Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

ОТЧЕТ

к лабораторной работе №5

**«Адаптеры контейнеров библиотеки STL»**

|  |  |
| --- | --- |
|  | |
|  |
|  |
|  |
|  |  | | |
|  | | |  | |  |
|  | | |  | |  |
| Студент гр. 324402 | | |  | | Цевелюк А.И. |
| Проверила | | |  | | Купрейчик А.С. |

Минск 2024

**1 Цель работы**

Цель работы:изучить адаптеры контейнеров библиотеки STL в языке C++, их особенности, функциональности и применения. Лабораторная работа направлена на освоение работы с контейнерами-адаптерами stack, queue и priority\_queue, их методов и возможностей для решения задач, требующих специфической структуры данных.

**2 Теоретические сведения по лабораторной работе**

Контейнеры-адаптеры позволяют организовать работу с различными структурами данных, такими как стек, очередь и очередь с приоритетом, на основе других контейнеров STL. Основное предназначение контейнеров-адаптеров — это предоставление упрощённого интерфейса для решения специфических задач с ограниченным набором операций.

Контейнеры-адаптеры

Контейнер-адаптер — это разновидность контейнера, которая предоставляет ограниченный интерфейс для доступа к элементам, по сравнению с более универсальными контейнерами. Адаптеры позволяют использовать контейнеры таким образом, чтобы они вели себя как другие структуры данных. Основными контейнерами-адаптерами в библиотеке STL являются:

- stack (стек),

- queue (очередь),

- priority\_queue (очередь с приоритетом).

Все контейнеры-адаптеры ограничены в своей функциональности и не поддерживают итераторы, что делает их неподходящими для использования в алгоритмах STL, которые требуют итераторов.

Понятие адаптера

В объектно-ориентированном программировании адаптер представляет собой сущность, которая предоставляет альтернативный интерфейс для другой сущности, не реализуя собственной функциональности. Адаптеры бывают нескольких видов: контейнерные адаптеры, итераторные адаптеры и адаптеры функций.

Контейнер-адаптер позволяет преобразовать стандартный контейнер в другую структуру данных, например, стек или очередь, предоставляя ограниченный интерфейс для работы с элементами.

Виды контейнеров-адаптеров

1. stack (стек)

Стек работает по принципу LIFO (Last In, First Out — последним вошел, первым вышел). Элементы добавляются в стек и извлекаются из него только с одного конца — вершины стека. Примером использования стека может быть моделирование рекурсии или выполнение инструкций в обратном порядке.

Основные методы контейнера stack:

- empty() — проверка, пуст ли стек.

- push(const Type& val) — добавление элемента в стек.

- pop() — удаление верхнего элемента.

- top() — доступ к верхнему элементу без его удаления.

- size() — возвращает количество элементов в стеке.

2. queue (очередь)

Очередь работает по принципу FIFO (First In, First Out — первым вошел, первым вышел). Элементы добавляются в конец очереди и извлекаются с начала. Примером использования может быть организация обработки задач в порядке поступления.

Основные методы контейнера queue:

- empty() — проверка, пуста ли очередь.

- push(const Type& val) — добавление элемента в очередь.

- pop() — удаление элемента из начала очереди.

- front() — доступ к первому элементу без его удаления.

- back() — доступ к последнему элементу.

- size() — возвращает количество элементов в очереди.

3. priority\_queue (очередь с приоритетом)

Очередь с приоритетом также основана на принципе FIFO, но элементы в ней извлекаются в порядке их приоритета, а не в порядке добавления. Элемент с самым высоким приоритетом будет извлечен первым. Примером использования может быть система управления задачами с различными уровнями важности.

Основные методы контейнера priority\_queue:

- empty() — проверка, пуста ли очередь.

- push(const Type& val) — добавление элемента с приоритетом.

- pop() — удаление элемента с самым высоким приоритетом.

- top() — доступ к элементу с наивысшим приоритетом без его удаления.

- size() — возвращает количество элементов в очереди.

