# Atmosphere Program Documentation

## UML Design and Data Dictionary

**UML Design (as seen in Atmosphere.mdj):**

A diagram of a data flow

AI-generated content may be incorrect.

**Data Dictionary:**

The program’s design is structured around modularity, cohesion and reusability. Each component, from input and output handling, file reading and writing, data storage, and statistical calculations, was separated into appropriate classes, structs or functions to allow for clear division of responsibilities and ensures each component focuses on a single responsibility.

The Vector class was implemented to manage dynamically allocated arrays that growth automatically during run-time, while ensuring safe access and modification of elements. This replaces raw arrays with a reusable container that can be templated to any data type, allowing for generic storage.

Atmospheric data records were stored using the AtmosRecType struct, which serves as a passive data holder without internal behaviour, allowing logic to be external and flexible, which in the case of this program is done through a Vector of the struct.

Classes such as MyTime and Date encapsulate date and time data with appropriate mutators and access to allow for maintainability and separation of data storage.

Functions associated with file input and output, menu logic, and statistical calculations were relegated to their own files. Statistical calculations were made generic through template functions to allow for processing in contexts both associated with the atmosphere program as well as other programs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *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. |

## Algorithm in Pseudocode

Main()

1 open read src = “data/data\_source.txt”

2 if not(src)

3 error “src error”

4 while read src, inFilename

5 open read inFile = inFilename

6 if not(inFile)

7 error “inFile error”

8 else

9 ReadAtmosphereData(inFile, atmos\_data)

10 choice = DisplayMenu()

11 while choice != 5

12 if choice == 1

13 OptionOne(atmos\_data)

14 else if choice == 2

15 OptionTwo(atmos\_data)

16 else if choice == 3

17 OptionThree(atmos\_data)

18 else if choice == 4

19 OptionFour(atmos\_data)

20 choice = DisplayMenu()

ReadAtmohsphereData(file, atmosData)

1 if (not(read file, line) or

not(GetColumnIndices(line,wastIndex,sIndex,tIndex,srIndex))

2 error “Column missing”

3 while read file, line

4 ReadRowData(line,wastData,wastIndex,sData,sIndex,tData,tIndex,

srData,srIndex)

5 if not(wastData.length == 0)

6 space = -1

7 for i = 1 to wastData.length

8 if wastData[i] == ' '

9 space = i

10 break

11 for i = 1 to space

12 tempStr = tempStr + wastData[i]

13 ParseDataRecord(tempStr,dateTemp)

14 tempStr = “”

15 for i = space + 1 to wastData.length

16 tempStr = tempStr + wastData[i]

17 ParseTimeRecord(tempStr,timeTemp)

18 if sData != “” and sData != “N/A”

19 speedTemp = convertToFloat(sData)

20 else

21 speedTemp = -1

22 if tData != “” and tData != “N/A”

23 temperatTemp = convertToFloat(tData)

24 else

25 temperatTemp = -1

26 if srData != “” and srData = “N/A”

27 srTemp = convertToFloat(srData)

28 else

29 srTemp = -1

30 a.date = dateTemp

31 a.time = timeTemp

32 a.speed = speedTemp

33 a.temperature = temperatTemp

34 a.solar\_rad = srTemp

35 atmosData.PushBack(a)

GetColumnIndices(headerline,wastIndex,sIndex,tIndex,srIndex)

1 start = columnIndex = 0

2 end = -1

3 for i = 1 to headerline.length

4 if wastData[i] == ','

5 end = i

6 break

7 while end != -1

8 for i = start to end

9 tempStr = tempStr + wastData[i]

10 if data == “WAST”

11 wastIndex = columnIndex

12 else if data = “S”

13 sIndex = columnIndex

14 else if data = “T”

15 tIndex = columnIndex

16 else if data = “SR”

17 srIndex = columnIndex

18 start = end + 1

19 columnIndex = columnIndex + 1

20 return wastIndex != -1 && sIndex != -1 && tIndex != -1 && srIndex != -1

ReadRowData(line,wastdata,wastIndex,sData,sIndex,tData,tIndex,srData,srIndex)

