Министерство Образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

**СТРУКТУРЫ ДАННЫХ**

**Лабораторная работа №1**

**Перечисление. Объединение. STL контейнеры. Массив. Дек. Одно/двух связный список. Вектор.**

Студент:

1курса 2 группы 1 подгруппы

Марушко Тимофей Фёдорович

Проверяющий:

Белодед Николай Иванович

**ОГЛАВЛЕНИЕ**

[**Теоритические сведения 4**](#_Toc191238788)

[**1. Deque (Двусторонняя очередь) 4**](#_Toc191238789)

[**2. Map (Ассоциативный контейнер) 5**](#_Toc191238790)

[**3. Односвязный и двусвязный список 6**](#_Toc191238791)

[**4. Vector (Динамический массив) 7**](#_Toc191238792)

[**5. Set (Множество) 8**](#_Toc191238793)

[**6. Multiset (Множество с дубликатами) 9**](#_Toc191238794)

[**РЕШЕНИЕ ПРАКТИЧЕСКИХ ЗАДАЧ 10**](#_Toc191238795)

[**1. Написать программу, которая, в зависимости от выбора пользователя, в консоли должна выводить названия стран мира, с использованием перечислений. 10**](#_Toc191238796)

[**2. Написать программу, в которой в конец/в начало массива добавляется n элементов и вывести на экран. 11**](#_Toc191238797)

[**3. Написать программу, в которой нужно найти среднее арифметическое элементов вектора и вывести его на экран. 12**](#_Toc191238798)

[**4. Написать программу, в которой нужно найти максимальный/минимальный элемент дэки и добавить его в конец/в начало, с выводом на экран. 13**](#_Toc191238799)

[**5. Написать программу, в которой нужно удалить все элементы односвязного списка и добавить n новых, с выводом на экран. 14**](#_Toc191238800)

[**6. Написать программу, в которой в начало/в конец двусвязного списка добавляется n элементов и вывести на экран. 15**](#_Toc191238801)

[**7. Написать программу, которая будет искать введённое пользователем число в map. 16**](#_Toc191238802)

[**РЕШЕНИЕ ДОПОЛЬНИТЕЛЬНЫХ ЗАДАЧ 17**](#_Toc191238803)

[**1. Объедините два массива целых чисел в один и выведите его элементы, используя вектор. 17**](#_Toc191238804)

[**2. Создайте дек перечислений, представляющий месяцы года, и добавьте в него несколько элементов. Затем удалите первый и последний элементы и выведите содержимое дека. 18**](#_Toc191238805)

[**3. Создайте массив строк и отсортируйте его по длине строк. Если длина строк одинакова, то сортируйте их в лексикографическом порядке. Затем выведите содержимое отсортированного массива. 19**](#_Toc191238806)

[**ВЫВОДЫ 20**](#_Toc191238807)

Теоритические сведения

1. Deque (Двусторонняя очередь)

**Что это?**

Deque — это контейнер для двусторонней очереди, которая позволяет добавлять и удалять элементы с обеих сторон (в начале и в конце).

**Где используется?**

Используется для сценариев, где необходимо быстро работать с обеими сторонами данных (например, реализация очередей с приоритетом).

**+** Преимущества: высокая производительность операций добавления/удаления с обеих сторон.

**-** Недостатки: нельзя использовать указатели на элементы, так как контейнер может перемещать элементы при изменении размера.

**Пример реализации:**

**Задача**: Реализуем очередь, в которой можно обрабатывать данные с обеих сторон, например, с помощью двусторонней очереди можно создать очередь с приоритетом для обработки заданий с разными приоритетами (в начале или в конце очереди).

