Assignment 5: IntVector

Collaboration Policy

You may not use code from any source (another student, a book, online, etc.) within your solution to this assignment. In fact, you may not even look at another student's solution or partial solution to this assignment. You also may not allow another student to look at any part of your solution to this exercise. You should get help on this assignment by coming to the instructor's or TA's office hours or by posting questions on Piazza (you still must not post assignment code publically on Piazza.) See the full Course Collaboration Policy here: Collaboration Policy

---

Assignment specs:

You are going to build a smaller, simpler version of the STL vector class that is capable of only storing integers. Appropriately, it will be called IntVector. This class will encapsulate a dynamically allocated array of integers.

You are required to come up with a header file (IntVector.h) that declares the class interface and a separate implementation file (IntVector.cpp) that implements the member functions. You will also create a test harness (main.cpp) that you use to test each function you are currently developing. **Never implement more than 1 or 2 member functions without fulling testing them with your own test harness.** The final test harness (main.cpp) file should include all of the tests you performed on each member function. Feel free to comment out previous tests while you are working on later member functions. However, the final test harness should test ALL of your member functions thoroughly.

Encapsulated (Private) Data Fields

- sz: unsigned
- cap: unsigned
- data: int *

Public Interface (Public Member Functions)

- IntVector()
- IntVector(unsigned size)
- IntVector(unsigned size, int value)
- ~IntVector()

- unsigned size( ) const
- unsigned capacity( ) const
- bool empty( ) const
- const int & at(unsigned index) const
- int & at(unsigned index)
- void insert(unsigned index, int value)
- void erase(unsigned index)
- const int & front( ) const
- int & front( )
- const int & back( ) const
- int & back( )
- void assign(unsigned n, int value)
- void push_back(int value)
- void pop_back( )
- void clear( )
- void resize(unsigned size)
- void resize(unsigned size, int value)
- void reserve(unsigned n)

Private Helper Functions

- void expand()
- void expand(unsigned amount)

You will not be implementing iterators. On the most part, though, the above functions should work just like the STL vector class member function with the same name. Please see the following pages to get a better idea of what each function should do: STL Vector.

Constructors and Accessors

Constructors

IntVector( ) - the default constructor

This function should set both the size and capacity of the IntVector to 0 and will not allocate any memory to store integers (make sure you do not have a dangling pointer).

IntVector(unsigned size)

Sets both the size and capacity of the IntVector to the value of the parameter passed in and dynamically allocates an array of that size as well. (Don't forget to initialize all values in your array to 0.)

IntVector(unsigned size, int value)

Sets both the size and capacity of the IntVector to the value of the parameter passed in and dynamically allocates an array of that size as well. This function should initialize all elements of the array to the value of the 2nd parameter.

Accessors

unsigned size( ) const

This function returns the current size (not the capacity) of the IntVector object, which is the value stored in the sz data field.

unsigned capacity( ) const

This function returns the current capacity (not the size) of the IntVector object, which is the value stored in the cap data field.

bool empty( ) const

Returns whether the vector is empty (the sz field is 0).

const int & at(unsigned index) const

This function will return the value stored in the element at the passed in index position. Your function should cause the program to exit if an index greater than or equal to the size is passed in. Because the index is an unsigned int (rather than just int) the compiler will not allow for a negative value to be passed in.

const int & front( ) const

This function will return the value stored in the first element.

const int & back( ) const

This function will return the value stored in the last element.

Destructor, Mutators, & Private Helper Functions

Destructor

~IntVector( )

This function should deallocate all remaining dynamically allocated memory.

void expand()

This function will double the capacity of the vector. This function should reallocate memory for the dynamically allocated array and update the value of capacity.

Make sure you don't create a memory leak here.

void expand(unsigned amount)

This function will expand the capacity of the vector by amount passed in. This function should reallocate memory for the dynamically allocated array and update the value of capacity.

Make sure you don't create a memory leak here.

void insert(unsigned index, int value)

This function inserts data at location index. To do this, all values currently at position index and greater are shifted to the right by 1 (to the element right after its current position).

