Group 5 (Hamilton) Class Specification - CSE 498

- **Datum**
  - Class Description
    - The Datum class functions as a storage and conversion for doubles and strings. Its goal is to store a value as a double or string and to convert it into the other data type when needed.
  - Similar Classes
    - Somewhat similar to std::variant
    - Classes Datum will use:
      - std::string
      - std::optional
      - std::variant
  - Key Functions
    - variant<double, string> mValue;
      - Note: **Not** a function, but a member variable that stores the value
    - Datum(double value)
      - Constructor for a double
    - Datum(string value)
      - Constructor for a string
    - void AsString()
      - Coverts the value as a string
    - void AsDouble()
      - Coverts the value as a double
    - bool IsString() const
      - Returns true if the value is a string, returns false otherwise (if it's a double)
    - bool IsDouble() const
      - Returns true if the value is a double, returns false otherwise (if it's a string)
    - void SetStringValue(string value)
      - Sets the value of the member variable to a string
    - void SetDoubleValue(double value)
      - Sets the value of the member variable to a double
    - variant<double, string> GetVariant() const
      - Return the member variable value
        - Returns as a variant
    - optional<string> GetString() const
      - Return string if the variant contains a string, else returns empty optional
    - optional<double> GetDouble() const
      - Return double if the variant contains a double, else returns empty optional

  Note: Used How to Deal with OPTIONAL Data in C++ to help with the optional

  - Error Conditions
    - User Error: Converting a string that has no numeric value
      - Returns NaN

        - User Error: Converting a NaN into a string
            - Return empty string
        - **From ChatGPT**: Programming Error: The programmer attempts to perform arithmetic or comparisons between mismatched types (e.g., compare a double Datum to a string Datum).
            - We need to: Document the behavior clearly and optionally trigger an assertion if a comparison or operation is invalid.
    - Challenges
        - Overall, I don't think this class will be too difficult to develop since it's a simple data class. We've never used variants before, so we will need to do some research on using them. The most challenging part will be detecting and determining what to do for edge cases mentioned in the *Error Conditions* section.
    - Other Class Projects
        - We don't need classes from other groups to develop the Datum class, but Group 2's DataMap class is similar to ours because it also stores unknown types of data.
        - Group 7's 'Output Log' class could be useful for the development of every class because it will provide useful debugging logs.
- **Reference Vector**
    - Class Description
        - The **ReferenceVector** class represents a collection of **Datum** objects stored as references. Unlike traditional vectors, which store objects directly, **ReferenceVector** keeps pointers to **Datum** instances under the hood. This design allows the manipulation of entries without duplicating data, ensuring consistency across the application when a single **Datum** object is updated in multiple contexts. It supports dynamic sizing, iteration, and proxy access for flexible operations.
    - Similar Classes
        - std::vector
        - std::deque - Efficient insertion/removal at the front and back. Has bidirectional iteration.
        - std::list - doubly linked list, which allows efficient insertion and deletion at arbitrary positions.
        - std::set/std::unordered_set - Quick lookup functionality for **Datum*** elements.
        - std::map/std::unordered_map - Could handle cases where an uninitialized **Datum*** needs to be accessed or created on demand.
    - Key Functions
        - Inherits all vector functions
            - Modified indexing to return **Datum*** elements
    - Error Conditions
        - Programmer error: Accessing out-of-bounds index
            - Throw an std::out_of_range exception
        - User error: Accessing null elements
            - Return nullptr

    - Challenges
        - Proxy object design
        - Handling null operations
        - Integration with other classes
    - Other Class Projects
        - DataMap from group 2 could be useful. Similar data structure probably with similar manipulation tactics.
        - FunctionSet from group 3 is a set of functions instead of a vector of **Datum*** elements
        - BitVector from group 4 is a vector that allows size changing
        - StringSet from group 6 has a set of strings with many operations like filter (to keep only strings that meet a certain criterion), filter out (to remove strings that meet a criterion) and well as union, intersection, and difference between two different StringSets.
        - RandomAccessSet from group 8 may help us see how to access elements at will
        - EventQueue from group 9 is an event container with priority features
        - StaticVector from group 10 is an efficient vector with a set maximum size and allocates all memory at creation.
- **DataGrid**
    - Class Description
        - Manages a table or 2D grid of Datum values.
        - Structured as a vector of reference vectors, where each sub-vector represents a row.
        - A simple interface for manipulating and querying tabular data.
    - Similar Classes
        - std::vector or std::array
            - This class is a vector of vectors at its core.
        - Similar to Pandas' DataFrame
    - Key Functions
        - DataGrid()
            - Constructor
        - ~DataGrid()
            - Destructor to deallocate memory
        - std::tuple<int, int> Shape()
            - Returns shape of DataGrid (ie. number of rows and columns)
        - ReferenceVector GetRow(int row)
            - Returns ReferenceVector row at given index
        - ReferenceVector GetColumn(int col)
            - Return a ReferenceVector of column at given index
                - ReferenceVector will need to be created, as existing ReferenceVectors in DataGrid are rows and not columns
        - Datum GetValue(int row, int col)
            - Returns Datum at given index
        - void InsertColumn(ReferenceVector v, int col)
            - Inserts reference vector as a column at column position j

