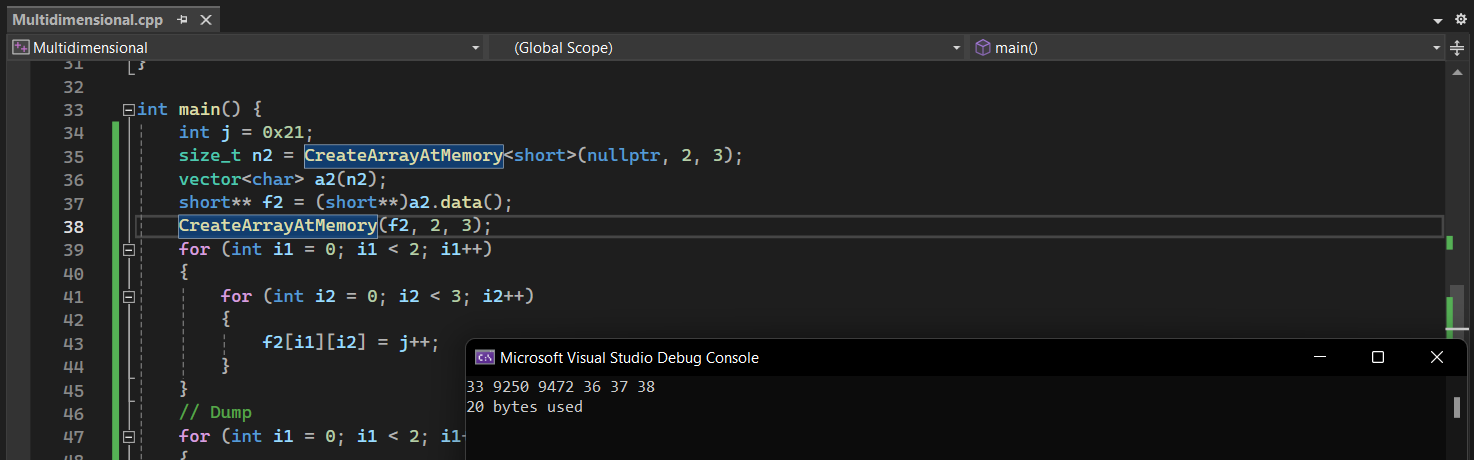
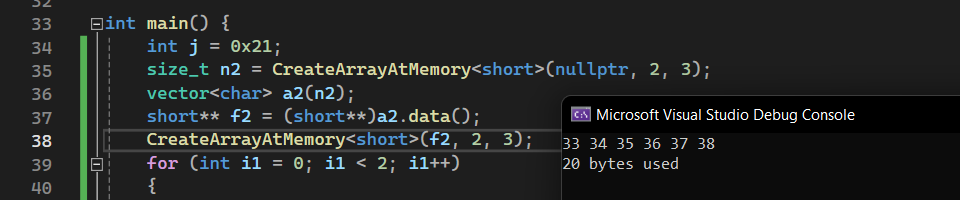
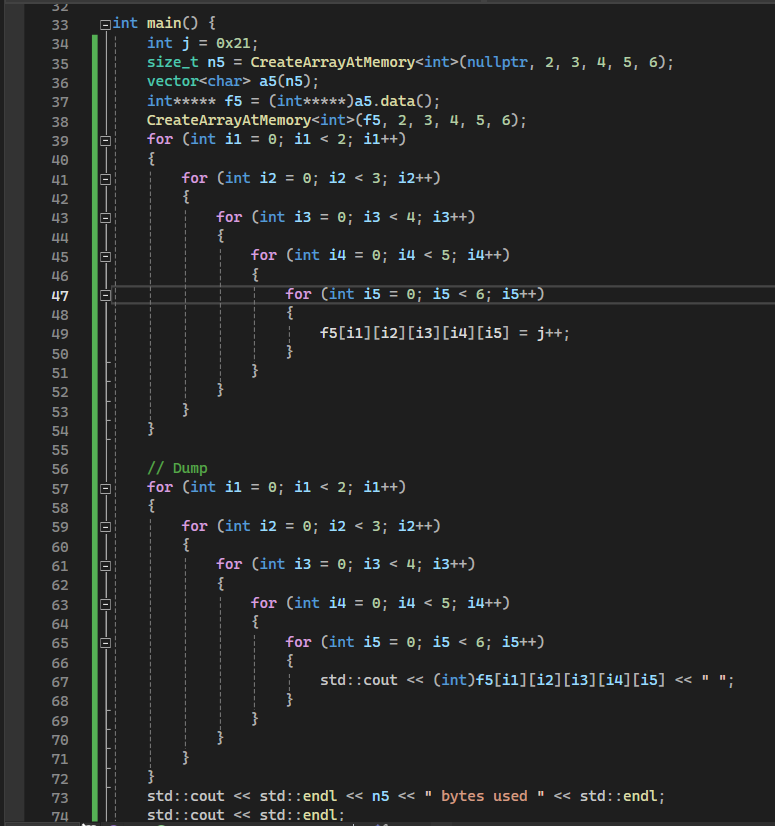
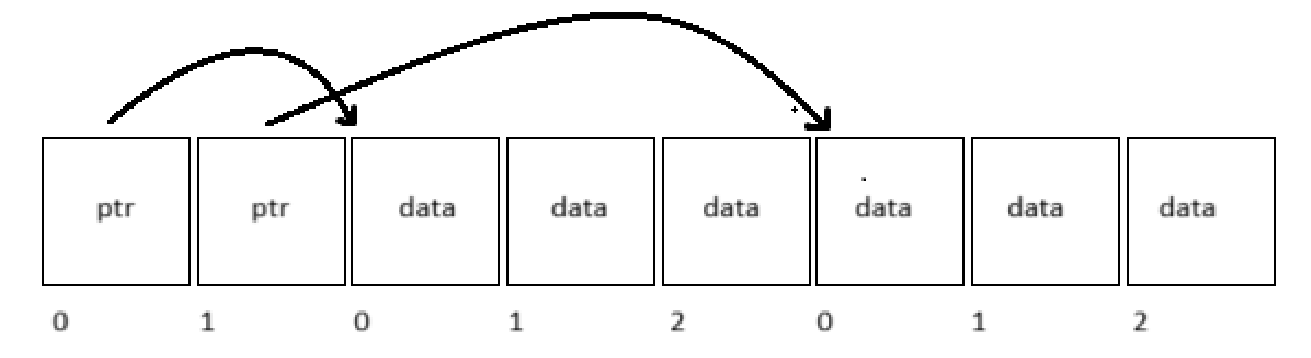
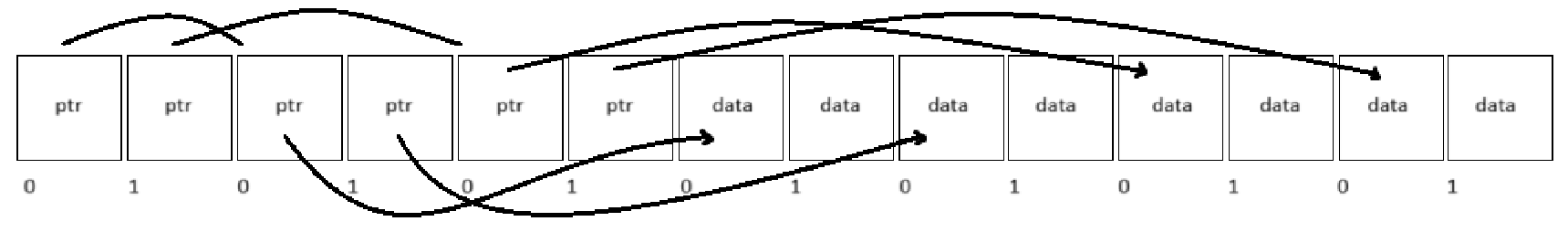
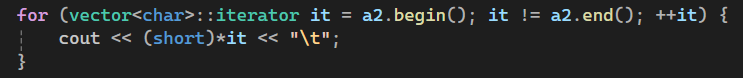
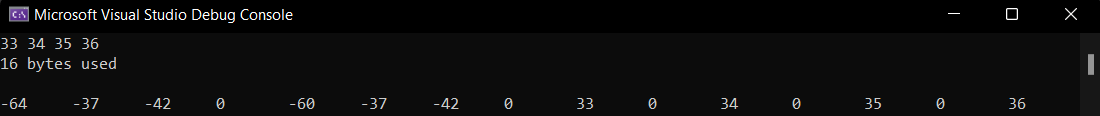