*#include* <iostream>

*#include* <deque>

using namespace std;

int main() {

*// Создаем двустороннюю очередь целых чисел*

    deque<int> dq;

*// Добавляем элементы в начало и конец*

    dq.push\_front(5); *// Добавляем в начало очереди*

    dq.push\_back(10); *// Добавляем в конец очереди*

    dq.push\_front(1); *// Добавляем еще в начало*

*// Выводим все элементы очереди*

    cout << "Элементы очереди: ";

*for* (int num : dq) {

        cout << num << " "; *// 1 5 10*

    }

    cout << endl; *// Удаляем элементы из очереди (с начала и с конца)*

    dq.pop\_front(); *// Удаляем первый элемент (1)*

    dq.pop\_back(); *// Удаляем последний элемент (10)*

    cout << "После удаления: ";*// Снова выводим элементы после удаления*

*for* (int num : dq) {

        cout << num << " "; *// 5*

    }

    cout << endl;

*return* 0;

}

1. Map (Ассоциативный контейнер)

**Что это?**

Map — контейнер, хранящий пары "ключ-значение", где ключи уникальны и отсортированы по возрастанию.

**Где используется?**

Применяется для быстрого поиска значений по ключу.

**+** Преимущества: быстрый поиск и сортировка по ключу.

**-** Недостатки: требуется дополнительная память для хранения пар ключ-значение.

**Пример реализации:**

**Задача**: Используем map для хранения телефонных номеров по именам, чтобы быстро искать номер по имени.

*#include* <iostream>

*#include* <deque>

using namespace std;

int main() {

*// Создаем двустороннюю очередь целых чисел*

   map<string, string> phoneBook;

*// Добавляем записи*

   phoneBook["John"] = "+1234567890";

   phoneBook["Alice"] = "+0987654321";

   phoneBook["Bob"] = "+1112233445";

*// Выводим номер по имени*

   cout << "Телефон Alice: " << phoneBook["Alice"] << endl; *// +0987654321*

*// Перебираем все записи и выводим*

   cout << "Все записи в телефонной книге:" << endl;

*for* (auto& entry : phoneBook) {

        cout << entry.first << ": " << entry.second << endl;

   }

*// Поиск элемента по ключу*

*if* (phoneBook.find("Bob") != phoneBook.end()) {

         cout << "Номер Bob найден: " << phoneBook["Bob"] << endl; *// +1112233445*

   } *else* {

         cout << "Bob не найден!" << endl;

}

*return* 0;

}

1. Односвязный и двусвязный список

**Что это?**

Списки, состоящие из узлов, каждый из которых содержит значение и указатель на следующий (односвязный) или и на следующий, и на предыдущий (двусвязный).

**Где используется?**

**Используются в задачах, где требуется динамическое изменение размера коллекции, например, при добавлении**/удалении элементов в середине.

**+** Преимущества: гибкость в изменении структуры списка, экономия памяти (особенно в случае двусвязного списка).

**-** Недостатки: медленный доступ к произвольным элементам, высокая стоимость операций в середине списка (для односвязного списка).

**Пример реализации:**

**Задача**: Используем односвязный список для добавления и удаления элементов. Будем моделировать очередь с элементами, которые могут быть добавлены в произвольное место.

*#include* <iostream>

*#include* <forward\_list>

using namespace std;

int main() {

*// Создаем односвязный список с начальными значениями*

    forward\_list<int> flist = {1, 2, 3, 4, 5};

*// Добавляем элемент после первого*

    auto iter = flist.begin();

    flist.emplace\_after(iter, 6); *// После 1 добавим 6*

*// Выводим элементы*

    cout << "После добавления элемента: ";

*for* (int n : flist) {

        cout << n << " "; *// 1 6 2 3 4 5*

    }

    cout << endl;

*// Удаляем первый элемент*

    flist.pop\_front();

    cout << "После удаления первого элемента: ";

*for* (int n : flist) {

        cout << n << " "; *// 6 2 3 4 5*

    }

    cout << endl;

*return* 0;

}

1. Vector (Динамический массив)

**Что это?**

Vector — динамический массив, размер которого может изменяться во время выполнения.