1 start = columnIndex = 0

2 for columnIndex = 0 to 18 and (wastData.length != 0 or sData.length != 0 or

tData.length != 0 or srData.length != 0)

3 end = -1

4 for i = start to line.length

5 if line[i] == ‘,’

6 end = i

7 break

8 if end == -1

9 for i = start to line.length

10 data = data + line[i]

11 else

12 for i = start to end

13 data = data + line[i]

14 if columnIndex == wastIndex

15 wastData = data

16 else if columnIndex == sIndex

17 sData = data

18 else if columnIndex == tIndex

19 tData = data

20 else if columnIndex == srIndex

21 srData = data

22 if end == -1

23 start = line.length

24 else

25 start = end + 1

ParseDataRecord(dateStr, d)

1 for i = 0 to dateStr.length

2 if dateStr[i] == ‘/’

3 firstSlashIndex = i

4 break

5 for i = firstSlashIndex to dateStr.length

6 if dateStr[i] == ‘/’

7 secondSlashInd = i

8 break

9 tempStr = “”

10 for i = 0 to firstSlashInd

11 tempStr = tempStr + dateStr[i]

12 day = convertToInt(tempStr)

13 tempStr = “”

14 for i = firstSlashInd to secondSlashInd

15 tempStr = tempStr + dateStr[i]

16 month = convertToInt(tempStr)

17 tempStr = “”

18 for i = secondSlashInd to dateStr.length

19 tempStr = tempStr + dateStr[i]

20 year = convertToInt(tempStr)

21 d.SetDay(day)

22 d.SetMonth(month)

23 d.SetYear(year)

ParseTimeRecord(timeStr, t)

1 for i = 0 to timeStr.length

2 if timeStr[i] == ‘:’

3 colonInd = i

4 break

13 tempStr = “”

14 for i = 0 to colonInd

15 tempStr = tempStr + timeStr[i]

16 hour = convertToInt(tempStr)

13 tempStr = “”

14 for i = colonInd to timeStr.length

15 tempStr = tempStr + timeStr[i]

16 minute = convertToInt(tempStr)

17 t.SetHour(hour)

18 t.SetMinute(minute)

DisplayMenu()

1 write con,"1. Wind Speed Average and Standard Deviation for a Specified

Month and Year"

2 write con,"2. Air Temperature Average and Standard Deviation for Each Month

in a Specified Year"

3 write con,"3. Total Solar Radiation (in kWh/m^2) for Each Month in a

Specified Year"

4 write con,"4. Export monthly Wind, Temperature & Solar Radiation stats for a

Specified Year to WindTempSolar.csv"

5 write con,"5. Exit"

6 write con,”Option:”

7 read kb, choice

8 return choice

OptionOne(a)

1 month = PromptMonth()

2 year = PromptYear()

3 GatherSpeedValues(a, speedVec, year, month)

4 if (speedVec.GetSize() > 0)

5 mean = CalculateMean(speedVec,speedVec.GetSize())

6 stddev = CalculateStandardDeviation(speedVec,speedVec.GetSize(),mean)

7 write con, MonthToString(month), “ ”, year, “:”

8 write con, “Average Speed: “, (mean \* 3.6f), “km/h”

9 write con, “Sample stddev: “, stddev

10 else

11 write con, MonthToString(month), “ “, year, “: No Data”

OptionTwo(a)

1 year = PromptYear()

2 write con, year

3 for month = 1 to 12

4 tempVec.Clear()

5 GatherTempValues(a, tempVec, year, month)

6 if tempVec.GetSize() > 0

7 mean = CalculateMean(tempVec,tempVec.GetSize())

8 stddev = CalculateStandardDeviation(tempVec,tempVec.GetSize(),

mean)

9 write con, MonthToString(month), “:”

10 write con, “average: “, mean, “degrees C,”

11 write con, “stddev: “, stddev

12 else

13 write con, MonthToString(month), “: No Data”

OptionThree(a)

1 year = PromptYear()

2 write con, year

3 for month = 1 to 12

4 srVec.Clear()