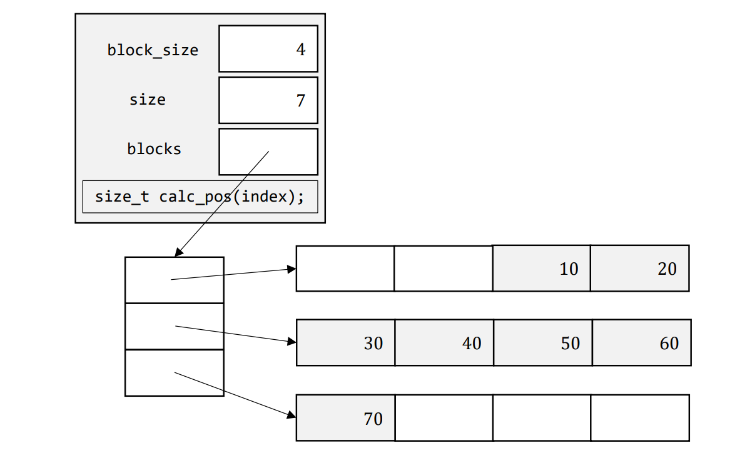
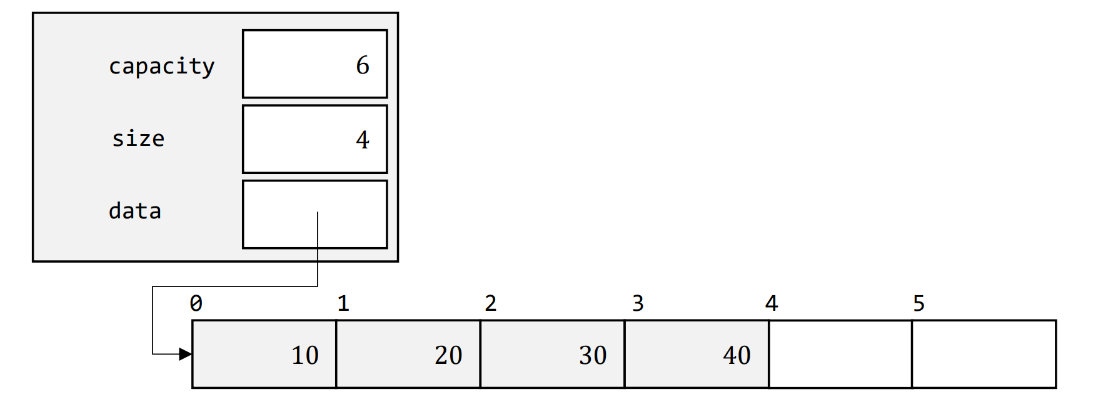
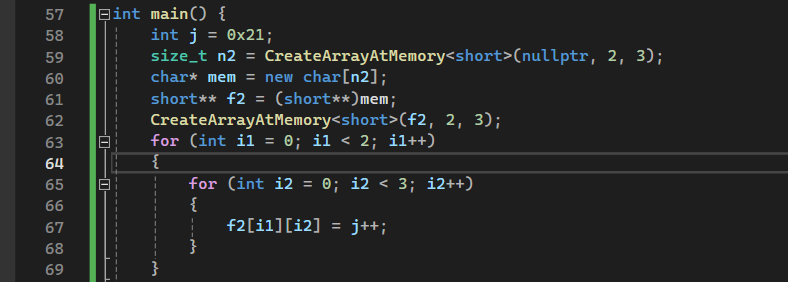
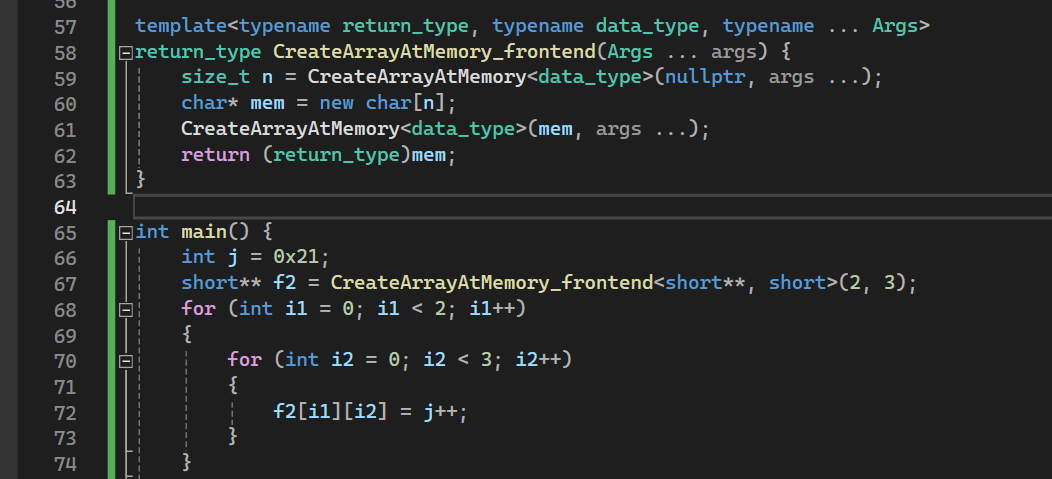
1. <https://www.codeproject.com/Members/vsh51> (посилання)
2. Автор швидше за все наводив приклади використовуючи 32-розрядну систему, оскільки при запуску другого та третього шаблонів на х64 платформі, масив використовує 28 та 4096 байтів відповідно, змінивши active solution platform на х86 (32-розрядна система) програма використовує таку ж кількість пам’яті як і у автора. Також стикнувся з проблемою приведення типу:  
     
