CSCI 340 Assignment 1 – Dynamic Array Creation

Abstract

In this assignment, you will write a C++ program that will create a vector like Dynamic Array. The objective of this assignment is to give students an understanding of how memory allocation works and how templated types work. Overall, this will show the base functionality of the STL vector class.

1 What is an STL vector?

The STL (Standard Template Library) `vector` container in C++ is a dynamic array that provides efficient random access to its elements while automatically managing memory to accommodate growth. It stores elements in contiguous memory locations, enabling constant-time access via indexing, similar to arrays. Unlike arrays, `vector` can dynamically resize itself when elements are added or removed, handling memory allocation and reallocation seamlessly. It supports a wide range of operations, including inserting, deleting, and iterating over elements (although you will not implement this feature), and integrates well with STL algorithms. Vectors are highly versatile and provide functions like `push_back`, `pop_back`, `size`, `capacity`, and `reserve` to manage and optimize their behavior efficiently, making them a cornerstone of modern C++ programming.

2 Files You Must Write

You will write a C++ program suitable for execution on `hopper.cs.niu.edu` (or `turing.cs.niu.edu`). We have provided substantial starter code for you. However, there are some functions that are not present that you must implement.

A MakeFile *MUST* be present or you will receive zero points for this assignment. You may refer to assignment 0, and the numerous resources provided to create a makefile.

- `makefile`
- `dynamicarray.h`
- `dynamicarray.cpp`

2.1 `dynamicarray.h` and `dynamicarray.cpp`

The purpose of this class is to create a templated dynamic array that will work with the main driver program. To do this, you will implement numerous member functions of the `DynamicArray` class.

Recall that operator overloading gives certain operators new functionality when called on an object. You will be provided with skeletons of some operators, but must provide others that are required for the program to run. If you would like a more in-depth refresher about these concepts, you may use this: `https://www.geeksforgeeks.org/operator-overloading-cpp/`, or `https://en.wikipedia.org/wiki/Operator_overloading`.

The `DynamicArray` class is a templated container class which means it can store any data type inside of it. Templates in C++ are a powerful feature that allows the creation of generic and reusable code. They enable functions, classes, or entire algorithms to work with any data type, reducing redundancy and enhancing flexibility. Templates are defined using the `template` keyword, followed by one or more type parameters enclosed in angle brackets (e.g., `template<typename T>`). The compiler generates specific implementations for the required types during compilation, ensuring type safety and performance equivalent to manually written code. Templates are widely used for creating type-independent containers (like those in the Standard Template Library), generic algorithms, and utility classes, making them an essential tool for writing efficient and maintainable C++ programs. For a more in-depth refersher on templates in c++ use this link: https://www.geeksforgeeks.org/templates-cpp/

DynamicArray class:

- `DynamicArray()`

  Default constructer. This function should set data to `nullptr`, capacity to 0, and size to 0;

- `DynamicArray(const int length, const int value)`

  Integer Constructer. This function will use `new` to create an array of size `length` and fill it with `value`. Additionally, it will set capacity to length, and size to length.

- `DynamicArray(const string& value)`

  Character Constructer. This function takes the string `value` and create an array that stores the characters of `value` in the appropriate index. You should get the length of the string; set the capacity and size to that length. You should then allocate memory using `new` to create an array of size `length`. Finally, you should loop through all of the characters in `value` and copy them at the same index in `data`.

- `~DynamicArray()`

  Destructor. This function will free memory by using `delete` to delete the data array.

- `changeCapacity(size_t newCapacity)`

  This function should increase the capacity of `data` to the length of `newCapacity`. You should do this by creating a `new` temporary array of size `newCapacity`, copying `data` into the temporary array, deleting `data`, and setting `data` to the temporary array.

  After you copy over data, you should set the capacity to `newCapacity`.

- `void push_back(const T& value)`

  This function should add `value` to the back to the array. If the current size is equal to capacity, and the capacity of `DynamicArray` is 0, call `changeCapacity()` to set the capacity to 1. Otherwise, call `changeCapacity()` and set the capacity to double the current capacity (`capacity * 2`).

  Afterwards, place `value` at that index `size`, then increment `size`.

- `void pop_back();`

  This function “removes” the last element in the array. To do this, check if the array is empty (`size==0`). If it is, you should `cout` “Array is empty\n” and then `return`. Otherwise, decrement size.

  Note: you do not need to `delete` anything with this function. Reducing size ensures that random access cannot reach it.

- `get_size() const`

  returns the size of the array. Note: The constant at the end ensures that the function will not change anything (read only method). For more information about what const means refer to this: https://learn.microsoft.com/en-us/cpp/cpp/const-cpp?view=msvc-170.

- `get_capacity() const`

  returns the capacity of the array.

- `oparator[](size_t index)`

  This function should return the value at `index`. To do this, you should first check if the requested index is greater than or equal to `size`. If it is, `cout` "Index out of range\n", then `return` the value at `index`.

  Note: Generally, you would throw an exception here to try to prevent the array from accessing data outside of the index range. For this assignment, we will not. the main function is written in a way that this is never the case.

- `DynamicArray<T>& operator=(const DynamicArray<T>* other)`

  This function should "assign" one `DynamicArray` to another. To do this, first check to see if `other` is the current `this` (self-assignment). If it is, simply `return *this`.

  Otherwise, delete `data`, set size to the size of `other`, set capacity to the capacity of `other`, create a `new` array with the new capacity, copy over all information in `other` to `data`. Finally, `return *this`.

3 How To Hand In Your Program

When you are ready to turn in your assignment, ensure that your program runs with the "make" command on turing/hopper. Additionally, make sure that you document *ALL* files according to NIU standards or the standards discussed in class or found on BlackBoard. The inability to write documentation can indicate the lack of understanding of the code you have written.

After you have ensured that your makefile works and that everything runs on turing/hopper, file-transfer to your local device, zip all source code files and the makefile into a zip file with the following name:

`yourzid_assign1.zip`

Submit the zip file to blackboard. **DO NOT INCLUDE ANY EXECUTABLE FILES OR OUTPUT FILES.**

4 Grading

The grade you receive on this programming assignment will be scored as follows: 20% documentation/coding standards, 80% output matching.

When we grade your assignment, we will compile it on `hopper.cs.niu.edu` using your makefile. We check to see if your program generates the correct output, has the proper documentation, and if your coding standards are proper.

5 Hints

As always, build up a solution one step at a time. Some times you can start with what you already have and build upon it. Other times you must create something new and (should) unit test it before trying to integrate it with the rest of your code.

- Start by un-commenting a few lines of code in the `main()` function found in `main.cpp`. I recommend working on the default constructer, `push_back()`, `get_size()`, `get_capacity()`, and operator[ ] overload first. After that, you should slowly progress through the lines of code in `main.cpp` and implement each function you encounter that has not been implemented yet.