5 GatherSolarRadValues(a, srVec, year, month)

5 if srVec.GetSize() > 0

6 total = CalculateTotal(srVec, srVec.GetSize())

7 write con, MonthToString(month), “:”, (total \* (10.0f / 60.0f) /

1000.0f), “kWh/m^2”

8 else

9 write con, MonthToString(month), “: No Data”

OptionFour(a)

1 year = PromptYear()

2 dataPrinted = false

3 open write out = “data/WindTempSolar.csv”

4 if not(out)

5 error “out error”

6 write out, year

7 for month 1 to 12

8 speedVec.Clear()

9 tempVec.Clear()

10 srVec.Clear()

11 GatherSpeedValues(a,speedVec,year,month)

12 GatherTempValues(a,tempVec,year,month)

13 GatherSolarRadValues(a,srVec,year,month)

14 if speedVec.GetSize()>0 or tempVec.GetSize()>0 or srVec.GetSize()>0

15 dataPrinted = true

16 write out, MonthToString(month), “,”

17 if speedVec.GetSize() > 0

18 mean = CalculateMean(speedVec,speedVec.GetSize())

19 stddev = CalculateStandardDeviation(speedVec,

speedVec.GetSize(),mean)

20 write out, mean, “(”, stddev, “),”

21 else

22 write out, “,”

23 if tempVec.GetSize() > 0

24 mean = CalculateMean(tempVec, tempVec.GetSize())

25 stddev = CalculateStandardDeviation(tempVec,

tempVec.GetSize(),mean)

26 write out, (mean \* 3.6), “(”, stddev, “),”

27 else

28 write out, “,”

29 if srVec.GetSize() > 0

30 total = CalculateTotal(srVec, srVec.GetSize())

31 write out, (total \* (10.0f / 60.0f) / 1000.0f)

32 else

33 write out, “\n”

34 if not(dataPrinted)

35 write out, “No Data”

PromptMonth()

1 write con, “Enter the month (1-12):”

2 read kb, m

3 while m <= 0 or m > 12

4 write con, “Enter the month (1-12):”

5 read kb, m

6 return m

PromptYear()

1 write con, “Enter the year:”

2 read kb, y

3 return y

MonthToString(monthNum)

1 invalid = “Invalid Month”

2 months[1..12] = ["January", "February", "March", "April", "May", "June",

"July", "August", "September", "October",

"November","December"]

3 if monthNum >= 1 and monthNum <= 12

4 return months[monthNum]

5 else

6 return invalid

GatherSpeedValues(source,vec,year,month)

1 for i = 0 to source.GetSize()

2 if source[i].date.GetYear() == year &&

source[i].date.GetMonth() == month &&

source[i].speed != -1.0f

3 vec.PushBack(source[i].speed)

GatherTempValues(source,vec,year,month)

1 for i = 0 to source.GetSize()

2 if source[i].date.GetYear() == year &&

source[i].date.GetMonth() == month &&

source[i].temperature != -1.0f

3 vec.PushBack(source[i].temperature)

GatherSolarRadValues(source,vec,year,month)

1 for i = 0 to source.GetSize()

2 if source[i].date.GetYear() == year &&

source[i].date.GetMonth() == month &&

source[i].solar\_rad >= 100.0f

3 vec.PushBack(source[i].solar\_rad)

CalculateMean(vec, total)

1 sum = 0

2 for i = 0 to total

3 sum = sum + vec[i]

4 return sum/total

CalculateStandardDeviation(vec, N, mean)

1 sum = 0

2 for i = 0 to N

3 diff = vec[i] – mean

4 sum = sum + (diff \* diff)

5 return sqrt(sum/(N – 1))

CalculateTotal(vec, size)

1 sum = 0

2 for i = 0 to size

3 sum = sum + vec[i]

4 return sum

PushBack(val)

1 if m\_capacity == 0

2 m\_capacity = 1

3 m\_data = allocate array of size m\_capacity

4 m\_data[m\_size] = val

5 m\_size = m\_size + 1

6 if m\_size > (m\_capacity / 2)