**Где используется?**

Используется в случаях, когда нужно хранить данные с возможностью быстрого доступа по индексу и изменения размера массива.

+ Преимущества: быстрый доступ к элементам по индексу, эффективное использование памяти.

- Недостатки: изменение размера может требовать перераспределения памяти.

**Пример реализации:**

**Задача:** Сортируем и выводим данные с помощью вектора, делая динамическое добавление элементов.

*#include* <iostream>

*#include* <forward\_list>

using namespace std;

int main() {

*// Создаем односвязный список с начальными значениями*

    forward\_list<int> flist = {1, 2, 3, 4, 5};

*// Добавляем элемент после первого*

    auto iter = flist.begin();

    flist.emplace\_after(iter, 6); *// После 1 добавим 6*

*// Выводим элементы*

    cout << "После добавления элемента: ";

*for* (int n : flist) {

        cout << n << " "; *// 1 6 2 3 4 5*

    }

    cout << endl;

*// Удаляем первый элемент*

    flist.pop\_front();

    cout << "После удаления первого элемента: ";

*for* (int n : flist) {

        cout << n << " "; *// 6 2 3 4 5*

    }

    cout << endl;

*return* 0;

}

1. Set (Множество)

**Что это?**

Set — контейнер для хранения уникальных элементов, которые автоматически сортируются.

**Где используется?**

Используется для хранения и поиска уникальных элементов без дубликатов.

+ Преимущества: автоматическая сортировка элементов, быстрые операции поиска и удаления.

- Недостатки: не поддерживает дубликаты элементов.

**Пример реализации:**

**Задача:** Пример использования set для хранения уникальных элементов, где дубликаты автоматически исключаются.

*#include* <iostream>

*#include* <set>

using namespace std;

int main() {

*// Создаем set для хранения уникальных чисел*

    set<int> uniqueNumbers;

*// Добавляем элементы, включая дубликаты*

    uniqueNumbers.insert(5);

    uniqueNumbers.insert(3);

    uniqueNumbers.insert(5); *// Этот элемент будет проигнорирован*

    uniqueNumbers.insert(8);

*// Выводим элементы множества*

    cout << "Элементы множества: ";

*for* (int num : uniqueNumbers) {

    cout << num << " "; *// 3 5 8 (автоматически отсортировано)*

    }

    cout << endl;

*return* 0;

}

1. Multiset (Множество с дубликатами)

**Что это?**

Multiset похож на set, но допускает хранение нескольких одинаковых элементов.

**Где используется?**

Используется, когда требуется хранить несколько одинаковых элементов.

+Преимущества: поддержка дубликатов элементов.

- Недостатки: те же, что и у set, но с добавлением поддержания нескольких одинаковых элементов.

**Пример реализации:**

**Задача:** Пример использования multiset для хранения элементов с возможностью дублирования.

*#include* <iostream>

*#include* <set>

using namespace std;

int main() {

*// Создаем multiset для хранения чисел с дубликатами*

    multiset<int> numbers;

*// Добавляем числа, включая дубликаты*

    numbers.insert(5);

    numbers.insert(3);

    numbers.insert(5); *// Дубликат сохраняется*

    numbers.insert(8);

*// Выводим элементы множества*

    cout << "Элементы multiset: ";

*for* (int num : numbers) {

        cout << num << " "; *// 3 5 5 8*

    }

    cout << endl;

*return* 0;

}

РЕШЕНИЕ ПРАКТИЧЕСКИХ ЗАДАЧ

1. Написать программу, которая, в зависимости от выбора пользователя, в консоли должна выводить названия стран мира, с использованием перечислений.

