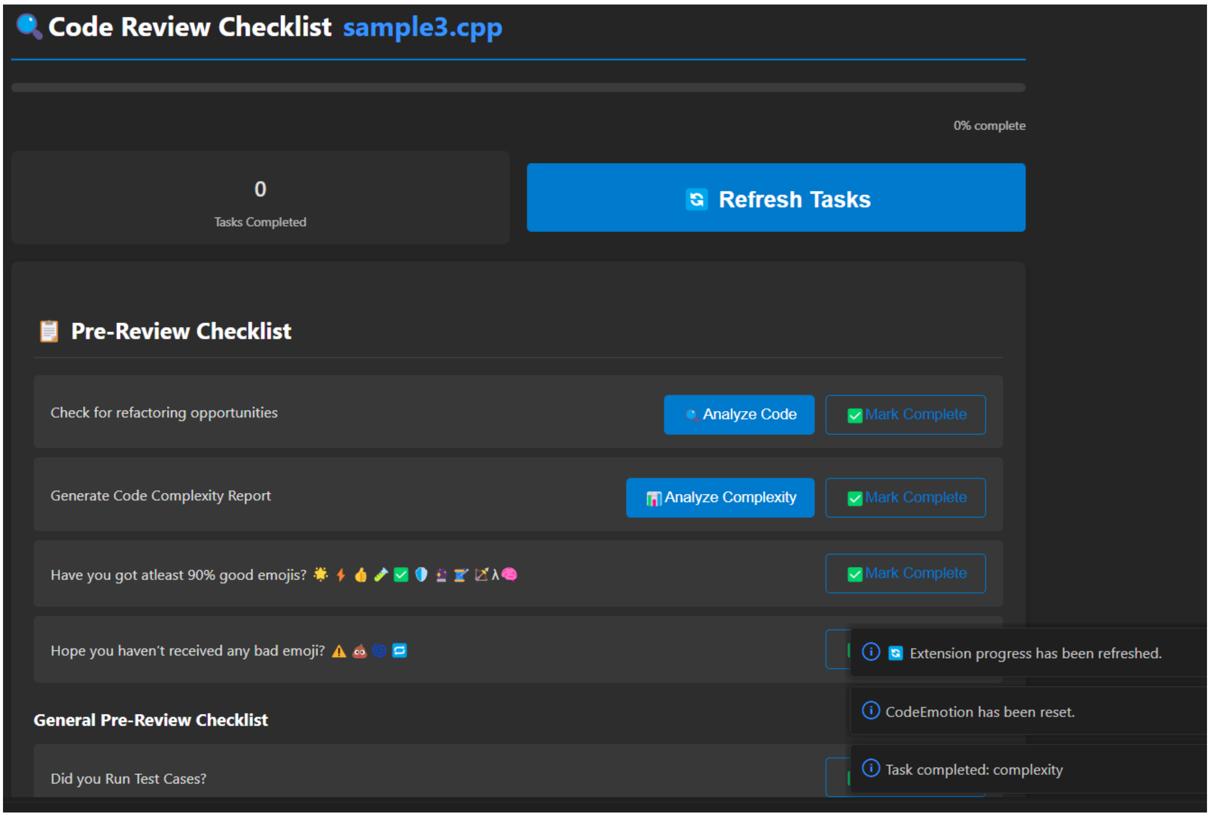
Hope you haven't received any bad emoji? ⚠️ 💩 🌀 🔁

General Pre-Review Checklist

Did you Run Test Cases?

