Table of Contents:

Introduction

Overview:

The objective of this project is to design and develop a software application capable of reading two text files and formatting the data into a new file. The programming language of choice for this application is C++, and the data structures employed will be array and heap. Additionally, the application will utilize abstraction and encapsulation principles of object-oriented programming (OOP) to enhance its functionality and maintainability. “CPlusPlus.com. (2000-2023)”

Objectives:

- To ensure accurate and consistent parsing of supplier and product data from text files
- To generate an inventory representation that aligns with the stakeholder’s requirements
- To ensure application robustness and error handling capabilities
- To maintain a well-documented and maintainable codebase

Scope:

- Text file processing
- Data parsing from supplier and product data
- Conversion of parsed data into a comprehensive inventory.txt representation
- Basic presentation of generated inventory

Software Architecture Overview/High-Level Overview

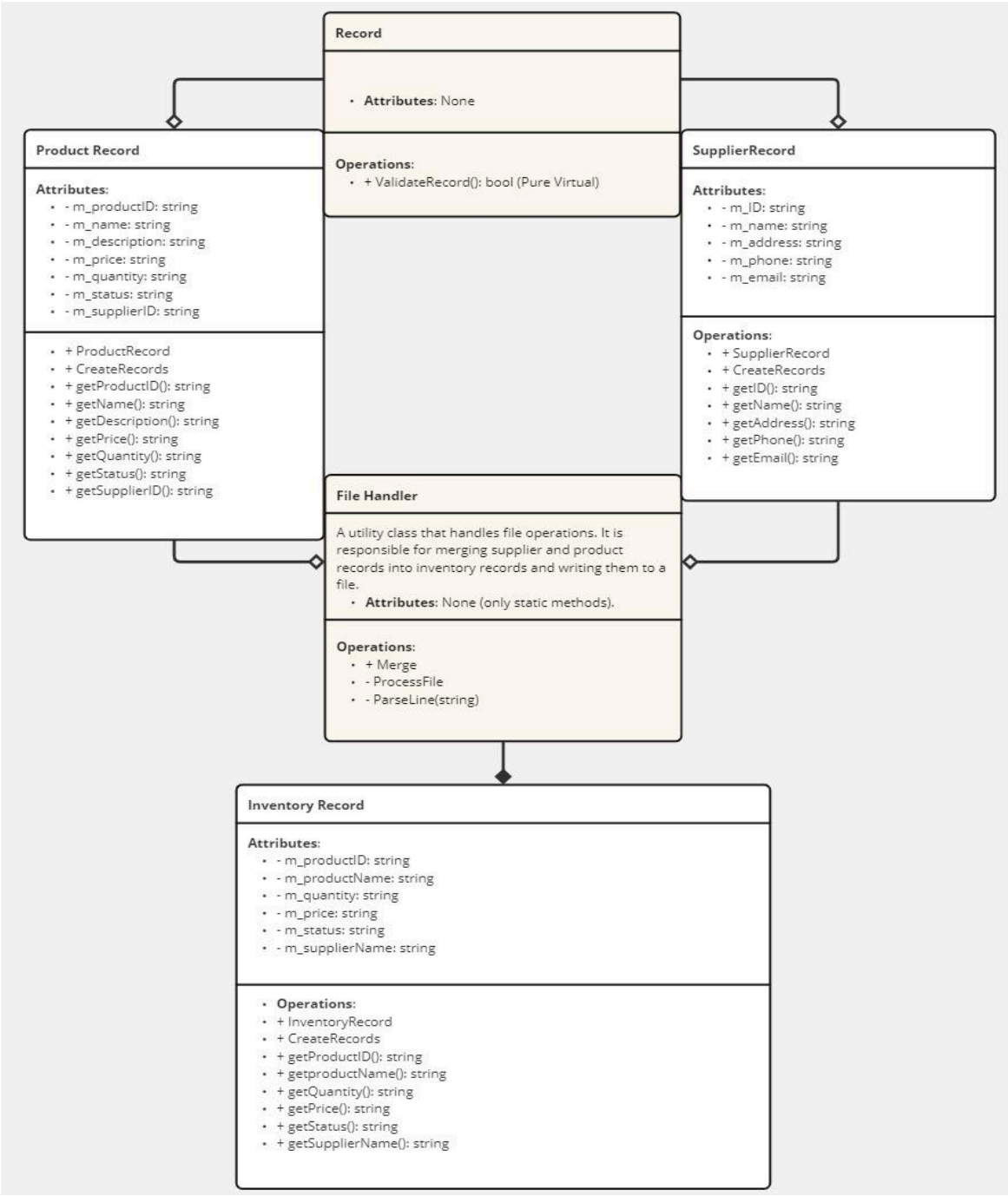
Architecture Design:

- The architecture of our application is based on the component-based design approach, in which the software system is broken down into discrete, reusable modules. Each module encapsulates a specific functionality and can be developed, tested, and maintained separately.
- Instead of creating a single, monolithic module, we have modularized it into submodules, each responsible for a specific feature of the application.
- For instance, we have a module named FileHandler, which contains various methods dedicated to handling input files in the program.
- By adhering to component-based architecture principles, our codebase gains the benefits of modularity, reusability, scalability, testability, and maintainability.