*#include* <map>

*#include* <list>

*#include* <iostream>

*#include* <vector>

*#include* <deque>

using namespace std;

*// Задача 1: Использование перечислений для выбора страны*

enum class Country {

    USA, Canada, Mexico, Brazil, Germany, France, Japan, India, China, Russia

};

*// Функция task1: Демонстрация работы с перечислением стран*

void task1() {

    cout << "\*\*\* TASK1 \*\*\*" << endl;

    int choice;

    cout << "Select a country:\n1. USA\n2. Canada\n3. Mexico\n4. Brazil\n5. Germany\n6. France\n7. Japan\n8. India\n9. China\n10. Russia\n";

    cin >> choice;

*// Используем switch, чтобы в зависимости от выбора пользователя вывести страну*

*switch* (choice) {

*case* 1: cout << "USA"; *break*;

*case* 2: cout << "Canada"; *break*;

*case* 3: cout << "Mexico"; *break*;

*case* 4: cout << "Brazil"; *break*;

*case* 5: cout << "Germany"; *break*;

*case* 6: cout << "France"; *break*;

*case* 7: cout << "Japan"; *break*;

*case* 8: cout << "India"; *break*;

*case* 9: cout << "China"; *break*;

*case* 10: cout << "Russia"; *break*;

*default*: cout << "Invalid choice"; *break*; *// Если выбор не соответствует ни одному из случаев*

    }

    cout << endl;

}

1. Написать программу, в которой в конец/в начало массива добавляется n элементов и вывести на экран.

*// Задача 2: Добавление элементов в начало и конец массива*

void task2() {

    cout << "\*\*\* TASK2 \*\*\*" << endl;

    vector<int> vec = { 1, 2, 3, 4, 5 }; *// Инициализируем исходный вектор*

    int n = 3; *// Количество добавляемых элементов*

    vector<int> toAdd = { 10, 20, 30 }; *// Вектор элементов для добавления*

*// Добавляем элементы в начало вектора*

    vec.insert(vec.begin(), toAdd.begin(), toAdd.end());

*// Добавляем элементы в конец вектора*

    vec.insert(vec.end(), toAdd.begin(), toAdd.end());

*// Выводим изменённый вектор*

    cout << "Updated vector: ";

*for* (int num : vec) {

        cout << num << " "; *// Печатаем каждый элемент вектора*

    }

    cout << endl;

}

1. Написать программу, в которой нужно найти среднее арифметическое элементов вектора и вывести его на экран.

*// Задача 3: Рассчёт среднего арифметического элементов вектора*

void task3() {

    cout << "\*\*\* TASK3 \*\*\*" << endl;

    vector<int> vec = { 10, 20, 30, 40, 50, 60 }; *// Исходный вектор*

    double sum = 0; *// Переменная для хранения суммы элементов*

*// Суммируем все элементы вектора*

*for* (auto num : vec) sum += num;

*// Рассчитываем и выводим среднее значение*

    cout << "Average value of vec: " << sum / vec.size() << endl << endl; *// Делим сумму на количество элементов*

}

1. Написать программу, в которой нужно найти максимальный/минимальный элемент дэки и добавить его в конец/в начало, с выводом на экран.

*// Задача 4: Поиск минимальных и максимальных элементов в deque*

void task4() {

    cout << "\*\*\* TASK4 \*\*\*" << endl;

    deque<int> dq = { 10, 20, 30, 40, 50, 60 }; *// Инициализируем deque*

    int max\_elem = INT32\_MIN; *// Инициализация максимального элемента*

    int min\_elem = INT32\_MAX; *// Инициализация минимального элемента*

*// Итерация через deque с использованием итераторов*

*for* (auto it = dq.begin(); it != dq.end(); ++it) {

*if* (\*it > max\_elem) max\_elem = \*it; *// Если текущий элемент больше max\_elem, обновляем max\_elem*

*if* (\*it < min\_elem) min\_elem = \*it; *// Если текущий элемент меньше min\_elem, обновляем min\_elem*

    }

*// Добавляем найденные элементы в конец и начало deque*

    dq.push\_back(max\_elem); *// Максимальный элемент в конец*

    dq.push\_front(min\_elem); *// Минимальный элемент в начало*

*// Выводим элементы deque*

*for* (auto num : dq) cout << num << " "; *// Проходим по всем элементам deque и выводим их*

    cout << endl << endl;

}

1. Написать программу, в которой нужно удалить все элементы односвязного списка и добавить n новых, с выводом на экран.