Особенности контейнеров-адаптеров

Контейнеры-адаптеры могут быть реализованы на основе последовательных контейнеров, таких как vector, deque, или list. Например:

- Контейнер stack по умолчанию реализован на основе контейнера deque, но может быть реализован и на основе vector или list;

- Контейнер queue может быть реализован на основе deque или list, но не vector, так как для очереди требуется поддержка функций back() и front();

- Контейнер priority\_queue обычно реализован на основе vector или deque, так как эти контейнеры поддерживают необходимый набор операций.

Ограничения на использование контейнеров-адаптеров

Контейнеры-адаптеры имеют некоторые ограничения:

- Они не могут использовать массивы (array) и однонаправленные списки (forward\_list), так как адаптеры требуют возможности добавления, удаления и доступа к последнему элементу контейнера;

- Для каждого адаптера могут быть применены только те функции, которые предоставлены его интерфейсом; прямой доступ к функциям базового контейнера невозможен.

Пример использования стека:

#include <stack>

#include <iostream>

using namespace std;

int main() {

stack<int> stk;

stk.push(10);

stk.push(20);

cout << "Top element: " << stk.top() << endl; // Выведет 20

stk.pop();

cout << "Top element after pop: " << stk.top() << endl; // Выведет 10

return 0;

}

Пример использования очереди:

#include <queue>

#include <iostream>

using namespace std;

int main() {

queue<int> q;

q.push(10);

q.push(20);

cout << "Front element: " << q.front() << endl; // Выведет 10

q.pop();

cout << "Front element after pop: " << q.front() << endl; // Выведет 20

return 0;

}

Пример использования очереди с приоритетом:

#include <queue>

#include <iostream>

using namespace std;

int main() {

priority\_queue<int> pq;

pq.push(10);

pq.push(20);

pq.push(15);

cout << "Top element: " << pq.top() << endl; // Выведет 20

pq.pop();

cout << "Top element after pop: " << pq.top() << endl; // Выведет 15

return 0;

}

Контейнеры-адаптеры являются мощным инструментом в языке C++, предоставляющим эффективные средства для работы с часто используемыми структурами данных. Они позволяют реализовать такие структуры, как стек и очередь, на основе других последовательных контейнеров, упрощая разработку и улучшая читаемость кода.

**3 Формулировка индивидуального задания**

Во всех вариантах индивидуальных заданий необходимо реализовать работу с контейнерами queue, stack, priority\_queue, которые будут хранить объекты классов по предметной области, указанной в таблице ниже (класс должен содержать функционал по предметной области (добавление, удаление, редактирование, поиск, фильтр данных)). Для контейнера реализовать добавление, удаление, редактирование, вывод содержимого контейнера на экран и в файл, поиск и сортировку элементов. Необходимо создать удобное пользовательское меню для тестирования всех реализованных функций.

**4 Код решения индивидуального задания**

// ReSharper disable CppClangTidyClangDiagnosticInvalidSourceEncoding

// ReSharper disable CppClangTidyConcurrencyMtUnsafe

// ReSharper disable CppClangTidyCertErr33C

#include <iostream>

#include <queue>

#include <stack>

#include <string>

#include <fstream>

#include <vector>

#include <algorithm>

class document

{

int id\_;

std::string title\_;

std::string content\_;

int priority\_;

public:

document(const int id, std::string title, std::string content, const int priority)

: id\_(id), title\_(std::move(title)), content\_(std::move(content)), priority\_(priority) {}

int get\_id() const { return id\_; }

std::string get\_title() const { return title\_; }

std::string get\_content() const { return content\_; }

int get\_priority() const { return priority\_; }

void set\_id(const int new\_id) { id\_ = new\_id; }

void set\_title(const std::string& new\_title) { title\_ = new\_title; }

void set\_content(const std::string& new\_content) { content\_ = new\_content; }

void set\_priority(const int new\_priority) { priority\_ = new\_priority; }

void display() const

{

std::cout << "ID: " << id\_ << "\nНазвание: " << title\_ << "\nСодержание: " << content\_ << "\nПриоритет: " << priority\_ << '\n';

}

bool operator<(const document& other) const { return id\_ < other.id\_; }

bool operator>(const document& other) const { return id\_ > other.id\_; }

};

class compare\_priority

{

public:

bool operator()(const document& d1, const document& d2) const

{

return d1.get\_priority() < d2.get\_priority();

