ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШАГО ОБРАЗОВАНИЯ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №2**

**«Реализация последовательного хранения в процедурном подходе»**

по курсу

«Прикладное объектно-ориентированное программирование»

Вариант № 5

Выполнил:

Студент: Кудаков Т. Г.

Группы: М4О-205Б-23

Проверил:

Дубовский А. А.

Отметка о защите:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2025

Теоретическая часть

В лабораторной работе используются различные составляющие:

* пространства имен *namespace* – это группа взаимосвязанных функция, переменных, констант, классов, объектов и других компонентов программы;
* Структуры *struct* – пользовательский тип данных, который позволяет объединить переменные разных типов под одним именем;
* Динамический массив – упорядоченный набор элементов, каждый из которых хранит значение, индицируемое с помощью одного или нескольких индексов, при этом размер массива может меняться во время исполнения программы;
* Динамический массив указателей – массив, каждый элемент которого содержит указатель на переменную (объект);
* Односвязный список – структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей.

Описание функционала каждого хранилища описаны в таблице 1.

Таблица 1 (начало)

|  |  |  |
| --- | --- | --- |
| № | Тип функционала | Описание |
| 1 | Получение элемента по заданному индексу (at) | Возвращает элемент по указанному индексу. Если индекс выходит за пределы контейнера, генерируется исключение. Сложность операции O(1) для массива и O(n) для списка. |
| 2 | Добавление элемента в конец контейнера (push\_back) | Добавляет новый элемент в конец контейнера. При необходимости происходит увеличение размера контейнера. Сложность O(1) амортизированная для массива и O(1) для списка. |

Таблица 1 (продолжение)

|  |  |  |
| --- | --- | --- |
| 3 | Добавление элемента в начало контейнера (push\_front) | Добавляет новый элемент в начало контейнера. Для массива требуется сдвиг всех элементов (O(n)), для списка - O(1). |
| 4 | Вставка нового элемента по произвольному индексу (insert) | Вставляет новый элемент по указанному индексу. Все последующие элементы сдвигаются вправо. Сложность O(n) для массива и списка. |
| 5 | Объединение двух контейнеров в один (concatenate) | Объединяет текущий контейнер с другим, добавляя все элементы второго контейнера в конец первого. Сложность O(n) для обоих типов. |
| 6 | Возврат количества элементов (count) | Возвращает текущее количество элементов в контейнере. Сложность O(1), так как размер отслеживается в отдельной переменной. |
| 7 | Удаление элемента с конца контейнера (pop\_back) | Удаляет последний элемент из контейнера. Сложность O(1) для обоих типов контейнеров. |
| 8 | Удаление элемента с начала контейнера (pop\_front) | Удаляет первый элемент из контейнера. Для массива требуется сдвиг элементов (O(n)), для списка - O(1). |

Таблица 1 (продолжение)

|  |  |  |
| --- | --- | --- |
| 9 | Удаление элемента по произвольному индексу (pop\_out) | Удаляет элемент по указанному индексу. Последующие элементы сдвигаются влево. Сложность O(n) для обоих типов. |
| 10 | Замена элемента другим, по заданному индексу (replace) | Заменяет элемент по указанному индексу на новый. Сложность O(1) для массива и O(n) для списка (для достижения нужной позиции). |
| 11 | Перемена мест двух элементов (swap) | Меняет местами два элемента по указанным индексам. Сложность O(1) для массива и O(n) для списка. |
| 12 | Вывод всего содержимого контейнера (print) | Выводит все элементы контейнера в консоль или файл. Проходит по всем элементам, сложность O(n) для обоих типов контейнеров. |

Полученные данные

Выполним код программы для каждой операции, после чего запишем полученные значения в таблицу 2.

Таблица 2

|  |  |  |  |
| --- | --- | --- | --- |
| Тип операции | SimpleDynamicArray | PointersArray | List |
| at | 1 | 0 | 0 |
| push\_back | 1 | 0 | 0 |
| push\_front | 24 | 0 | 0 |
| insert | 25 | 1 | 1 |
| concatenate | 183 | 2 | 0 |
| count | 1 | 0 | 0 |
| pop\_back | 0 | 0 | 0 |
| pop\_front | 3 | 0 | 0 |
| pop\_out | 1 | 0 | 0 |
| replace | 1 | 0 | 1 |
| swap | 1 | 0 | 0 |
| print | 1041 | 616 | 1260 |
| move | 12 | 0 | 0 |

Вывод и используемая литература

1. Вывод

Проведенные эксперименты подтвердили теоретические оценки сложности операций для динамического массива, массива указателей и односвязного списка. Наиболее значимые различия наблюдались в операциях, требующих сдвига элементов:

* Операция push\_front;
* Операция print.

Результаты concatenate для SimpleDynamicArray объясняются копированием элементов при расширении, что соответствует амортизированной сложности O(n). Для списка операция объединения может быть реализована за O(1), если сохранять указатель на конец.

Практические данные подтверждают, что выбор структуры данных должен основываться на преобладающих операциях: массивы эффективны для частого доступа по индексу, а списки — для частых вставок/удалений в начале.

1. Используемая литература
   1. Массивы (C++) | Microsoft Learn // Microsoft Learn URL: <https://learn.microsoft.com/ru-ru/cpp/cpp/arrays-cpp?view=msvc-170>
   2. Указатели (C++) | Microsoft Learn // Microsoft Learn URL: <https://learn.microsoft.com/ru-ru/cpp/cpp/pointers-cpp?view=msvc-170>
   3. Динамические массивы | METANIT.COM // METANIT.COM URL: <https://metanit.com/cpp/tutorial/4.12.php>