Task completed: complexity

Task completed: refactor



🔍 Code Review Checklist sample3.cpp

0% complete

0
Tasks Completed

🔄 Refresh Tasks

📋 Pre-Review Checklist

Check for refactoring opportunities — 🔍 Analyze Code — ✅ Mark Complete

Generate Code Complexity Report — 📊 Analyze Complexity — ✅ Mark Complete

Have you got atleast 90% good emojis? 🌟 ⚡ 👍 🧪 ✅ 🛡️ 🔮 🧵 🏹 λ 🧠 — ✅ Mark Complete

Hope you haven't received any bad emoji? ⚠️ 💩 🌀 🔁

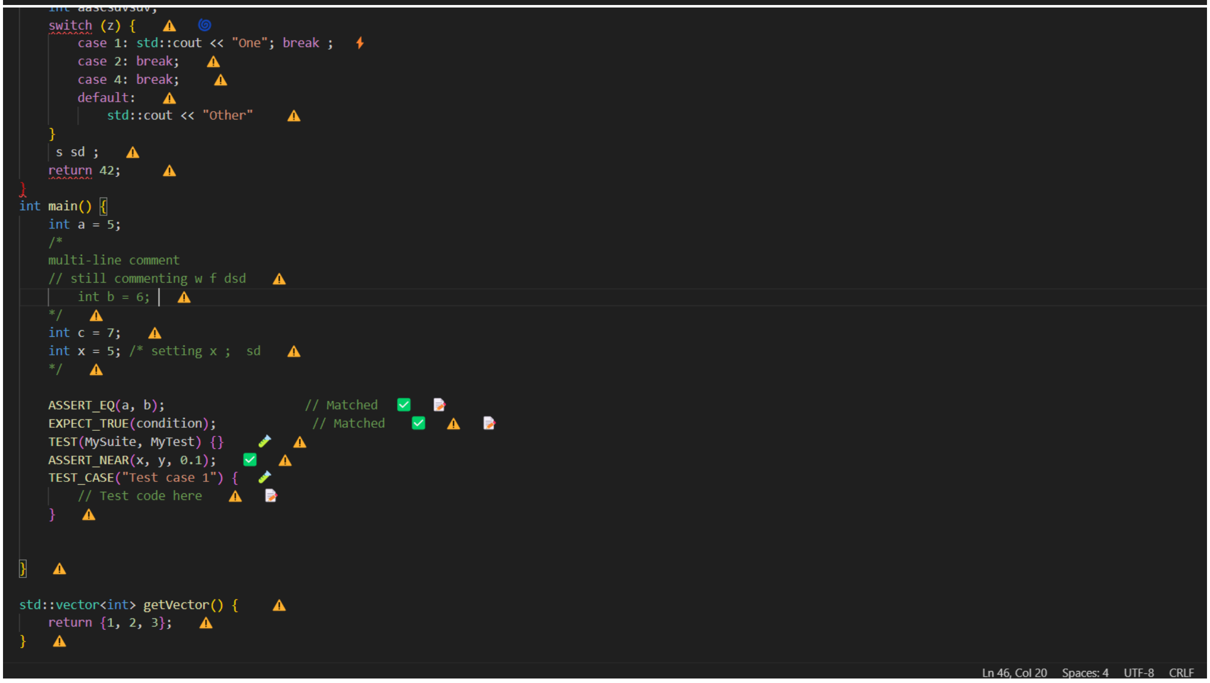
General Pre-Review Checklist

Did you Run Test Cases?

ⓘ 🔄 Extension progress has been refreshed.

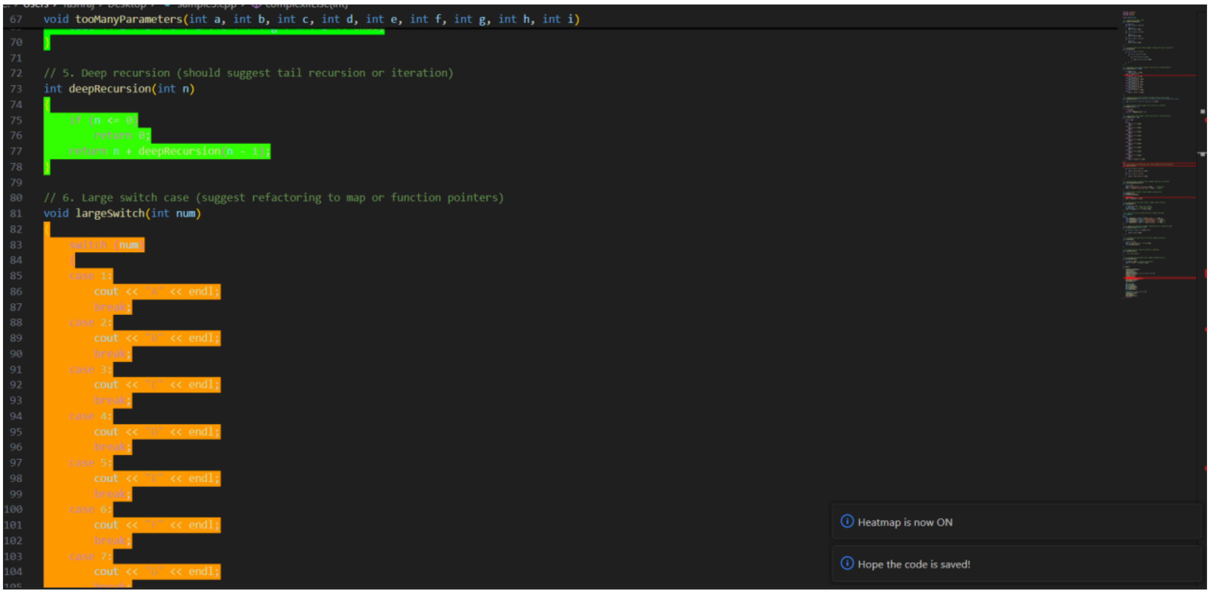
ⓘ CodeEmotion has been reset.