}

};

void edit\_document(document& doc)

{

int new\_id, new\_priority;

std::string new\_title, new\_content;

std::cout << "Введите новый ID: ";

std::cin >> new\_id;

std::cin.ignore();

std::cout << "Введите новое название: ";

getline(std::cin, new\_title);

std::cout << "Введите новое содержание: ";

getline(std::cin, new\_content);

std::cout << "Введите новый приоритет: ";

std::cin >> new\_priority;

doc.set\_id(new\_id);

doc.set\_title(new\_title);

doc.set\_content(new\_content);

doc.set\_priority(new\_priority);

}

void find\_document\_in\_queue(const std::queue<document>& doc\_queue, const int id)

{

std::queue<document> temp\_queue = doc\_queue;

bool found = false;

while (!temp\_queue.empty())

{

const document& doc = temp\_queue.front();

if (doc.get\_id() == id)

{

doc.display();

found = true;

break;

}

temp\_queue.pop();

}

if (!found)

std::cout << "Документ с ID " << id << " не найден.\n";

}

void filter\_documents\_by\_priority(const std::queue<document>& doc\_queue, const int min\_priority)

{

std::queue<document> temp\_queue = doc\_queue;

std::cout << "Документы с приоритетом выше " << min\_priority << ":\n";

while (!temp\_queue.empty())

{

const document& doc = temp\_queue.front();

if (doc.get\_priority() >= min\_priority)

doc.display();

temp\_queue.pop();

}

}

void sort\_queue(std::queue<document>& doc\_queue)

{

std::vector<document> doc\_vector;

while (!doc\_queue.empty())

{

doc\_vector.push\_back(doc\_queue.front());

doc\_queue.pop();

}

std::sort(doc\_vector.begin(), doc\_vector.end(), [](const document& d1, const document& d2) { return d1.get\_id() < d2.get\_id(); });

std::cout << "Отсортированные документы:\n";

for (const auto& doc : doc\_vector)

{

doc.display();

doc\_queue.push(doc);

}

}

void display\_queue(std::queue<document> doc\_queue)

{

while (!doc\_queue.empty())

{

doc\_queue.front().display();

doc\_queue.pop();

}

}

void display\_stack(std::stack<document> doc\_stack)

{

while (!doc\_stack.empty())

{

doc\_stack.top().display();

doc\_stack.pop();

}

}

void display\_priority\_queue(std::priority\_queue<document, std::vector<document>, compare\_priority> doc\_priority\_queue)

{

while (!doc\_priority\_queue.empty())

{

doc\_priority\_queue.top().display();

doc\_priority\_queue.pop();

}

}

void save\_to\_file(std::queue<document> doc\_queue, const std::string& filename)

{

std::ofstream out\_file(filename);

while (!doc\_queue.empty())

{

out\_file << "ID: " << doc\_queue.front().get\_id() << " | Название: " << doc\_queue.front().get\_title()

<< " | Содержание: " << doc\_queue.front().get\_content() << " | Приоритет: " << doc\_queue.front().get\_priority() << '\n';

doc\_queue.pop();

}

out\_file.close();

std::cout << "Данные сохранены в файл: " << filename << '\n';

}

void end()

{

system("pause");

system("cls");

}

int main()

{

setlocale(LC\_ALL, "Russian");

std::queue<document> doc\_queue;

std::stack<document> doc\_stack;

std::priority\_queue<document, std::vector<document>, compare\_priority> doc\_priority\_queue;

int choice = 0;

do

{

if (choice) end();

std::cout << "\nМеню:\n";

std::cout << "1. Добавить документ\n";

std::cout << "2. Удалить документ (очередь)\n";

std::cout << "3. Показать все документы (очередь)\n";

std::cout << "4. Сохранить документы в файл\n";

std::cout << "5. Добавить документ в стек\n";

std::cout << "6. Показать все документы (стек)\n";

std::cout << "7. Добавить документ в очередь с приоритетом\n";

std::cout << "8. Показать все документы (приоритетная очередь)\n";

std::cout << "9. Поиск документа по ID\n";

std::cout << "10. Фильтрация документов по приоритету\n";

std::cout << "11. Редактировать документ (очередь)\n";

std::cout << "12. Сортировка документов (очередь)\n";

std::cout << "13. Выход\n";

std::cout << "Выберите действие: ";

std::cin >> choice;

switch (choice)