     
     
   числа 33 та 34 відображаються неправильно (чомусь не відбулось приведення типу для пари цих чисел). Помилки не виникає, якщо явно вказувати тип (short) в шаблоні CreateArrayAtMemory (рядок 38):  
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
   Така ж проблема виникає й в третьому прикладі, тільки вказувати потрібно тип int. До того ж, в третьому прикладі є помилка (вектор а5 використовується ще до оголошення), провівши аналогію з викликами   
   шаблону в попередніх прикладах, виправив помилку та вказав замість а5 - nullptr. Виправлений код четвертого прикладу виглядає так:  
     
   
3. Шаблон функції CreateArrayAtMemory, що приймає два параметри, обчислює розмір корисної інформації, що може зберігати один вимір масиву, в байтах та є базовим випадком рекурсії в перевантаженому своєму ж варіанті. Складніша версія шаблону розміщує вказівники в переданій ділянці пам’яті так, щоб її можна було опрацьовувати оператором квадратні дужки, тобто трактувати як n-вимірний масив. Детально зрозуміти як працює логіка розміщення вказівників допоможуть дві відносно прості ілюстрації.  
     
     
     
     
   n-вимірний масив 2х3  
     
     
   n-вимірний масив 2х2х2  
     
     
   Також можна подивитись на те, як зберігаються дані в контейнері, який автор використав для виділення пам’яті (vector) на прикладі n-вимірного масиву 2x2 з елементами типу short:  
     
     
     
     
     
   Числа -64 -37 -42 0 є вказівником на масив який містить числа 33 та 34.  
   Виведення працює так, оскільки вказівник на масив елементів типу short займає чотири байти, елемент вектора на якому будується матриця – 1 байт. Відповідно вказівник займає місце 4 елементів в пам’яті вектора і при виведенні відображається інформація з кожного байта який займає цей вказівник приведена до типу short. З числами 33, 34, 35, 36 цікавіше. Вони займають 2 байти в пам’яті і тому після 33 ще бачимо 0, оскільки старші біти числа не несуть в собі інформацію (занадто маленьке число). До прикладу якщо взяти число близьке до максимального що можна зберегти в типі short на 32-розрядній системі, ми не побачимо власне цих чисел в векторі при такому виведенні, оскільки важливо буде мати інформацію вже зі всіх бітів.  
   Шаблон використовує пам’ять виділену контейнером задля уникнення повторних викликів оператора new.  
   Щодо покращень, оскільки для роботи шаблонної функції потрібно мати просто виділену ділянку пам’яті, на мою думку, можна скористатись простішим методом для її отримання – звичайний одновимірний динамічний масив. Також можна уникнути рекурсивних викликів зберігаючи аргументи (виміри) в вектор та обчислюючи зміщення вказівника в циклі:
4. Плавно перейшовши до наступного питання хочу наголосити на тому, що в ролі своєрідного алокатора пам’яті в даній реалізації може слугувати стандартний контейнер з бібліотеки stl, проте лише за умови що він зберігає елементи в суцільній послідовній ділянці пам’яті. Отже використати deque не вийде, бо він зберігає дані в складнішій структурі:  
     
     
     
     
     
     
     
     
   Звичайно подібний алгоритм можна імплементувати на основі будь-якого контейнера, просто врахувавши особливості структури його пам’яті, але це виглядає як надлишкова і непотрібна складність. Вектор схематично містить таку структуру, інкапсулює динамічний масив:  
     
     
     
     
     
     
   Тому використати можна просто динамічний масив, так як говорив в пункті 3:  
     
     
     
   Все працює точно так само як і у автора. Цей варіант навіть кращий, бо як і писав раніше, вектор слугує тільки своєрідною обгорткою для виділення пам’яті. Це можна зробити вручну і значно ефективніше.

*(ілюстрація з лекції 26 Послідовні контейнери)*

1. Front-end функція для створення n-вимірного масиву:  
     
     
     
   Принцип роботи нічим не відрізняється від варіанту автора. Нічого складного, лише всі виклики запакував у шаблонну функцію. Мета її існування – простий інтерфейс створення багатовимірного масиву. Також на скріншоті наводжу приклад використання такої функції.
2. Варіант автора здався мені достатньо складним на моменті розставляння вказівників всередині суцільної ділянки пам’яті, тому спробував реалізувати простішу шаблонну функцію яка не зовсім відповідає тим вимогам, які поставив перед собою автор, а саме – обійтись без багаторазових виділень пам’яті, проте є в рази простішою для розуміння. Принцип дуже подібний: використовується рекурсивне виділення пам’яті для кожного виміру масиву. Тобто в моєму варіанті ділянка пам’яті не є суцільною, а всю роботу з вказівниками спрощено виконує оператор new. Проект разом з модульними тестами та міні документацією завантажив на гітхаб: