# Data Dictionary for Atmosphere Program

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Name* | *Type* | *Protection* | *Description* | *Rationale* |
| **Date** | **Class** |  | **Represents a calendar date using integers that represent when marks were obtained for a unit.** | **Separates the storing handling of dates to a separate object. Promotes modularity and reusability.** |
| M\_day | Integer | Private | The day of the month as a number. | Storing the day as an integer provides consistency in representation and avoids problems associated with strings such as capitalization or formatting. |
| M\_month | Integer | Private | The month as a number. | Storing the month as an integer provides consistency in representation and avoids problems associated with strings such as capitalization or formatting. |
| M\_year | Integer | Private | The year as a number. | Storing the year as an integer provides consistency in representation and avoids problems associated with strings such as capitalization or formatting. |
| GetDay() | Method | Public | Returns the value of the day variable as an integer. |  |
| SetDay(int day) | Procedure | Public | Sets the value of the day variable. |  |
| GetMonth() | Method | Public | Returns the value of the month variable as an integer. |  |
| SetMonth(int month) | Procedure | Public | Sets the value of the month variable. |  |
| GetYear() | Method | Public | Returns the value of the year variable as an integer. |  |
| SetYear(int year) | Procedure | Public | Sets the value of the year variable. |  |
|  |  |  |  |  |
| **Vector** | **Class** |  | **This class provides a basic implementation of a dynamic array similar to std::vector, with support for element insertion, access, copying, and dynamic resizing. It manages its own memory, grows based on size heuristics, and ensures safe access using bounds assertions. The class supports copy construction, assignment, and const-based indexing.** | **Encapsulates dynamically allocated array that manages memory safely, providing minimal but complete methods for insertion, resizing, access and modification, and copying and clearing memory. It promotes reusability and modular design through templating, allowing for the storage of any data type or class, in place of raw arrays.** |
| M\_data | T\* | Private | A dynamic array of T objects stored by the Vector class. | Serves as the key component of the Vector by providing storage of data by dynamically allocating memory on the heap. Provides flexibility and allows the Vector to be generic by allowing the storage of various data types, making it reusable for different contexts. |
| M\_size | Int | Private | The number of items currently in the internal array. | Separates logical size from allocated capacity, allowing for safe indexing and support of operations such as iteration and resizing. |
| M\_Capacity | Int | Private | The total amount of space available in the internal array. | Tracks the total size of the internal array, which, in turn, supports growth logic by allowing future insertions without the need for frequent reallocation of memory. Allows the vector to grow predictably as more elements are added, thereby improving performance. |
| Vector() | Constructor | Public | This constructor will construct a new object with size and capacity set to 0. | Provides a valid base state for the object without allocating memory prematurely. |
| Vector(int n) | Constructor | Public | This constructor will overload the default constructor, passing value for the capacity. It will then assign the capacity value and dynamically allocate space on the heap for the internal array using the capacity value. It also sets the value of size to 0. | Allows the caller to reserve memory for future insertions. |
| Vector(const Vector<T>& other) | Constructor | Public | This constructor will overload the default constructor, passing a Vector object. The passed object's data is copied to the constructed object's, and creating a new array on the heap that copies the data from the passed object's array. | Enables copying while preserving data independence between objects by making deep copies. |
| ~Vector() | Destructor | Public | This destructor will free the dynamically allocated memory on the heap and set the m\_data pointer to nullptr. | Prevents memory leaks by deallocating dynamic memory when the object is destroyed. |
| operator=(const Vector<T>&) | Operator | Public | This function overloads the assignment operator, copying the data in the other object into the assigned object. | Supports object assignment with memory safety and making separate deep copies. |
| operator[](int index) | Operator | Public | This function provides access to the element at the specified index in the Vector. The returned reference allows for modification of the element at that position. | Allows direct access and modification of stored data with bounds safety while ensuring safe bounds checking through assertions. This balances access convenience with safety for data manipulation. |
| operator[](int index) const | Operator | Public | This function provides read-only access to the element at the specified index in the Vector. The returned reference is const, ensuring that the element cannot be modified through this operator. | Enables element access in const contexts, ensuring data cannot be accidentally modified. This improves reliability and supports safe design. |
| PushBack(const T& val) | Procedure | Public | This function inserts the passed element into the end of the internal array and increases the size by 1. If size is greater than half the total capacity, the internal array is grown to have at least more than half capacity available. | Provides a safe and user-friendly way to append elements by automatically managing memory and growth internally, thus eliminating the need for users to handle allocation or growth logic. |
| GetSize() | Method | Public | Retrieves the current value of the size member variable as an integer. |  |
| GetCapacity() | Method | Public | Retrieves the current value of the capacity member variable as an integer. |  |
| Clear() | Procedure | Public | This function releases the memory previously allocated on the heap and sets the internal pointer to nullptr. It also resets the size and capacity values. | Serves as a re-usable and cohesive function for the deallocation of memory and resetting of private member values. This prepares the object for reuse or for clearing data when vector destructor is called. |
| Copy(const Vector<T>& other) | Procedure | Private | This function copies the private member values and dynamic array of another Vector object into the current object. It allocates a new array on the heap and copies the data from the passed Vector object. | Serves as a re-usable and cohesive function of the deep-copy logic to be used by copy constructor and assignment operator. |
|  |  |  |  |  |
| **MyTime** | **Class** |  | **This class represents a time value consisting of an hour and minute, using 24-hour format. It allows for setting and retrieving time values.** | **Encapsulates Time data into a single class to allow for separation of Time storage, reusability and modularity.** |
| M\_hour | Int | Private | The hour as a two-digit number in 24 hour format. | Storing the hour as an integer provides consistency in representation and avoids problems associated with strings such as capitalization or formatting. |
| M\_minute | Int | Private | The minute as a two-digit number. | Storing the minute as an integer provides consistency in representation and avoids problems associated with strings such as capitalization or formatting. |
| GetHour() | Method | Public | Retrieves the current value of the hour member variable as an integer. |  |
| SetHour(int hour) | Procedure | Public | Updates the hour member variable to the specified value. |  |
| GetMinute() | Method | Public | Retrieves the current value of the minute member variable as an integer. |  |
| SetMinute(int minute) | Procedure | Public | Updates the minute member variable to the specified value. |  |
|  |  |  |  |  |
| **AtmosRecType** | **Struct** |  | **Represents a atmospheric data record containing date, wind speed, wind temperature and solar radiation.** | **Stores data as a struct because AtmosRecType serves as a passive data holder for each record of atmospheric data. It contains no behavior, logic, or processing, as all operations are performed externally, typically through its use in a Vector. This design maintains high cohesion and low coupling.** |
| Date | Date | Public | Represents the numeric calendar date of the atmospheric measurement. In format: dd/mm/yyyy | Stores data into the class with the same name, allowing for consistent storage, modification, formatting, and access. |
| Time | MyTime | Public | Represents the time of day (in 24-hour format) when the measurement was taken. In format: HH:MM | Stores data into the class with the same name, allowing for consistent storage, modification, formatting, and access. |
| Speed | Float | Public | Records the wind speed in meters per second. | Floating-point precision allows readings to stored and represented accurately. Public access supports simplicity of data handling in the struct. |
| Temperature | Float | Public | Records the ambient air temperature in degrees Celsius. | Floating-point precision allows readings to stored and represented accurately. Public access supports simplicity of data handling in the struct. |
| Solar\_rad | Float | Public | Records solar radiation in W/m^2 for the date-time. | Floating-point precision allows readings to stored and represented accurately. Public access supports simplicity of data handling in the struct. |