Assignment 2, Week 2

Gordon Erlebacher, GPT-5

Release date: 2025-09-11, **Due date:** 2024-09-18, 11:59 pm

In this assignment, we will explore the creation of a 3D Array class (named Grid) where the array is created in three different ways. You will learn to uses `.h` files together with C++ files to create classes. You will create tests to check functionality, and answer ten questions about the homework.

All include files related to the 3D grid (i.e., array) class are provided in this assignment. Your task is to develop the three corresponding `cpp` files. The Makefile is also included and should not be changed. As usual, you will create tests to verify that the different functions are running as expected. What tests to include is up to you. They should be documented in a README.md file (in markdown), or a pdf file (created with your favorite tool). Leave the `README.md` or pdf file in the repository and upload a pdf to Canvas.

Work is submitted via your github repository, which you'll use for all your assignments. Create a folder: `homework3/` at the top level of your repository. Beware: we download your repository when the homework is due so changes after the due date/time will not be taken into account by the TA.

As usual, when details are not clear, either ask us, or make your own decision. However, document this decision in your report (README) so that we can follow your train of thought. You must provide a justification for any decision made and reported.

Tasks

As discussed in class, you will consider the following three appraoches:

1. **1D array :** Create the file `grid3d_1d_array.cpp`
2. **std::vector :** Create the file `grid3d_vector.cpp`
3. **new operator :** Create the file `grid3d_new.cpp`

You will implement the following functions in all three files:

1. Constructor: construct the grid and initialize variables via `Grid(int i, int j, int k)`.
2. Copy constructor

3. Destructor: destroy the grid
4. Assignment operator
5. `getSize()` : retrieve the total number of elements in the 3D array
6. `getMemory()`: return the memory (in bytes) used by the array (the elements themselves and any additional memory used)
7. `operator()`: access a particular element. The arguments are "i, j, k": element index in each of the three directions. You should be able to access an element via `double my_var = grid(3, 4, 5);`, and set the element via `grid(3,4,5) = 7.;`
8. `set()` : set the value of a particular element:
   `double value; set(3, 4, 5, value);`
9. `operator+`: add two 3D arrays
10. `operator*`: multiply an array by a double:
    `grid = 3.0 * grid`
    `grid = grid * 3.0`
11. `operator++`: Increment all elements of an array (in place) by a double
12. `operator+=`: Increment an array by another array in place.
13. `operator<<`: overload `std::cout`

`main.cpp` should be run as is, and it should run without error.

Expected Deliverables

20 pts for each file] Implementation of all the functions stated above in the three `cpp` files.

10 pts Demonstrate that the Makefile executes without error (Kunal will simply execute the Makefile. Either it runs properly or doesn't.

15 pts A collection of unit tests (at least one test per operator) to help establish that your code is running properly. Run the same tests on each of the three grid types. I accept slight modifications if necessary to accommodate unexpected issues with your classes. Use the AI to help you (as usual).

10 pts Documentation of all your functions. The documentation should be in the header files, not in the source (cpp) files.

10 pts Use the `<chrono>` header file to time the following from inside main: The execution of the summation operator for arrays of size `n, n, n`, where n takes the values 10, 100, 1000, and create a single plot that displays the time to execute the summation of two identical grids:

```
Grid1 grid(n, n, n);
Grid1 grid_sum = grid + grid;
```

Execute each timing 5 times, and do not consider the first timing. Make sure you include labels on the horizontal and vertical axes, as well as a title. Draw a grid if you can.

10 pts Answer to the 10 graded questions below.

20 pts Description of your work in the README.md or pdf files, included in your repository, incluing the answers to the ten questions below.

1 C++ Homework Understanding Assessment, graded

author: AI Assistant

This assessment is graded, but will help you prepare for the quizzes given at a later time. Use the AI if you do not know the answers, and make sure you understand the responses. If you do not agree with the responses, ask the same question with different wording. Talk to Kunal (TA) or myself (instructor) if you are stuck and need help.

1. How is a 1D array used to represent a 3D array in memory? Explain the mapping between the 3D indices (i, j, k) and the 1D index.
2. What are the advantages and disadvantages of using a 1D array for 3D data storage compared to using multiple vectors of vectors?
3. When using a 1D array to represent a 3D grid, what formula would you use to access the element at position (i, j, k) given dimensions (nx, ny, nz)?
4. How can you implement the constructor for a 1D array-based 3D grid class to allocate memory?
5. How do you calculate the total memory usage (in bytes) of a 3D grid stored in a 1D array of doubles?
6. Explain how memory allocation differs between the three methods: a 1D array of doubles, a vector of vectors of vectors, and a dynamically allocated 3D array using `new`.
7. When using

   ```
   std::vector<std::vector<std::vector<double>>>
   ```

   how does the memory layout differ from using a 1D array?

8. How would you initialize the 3D vector structure

```
std::vector<std::vector<std::vector<double>>>
```

   to match a 3D grid of size (nx, ny, nz)?

9. Describe how you can access an element at position (i, j, k) in a 3D vector of vectors of vectors.

10. How would you implement a destructor for a 3D grid stored using dynamically allocated memory with `double***` to prevent memory leaks?

11. What is the correct syntax to allocate and deallocate memory for a 3D grid using `new double***`?

12. Explain the role of operator overloading in C++. How would you overload the `operator()` to access elements in a 3D grid for all three storage methods?

13. Implement the `operator()` for accessing elements in a 3D grid stored as a 1D array of doubles. What is the formula used to translate 3D indices to a 1D index?

14. How would you overload the `+` operator to add two 3D arrays of the same dimensions? Provide the pseudocode for this implementation.

15. Describe the steps necessary to overload the `operator<<` for printing the contents of a 3D grid in a readable format.

16. What considerations are important when overloading `operator+` to ensure that it works correctly with dynamically allocated 3D arrays using `new`?

17. How can you implement a `getMemory()` function to calculate the memory usage for a 3D grid stored in a vector of vectors of vectors?

18. How do you test whether two 3D arrays of different storage types (1D array, vector of vectors, dynamic `new` array) contain the same data?

19. What kind of memory issues can arise if the destructor of a dynamically allocated 3D grid class using `new` is not implemented correctly?

20. Provide a timing analysis to compare the memory access times for a 1D array,

```
std::vector<std::vector<std::vector<double>>>
```

   and a dynamically allocated 3D array using `new`. Which method do you expect to perform better, and why?

2 Additional C++ Understanding Assessment, no grade

Topics: Stack Frame, Heap, Function Arguments, Friend Functions, Overloaded Operators, and Overloaded Functions

1. What is a stack frame, and what types of data are typically stored in it during function execution?
2. How does the stack differ from the heap in terms of memory management and lifetime of variables?
3. Explain what happens to the stack frame when a function is called and how it is cleaned up when the function returns.
4. Describe the advantages and disadvantages of using heap memory compared to stack memory for dynamic data allocation.
5. What happens if too much memory is allocated on the stack? Explain stack overflow and how it can occur.
6. How can you explicitly allocate and deallocate memory on the heap in C++? Give an example using `new` and `delete`.
7. Describe the role of function arguments in the creation of a stack frame. What happens to function parameters in terms of memory when a function is called?
8. What are the differences between passing arguments by value, by reference, and by pointer? Provide examples of each.
9. How does the use of the `const` keyword with reference or pointer function arguments affect the function?
10. How does C++ handle variable arguments in functions? Describe how `std::initializer_list` or variadic templates can be used to pass variable arguments to functions.
11. Explain the purpose of friend functions in C++. Why might you declare a function as a friend of a class, and how does this affect encapsulation?
12. How do friend functions differ from member functions in terms of access to private and protected data?
13. Write an example of a friend function that accesses private data from two different classes.
14. How does operator overloading enhance the usability of classes in C++? Provide an example where overloading an operator simplifies code.

15. What are the rules and limitations of operator overloading in C++? Which operators cannot be overloaded?

16. How do you implement the `+=` operator for a class that represents a 3D array? Describe the relationship between `+` and `+=` in terms of operator overloading.

17. What is the difference between overloading the `<<` operator for output and the `+` operator for addition in terms of syntax and functionality?

18. What are overloaded functions in C++? How do they differ from templates, and in what scenarios would you use each?

19. Describe how function overloading works with respect to the number and types of parameters. What are some potential pitfalls when overloading functions?

20. Provide an example where you would overload a function to handle both integer and floating-point input. What considerations must you keep in mind regarding return types?