# AI Code Correction

```
// This is a sample code for a VSCode extension that uses an AI API to correct errors in C language

// The extension uses the OpenAI Codex API to generate suggestions for fixing syntax and semantic errors in C code

// The extension requires an OpenAI API key and a VSCode account to work

// The extension registers a command called "aiFix" that can be invoked by the user or by a keyboard shortcut

// The extension also adds a code action provider that can suggest fixes for errors detected by the C/C++ extension


// Import the required modules

const vscode = require('vscode');

const axios = require('axios');


// Define the OpenAI API key and the Codex endpoint

const OPENAI_API_KEY = "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"; // Replace with your own key

const CODEX_ENDPOINT = "https://api.openai.com/v1/engines/davinci-codex/completions";


// Define the Codex parameters

const CODEX_PARAMS = {

    "prompt": "", // The prompt will be dynamically generated based on the code and the error

    "max_tokens": 100, // The maximum number of tokens to generate

    "temperature": 0.1, // The randomness of the generation

    "stop": "\n\n" // The stop sequence to end the generation

};


// Define the Codex headers

const CODEX_HEADERS = {

    "Authorization": `Bearer ${OPENAI_API_KEY}`,

    "Content-Type": "application/json"

};
```

```
// Define the regex patterns to extract the error message and the line number from the C/C++ extension
const ERROR_PATTERN = /(.+?):(\d+):(\d+):\s+(error|warning):\s+(.+)/;
const LINE_NUMBER_PATTERN = /line (\d+)/;

// Define the command name
const COMMAND_NAME = "aiFix";

// Register the command
function activate(context) {
    let disposable = vscode.commands.registerCommand(COMMAND_NAME, async function () {
        // Get the active text editor and the document
        let editor = vscode.window.activeTextEditor;
        let document = editor.document;

        // Get the current selection or the whole document
        let selection = editor.selection;
        let code = document.getText(selection.isEmpty ? undefined : selection);

        // Get the diagnostics for the current document
        let diagnostics = vscode.languages.getDiagnostics(document.uri);

        // Check if there are any errors or warnings
        if (diagnostics.length === 0) {
            vscode.window.showInformationMessage("No errors or warnings found.");
            return;
        }

        // Get the first error or warning
        let diagnostic = diagnostics[0];
```

```
    // Get the error message and the line number
    let errorMessage = diagnostic.message;
    let lineNumber = diagnostic.range.start.line + 1;

    // Extract the error details from the error message using regex
    let errorMatch = errorMessage.match(ERROR_PATTERN);
    if (!errorMatch) {
        vscode.window.showErrorMessage("Could not parse the error message.");
        return;
    }

    // Get the error type, the error description, and the error location
    let errorType = errorMatch[4];
    let errorDescription = errorMatch[5];
    let errorLocation = errorMatch[1] + ":" + errorMatch[2] + ":" + errorMatch[3];

    // Generate the prompt for Codex based on the code and the error
    let prompt = `// This is a C code with a ${errorType} at ${errorLocation}:\n${code}\n// The ${errorType} message is: ${errorDescription}\n// Please fix the ${errorType} and return the corrected code:\n`;
    CODEX_PARAMS.prompt = prompt;

    // Call the Codex API with the prompt and the parameters
    try {
        let response = await axios.post(CODEX_ENDPOINT, CODEX_PARAMS, {headers: CODEX_HEADERS});
        let data = response.data;

        // Check if the response is valid and has a completion
        if (data && data.choices && data.choices.length > 0) {
            let choice = data.choices[0];
```

```
            // Get the generated text from the choice
            let text = choice.text;

            // Check if the text is not empty
            if (text) {
                // Replace the code with the generated text
                editor.edit(editBuilder => {
                    editBuilder.replace(selection.isEmpty ? new vscode.Range(0, 0, document.lineCount, 0) : selection, text);
                });

                // Show a success message
                vscode.window.showInformationMessage("The code has been fixed by Codex.");
            } else {
                // Show a failure message
                vscode.window.showErrorMessage("Codex could not fix the code.");
            }
        } else {
            // Show a failure message
            vscode.window.showErrorMessage("Codex did not return a valid response.");
        }
    } catch (error) {
        // Show a failure message
        vscode.window.showErrorMessage("Codex encountered an error: " + error.message);
    }
});

context.subscriptions.push(disposable);
}

// Add a code action provider that can suggest fixes for errors detected by the C/C++ extension
```

```
class AIFixProvider {
    provideCodeActions(document, range, context, token) {
        // Get the diagnostics for the current range
        let diagnostics = context.diagnostics;

        // Check if there are any errors or warnings
        if (diagnostics.length === 0) {
            return;
        }

        // Get the first error or warning
        let diagnostic = diagnostics[0];

        // Get the error message and the line number
        let errorMessage = diagnostic.message;
        let lineNumber = diagnostic.range.start.line + 1;

        // Extract the error details from the error message using regex
        let errorMatch = errorMessage.match(ERROR_PATTERN);
        if (!errorMatch) {
            return;
        }

        // Get the error type and the error description
        let errorType = errorMatch[4];
        let errorDescription = errorMatch[5];

        // Create a code action that can invoke the command
        let action = new vscode.CodeAction(`Fix ${errorType} with Codex`,
vscode.CodeActionKind.QuickFix);
        action.command = {
```

```
            command: COMMAND_NAME,
            title: `Fix ${errorType} with Codex`
        };
        action.diagnostics = [diagnostic];
        action.isPreferred = true;

        // Return the code action
        return [action];
    }
}

// Register the code action provider for C and C++ files
vscode.languages.registerCodeActionsProvider(["c", "cpp"], new AIFixProvider());

// Export the activate function
exports.activate = activate;
```

x

This is a possible code for a VSCode extension that uses an AI API to correct errors in C language. The code is based on the OpenAI Codex API[1], which is an improved version of OpenAI Codex, an AI system that translates natural language to code[2]. The code also uses the axios module to make HTTP requests to the Codex API, and the vscode module to interact with the VSCode editor. The code comments explain the main logic and functionality of the extension.