**Programming Language Practical Project**

**Phase 4**

Nouraldin Hassan

Contents

[Evidence of Learning 3](#_Toc179647756)

[Program Architecture 7](#_Toc179647757)

[Program Demonstration via Screen Shots 9](#_Toc179647758)

[Unit Testing Demonstration via Screen Shots 10](#_Toc179647759)

[Source Code Commenting Example 10](#_Toc179647760)

# Evidence of Learning

**NOTICE:** program.js is the only file significantly changed.

**Variables (and use of API Libraries, Arrays):**

program.js:

302 const chartData = records.map(record => ({  
303 id: record.id,  // Assuming the record ID is stored in the 'id' field  
304 x: record[xAxis],  
305 y: parseFloat(record[yAxis]) || 0  
306 }));

chartData is declared for the mapped records to then be utilized with the sort function for ordering content in the data by value or id.

**Snippet: function generateChart(chartData):**

program.js:  
334 function generateChart(chartData) {  
335     const maxBarLength = 50; // Adjust as needed for chart width  
336     console.log('Horizontal Bar Chart:');  
337     chartData.forEach(dataPoint => {  
338         const barLength = Math.round(dataPoint.y / Math.max(...chartData.map(d => d.y)) \* maxBarLength);  
339         const bar = '█'.repeat(barLength);  // Use a character for the bar (e.g., "█")  
340         console.log(`${dataPoint.id} | ${dataPoint.x}: ${bar} (${dataPoint.y.toFixed(2)})`);  
341     });  
342 }

The function provided executes within generateHorizontalBarChart() in a loop to get all data used in comparing the records provided. The bar length is calculated to get the value based on what the dataset or records table provides.

**Snippet: Array matching (if-statement):**

270 if (!Array.isArray(records)) {  
271             console.error('Expected an array, but got:', typeof records);  
272             rl.close();  
273             promptUser();  
274             return;  
275        }

The statement given has been chosen to determine if the records is not an array, which is never the case but it is a just-in-case measure. The type of the records is determined (i.e. String or list or any other value).

**Snippet: Stacked Readlines:**

284       rl.question('Choose a column for the X-axis (e.g., company, product, throughput): ', (xAxis) => {  
285            rl.question('Choose a column for the Y-axis (e.g., throughput, committedVolumes,  
uncommittedVolumes): ', (yAxis) => {  
286                rl.question('How would you like to sort the data? (value/id/none): ', (sortChoice) => {  
287                    rl.question('Sort in ascending or descending order? (asc/desc): ', (orderChoice) => {  
288                        if ('company', 'product', 'throughput', 'committedVolumes', 'uncommittedVolumes'].includes(xAxis) || !['throughput', 'committedVolumes', 'uncommittedVolumes'].includes(yAxis)) {  
289                            console.log('Invalid columns selected.');  
290                            rl.close();  
291                            promptUser();  
292                            return;  
293                        }  
 294                       if (!['value', 'id', 'none'].includes(sortChoice) || !['asc', 'desc'].includes(orderChoice)) {  
295                            console.log('Invalid sort or order choice.');  
296                            rl.close();  
297                            promptUser();  
298                            return;  
299                        }  
300                        const chartData = records.map(record => ({  
301                            id: record.id,  // Assuming the record ID is stored in the 'id' field  
302                            x: record[xAxis],  
303                            y: parseFloat(record[yAxis]) || 0  
304                        }));  
305                        // Sorting logic based on user choice  
306                        if (sortChoice === 'value') {  
307                            chartData.sort((a, b) => orderChoice === 'asc' ? a.y - b.y : b.y - a.y);  
308                        } else if (sortChoice === 'id') {  
309                            chartData.sort((a, b) => orderChoice === 'asc' ? a.id - b.id : b.id - a.id);  
310                        }  
311                        generateChart(chartData);  
312                        rl.close();  
313                        promptUser();  
314                    });  
315                });  
316            });  
317        });

This is not the best convention to use for readlines, but has worked the best for the author. Examples are provided as part of the first two readline prompts. Do note that some inputted columns will not work since the working columns are hard-coded. The sort choice dictates how the horizontal bar graph will display in the given order.