7 newCapacity = m\_capacity + m\_size + 1

8 newData = allocate array of size newCapacity

9 for i = 0 to m\_size

10 newData[i] = m\_data[i]

11 deallocate m\_data

12 m\_data = newData

13 m\_capacity = newCapacity

Operator=(other)

1 if this != other

2 Clear()

3 Copy(other)

4 return this

Operator[](index)

1 assert(index < m\_size and index >= 0)

2 return m\_data[index]

Clear()

1 deallocate m\_data

2 m\_data = NIL

3 m\_size = 0

4 m\_capacity = 0

Copy(other)

1 m\_size = other.GetSize()

2 m\_capacity = other.GetCapacity()

3 m\_data = allocate memory of size m\_capacity

4 for i = 0 to m\_size

5 m\_data[i] = other[i]

## Test Plan

The testing approach for this program is structured around two major parts: the overall program and the three classes used by the program. Each class was unit tested, including tests for default and overloaded constructors, getters, setters, as well as methods and internal validation unique to each class such as PushBack in Vector. The program was then tested to ensure that it functions according to the required specifications such as ensuring that each menu option outputs the expected results.

* Program Test Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description (and why) | Actual Test Data | Expected Output | Pass/Fail |
| 1 | Option 1 prints wind speed average and standard deviation for valid month and year. | Test1-6.csv file containing speed data for October 2015.  Option 1 input:  Month: 10  Year: 2015 | October 2015:  Average Speed: 24.1 km/h  Sample stddev: 5.4 | Pass |
| 2 | Option 1 prints “No Data” for invalid month and/or year. | Test1-6.csv file containing speed data for October 2015.  Option 1 input:  Month: 4  Year: 2005 | April 2005: No Data | Pass |
| 3 | Option 2 prints the monthly temperature average and standard deviation for a valid year, or “No Data” for months without data. | Test1-6.csv file containing temperature data for April and October 2015.  Option 2 input:  Year: 2015 | 2015  January: No Data  …  April: average: 18.6 degrees C, stddev: 0.8  …  October: average: 12.3 degrees C, stddev: 7.2  …. | Pass |
| 4 | Option 3 prints the monthly total solar radiation for a valid year, or “No Data” for months without data. | Test1-6.csv file containing temperature data for 2015.  Option 3 input:  Year: 2005 | 2015  January: No Data  …  April: 0.7 kWh/m^2  …  October: 0.8 kWh/m^2  …. | Pass |
| 5 | Option 4 exports data to WindTempSolar.csv file for a valid year. | Test1-6.csv file containing solar radiation data for 2015.  Option 4 input:  Year: 2015 | 2015  April,38.4(1.4),18.6(0.8),0.7  October,24.1(5.4),12.3(7.2),0.8 | Pass |
| 6 | Option 4 writes “No Data” in WindTempSolar.csv for an invalid year. | Test1-6.csv file containing solar radiation data for 2015.  Option 4 input:  Year: 2005 | 2005  No Data | Pass |
| 7 | Option 4 exports data to WindTempSolar.csv file for a valid year with columns that are missing values and no data for every month. | Test7.csv file with no data available for solar radiation and only contains data for March 2015. | 2015  March,31.4(2.6),24.5(0.6), | Pass |
| 8 | Program ignores Solar Radiation values below 100 during file reading. | Test8.csv contains a mix of solar radiation values above and below 100. | 2015  …  October: 0.2 kWh/m^2  …. | Pass |
| 9 | Program handles missing or unreadable input file gracefully without crashing. Ensures robustness when data source is missing or filename in data\_source.txt is invalid. | Data\_source.txt contains Random.csv | Process returns -1. | Pass |
| 10 | Option 5 exits the program without error. | Entering 5 when prompted for menu choice. | Process returns 0. | Pass |
| 11 | Program loads data from multiple valid files listed in data\_source.txt (random order). | Data\_source.txt contains:  Metdata-Jan-Dec2007.csv  Metdata-Jan-Dec2008.csv  Metdata-Jan-Dec2009.csv  Metdata-Jan-Dec2016.csv  Option 3 input:  Year: 2007 | 2007  January: 237.9 kWh/m^2  …  March: No Data  …  December: 236.3 kWh/m^2 | Pass |
| 12 | Program skips unreadable or missing files and continues loading other valid ones. | Data\_source.txt contains:  Metdata-Jan-Dec2007.csv  Metdata-Jan-Dec2008.csv  Missing.csv  Metdata-Jan-Dec2009.csv  Metdata-Jan-Dec2016.csv  Option 3 input:  Year: 2016 | Unable to open input file missing.csv  (Menu Printed to Console)  2016  January: 223.4 kWh/m^2  …  December: 262.4 kWh/m^2 | Pass |
| 13 | Program handles an empty data\_source.txt file without crashing. | Data\_source.txt is empty.  Option 3 input:  Year: 2016 | 2016  January: No Data  …  December: No Data | Pass |
| 14 | Program handles a valid but empty CSV file. | Data\_source.txt contains:  Test14.csv  Option 2 input:  2015 | 2015  January: No Data  …  December: No Data | Pass |