*// Задача 5: Работа с односвязным списком*

struct Node { *// Структура для узла односвязного списка*

    int data; *// Данные, хранящиеся в узле*

    Node\* next; *// Указатель на следующий узел*

    Node(int *val*) : data(*val*), next(nullptr) {} *// Конструктор для инициализации узла значением val*

};

*// Функция для удаления всех узлов списка*

void deleteList(Node\*& *head*) {

*while* (*head* != nullptr) {

        Node\* temp = *head*; *// Сохраняем текущий узел*

*head* = *head*->next; *// Перемещаем голову на следующий узел*

        delete temp; *// Удаляем текущий узел*

    }

}

*// Функция для добавления n узлов в начало списка*

void addNodes(Node\*& *head*, int *n*) {

*for* (int i = 1; i <= *n*; ++i) {

        Node\* newNode = new Node(i); *// Создаём новый узел*

        newNode->next = *head*; *// Новый узел указывает на текущую голову списка*

*head* = newNode; *// Обновляем голову списка*

    }

}

*// Функция для вывода списка*

void printList(Node\* *head*) {

*// Пока список не пуст*

*while* (*head* != nullptr) {

        cout << *head*->data << " "; *// Печатаем данные текущего узла*

*head* = *head*->next; *// Переходим к следующему узлу*

    }

    cout << endl;

}

*// Задача 5: Демонстрация работы с односвязным списком*

void task5() {

    cout << "\*\*\* TASK5 \*\*\*" << endl;

    Node\* head = nullptr; *// Инициализируем голову списка (пустой список)*

    addNodes(head, 5); *// Добавляем 5 узлов в начало списка*

    cout << "List after adding nodes: ";

    printList(head); *// Печатаем список после добавления узлов*

    deleteList(head); *// Удаляем все узлы из списка*

    cout << "List after deletion: ";

    printList(head); *// Печатаем список после удаления узлов*

}

1. Написать программу, в которой в начало/в конец двусвязного списка добавляется n элементов и вывести на экран.

*// Задача 6: Работа с двусвязным списком (std::list)*

void task6() {

    cout << "\*\*\* TASK6 \*\*\*" << endl;

    list<int> myList = { 1, 2, 3, 4, 5 }; *// Инициализируем двусвязный список*

    int n = 3; *// Количество добавляемых элементов*

    list<int> toAdd = { 10, 20, 30 }; *// Список элементов для добавления*

*// Добавляем элементы в начало списка*

    myList.insert(myList.begin(), toAdd.begin(), toAdd.end());

*// Добавляем элементы в конец списка*

    myList.insert(myList.end(), toAdd.begin(), toAdd.end());

*// Печатаем обновлённый список*

    cout << "Updated list: ";

*for* (int num : myList) {

        cout << num << " "; *// Печатаем каждый элемент*

    }

    cout << endl;

}

1. Написать программу, которая будет искать введённое пользователем число в map.

*// Задача 7: Поиск по map*

void task7() {

    cout << "\*\*\* TASK7 \*\*\*" << endl;

    map<int, string> myMap = { {1, "One"}, {2, "Two"}, {3, "Three"} }; *// Инициализация map*

    int searchValue;

    cout << "Enter a number to search in the map: ";

    cin >> searchValue;

*// Поиск значения по ключу в map*

    auto it = myMap.find(searchValue);

*if* (it != myMap.end()) { *// Если элемент найден*

        cout << "Found: " << it->first << " -> " << it->second << endl;