# Program Changes

A new input option and two new functions in program.js was the only significant change in the entire project.

# Program Demonstration via Screen Shots

**Execution:**Phase 3 execution demonstration is added here as per the instructions, but is not up-to-date.

![A screenshot of a computer

Description automatically generated  
Pressing 1 reloads the data and the console texts remain the same. Reloaded data means the data is set back to default (i.e. Add/Edit/Delete functions are undone).

A screenshot of a computer

Description automatically generated

Pressing 2 saves the data to program/data where output-from-database.csv is created and data is from keystone-throughput-and-capacity.  
  
A screenshot of a computer

Description automatically generated

The database table known as “records” has been truncated and re-populated, so the row id is set back to 1 (assignment instructions said not to show the database).  
  
Pressing 3 provides the user with another prompt to choose a display option for the records to present.

A screenshot of a computer

Description automatically generated  
Pressing 1 from there brings the user to prompt an input of a record ID to display. The number to input for the ID represents a row to select.

A black screen with white text

Description automatically generated  
The row that corresponds to the ID will be displayed, and the user will go back to the main option selection prompt.

A screenshot of a computer program

Description automatically generated  
Repeating the steps to display a single record with an index of 99 yields the following result:

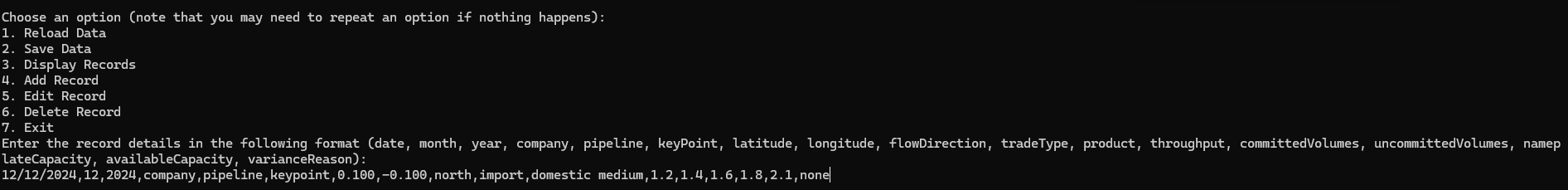
A screenshot of a computer

Description automatically generated   
Pressing 3 followed by 2 will show all records in the database (note that date and some other fields do not show since they would appear as undefined).

A screenshot of a computer program

Description automatically generated

Pressing 4 shows a prompt to enter record details to add to the record data. After pressing enter, a new record added message appears.

  
A screen shot of a computer

Description automatically generated  
A screenshot of a computer program

Description automatically generated  
Pressing 5 edits a record, prompting for an ID to select followed by entering new record details.

A screenshot of a computer

Description automatically generated   
Repeating the process with some different fields changes the row at the given point and therefore the results, which will be seen later.

Pressing 6 prompts for an ID to delete data from row corresponding to said ID.

A computer screen shot of a black screen

Description automatically generated  
Pressing 7 exits the program with a code of 0, meaning successful run.  
  
Checking the output (output-from-database.csv) after saving from the above:  
A screenshot of a computer

Description automatically generated  
Row 1 has changed and row 354 and 355 now has data. actual IDs start at 1 and the header does not count, so row 2 is index 3, id 2.

**--- Updated executions start here ---**

A screenshot of a computer

Description automatically generated  
Pressing 7 creates a horizontal bar chart to display. Options are provided along the way to compare data and sort the data in a particular order.

A screenshot of a computer

Description automatically generated

Entering “company” adds the “company” record column to to X-Axis. The same for another record column applies to the Y-Axis.  
  