            - Append to end as final columns as default
        - void InsertRow(ReferenceVector v, int row)
            - Inserts reference vector as a row at row position i
                - Append to end as final row as default
        - ReferenceVector RemoveColumn(int col)
            - Remove and return column ‘col’
        - ReferenceVector RemoveRow(int row)
            - Remove and return row ‘row’
        - ReferenceVector operator[](int row)
            - Returns a ReferenceVector at the given row index
                - Same as GetRow()
        - Future Function Ideas
            - DataGrid CreateTable(int i, int j, defaultValue)
            - Resize
            - Clear
            - isEmpty
            - isValidIndex
            - FillNA
    - Error Conditions
        - *Programming Error:* Indexing out of bounds for querying, manipulating and inserting is a possibility for nearly every member function
            - Throw error
    - Challenges
        - Efficiently handling columns operations, especially insertions
            - This is a challenge because the DataGrid is organized as a collection of rows and not columns
        - Dynamic memory management for large DataGrids
        - Robust error handling for out of bounds indexing
    - Other Class Projects
        - Group 2: Ritchie
            - Class: StateGrid
                - This class and our class both deal with querying and manipulating tabular data
- **CSVfile**
    - Class Description
        - Responsible for managing interactions between CSV files and DataGrid
        - Ensures that data is loaded into the grid, with logical differentiation between string and numeric types
        - Offers ways for importing, exporting, and validating CSV data which includes managing edge cases, for example special characters or empty cells
    - Similar Classes
        - std::ifstream & std::ofstream
            - Standard file stream classes are used for reading and writing CSV files
        - std::stringstream

            - Used for parsing each line of the CSV file by splitting values using the delimiter
        - std::vector<std::vector<std::stream>>
            - Can be used to store tabular data in memory
        - Python's pandas library
            - Provides some robust CSV handling capabilities such as read_csv or to_csv for import/export
    - Key Functions
        - LoadCSV
            - Purpose: Reads a CSV file and interact with DataGrid
            - Parameters:
                - const std::string& fileName: The name of the CSV file to load
                - char delimiter = ',' The delimiter used in the CSV file (default is a comma)
            - Returns: a DataGrid populated with the parsed data from the CSV file.
            - Responsibilities:
                - Parsing the file line by line, splitting each line into individual values based on the provided delimiter and identifying whether each value is a string or double and populating the grid with Datum objects accordingly. Also, handling the edge cases. If a file can't be loaded, throw an exception.
        - ExportCSV
            - Purpose: Exports the contents of a DataGrid to a CSV file
            - Parameters:
                - const std::string& fileName: The name of the file to save.
                - const DataGrid& grid: The grid to export
            - Returns: bool (true or false)
            - Responsibilities:
                - Iterating through the grid rows and write values to the file, and cleaning/sanitizing the strings in the output file
    - Error Conditions
        - Loading Errors
            - Invalid file path (user error): File can't be opened or doesn't exist.
            - Data Type Ambiguity (user error): If a value can't be parsed as a string or double, decide whether to throw an error, skip, or use a default value.
        - Exporting Errors
            - Formatting errors (user error): Strings with some special characters (comma or quotes) can cause corruption for the CSV format.
    - Challenges
        - Differentiating Strings and Doubles when loading a csv file
    - Other Class Projects
        - ErrorManager (Group 6: Lamport)

                - Handling errors such as invalid file paths, corrupted files, or write permissions for the CSV file class to ensure a user-friendly error reporting system.
            - DataFileManager (Group 7: Euler)
                - It can handle the incremental updates efficiently if CSV needs to periodically update or log changes to a file.
- **ExpressionParser**
    - Class Description
        - Parses and performs mathematical expressions from string using given syntax
        - Must allow for simple equations using PEMDAS
    - Similar Classes
        - std::div()
        - std::exp()
    - Key Functions
        - bool assertEquationValidity(str equation)
            - Takes the string representing the equation
            - Returns a bool representing whether the equation is valid
        - referencedVector createPriorityQueue(str equation)
            - Private function
            - Takes the string representing the equation
            - Assigns priority and values to a priority queue, which will be used to run the equation in order
            - Returns the priority queue, likely a referenced vector
        - int evaluateEquation(str equation)
            - Takes the string representing the equation
            - Calls createPriorityQueue
            - Runs the equation in the order of the priority queue
            - Returns final value
    - Error Conditions
        - InvalidSyntax (user error)
            - If violate syntax, throws an exception
        - DivideByZero (user error)
            - If program tries to divide by zero, throw and exception
        - BitOverflow (recoverable error)
            - If result is too large and overflows, throw an error
    - Challenges
        - How to handle order of operations (priority queue?)
        - How to hold the values on each side of an operation as they change.
    - Other Class Projects
        - CommandLine (Group 6: Lamport)
            - Must follow instructions based on symbols similar to the expression parser