* VectorTest.cpp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description (and why) | Actual Test Data | Expected Output | Pass/Fail |
| 1 | Default constructor sets size and capacity to 0. | N/A – Default constructor called. | Size = 0  Capacity = 0 | Pass |
| 2 | Parameterized constructor sets initial capacity. | Vector<int> v(5) | Size = 0  Capacity = 5 | Pass |
| 3 | PushBack() adds elements correctly and updates size. | Vector<int> v(5)  v.PushBack(1) | V[0] = 1  Size = 1  Capacity = 5 | Pass |
| 4 | PushBack() increases capacity when needed. | Vector<int> v(4)  v.PushBack(1)  v.PushBack(2)  v.PushBack(3) | V[0] = 1  V[1] = 2  V[2] = 3  Size = 3  Capacity = 8 | Pass |
| 5 | Copy constructor copies all elements, size, and capacity. | Vector<int> v1(5)  v1.PushBack(1)  v1.PushBack(2)  Vector<int> v2(v1) | V1[0] = 1  V1[1] = 2  Size = 2  Capacity = 5  V2[0] = 1  V2[1] = 2  Size = 2  Capacity = 5 | Pass |
| 6 | Assignment operator copies all elements, size, and capacity. | Vector<int> v1(5)  v1.PushBack(1)  v1.PushBack(2)  Vector<int> v2 = v1 | V1[0] = 1  V1[1] = 2  Size = 2  Capacity = 5  V2[0] = 1  V2[1] = 2  Size = 2  Capacity = 5 | Pass |
| 7 | Subscript ([]) operator returns reference to element. | Vector<int> v1(5)  v1.PushBack(1) | Memory address and values of V[0] are printed correctly | Pass |
| 8 | GetSize() returns the number of inserted elements. | Vector<int> v1(5)  v1.PushBack(1)  v1.PushBack(2) | Size = 2 | Pass |
| 9 | GetCapacity() returns correct capacity after expansion. | Vector<int> v1(5)  v1.PushBack(1)  v1.PushBack(2) | Capacity = 5 | Pass |
| 10 | Copy() in Assignment Operator and Copy Constructor creates a deep copy of the Vector object. | Vector<int> v1(5)  v1.PushBack(1)  v1.PushBack(2)  Vector<int> v2 = v1  Vector<int> v3(v1)  V1[0] = 99 | Memory addresses of the same elements (v1[0], v2[0], v3[0]) are different. The value of v1 at index 0 is different to v2 and v3. | Pass |
| 11 | Vector object is copied correctly when passed by value. | Vector<int> v1(5)  v1.PushBack(1)  V\_copy[0] = 2 (in function) | V\_copy[0] = 2  V1[0] = 1 | Pass |
| 12 | Vector object is correctly modified when passed by non-const reference. | Vector<int> v1(5)  v1.PushBack(1)  V\_ ref[0] = 2 (in function) | V\_ ref[0] = 2  V1[0] = 1 | Pass |
| 13 | Vector object data can be accessed but not modified when passed by const reference. | Vector<int> v1(5)  v1.PushBack(1)  V\_ const\_ref[0] = 2 (in function) | Compiler prints an error when setter in PassByConstRef() is left uncommented and called. | Pass |
| 14 | Vector is correctly returned from a function by value. | Vector object in ReturnObjectCopy()  Vector<int> v(5)  v.PushBack(1) | V[0] In Function: 1  V[0] From Return Function: 1 | Pass |
| 15 | Accessing element at an index greater than size using subscript ([]) operator throws an assertion error. | Vector<int> v(5)  v.PushBack(1)  v[3] = 2 | Compiler prints an error when setter in “v[3] = 2” is left uncommented and called. | Pass |
| 16 | PushBack() handles insertion correctly when capacity is 0 (ie. Default constructor is called for new vector object). | Vector<int> v  v.PushBack(1) | V[0] = 1  Size: 1  Capacity: 3 | Pass |
| 17 | Read, store and print integer data from a file. | 45  2213  235  877  1215  321445  32167  4332 | 45  2213  235  877  1215  321445  32167  4332 | Pass |
| 18 | Read, store and print Date data from a file. | 01/01/2020  15/03/2021  31/12/2019  29/02/2024  04/07/2022  10/10/2010  25/12/2025  30/06/2023  11/11/2011  05/05/2005 | 01/01/2020  15/03/2021  31/12/2019  29/02/2024  04/07/2022  10/10/2010  25/12/2025  30/06/2023  11/11/2011  05/05/2005 | Pass |
| 19 | Read, store and print Unit data from a file. | ICT100 IntroProgramming 3  ICT159 FundamentalsCS 3  ICT283 DataStructures 3  ICT374 Algorithms 3  ICT289 SoftwareEng 3  ICT207 WebDev 3  ICT302 CyberSecurity 3  ICT115 Databases 3  ICT203 OOP 3  ICT210 OSFundamentals 3 | ICT100 IntroProgramming 3  ICT159 FundamentalsCS 3  ICT283 DataStructures 3  ICT374 Algorithms 3  ICT289 SoftwareEng 3  ICT207 WebDev 3  ICT302 CyberSecurity 3  ICT115 Databases 3  ICT203 OOP 3  ICT210 OSFundamentals 3 | Pass |
| 20 | Constructing a vector using capacity < 0 will not work. | Vector<int> v(-1) | Compiler prints an error when improper constructor is left uncommented and called. | Pass |