Description:

The program will take two input files, one containing supplier data and the other containing product data. These two files will be combined based on the supplier ID to create an inventory file, where the inventories are sorted by the ID numbers.

UML Diagram:



Modeled after *UML class and Object Diagrams Overview*, Fakhroutdinov, K. (2018, April 21)

Detailed Design

Data Structures:

- Array and Heaps - C++ vectors are dynamic arrays. Heaps are used in the STD library sort function of C++
- The vector elements consist of columns from the supplier and product files. Sorting operations are performed on these vectors to arrange the rows in the inventory file by ID. This sorting process utilizes a combination of Quicksort, Heapsort, and Insertion sort algorithms implemented using arrays and heaps.

OOP Designs:

a. Encapsulation

Examples from code

Encapsulation is demonstrated in the class FileHandler. Specifically, in FileHandler, we have the private static member functions ParseLine and ProcessFile encapsulate the details of parsing lines and processing files, respectively. These functions are set "private" to prevent direct access outside the FileHandler, which shows data hiding and access control. Encapsulation is also shown through the use of private variables for ProductRecord, SupplierRecord, and InventoryRecord classes. Direct access to data is prohibited and instead, attributes are retrieved with public get methods.

Benefits

Encapsulation helps maintain data integrity and prevent unauthorized access to the inner workings of a class. By encapsulating functionality within the class, we are able to control how data is manipulated and accessed. This reduces the risk of unintended side effects and enhances code maintainability. Additionally, encapsulation promotes modularity and code organization, making it easier to understand and maintain the codebase.

Why it is necessary

Without encapsulation, we may expose the inner details to the external files. This could lead to potential misuse of critical functionality, which leads to stability and security issues of the application. Additionally, any changes to the internal implementation would need changes across multiple parts of the codebase, making it hard to maintain.

b. Abstraction

Examples from code

Abstraction is shown in the Merge function of the FileHandler class. This function “abstracts away”, or hides the details of file merging and sorting, providing a high-level interface for merging supplier and product files into an inventory file. Users of the FileHandler class do not need to know how the Merge and Sort methods are implemented.

Benefits

Abstraction simplifies the complexity of a system by hiding unnecessary details and exposing only essential features to the user. By abstracting the file merging and sorting process, the Merge function provides a clear and concise interface for users to interact with. This promotes code readability, makes it easier to use, easier to maintain and extend the codebase.

Why it is necessary

Without abstraction, the Merge function would expose the underlying implementation details of file merging and sorting to the users of the FileHandler class. This would increase the complexity of using the Merge function, making it harder to understand and maintain. Additionally, any changes to the implementation details would require modifications to other parts of the code, leading to a lack of separation of concerns and decreased code flexibility.

Programming Language:

C++ is an object-oriented programming language, which makes it suitable for this task. It contains built-in data structures which are necessary for the functionality of the program, such as heaps and arrays.

APIs:

- Utilization of C++ APIs for specific tasks
    - IOstream
    - Fstream
    - Vector
    - Algorithm
    - String
    - Sstream

Offensive Programming:

1. Assert Pre-Conditions
    a. Pre-conditions constitute checks performed at the beginning of a function to validate the assumptions made by the subsequent code. These assumptions may encompass the state of input parameters, global variables, or any condition that must be met for the function to execute correctly.
    b. Asserting pre-conditions facilitates the early detection of errors by prompting a prompt failure if the conditions are not satisfied. This is particularly advantageous during development and testing phases, as it can prevent undefined behavior or difficult-to-diagnose bugs later in the execution process.
2. Validate Data
    a. Validating data helps to ensure that only well-structured and expected data is processed by the application, thereby mitigating the risk of errors and crashes.
3. Assert Post-Conditions
    a. Post-conditions denote the anticipated conditions to hold true following the execution of a code block. Asserting post-conditions establishes that the code's effect aligns with its intended purpose and aids in verifying the function's successful execution.
    b. These assertions serve as a protective mechanism, detecting logical errors and ensuring that subsequent parts of the program can operate reliably on the function's outcomes.
4. Use Exceptions for Error Handling
    a. Exceptions offer a structured method for managing errors and exceptional circumstances. While assertions are commonly used during debugging, exceptions are essential for handling runtime errors in production code.
    b. Utilizing exceptions enables the separation of error handling from regular code paths, enhancing code readability and comprehension. This approach allows for error propagation up the call stack, facilitating the handling of errors at appropriate levels with sufficient context for corrective measures or graceful failure.

