C# Fundamentals

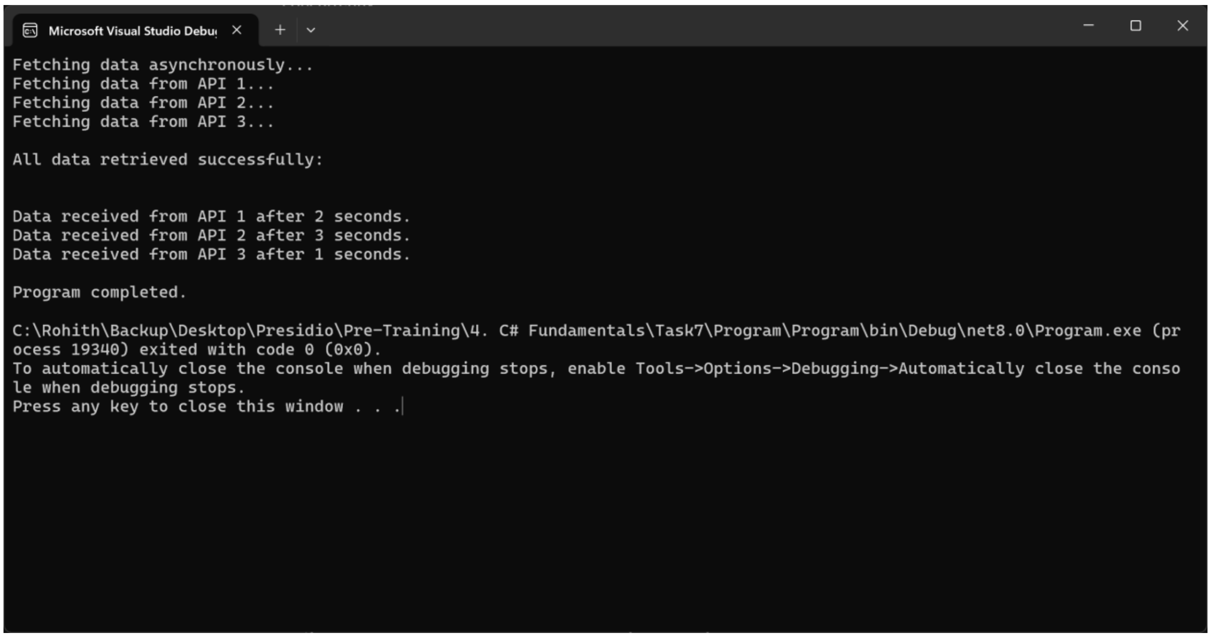
7. **Asynchronous Programming and Multi-threadingObjective:Requirements:**
   - **Develop a console application that performs multiple asynchronous operations concurrently.**
   - **Use async and await to fetch data from multiple simulated sources (e.g., using Task.Delay to mimic API calls).**
   - **Aggregate the results once all tasks are complete.**
   - **Handle exceptions that may occur during asynchronous operations.**

Program code:

```
using System;
using System.Threading.Tasks;


class Program
{
   // Declares the Main() method as async so we can use await inside it.
   // Task represents an asynchronous operation.
   static async Task Main()
   {
      Console.WriteLine("Fetching data asynchronously...");


      try
      {
         // Start fetching multiple data sources asynchronously
         // instead of waiting for one task to complete before starting the next, all tasks start
simultaneously
         Task<string> task1 = FetchDataFromSourceAsync("API 1", 2000);
         Task<string> task2 = FetchDataFromSourceAsync("API 2", 3000);
         Task<string> task3 = FetchDataFromSourceAsync("API 3", 1000);


         // Task.WhenAll() runs all tasks in parallel and waits until all are finished
```

```
            // Returns a Task that completes when all provided tasks finish
            string[] results = await Task.WhenAll(task1, task2, task3);

            // Display aggregated results
            Console.WriteLine("\nAll data retrieved successfully:\n\n");
            foreach (var result in results)
            {
                Console.WriteLine(result);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine("\nProgram completed.");
    }

    // Simulated async method to fetch data
    static async Task<string> FetchDataFromSourceAsync(string source, int delay)
    {
        Console.WriteLine($"Fetching data from {source}...");
        await Task.Delay(delay); // Simulating network delay
        return $"Data received from {source} after {delay / 1000} seconds.";
    }
}

// A Task in C# represents an asynchronous operation that may complete in the future.
```

Output:



```
Fetching data asynchronously...
Fetching data from API 1...
Fetching data from API 2...
Fetching data from API 3...

All data retrieved successfully:


Data received from API 1 after 2 seconds.
Data received from API 2 after 3 seconds.
Data received from API 3 after 1 seconds.

Program completed.

C:\Rohith\Backup\Desktop\Presidio\Pre-Training\4. C# Fundamentals\Task7\Program\Program\bin\Debug\net8.0\Program.exe (pr
ocess 19340) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```