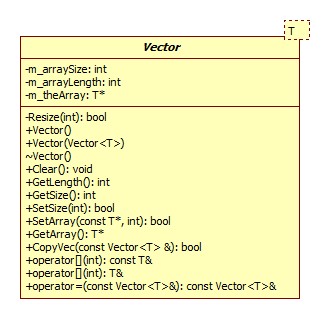
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Protection | Description | Rational |
| m\_arraySize | int | private | An int to store the size of the array | Allows the user to see how large the dynamic array is. |
| m\_arrayLength | int | private | An int to store the length of the array | I added this so it’s possible to loop through just the length that is being used as opposed to the entire size of the array. |
| m\_theArray | T\* | private | A pointer to allow access to a dynamic array. | Needed to have a dynamic array. |
| Resize(int) | bool | private | Used to Resize the array keeping the old data. If the Resize is smaller than the current array or the Resize fails it returns false. | I chose to make this private so it is only called for the overloaded [] operator. There is no need for the user to access it. This is not the most efficient function as it makes a copy of the original array first then resizes and then copies it back. |
| Vector() |  | public | Sets size and length to 0 the pointer to NULL. | Default constructor |
| ~Vector() |  | public | Destructor. Runs Clear(). See below. | Needed. |
| Vector(Vector<T>) |  | public | Copy constructor | Needed to stop the default one from shallow copying. |
| Clear() | void | public | Clears length and size, if there is an array it deletes it then sets the pointer to NULL. | Used to empty the Vector. Handy if you don’t want to create keep creating Vector objects. |
| SetSize(int) | bool | public | Clears the array if it’s there and creates a new array of the desired size. If this fails it returns false. | Needed to set the size of the array. If there is an array already it clears it and creates a new one. It checks to ensure the memory was created. |
| CopVec(const Vector<T>&) | bool | public | Copies one Vector to another as long as it’s not the same thing. | This is required for the overloaded = operator and the copy constructor to help prevent a shallow copy of the Vector. It checks to make sure it doesn’t self-copy. |
| GetArray() const | T\* | public | Copies the array contained in the Vector and return a pointer to it. | Used to Get the array. Needed for the Copy function and resize. |
| SetArray(const T\*, int) | bool | public | Sets the array. If the Vector is big enough it will copy it in otherwise it will resize and copy. Returns false if it fails. Use Clear() to save memory before the copy. | This is needed to set the array. I’ve allowed to set if the Vector is larger than the array. This may waste memory but allows for my Resize function to work and if memory is an issue Clear() can be called before setting the array. |
| GetSize() | int | public | Returns the size by value. | Basic getter. |
| GetLength() | int | public | Returns the length by value. | Basic getter. |
| operator [] (int) const | Const T& | public | Used to get an index. Has bounds checking. | Used for const access. Does not resize and write access is denied. I used assert from the cassert header to do the bounds checking. I’m not sure if that’s the best way to do it. |
| operator [] (int) | T& | public | Used to get an index. Has bounds checking for below zero but none for above. It will automatically resize is the index is greater than the size. | I only have bounds checking for under zero so when retrieving from the Vector it will resize and retrieve “garbage”. I wasn’t sure how to prevent this so I left it in. But otherwise it functions well. |
| operator =(const Vector<T>&) | const Vector<T> & | public | Used to copy another Vector | Needed to prevent a shallow copy from the default version. Calls CopVec(). |

**UML**



**ALGORITHMS**

void Clear()

m\_arraySize = 0

m\_arrayLength = 0

if m\_theArray is not equal to NULL

delete all the elemnts that m\_theArray points to

m\_theArray = NULL

bool SetSize(int arrSize)

if m\_theArray is not NULL

run Clear()

if m\_theArray is equal to NULL and arrSize is greater than 0

set m\_arraySize to size

create dynamic array of size arrSize.

if m\_theArray is still NULL

return false

else

return true

bool CopyVec(const Vector<T> &newVec)

if newVec is not this object

run Clear()

m\_arrayLength = newVec.GetLength()

m\_arraySize = newVec.GetSize()

m\_theArray = newVec.GetArray()

return true

else

return false.

T\* GetArray()

newT is a T\*

create a dynamic array the size of m\_theArray

store the address in newt

for i = 0 to m\_arrayLength

newt[i]= m\_theArray[i]

return newt

bool SetArray(const T \*newArr, int siz)

if siz is greater than 0 and less than or equal to m\_arraySize

for i = 0 to siz

m\_theArray[i] = newArr[i]

m\_arrayLength = siz

return true

else if siz is greater than 0 and SetSize() returns true

for i = 0 to siz

m\_theArray[i] = newArr[i]

m\_arrayLength + 1

return true

else

return false

const operator[](int index) const

if index is less than 0 or greater than m\_arraySize throw an exception

return m\_theArray[index]

operator[](int index)

if index is less than 0 throw an exception

if m\_theArray == NULL

run SetSize(index + 1)

while m\_arraySize is less than or equal to index

run Resize(2 \* m\_arraSize)

if index is greater than or equal to m\_arrayLength

m\_arrayLength = index + 1

return m\_theArray[index]

bool Resize(int newSize)

if newSize is greater than m\_arraySize

T \*temp = GetArray();

int templength = m\_arrayLength

return true if

SetSize() and SetArray return true

else

return false