Appendix

ProductFile.txt
Interface:

Property	Description
ProductID (int)	A unique identifier for each product
ProductName(string)	Name of the product
Description(string)	Brief description of product
Price (string)	Price of the inventory
Quantity (int)	Inventory Quantity
Status (Char)	A character indicating the product's status
SupplierID(string)	Foreign Key (the product is supplied by the supplier with this ID)

(output file must be sorted by product ID)

Product File Sample

2591, Camera, Camera, $799.9, 50, B, 7890
3374, Laptop, MacBook Pro, $1799.9, 30, A, 9876
3034, Telephone, Cordless Phone, $299.99, 40, A, 3456
3034, Telephone, Home telephone, $99.9, 25, A, 8765
1234, TV, Plate TV, $799.9, 20, C, 9144
1234, TV, Plate TV, $1499.99, 5, A, 7671
2591, Camera, Instant Camera, $179.5, 30, C, 8642
1516, Mouse, Wireless Mouse, $99.5, 30, A, 3579
3034, Telephone, Home Telephone, $169.99, 15, A, 8692
2591, Camera, Digital Camera, $499.9, 10, B, 9512
3034, Telephone, Home Telephone, $59.5, 20, A, 8655
2591, Camera, Digital Camera, $449.4, 50, A, 3592
1234, TV, Plate TV, $699.7, 5, B, 7084
1516, Mouse, Wireless Mouse, $69.9, 25, C, 2345
3374, Laptop, Laptop, $1399.2, 10, B, 1357
3374, Laptop, Refurbished Laptop, $1099.1, 20, A, 6954
1516, Mouse, Wireless Mouse, $49.4, 50, B, 9794
1516, Mouse, Wireless Mouse, $69.5, 20, A, 7807
1234, TV, Plate TV, $599.3, 5, B, 8672
3374, Laptop, Laptop, $1369.9, 15, A, 4567

SupplierFile.txt

Property	Description
SupplierID (int)	A unique identifier for each supplier

SupplierName(string)	Name of the supplier
Address(string)	Address of the supplier
Phone (string)	Phone number for the supplier
Email (string)	Email address for the supplier

Supplier File Sample

9512, Acme Corporation, 123 Main St, 205-288-8591, info@acme-corp.com
8642, Xerox Inc., 456 High St, 505-398-8414, info@xrx.com
3579, RedPark Ltd., 789 Park Ave, 604-683-2555, info@redpark.ca
7890, Samsung, 456 Seoul St, 909-763-4442, support@samsung.com
7671, LG Electronics, 789 Busan St, 668-286-5378, support@lge.kr
9876, Toshiba, 246 Osaka St, 90-6378-0835, support@toshiba.co.jp
3456, Panasonic, 246 Osaka St, 443-887-9967, support@panasonic.co.jp
8765, Philips, 789 Amsterdam St, 61-483-898-670, support@philips.au
1357, Sharp, 123 Tokyo St, 80-4745-3107, support@sharp.co.jp
9144, Fujitsu, 456 Tokyo St, 03-3556-7890, support@fujitsu.co.jp
8655, Dell, 246 Austin St, 505-351-3181, support@dell.com
3592, IBM, 456 New York St, 201-335-9423, support@ibm.com
7084, Acer, 135 Taipei St, 905-926-031, support@acer.tw
2345, MSI, 789 Mofan St, 943-299-465, support@msi.tw
6954, Apple, 246 Cupertino St, 202-918-2132, support@apple.com
9794, Amazon, 246 Seattle St, 555-343-8950, support@amazon.com
8692, Microsoft, 123 Redmond St, 505-549-0420, support@microsoft.com
7807, Intel, 2200 Mission College Blvd, 408-646-7611, support@intel.com
8672, AMD, 246 Santa Clara St, 312-866-2043, support@amd.com
4567, Qualcomm, 456 San Diego St, 44-7700-087231, info@qualcomm.co.uk

InventoryFile.txt

Interface:

Property	Description
ProductID (int)	A unique identifier for each product
ProductName(string)	Name of the product
Quantity (int)	Inventory Quantity
Price (string?!)	Price of the inventory
Status (Char)	A character indicating the product's status
SupplierName(string)	Name of the supplier

Note: Since ProductFile has SupplierID as the foreign key, we can match the specific supplier with the product in ProductFile based on the SupplierID, which helps us get the Inventory File as output.

References

- CPlusPlus.com. (2000-2023). C++ language tutorial. https://cplusplus.com/doc/tutorial/files/

- Fakhroutdinov, K. (2018, April 21). *UML class and Object Diagrams Overview*. UML graphical notation overview, examples, and reference. https://www.uml-diagrams.org/class-diagrams-overview.html

- *Defensive & Offensive Programming*. Programming Duck. (2022, January 16). https://programmingduck.com/articles/defensive-programming