Size is increased by 1. If size becomes larger than capacity, this function needs to double the capacity. In other words, if capacity and size are both 20 when this function is called, capacity will become 40 and size will be set to 21.

Since other functions will also potentially need to expand (double) the capacity, implement the private helper function named expand (see above) to do this for you. This function should call exit(1) if an index value greater than sz is passed in.

void erase(unsigned index)

This function removes the value at position index and shifts all of the values at positions greater than index to the left by one (to the element right before its current position).

Size is decreased by 1. This function should call exit(1) if an index value greater than or equal to sz is passed in.

void push_back(int value)

This function inserts a value at the back end of the array.

Size is increased by 1. If size becomes larger than capacity, this function needs to double the capacity. In other words, if capacity and size are both 20 when this function is called, capacity will become 40 and size will be set to 21.

Since other functions will also potentially need to expand (double) the capacity, implement the private helper function named expand (see above) to do this for you.

void pop_back()

This function just needs to decrease size by 1.
This function should call exit(1) if pop_back is called on an empty vector.

void clear()

This function reduces the size of the vector to 0.

void resize(unsigned size)

This function resizes the vector to contain size elements.

If size is smaller than the current size(sz), this function just reduces the value of sz to size.

If size is greater than the current size(sz), then the appropriate number of elements are inserted at the end of the vector, giving all of the new elements a value of 0.

If the new value of size will be larger than capacity, then the capacity should be expanded by either doubling (expand()) or by increasing the capacity by a specific amount (expand(size - cap)), whichever results in the largest capacity.

void resize(unsigned size, int value)

This function resizes the vector to contain size elements.

If size is smaller than the current size (sz), this function just reduces the value of sz to size.

If size is greater than the current size (sz), then the appropriate number of elements are inserted at the end of the vector, giving all of the new elements the value passed in through the 2nd parameter (value).

If the new value of size will be larger than capacity, then the capacity should be expanded by either doubling (expand()) or by increasing the capacity by a specific amount (expand(size - cap)), whichever results in the largest capacity.

void reserve(unsigned n)

This function requests that the capacity (the size of the dynamically allocated array) be set to n ***at minimum***. This informs the vector of a planned increase in size, although notice that the parameter n informs of a minimum, so the resulting capacity may be any capacity equal or larger than this.

This function should **NOT** ever reduce the size of the vector. If n is larger than the current capacity then the capacity should be expanded by either doubling (expand()) or by increasing the capacity by a specific amount (expand(n - cap)), whichever results in the largest capacity.

In any case, a call to this function never affects the elements contained in the vector, nor the vector size.

void assign(unsigned n, int value)

Assigns new content to the vector object, dropping all the elements contained in the vector before the call and replacing them by those specified by the parameters. The new value of size will be n and all values stored in the vector will have the value of the 2nd parameter.

If the new value of size will be larger than capacity, then the capacity should be expanded by either doubling (expand()) or by increasing the capacity by a specific amount (expand(n - cap)), whichever results in the largest capacity.

int & at(unsigned index)

This function will return the value stored at the element at the passed in index position. Your function should cause the program to exit (exit(1)) if an index greater than or equal to the sz is passed in. Because the index is an unsigned int (rather than just int) the compiler will not allow for a negative value to be passed in.

int & front( )

This function will return the value stored at the first element.

int & back( )

This function will return the value stored at the last element.

Compiling and Testing

You can compile the entire program, generating an executable named a.out, with the following command line:

```
g++ -W -Wall -Werror -ansi -pedantic main.cpp IntVector.cpp
```

But when you want to compile the class seperately, without generating an executable, use the command line:

```
g++ -W -Wall -Werror -ansi -pedantic -c IntVector.cpp
```

And then to generate the executable once you've already compiled the IntVector class separately, use the command line:

```
g++ -W -Wall -Werror -ansi -pedantic main.cpp IntVector.o
```

What to Submit

For this assignment you will turn in the following files (case sensitive) to R'Sub (galah.cs.ucr.edu):

- main.cpp (test harness)
- IntVector.h
- IntVector.cpp