ⓘ Task completed: complexity



```
    int aascsdvsdv;
    switch (z) {    ⚠️   🌀
        case 1: std::cout << "One"; break ;   ⚡
        case 2: break;    ⚠️
        case 4: break;     ⚠️
        default:    ⚠️
            std::cout << "Other"     ⚠️
    }
     s sd ;    ⚠️
    return 42;      ⚠️
}
int main() {
    int a = 5;
    /*
    multi-line comment
    // still commenting w f dsd    ⚠️
        int b = 6;    ⚠️
    */    ⚠️
    int c = 7;    ⚠️
    int x = 5; /* setting x ;  sd    ⚠️
    */    ⚠️

    ASSERT_EQ(a, b);                   // Matched   ✅    📝
    EXPECT_TRUE(condition);             // Matched    ✅    ⚠️    📝
    TEST(MySuite, MyTest) {}     🧪    ⚠️
    ASSERT_NEAR(x, y, 0.1);    ✅    ⚠️
    TEST_CASE("Test case 1") {   🧪
        // Test code here    ⚠️    📝
    }    ⚠️


}    ⚠️

std::vector<int> getVector() {     ⚠️
    return {1, 2, 3};    ⚠️
}    ⚠️
```

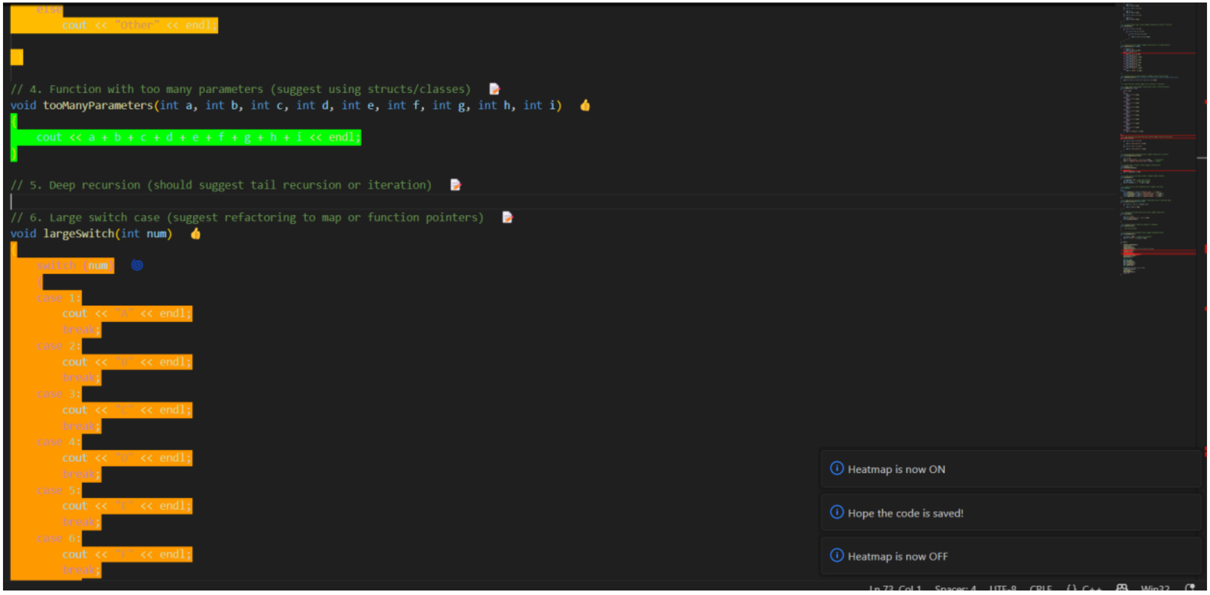
Ln 46, Col 20   Spaces: 4   UTF-8   CRLF



```cpp
void tooManyParameters(int a, int b, int c, int d, int e, int f, int g, int h, int i)
}

// 5. Deep recursion (should suggest tail recursion or iteration)
int deepRecursion(int n)
{
    if (n <= 0)
        return 0;
    return n + deepRecursion(n - 1);
}

// 6. Large switch case (suggest refactoring to map or function pointers)
void largeSwitch(int num)
{
    switch (num)
    {
    case 1:
        cout << "A" << endl;
        break;
    case 2:
        cout << "B" << endl;
        break;
    case 3:
        cout << "C" << endl;
        break;
    case 4:
        cout << "D" << endl;
        break;
    case 5:
        cout << "E" << endl;
        break;
    case 6:
        cout << "F" << endl;
        break;
    case 7:
        cout << "G" << endl;
```

Heatmap is now ON

Hope the code is saved!

