**Practical 11**

**AIM:**

Write a program that demonstrates asynchronous behavior using a callback function. For example, create a function that simulates fetching data from an API and invokes a callback with the fetched data.

**CODE:**

function fetchData(callback) {

  setTimeout(() => {

    const data = { message: "Data fetched successfully!" };

    callback(data);

  }, 2000);

}

function handleData(data) {

  console.log(data.message);

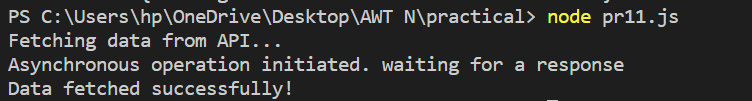
}

console.log("Fetching data from API...");

fetchData(handleData);

console.log("Asynchronous operation initiated. waiting for a response");

**OUTPUT:**



**Practical 12**

**AIM:**

Create a program that reads a file asynchronously using callbacks and displays its contents.

**CODE:**

const fs = require("fs");

function readFileAsync(filePath, callback) {

  fs.readFile(filePath, "utf8", (err, data) => {

    if (err) {

      callback(err);

      return;

    }

    callback(null, data);

  });

}

function FileContents(err, data) {

  if (err) {

    console.error(err);

  } else {

    console.log(data);

  }

}

const filePath = "example.txt";

console.log("reading file");

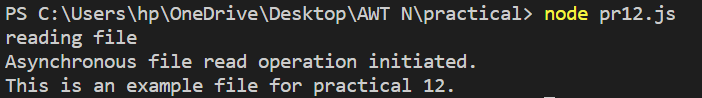
readFileAsync(filePath, FileContents);

console.log("Asynchronous file read operation initiated.");

**example.txt**

This is an example file for practical 12.

**OUTPUT:**



**Practical 13**

**AIM:**

Write a program that uses Promises to handle asynchronous operations. For example, create a function that returns a Promise to fetch data from an API and resolve it with the fetched data.

Implement error handling using Promises by rejecting a Promise with an error message in case of failure.

**CODE:**

const fetchFromAPI = () => {

  return new Promise((resolve, reject) => {

    const success = true;

    setTimeout(() => {

      if (success) {

        const data = { message: "Data fetched from the API" };

        resolve(data);

      } else {

        reject(new Error("Failed to fetch data from the API"));

      }

    }, 2000);

  });

};

fetchFromAPI()

  .then((data) => {

    console.log("API call successfully", data);

  })

  .catch((error) => {

    console.error("API call failed:", error.message);

  });

**OUTPUT:**

****

**Practical 14**

**AIM:**

Convert a Promise-based asynchronous function into an async/await style function. For example, rewrite a function that fetches data from an API using async/await.

Write a program that utilizes multiple async/await functions to fetch data from different APIs sequentially and display the combined results.

**CODE:**

const firstAPI = () => {

  return new Promise((resolve, reject) => {

    const success = true;

    setTimeout(() => {

      if (success) {

        const data = { message: "Data fetched from first API" };

        resolve(data);

      } else {

        reject(new Error("Failed to fetch data from first API"));

      }

    }, 2000);

  });

};

const secondAPI = () => {

  return new Promise((resolve, reject) => {

    const success = true;

    setTimeout(() => {

      if (success) {

        const data = { message: "Data fetched from second API" };

        resolve(data);

      } else {

        reject(new Error("Failed to fetch data from second API"));

      }

    }, 1500);

  });

};

const fetchDataSequentially = async () => {

  try {

    const apiData = await firstAPI();

    console.log("API 1 data:", apiData);

    const secondApi = await secondAPI();

    console.log("API 2 data:", secondApi);

  } catch (error) {

    console.error("Error:", error.message);

  }

};

fetchDataSequentially();

**OUTPUT:**