A black screen with white text

Description automatically generated  
A screen shot of a computer

Description automatically generated

Entering “value” sorts the chart data by value, where id sorts the chart data by row id.  
  
Entering “value” sets the sorting of data to value.

A screen shot of a computer

Description automatically generated  
Entering “asc” organizes the data in ascending order.

A screenshot of a computer

Description automatically generated  
Repeating the steps with the last option as descending yields the following result:

A screenshot of a computer

Description automatically generated   
Repeating the steps with “id” and “asc” will result in the following:

A screenshot of a computer

Description automatically generated

Repeating the process with “id” and “desc” will result in the following:  
A screenshot of a computer

Description automatically generated

# Source Code Commenting Example

I have updated the source code comments prior to adding them here.

// program.js  
/\*\*

 \* Main program.

 \* @author {Nouraldin Hassan}

\*/

const fs = require('fs');

const csvParser = require('csv-parser');

const readline = require('readline');

const { initializeRecords, reloadRecords, persistRecords, getRecords, getRecord, addRecord, updateRecord, deleteRecord } = require('../service/record-service');

const Record = require('../service/record');

const { filePath, loadRecords } = require('../data/file-handler');

const mysql = require('mysql2');

const { importCSV, exportToCSV } = require('../data/csv-db-handler');

/\*\*

 \* Prompts the user for input.

 \* @function

 \*/

function promptUser() {

    const rl = readline.createInterface({

        input: process.stdin,

        output: process.stdout

    });

    //rl.question('Choose an option (note that you may need to repeat an option if nothing happens):\n1. Reload Data\n2. Save Data\n3. Display Records\n4. Add Record\n5. Edit Record\n6. Delete Record\n7. Exit\n\n', (answer) => {

    rl.question('Choose an option (note that you may need to repeat an option if nothing happens):\n1. Reload Data\n2. Save Data\n3. Display Records\n4. Add Record\n5. Edit Record\n6. Delete Record\n7. Create Horizontal Bar Chart\n8. Exit\n\n', (answer) => {

            switch (answer) {

            case '1':

                // reloadRecords(() => {

                //     clearConsole(true);

                //     rl.close();

                //     promptUser();

                // });

                reloadDatabaseFromCSV(rl);

                break;

            case '2':

                //     clearConsole(true);

                //     persistRecords('./program/data/output.csv');

                //     console.log('Data saved.');

                //     rl.close();

                //     promptUser();

                exportDatabaseToCSV(rl);

                break;

            case '3':

                clearConsole(true);

                giveDisplayOptions(rl);

                //displayRecords();

                //rl.close();

                //promptUser();

                break;

            case '4':

                addNewRecord(rl);

                break;

            case '5':

                editRecord(rl);

                break;

            case '6':

                deleteRecordPrompt(rl);

                break;

            case '7':

                generateHorizontalBarChart(rl);

                break;

            case '8':

                rl.close();

                //console.log('Program by Nouraldin Hassan');

                process.exit();

                break;

            default:

                console.log('Invalid option.');

                rl.close();

                promptUser();

                break;

        }

    });

}

async function reloadDatabaseFromCSV(rl) {

    const filePath = 'program/data/keystone-throughput-and-capacity.csv'; // Update to match your structure

    try {

        //console.log(`Reloading database with data from CSV file: ${filePath}`);

        if (!fs.existsSync(filePath)) {

            console.error(`File not found: ${filePath}`);

            rl.close();

            return;

        }

        clearConsole(true);

        await importCSV(filePath); // This will trigger the CSV import

        //console.log('Database reloaded successfully.');

    } catch (error) {

        console.error('Failed to reload database:', error);

    }

    console.log();

    rl.close();

    promptUser();

}

/\*\*

 \* Exports the database records to a CSV file.

 \* @param {Interface} rl - The readline interface.

