Code Structure, Logic, and Techniques:

1.Code Structure:

- The code is organized into a Program class, with a nested BasicResponseSystem class.
- BasicResponseSystem encapsulates the chatbot's response logic.
- Main handles the program's execution flow, including file operations, user input, and output.

- **2.Logic:**
- **Initialization:**
- The program reads ASCII art from a file.
- It prompts the user for their name.
- It displays a list of example questions.
- **Chatbot Loop:**
- The program enters a while (true) loop to continuously accept user input.
- It reads the user's input using Console.ReadLine().
- If the input is empty, the loop breaks.
- It uses the BasicResponseSystem.GetResponse() method to get the chatbot's response.
- It formats and displays the response using Console.ForegroundColor , Console.WriteLine() , and the TypeWriterEffect() method.

- **End of Conversation:**
- After the loop ends, it displays an "End of Conversation" header.
- It prompts the user to press any key to exit.
- **Exception Handling:**
- A try-catch block handles potential file I/O errors and other exceptions.

3.Techniques:

- **Response System:**
- A Dictionary<string, string> is used to store the chatbot's responses.
- The StringComparer.OrdinalIgnoreCase ensures case-insensitive matching of user input.
- The responses.TryGetValue() method efficiently retrieves responses.

- **Formatting:**
- Console.ForegroundColor is used to change the text color for user input, chatbot responses, and headers, improving readability.
- Console.WriteLine() and Console.Write() are used to control console output.
- DisplayDivider() creates horizontal lines to separate input and output.
- DisplayHeader() creates formatted headers.
- TypeWriterEffect() simulates a typewriter effect by displaying text character by character with a delay.
- **File Handling:**
- File.ReadAllLines() reads the ASCII art from a text file.
- Path.Combine() is used to construct the file path safely.
- try-catch blocks handle file I/O exceptions.

- **File Handling:**
- File.ReadAllLines() reads the ASCII art from a text file.
- Path.Combine() is used to construct the file path safely.
- try-catch blocks handle file I/O exceptions.
- **User Input:**
- Console.ReadLine() reads user input from the console.
- string.IsNullOrWhiteSpace() checks for empty or whitespace-only input.
- **Looping:**
- A while (true) loop is used to create the interactive chatbot conversation.
- **Threading:**
- Thread.Sleep() is used inside the TypeWriterEffect() method to introduce delays.

- **Voice Integration:**
- This code *does not* include any voice integration. To add voice capabilities, you would need to use libraries like:
  - **System.Speech:** (Windows-specific) For text-to-speech (TTS) and speech recognition.
  - **Microsoft.CognitiveServices.Speech:** (Cross-platform, Azure Cognitive Services) More advanced TTS and speech recognition.
  - 3rd party libraries such as NAudio.
- Voice integration would involve:
  - Using a speech recognition library to convert spoken input to text.
  - Passing the converted text to the BasicResponseSystem.GetResponse() method.
  - Using a TTS library to convert the chatbot's text response to speech.
  - Playing the generated speech through the computer's speakers.

- Example of conceptual voice integration:
- //Conceptual example, requires external libraries.
- //string userInput = GetVoiceInput(); // Function that gets voice input and returns a string
- //string response = responseSystem.GetResponse(userInput);
- //SpeakResponse(response); //Function that speaks the response.