* DateTest.cpp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description (and why) | Actual Test Data | Expected Output | Pass/Fail |
| 1 | Default constructor sets marks to 0. | N/A – Default constructor called. | Day: 1  Month: 1  Year: 1 | Pass |
| 2 | Parameterized constructor correctly sets day, month and year. | Day: 13  Month: 3  Year: 2005 | Day: 13  Month: 3  Year: 2005 | Pass |
| 3 | SetDay() correctly updates the day. | Old Day: 13  New Day: 14 | Day Before Setting: 13  Day After Setting: 14 | Pass |
| 4 | GetDay() correctly returns the updated day. | Day: 13 | Day: 13 | Pass |
| 5 | SetMonth() correctly updates the month. | Old Month: 3  New Month: 4 | Month Before Setting: 3  Month After Setting: 4 | Pass |
| 6 | GetMonth() correctly returns the updated month. | Month: 3 | Month: 3 | Pass |
| 7 | SetYear() correctly updates the year. | Old Year: 2005  New Year: 2025 | Year Before Setting: 2005  Year After Setting: 2025 | Pass |
| 8 | GetYear() correctly returns the updated year. | Year: 2005 | Year: 2005 | Pass |
| 9 | SetDay() sets an invalid day due to lack of validation checking. | Old Day: 13  New Day: 33 | Day: 33 | Pass |
| 10 | SetMonth() sets an invalid month due to lack of validation checking. | Old Month: 3  New Month: 15 | Month: 15 | Pass |
| 11 | SetYear() sets an invalid year due to lack of validation checking. | Old Year: 2005  New Year: -2005 | Year: -2005 | Pass |
| 12 | Date object is copied correctly when passed by value. | D1 Date Object  Day: 13  Month: 3  Year: 2005  D\_copy Date Object (pass by value)  Day: 13  Month: 3  Year: 2005  D\_ copy modified Date Object  Day: 15  Month: 3  Year: 2005 | D1 and D\_copy (after modification) print different days  Day of d\_copy: 15  Day of d1: 13 | Pass |
| 13 | Date object is correctly modified when passed by non-const reference. | D1 Date Object  Day: 13  Month: 3  Year: 2005  D\_ref Date Object (pass by reference)  Day: 13  Month: 3  Year: 2005  D\_ref modified Date Object  Day: 15  Month: 3  Year: 2005 | D1 and D\_copy (after modification) print same days  Day of d\_copy: 15  Day of d1: 15 | Pass |
| 14 | Date object data can be safely accessed without modification when passed by const reference. | D1 Date Object  Day: 13  Month: 3  Year: 2005  D\_const\_ref Date Object  Day: 13  Month: 3  Year: 2005  D\_const\_ref attempts to change the day to 14 in the function | Compiler prints an error when setter in PassByConstRef() is left uncommented and called. | Pass |
| 15 | Date object is correctly copied and returned from a function via return-by-value. | Date Object in ReturnObjectCopy()  Day: 13  Month: 3  Year: 2005 | Day In Function: 13  Day From Return Function: 13 | Pass |