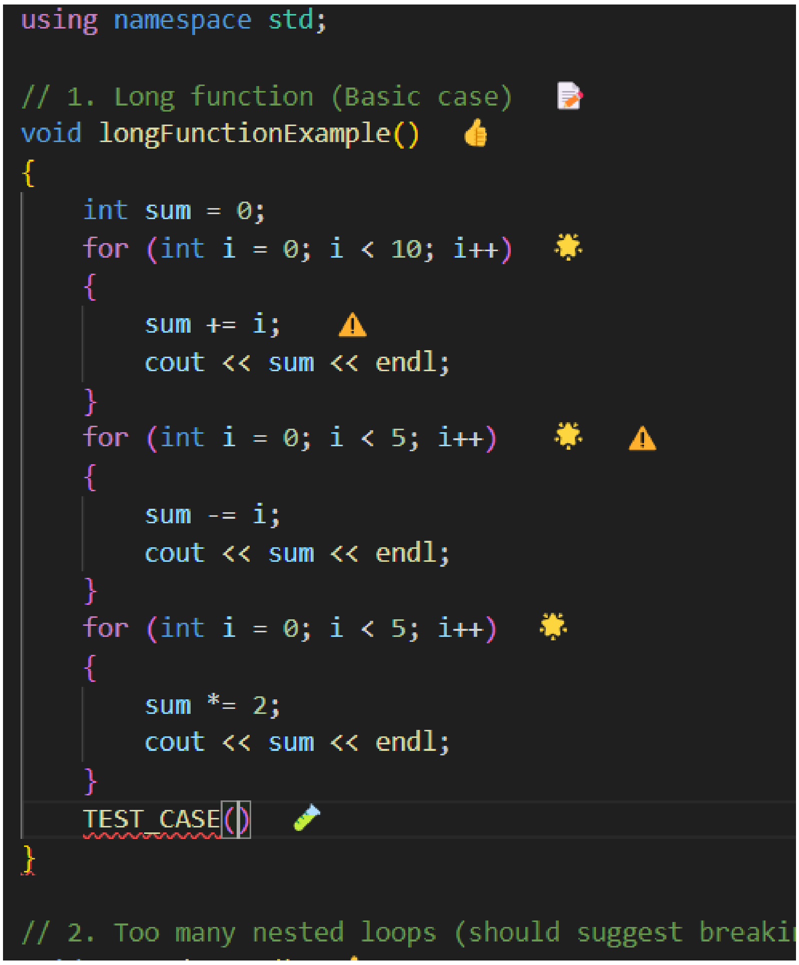


```cpp
    else
        cout << "Other" << endl;

}

// 4. Function with too many parameters (suggest using structs/classes)
void tooManyParameters(int a, int b, int c, int d, int e, int f, int g, int h, int i)
{
    cout << a + b + c + d + e + f + g + h + i << endl;
}

// 5. Deep recursion (should suggest tail recursion or iteration)

// 6. Large switch case (suggest refactoring to map or function pointers)
void largeSwitch(int num)
{
    switch (num)
    {
    case 1:
        cout << "A" << endl;
        break;
    case 2:
        cout << "B" << endl;
        break;
    case 3:
        cout << "C" << endl;
        break;
    case 4:
        cout << "D" << endl;
        break;
    case 5:
        cout << "E" << endl;
        break;
    case 6:
        cout << "F" << endl;
        break;
```

Heatmap is now ON

Hope the code is saved!

Heatmap is now OFF



```
using namespace std;

// 1. Long function (Basic case)   📝
void longFunctionExample()   👍
{
    int sum = 0;
    for (int i = 0; i < 10; i++)   🌟
    {
        sum += i;   ⚠
        cout << sum << endl;
    }
    for (int i = 0; i < 5; i++)   🌟   ⚠
    {
        sum -= i;
        cout << sum << endl;
    }
    for (int i = 0; i < 5; i++)   🌟
    {
        sum *= 2;
        cout << sum << endl;
    }
    TEST_CASE()   🧪
}

// 2. Too many nested loops (should suggest breaki
```