    }

*else* {

        cout << "Number not found in the map." << endl; *// Если элемент не найден*

    }

}

РЕШЕНИЕ ДОПОЛЬНИТЕЛЬНЫХ ЗАДАЧ

1. Объедините два массива целых чисел в один и выведите его элементы, используя вектор.

void additionalTask1() {

    cout << "\*\*\* TASK1 \*\*\*" << endl;

*// Инициализируем два вектора с целыми числами*

    vector<int> vec1 = { 1, 2, 3, 4, 5 };

    vector<int> vec2 = { 6, 7, 8, 9, 10 };

*// Объединяем два вектора в один с помощью insert*

    vec1.insert(vec1.end(), vec2.begin(), vec2.end());

*// Выводим элементы объединённого вектора*

    cout << "Merged vector: ";

*for* (int num : vec1) {

        cout << num << " "; *// Печатаем каждый элемент вектора*

    }

    cout << endl;

}

1. Создайте дек перечислений, представляющий месяцы года, и добавьте в него несколько элементов. Затем удалите первый и последний элементы и выведите содержимое дека.

enum Month {

    January, February, March, April, May, June, July, August, September, October, November, December

};

void additionalTask2() {

    cout << "\*\*\* TASK2 \*\*\*" << endl;

*// Создаём дек для хранения месяцев*

    deque<Month> months = { Month::January, Month::February, Month::March, Month::April, Month::May };

*// Добавляем новые месяцы в дек*

    months.push\_back(Month::June);

    months.push\_back(Month::July);

*// Удаляем первый и последний элемент*

    months.pop\_front();

    months.pop\_back();

*// Выводим оставшиеся месяцы*

    cout << "Updated deque of months: ";

*for* (auto month : months) {

*switch* (month) { *// Выводим название месяца*

*case* Month::January: cout << "January "; *break*;

*case* Month::February: cout << "February "; *break*;

*case* Month::March: cout << "March "; *break*;

*case* Month::April: cout << "April "; *break*;

*case* Month::May: cout << "May "; *break*;

*case* Month::June: cout << "June "; *break*;

*case* Month::July: cout << "July "; *break*;

*case* Month::August: cout << "August "; *break*;

*case* Month::September: cout << "September "; *break*;

*case* Month::October: cout << "October "; *break*;

*case* Month::November: cout << "November "; *break*;

*case* Month::December: cout << "December "; *break*;

        }

    }

    cout << endl;

}

1. Создайте массив строк и отсортируйте его по длине строк. Если длина строк одинакова, то сортируйте их в лексикографическом порядке. Затем выведите содержимое отсортированного массива.

void additionalTask3() {

    cout << "\*\*\* TASK3 \*\*\*" << endl;

*// Инициализируем массив строк*

    vector<string> strs = { "apple", "banana", "kiwi", "mango", "cherry", "peach" };

*// Сортируем строки по длине, а затем лексикографически*

    sort(strs.begin(), strs.end(), [](const string& *a*, const string& *b*) {

*if* (*a*.size() == *b*.size()) {

*return* *a* < *b*; *// Если длины одинаковые, сортируем лексикографически*

        }

*return* *a*.size() < *b*.size(); *// Сортируем по длине*

        });

*// Выводим отсортированный массив*

    cout << "Sorted array of strings: ";

*for* (const auto& str : strs) {

        cout << str << " "; *// Печатаем каждую строку*

    }

    cout << endl;

}

ВЫВОДЫ

Я изучил несколько важных аспектов работы с кодом и структурами данных в C++: как использовать различные контейнеры, такие как vector, deque, map, и др. для эффективной обработки данных. Я научился работать с базовыми операциями добавления, удаления и поиска элементов в этих структурах, а также с сортировкой данных и вычислением различных показателей.

Кроме этого, я освоил основы работы с динамической памятью, создав и удалив связанный список. Эти навыки важны для решения более сложных задач в программировании и помогут мне эффективно работать с данными.