 \*/

async function exportDatabaseToCSV(rl) {

    clearConsole(true);

    console.log('Exporting database records to CSV...');

    await exportToCSV()

        .then(() => {

            console.log('Database records successfully exported to program/data/output-from-database.csv.\n');

            rl.close();

            promptUser();

        })

        .catch((err) => {

            console.error('Error exporting database records to CSV:', err);

            console.log();

            rl.close();

            promptUser();

        });

}

/\*\*

 \* Provides display options to the user. Used exclusively to provide more options for displaying records.

 \* @param {readline.Interface} rl - The readline interface input.

 \*/

function giveDisplayOptions(rl) {

    clearConsole(true);

    rl.question('Choose a display option:\n1. Single: Display a single record\n2. Multiple: Display multiple records (options available)\n\n', (displayAnswer) => {

        switch (displayAnswer) {

            case '1':

                viewRecordById(rl);

                break;

            case '2':

                viewAllRecords(rl);

                break;

            default:

                console.log('Invalid display option.');

                rl.close();

                promptUser();

                break;

        }

    });

}

/\*\*

 \* Displays a record to the user.

 \* @param {readline.Interface} rl - The readline interface input.

 \*/

async function viewRecordById(rl) {

    rl.question('Enter record ID: ', async (id) => {

        try {

            const record = await getRecord(id);

            if (record) {

                console.log('Record details:');

                console.log(record);

            } else {

                console.log('Record not found.');

            }

        } catch (error) {

            console.error('Error fetching record:', error);

        }

        rl.close()

        promptUser();

    });

}

/\*\*

 \* Displays the records to the user.

 \* @param {readline.Interface} rl - The readline interface input to prompt the user on what kind of record list to print.

 \*/

async function viewAllRecords(rl) {

    try {

        const records = await getRecords();

        if (records.length === 0) {

            console.log('No records found.');

        } else {

            console.log('Records:');

            records.forEach((record, index) => {

                console.log(`${index + 1}. ${record.company} - ${record.product} - ${record.throughput}`);

            })

        }

    } catch (error) {

        console.error('Error fetching records:', error);

    }

    console.log("\nProgram by Nouraldin Hassan\n");

    rl.close();

    promptUser();

}

/\*\*

 \* Adds a new record.

 \* @param {readline.Interface} rl - The readline interface input for adding a record.

 \*/

async function addNewRecord(rl) {

    rl.question('Enter the record details in the following format (date, month, year, company, pipeline, keyPoint, latitude, longitude, flowDirection, tradeType, product, throughput, committedVolumes, uncommittedVolumes, nameplateCapacity, availableCapacity, varianceReason):\n', async (input) => {

        const details = input.split(',').map(item => item.trim());

        if (details.length !== 17) {

            console.log('Invalid input. Please provide all 17 fields.');

            rl.close()

            promptUser();

            return;

        }

        try {

            const newRecord = new Record(...details);

            await addRecord(newRecord);

            console.log('Record added successfully!');

        } catch (error) {

            console.error('Error adding record:', error);

        }

        rl.close()

        promptUser();

    });

}

/\*\*

 \* Edits a record.

 \* @param {readline.Interface} rl - The readline interface input for editing a record.

 \*/

async function editRecord(rl) {

    rl.question('Enter the record ID to update: ', async (id) => {

        rl.question('Enter the new record details (date, month, year, company, pipeline, keyPoint, latitude, longitude, flowDirection, tradeType, product, throughput, committedVolumes, uncommittedVolumes, nameplateCapacity, availableCapacity, varianceReason):\n', async (input) => {

            const details = input.split(',').map(item => item.trim());

            if (details.length !== 17) {

                console.log('Invalid input. Please provide all 17 fields.');

                rl.close()

                promptUser();

                return;

            }

            try {

                const updatedRecord = new Record(...details);

                await updateRecord(id, updatedRecord);

                console.log('Record updated successfully!');

            } catch (error) {

                console.error('Error updating record:', error);

            }

            rl.close()

            promptUser();

        });

    });

}

/\*\*

 \* Deletes a record

 \* @param {readline.Interface} rl - The readline interface input for deleting a record.