* MyTimeTest.cpp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description (and why) | Actual Test Data | Expected Output | Pass/Fail |
| 1 | Default constructor sets hour and minute to 0. | N/A – Default constructor called. | Time: 0:0 | Pass |
| 2 | Parameterized constructor stores hour and minute. | Hour: 4  Minute: 35 | Time: 4:35 | Pass |
| 3 | SetHour() updates hour value. | Old Hour: 4  New Hour: 12 | Hour Before Setting: 4  Hour After Setting: 12 | Pass |
| 4 | SetMinute() updates minute value. | Old Minute: 35  New Minute: 50 | Minute Before Setting: 35  Minute After Setting: 50 | Pass |
| 5 | GetHour() correctly returns the hour value. | Hour: 4 | Hour: 4 | Pass |
| 6 | GetMinute() correctly returns the minute value. | Minute: 35 | Minute: 35 | Pass |
| 7 | Can store arbitrary values outside 24h format. | Hour: 123  Minute: 90 | Time: 123:90 | Pass |
| 8 | Time object is copied correctly when passed by value. | T1 MyTime Object  Hour: 4  Minute: 35  T\_copy MyTime Object (pass by value)  Hour: 4  Minute: 35  T\_ copy modified MyTime Object  Hour: 5  Minute: 45 | Time in T\_copy: 5:45  Time in T1: 4:35 | Pass |
| 9 | Time object is correctly modified when passed by non-const reference. | T1 MyTime Object  Hour: 4  Minute: 35  T\_ref MyTime Object (pass by reference)  Hour: 4  Minute: 35  T\_ ref modified MyTime Object  Hour: 5  Minute: 45 | Time in T\_ ref: 5:45  Time in T1: 5:45 | Pass |
| 10 | Time object data can be safely accessed without modification when passed by const reference. | T1 MyTime Object  Hour: 4  Minute: 35  T\_const\_ref MyTime Object  Hour: 4  Minute: 35  T\_const\_ref attempts to change the hour to 14 in the function | Compiler prints an error when setter in PassByConstRef() is left uncommented and called. | Pass |
| 11 | Time object is correctly copied and returned from a function via return-by-value. | MyTime Object in ReturnObjectCopy()  Hour: 4  Minute: 35 | Time in Function: 4:35  Time From Return Function: 4:35 | Pass |

## Output of Tests

* Program Test

|  |  |
| --- | --- |
| Test ID | Screenshot of output |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |

* VectorTest.cpp

|  |  |
| --- | --- |
| Test ID | Screenshot of output |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |

* DateTest.cpp

|  |  |
| --- | --- |
| Test ID | Screenshot of output |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |

* MyTimeTest.cpp

|  |  |
| --- | --- |
| Test ID | Screenshot of output |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |