Meenakshi Gopalakrishnan
Student ID: 3682020

Assignment 3: Journal

Sources:

For this assignment, I primarily used the Think C++ textbook for reference. In my procedure, I first read through and took notes on chapter 15, as indicated in the Study Guide to strengthen my understanding of C++ concepts and complete these assignments. My notes as well as links to other sources I have referenced are listed below.

Think C++ Notes [Chapters 15]

Streams:

- Streams represent the flow of data from a source to a destination.
- Common streams include cin (for input from the keyboard) and cout (for output to the screen).
- File streams (ifstream for input and ofstream for output) are used for file operations.

File Input:

- To read data from a file, create an ifstream object with the file name as an argument.
- Use the >> operator or getline function to read data.
- Functions like good, eof, fail, and bad are used to check the status of the input stream.

File Output:

- Creating an ofstream object with the file name allows writing data to a file.
- Use the << operator to send data to the output file.
- It's important to check the status of the file streams to handle errors effectively.

Parsing Input:

- Parsing involves analyzing the structure of input to extract meaningful information.
- For example, parsing a file containing city distances formatted with city names in quotes and distances as integers.
- Use the find function to locate quotation marks and the substr function to extract substrings.

Parsing Numbers:

- Converting strings to integers often involves removing non-digit characters (e.g., commas in large numbers).
- The isdigit function helps identify digit characters.
- The atoi function converts a C string to an integer.

Set Data Structure:

- An ordered set ensures elements are unique and maintains their order.
- Adding elements to a set checks for uniqueness before insertion.

apmatrix:

- The apmatrix class represents a two-dimensional array with rows and columns.
- Constructors initialize the matrix with specified dimensions and optionally, initial values.
- Access matrix elements using row and column indices.
- Common operations include resizing the matrix and traversing it with nested loops.

Distance Matrix Example:

- Use the apmatrix class to store distances and a Set to manage city names.
- Functions handle reading data from a file, parsing it, and storing it in the matrix.

Improving Code Design:

- Encapsulation: Combine related data and functions into a single object (e.g., DistMatrix).
- Simplification: Refactor code to make it more readable and maintainable.
- Error checking: Implement accessor functions to perform error checking and maintain invariants.

Journal Entry 1: Word Counter [Problem 1]

Program Purpose:

The primary goal of this program is to count the number of whitespace-separated words in a specified file. When the program is executed, it prompts the user to enter a filename. Upon receiving the filename, the program reads the content of the file, processes it to count the number of words, and then outputs the total word count to the console. This program is a demonstration of basic file input/output (I/O) operations, string manipulation, and error handling in C++. It aims to provide a practical understanding of how to handle files, manage user inputs, and process text data in a structured and efficient manner.

Reflection on Assignment:

Developing the Word Count Program was a valuable learning experience that involved several key programming concepts and techniques. The assignment required me to implement file handling, string processing, and user interaction. Through this assignment, I gained hands-on experience with file streams, error handling, and text parsing. The process of designing, coding, and testing the program helped me reinforce my understanding of these concepts and apply them to solve a real-world problem.

Activities Undertaken:

Understanding Requirements:

- Carefully reviewed assignment specifications to understand program behavior.
- Identified the need for reading a file and counting the number of whitespace-separated words within it.
- Recognized the input and method requirements and implemented input validation to ensure accurate and reliable program execution.

Design Decisions:

- Opted for a straightforward procedural approach to maintain focus on file I/O and string processing tasks.
- Designed a dedicated function, countWordsInFile, to encapsulate the word-counting logic, ensuring the code remains modular and reusable.
- Implemented comprehensive error handling mechanisms to manage file access issues and provide clear, informative error messages to the user.
- Chose to handle user inputs and file processing in a way that ensures efficient resource utilization and prevents memory leaks.

Implementation:

- Implemented the countWordsInFile function, which takes a filename as input, reads the file, and returns the word count.
- Utilized file streams to open and read the specified file line by line, processing its content efficiently.
- Applied string manipulation techniques to split each line into individual words and count them accurately.
- Ensured the use of clear and descriptive names for variables and functions, accompanied by comments to enhance code readability and maintainability.
- Paid special attention to input validation to handle invalid filenames, providing informative error messages and ensuring a smooth user experience.

Testing and Debugging:

- Conducted testing across various scenarios, including valid and invalid user inputs.
- Verified the correctness of the program by using a variety of sample text files with different contents and structures.
- Iteratively refined the logic to ensure the program runs smoothly and efficiently, addressing any issues encountered during testing and debugging.

Difficulties Encountered:

File Handling:

- *Ensuring that the file was opened and read correctly required careful attention to file stream operations and error handling.*
- *Handling invalid filenames and providing clear and informative error messages was challenging, requiring thoughtful design and implementation.*

String Processing:

- *Splitting lines into words and counting them accurately presented challenges, particularly in handling various types of whitespace and special characters.*
- *Overcame these challenges through iterative refinement of the string processing logic, testing with different input scenarios to ensure accuracy and robustness.*

Learning Outcomes Fulfilled:

Consolidation of Core Concepts:

- Reinforced my understanding of fundamental C++ concepts, including file I/O, string manipulation, and error handling.
- Applied these concepts effectively to design and implement a solution to the assignment requirements.

Enhanced Debugging Skills:

- Improved problem-solving skills through iterative design, implementation, testing, and debugging processes.
- Developed the ability to identify and rectify logical errors and edge cases through rigorous testing and debugging processes.

Sources Used:

Reference for C++ file I/O: cppreference.com
Reference for C++ string manipulation: cppreference.com

Reflection on Programming Process:

- The development of the Word Count Program deepened my understanding of file handling and text processing in C++.
- Emphasized the significance of user interaction and input validation
- Gained valuable insights into modular, object-oriented programming and developed a more structured approach to software development.

Journal Entry 2: Text File Reader I [Problem 2]

Program Purpose:

The purpose of this program is to read and display the contents of a text file, one line at a time, waiting for user input between lines. The user provides the name of the text file via a prompt, and the program reads the file line by line, displaying each line only after the user presses the <Enter> key. This program demonstrates file I/O operations, user interaction, and control flow in C++.

Reflection on Assignment:

Developing the Text File Reader Program was an insightful experience that involved implementing user interaction, file handling, and control flow management. Through this assignment, I gained practical experience with prompting user input, processing text files, and controlling the flow of the program to wait for user actions. The assignment provided a hands-on opportunity to enhance my understanding of these fundamental C++ concepts.

Activities Undertaken:

Understanding Requirements:

- Analyzed the problem statement to understand the need for reading and displaying a text file line by line, with user interaction between lines.
- Identified the core functionalities required: prompting for a file name, reading the file, and waiting for user input between lines.

Design Decisions:

- Decided to implement a TextFileReader class to encapsulate the functionality of reading and displaying the file contents.
- Chose to use a simple loop to read and display lines from the file, with std::cin.ignore() to wait for user input between lines.

Implementation:

- Implemented the TextFileReader class with a method to read and display lines from the specified file.
- Used file streams (std::ifstream) to open and read the file.
- Incorporated user prompts and used std::cin.ignore() to pause the program and wait for the user to press <Enter> before proceeding to the next line.
- Ensured the use of clear and descriptive names for variables and functions, with comments to enhance code readability and maintainability.

Testing and Debugging:

- Verified the correctness of the program by using different sample text files with varying content.
- Iteratively refined the logic to ensure the program runs smoothly and efficiently, addressing any issues encountered during testing and debugging.

Difficulties Encountered:

File Handling:

- Ensuring that the file was opened and read correctly required careful attention to file stream operations and error handling.
- Handling invalid filenames and providing clear and informative error messages was challenging, requiring thoughtful design and implementation.

Learning Outcomes Fulfilled:

This assignment reinforced my understanding of core C++ concepts, including file I/O, user interaction, and control flow management. By designing and implementing a solution to meet the assignment requirements, I was able to apply these concepts effectively. Additionally, the iterative process of designing, implementing, testing, and debugging significantly improved my problem-solving skills.

Reflection on Programming Process:

Developing the Text File Reader Program deepened my understanding of file handling, user interaction, and control flow in C++. It highlighted the importance of providing informative user feedback and creating a smooth user experience. I gained valuable insights into writing modular and efficient code, benefiting future software projects. This assignment also helped me adopt a more structured and systematic approach to software development, emphasizing planning, testing, and iterative improvement.

Journal Entry 3: Text File Reader II [Problem 3]

Program Purpose:

The objective of this program is to read and display the contents of a text file, line by line, with line numbers. The program reads up to 100 lines from a specified file, concatenates them into a single string, and displays the lines with their respective line numbers. It waits for the user to press <Enter> after each line, ensuring a clear and controlled display of the file content.

Reflection on Assignment:

Developing the Text File Reader Program involved handling files, managing arrays, and displaying formatted output. This assignment allowed me to enhance my understanding of file I/O operations, array manipulation, and user interaction in C++. It provided practical experience in handling command-line arguments, reading and storing text data, and displaying it with additional formatting.

Activities Undertaken:

Understanding Requirements:

- Carefully reviewed the assignment specifications to understand the program's expected behavior.
- Identified the need for reading a file and displaying its contents with line numbers.
- Recognized the importance of handling command-line arguments for specifying the filename and implemented robust input validation.

Design Decisions:

- Designed a TextFileReader class to encapsulate the functionality of reading and storing lines from a file.
- Implemented methods within the class to concatenate the file contents into a single string and to display each line with its line number.

- Added error handling to manage issues related to file access and command-line arguments.

Implementation:

- Created the TextFileReader class with a constructor that takes the filename as input and reads up to 100 lines from the file.
- Used file streams to open and read the specified file, storing the lines in an array.
- Developed a method to concatenate the stored lines into a single string and another method to display each line with its line number.
- Ensured adherence to best coding practices, including clear and descriptive method names and meaningful comments for code readability.

Testing and Debugging:

- Verified the correctness of the program by using different sample text files with varying content and line counts.
- Debugged any encountered issues promptly, focusing on precision handling, formatting inconsistencies, and boundary conditions.

Difficulties Encountered:

File Handling:

- Ensuring the file opened and read correctly required careful attention to file stream operations and error handling.
- Handling invalid filenames and providing clear and informative error messages was challenging, necessitating thoughtful design and implementation.

Array Management:

- Storing up to 100 lines in an array and managing the array operations presented challenges.
- Refined the array management logic iteratively to ensure accurate storage and retrieval of lines.

Learning Outcomes Fulfilled:

Consolidation of Core Concepts:

- Reinforced my understanding of file I/O, array manipulation, and user interaction in C++.
- Successfully applied these concepts to design and implement a solution that meets the assignment requirements.

Reflection on Programming Process:

Developing the Text File Reader Program deepened my understanding of file handling, array management, and user interaction in C++. I gained valuable insights into writing modular and efficient code, which will benefit future software projects. This assignment also helped me adopt a more structured approach to software development, emphasizing the importance of planning, testing, and iterative improvement.

Journal Entry 4: Square Operations Program [Problem 4]

Program Purpose:

The purpose of this program is to demonstrate basic operations on arrays, including filling arrays with values, performing element-wise addition, and displaying the results. Specifically, it fills one array with loop counter values, another with the squares of those values, and a third with the sum of corresponding elements from the first two arrays. The results are displayed with formatted floating-point numbers.

Reflection on Assignment:

Developing the Array Operations Program was an insightful experience that involved working with arrays, performing arithmetic operations, and displaying formatted output. This assignment helped me strengthen my understanding of array manipulation, loop constructs, and formatting output in C++. It provided practical experience in creating, populating, and manipulating arrays, and in displaying results in a clear and readable format.

Activities Undertaken:

Understanding Requirements:

- Carefully reviewed the assignment specifications to understand the program's expected behavior.
- Identified the need to fill arrays with specific values, perform element-wise addition, and display the results with formatted output.

Design Decisions:

- Used a loop to fill the first array with loop counter values and the second array with their squares.

- Implemented element-wise addition to populate the third array with the sum of corresponding elements from the first two arrays.
- Utilized std::fixed and std::setprecision to format the output with two decimal places.

Implementation:

- Created three arrays of size 25 to store the values.
- Filled the first array with loop counter values and the second array with their squares.
- Calculated the sum of corresponding elements from the first two arrays and stored the results in the third array.
- Displayed the contents of all three arrays with formatted output to ensure readability.

Testing and Debugging:

- Conducted testing to verify the program's correctness by checking the output for standard input scenarios.
- Ensured that the arrays were populated correctly and the results displayed as expected.

Difficulties Encountered:

Output Formatting:

- Formatting the floating-point numbers for display required using std::fixed and std::setprecision correctly.
- Overcame this challenge by referring to documentation and testing different formatting options.

Learning Outcomes Fulfilled:

Consolidation of Core Concepts:

- Reinforced my understanding of array manipulation, arithmetic operations, and output formatting in C++.
- Successfully applied these concepts to design and implement a solution that meets the assignment requirements.

Enhanced Debugging Skills:

- Improved my problem-solving skills through iterative design, implementation, testing, and debugging processes.
- Developed the ability to identify and rectify logical errors and edge cases through rigorous testing and debugging practices.

Reflection on Programming Process:

Developing the Array Operations Program deepened my understanding of array manipulation and output formatting in C++. It highlighted the importance of clear and readable output for user interaction and provided valuable insights into writing modular and efficient code. This experience will benefit future software projects by emphasizing the importance of planning, testing, and iterative improvement. Overall, it helped me adopt a more structured and systematic approach to software development.

Journal Entry 5: Book Collection Management Program [Problem 5]

Program Purpose:

The purpose of this program is to manage a collection of books by creating book objects with various attributes, displaying them, and sorting them by title and year of publication. The program allows for the entry of book details, stores them in a collection, and provides functionality to display and sort the books.

Reflection on Assignment:

Developing the Book Collection Management Program enhanced my understanding of object-oriented programming concepts, such as classes and objects, and the use of comparators for sorting. This assignment helped me practice creating and managing a collection of objects, implementing sorting algorithms, and displaying formatted output.

Activities Undertaken:

Understanding Requirements:

- Reviewed the assignment specifications to understand the program's expected behavior.
- Identified the need to manage a collection of books, including creation, display, and sorting.

Design Decisions:

- *Designed a Book class to represent individual books with attributes like title, ISBN, author, edition, publisher, and year.*
- *Implemented a BookComparator class to sort books by title and year of publication.*

- *Created a Bookshelf class containing the main method to demonstrate the functionality of creating, displaying, and sorting books.*

Implementation:

- Defined the Book class with constructors, getter and setter methods, and a display method.
- Implemented the BookComparator class as a functor to compare two books.
- Developed the Bookshelf class to create a collection of books, display them, and sort them.
- Used std::vector to store the collection of books and std::sort for sorting them.

Testing and Debugging:

- Conducted testing to verify the creation and display of book objects.
- Tested sorting functionality to ensure books were sorted by title and year correctly.

Difficulties Encountered:

Object Management:

- Ensuring accurate creation and manipulation of book objects required careful attention to class methods and attributes.
- Addressed challenges by refining class designs and verifying outputs.

Sorting Logic:

- Implementing a comparator to sort books by multiple attributes required careful design.
- Overcame this challenge by testing and refining the comparator logic.

Learning Outcomes Fulfilled:

Consolidation of Core Concepts:

- Reinforced understanding of classes, objects, and sorting algorithms in C++.
- Successfully applied these concepts to manage a collection of books and implement sorting functionality.

Enhanced Debugging Skills:

- Improved problem-solving skills through iterative design, implementation, testing, and debugging processes.

- Developed the ability to identify and rectify logical errors and edge cases through rigorous testing and debugging.

Reflection on Programming Process:

Developing the Book Collection Management Program deepened my understanding of object-oriented programming and sorting algorithms. It highlighted the importance of clear and modular code design for managing collections of objects. I gained valuable insights into writing efficient and readable code, which will benefit future software development projects. This assignment also helped me adopt a more structured and systematic approach to software development, emphasizing planning, testing, and iterative improvement.