# Morse Code Generator

Internal - General Use

# Introduction

The Morse Code Generator is an application that automatically converts text into Morse code, a method of encoding characters as sequences of dots and dashes. With the assistance of OpenAI's Copilot, we can streamline the development process by utilizing its code suggestions and auto-completion capabilities.

It is a simple program designed to convert text into Morse code. Morse code is a method of encoding text characters as sequences of two different signal durations, called dots and dashes or Dits and dahs. It was widely used in telecommunication before the advent of modern digital communication systems.

The Morse Code Generator program automates the process of converting plain text into Morse code representations, allowing users to quickly and easily encode messages in this historic encoding scheme.

Internal - General Use

# Setting Up

To get started, make sure you have Visual Studio Code installed with the Copilot extension. You'll also need to have a GitHub Copilot subscription to access its features.

- **Install Visual Studio Code**: Download and install Visual Studio Code from the official website: https://code.visualstudio.com/

- **Install Copilot Extension**: Open Visual Studio Code, go to the Extensions marketplace, and search for "GitHub Copilot." Install the extension developed by GitHub.

- **GitHub Copilot Subscription**: Make sure you have a subscription to GitHub Copilot. You'll need to link your GitHub account to access its features.

# Creating the Morse Code Generator

In this section, we'll create a Morse code generator using Copilot. We'll use the basic structure provided by Copilot and enhance it with the Morse code conversion logic.

- **Open Visual Studio Code**.

- **Create a New File**: Create a new C++ file with the extension **.cpp**.

- **Initial Code**: Start by typing a simple comment at the top of the file to describe the program's purpose.

  **// write a program that converts characters to Morse code.**

1. **Include Necessary Headers:**

Copilot assists us in including the necessary headers. Begin typing **#inc** and Copilot autocomplete the header inclusion for us.

```
#include <iostream>      // Include the input-output stream library
#include <string>        // Include the string library
#include <algorithm>     // Include the algorithm library for using transform and other operations
```

2. **Create Morse Code:**

This section defines a function named **get_morse_code** that takes a character **c** as input and returns its Morse code representation.

The function uses a switch statement to map individual characters to their respective Morse code representations, converting the character to uppercase using **toupper(c)** before comparison.

Cases are defined for letters A to Z, digits 0 to 9, and a few special characters like space and '@'. If the character doesn't match any of the defined cases, "invalid" is returned.

```
// Function to convert a character to its Morse code representation
string get_morse_code(char c) {
    // Switch statement maps characters to their Morse code
    switch (toupper(c)) {
  // Convert the character to uppercase before comparison
 // Cases for individual characters and their corresponding Morse code
        case 'A': return ".-";
        case 'B': return "-...";
        case 'C': return "-.-.";

        // ... (similar cases for the rest of the characters)

        case ' ': return "/";  // Using "/" to represent space in Morse code
        default: return "invalid";  // Return "invalid" for unsupported characters
    }
}
```

3. **Main Function**:

Finally, create the **main** function. Begin typing the **main** function structure and Copilot will help us to complete it.

```
int main() {
    string text;
    while (true) {
        // Display menu and get user choice
        // ...
        if (choice == 1) {  // Convert text to Morse code
            // Get user input text
            // Convert each character to Morse code
            // Display Morse code
        } else if (choice == 2) {  // Quit
            // Inform the user and exit the loop
        } else {
            // Handle invalid choices
        }
    }
    return 0;
}
```

Internal - General Use

# Key Components

**Header Inclusions**: The necessary libraries are included:

**<iostream>**: Input-output stream library.
**<string>**: String manipulation library.
**<algorithm>**: Used for character transformations.

**Function get_morse_code**: This function takes a character as input and returns its corresponding Morse code representation.
It uses a switch statement to map characters to Morse code sequences.

**Main Function**: The main program execution starts here. It includes:

- A loop that repeatedly displays a menu to the user.
- User input for their choice (convert text or quit).
- Branches to handle user choices:

  Choice 1: Convert text to Morse code.
  Choice 2: Exit the program.
  Invalid choices: Handle errors.

# Usage

- **Menu**: The program presents a menu to the user:

  Option 1: Convert text to Morse code.
  Option 2: Quit the program.

- **Conversion**: If the user selects option 1, they are prompted to enter the text they want to convert to Morse code. The program then iterates through each character of the input text, converts it to Morse code using the **get_morse_code** function, and builds the Morse code string.

- **Display**: The generated Morse code is displayed to the user.
- **Exit**: If the user selects option 2, the program informs the user that it's exiting and terminates.

Internal - General Use

# Conclusion

The Morse code generator program provided is a simple illustration of how Morse code can be generated from user input using C++.

This concept can be expanded upon, such as by adding more special character cases, improving user interface design, or implementing additional features. Morse code, despite its historical origins, continues to captivate people's interest and can serve as a starting point for various creative applications.

In this document, we explored how to create a Morse code generator using GitHub Copilot in C++. Copilot provided valuable assistance by suggesting headers, completing code structures, and generating character mappings and functions.

This demonstrates the power of AI-assisted programming in enhancing your development process. As you further explore Copilot, you can apply similar techniques to more complex projects and improve your coding efficiency.

The Morse Code Generator created using Copilot showcases how AI-powered code assistance can simplify the development process. By leveraging Copilot's auto-completion, suggestions, and insights, developers can efficiently build applications like the Morse Code Generator with enhanced accuracy and productivity.

Internal - General Use