 \*/

async function deleteRecordPrompt(rl) {

    rl.question('Enter the record ID to delete: ', async (id) => {

        try {

            const record = await deleteRecord(id);

            if (record) {

            console.log('Record deleted successfully!');

            } else {

            console.log('Record not found')

            }

        } catch (error) {

            console.error('Error deleting record:', error);

        }

        rl.close()

        promptUser();

    });

}

/\*\*

 \* Generate a Horizontal Bar Chart for visualizing data.

 \* @param {readline.Interface} rl - The readline interface input.

 \*/

async function generateHorizontalBarChart(rl) {

    clearConsole(true);

    console.log('Generating Horizontal Bar Chart...');

    try {

        const records = await getRecords();

        // console.log('Fetched Records:', records);

        if (!Array.isArray(records)) {

            console.error('Expected an array, but got:', typeof records);

            rl.close();

            promptUser();

            return;

        }

        if (records.length === 0) {

            console.log('No records found.');

            rl.close();

            promptUser();

            return;

        }

        rl.question('Choose a column for the X-axis (e.g., company, product, throughput): ', (xAxis) => {

            rl.question('Choose a column for the Y-axis (e.g., throughput, committedVolumes, uncommittedVolumes): ', (yAxis) => {

                rl.question('How would you like to sort the data? (value/id): ', (sortChoice) => {

                    rl.question('Sort in ascending or descending order? (asc/desc): ', (orderChoice) => {

                        if (!['company', 'product', 'throughput', 'committedVolumes', 'uncommittedVolumes'].includes(xAxis) || !['throughput', 'committedVolumes', 'uncommittedVolumes'].includes(yAxis)) {

                            console.log('Invalid columns selected.');

                            rl.close();

                            promptUser();

                            return;

                        }

                        if (!['value', 'id'].includes(sortChoice) || !['asc', 'desc'].includes(orderChoice)) {

                            console.log('Invalid sort or order choice.');

                            rl.close();

                            promptUser();

                            return;

                        }

                        const chartData = records.map(record => ({

                            id: record.id,

                            x: record[xAxis],

                            y: parseFloat(record[yAxis]) || 0

                        }));

                        if (sortChoice === 'value') {

                            chartData.sort((a, b) => orderChoice === 'asc' ? a.y - b.y : b.y - a.y);

                        } else if (sortChoice === 'id') {

                            chartData.sort((a, b) => orderChoice === 'asc' ? a.id - b.id : b.id - a.id);

                        }

                        generateChart(chartData);

                        rl.close();

                        promptUser();

                    });

                });

            });

        });

    } catch (error) {

        console.error('Error generating chart:', error);

        rl.close();

        promptUser();

    }

}

/\*\*

 \* Generate the horizontal bar chart and print it to the console.

 \* @param {Array} chartData - The data to be plotted on the chart.

 \*/

function generateChart(chartData) {

    const maxBarLength = 50;

    console.log('Horizontal Bar Chart:');

    chartData.forEach(dataPoint => {

        const barLength = Math.round(dataPoint.y / Math.max(...chartData.map(d => d.y)) \* maxBarLength);

        const bar = '█'.repeat(barLength);

        console.log(`${dataPoint.id} | ${dataPoint.x}: ${bar} (${dataPoint.y.toFixed(2)})`);

    });

}

function clearConsole(showAuthorAtEnd = false) {

    console.clear();

    process.stdout.write("\u001b[3J\u001b[2J\u001b[1J");

    if (showAuthorAtEnd) {

        console.log('Program by Nouraldin Hassan\n');

    }

}

/\*\*

 \* Initializes records and prints the author's name

 \*/

initializeRecords(() => {

    clearConsole();

    console.log('Program by Nouraldin Hassan\n');

    promptUser();

});