{

case 1:

{

int id, priority;

std::string title, content;

std::cout << "Введите ID документа: ";

std::cin >> id;

std::cin.ignore();

std::cout << "Введите название документа: ";

getline(std::cin, title);

std::cout << "Введите содержание документа: ";

getline(std::cin, content);

std::cout << "Введите приоритет документа: ";

std::cin >> priority;

doc\_queue.emplace(id, title, content, priority);

break;

}

case 2:

{

if (!doc\_queue.empty())

{

std::cout << "Удален документ:\n";

doc\_queue.front().display();

doc\_queue.pop();

}

else

std::cout << "Очередь пуста.\n";

break;

}

case 3:

{

if (!doc\_queue.empty())

display\_queue(doc\_queue);

else

std::cout << "Очередь пуста.\n";

break;

}

case 4:

{

std::string filename;

std::cout << "Введите имя файла: ";

std::cin >> filename;

save\_to\_file(doc\_queue, filename);

break;

}

case 5:

{

int id, priority;

std::string title, content;

std::cout << "Введите ID документа: ";

std::cin >> id;

std::cin.ignore();

std::cout << "Введите название документа: ";

getline(std::cin, title);

std::cout << "Введите содержание документа: ";

getline(std::cin, content);

std::cout << "Введите приоритет документа: ";

std::cin >> priority;

doc\_stack.emplace(id, title, content, priority);

break;

}

case 6:

{

if (!doc\_stack.empty())

display\_stack(doc\_stack);

else

std::cout << "Стек пуст.\n";

break;

}

case 7:

{

int id, priority;

std::string title, content;

std::cout << "Введите ID документа: ";

std::cin >> id;

std::cin.ignore();

std::cout << "Введите название документа: ";

getline(std::cin, title);

std::cout << "Введите содержание документа: ";

getline(std::cin, content);

std::cout << "Введите приоритет документа: ";

std::cin >> priority;

doc\_priority\_queue.emplace(id, title, content, priority);

break;

}

case 8:

{

if (!doc\_priority\_queue.empty())

display\_priority\_queue(doc\_priority\_queue);

else

std::cout << "Приоритетная очередь пуста.\n";

break;

}

case 9:

{

int id;

std::cout << "Введите ID для поиска: ";

std::cin >> id;

find\_document\_in\_queue(doc\_queue, id);

break;

}

case 10:

{

int min\_priority;

std::cout << "Введите минимальный приоритет для фильтрации: ";

std::cin >> min\_priority;

filter\_documents\_by\_priority(doc\_queue, min\_priority);

break;

}

case 11:

{

if (!doc\_queue.empty())

{

auto& doc = const\_cast<document&>(doc\_queue.front());

std::cout << "Редактирование документа:\n";

doc.display();

edit\_document(doc);

}

else

std::cout << "Очередь пуста.\n";

break;

}

case 12:

{

if (!doc\_queue.empty())

sort\_queue(doc\_queue);

else

std::cout << "Очередь пуста.\n";

break;

}

case 13:

std::cout << "Выход из программы.\n";

break;

default:

std::cout << "Неверный выбор.\n";

break;

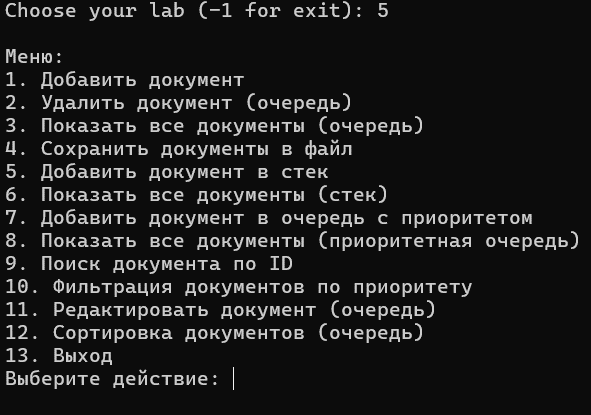
}

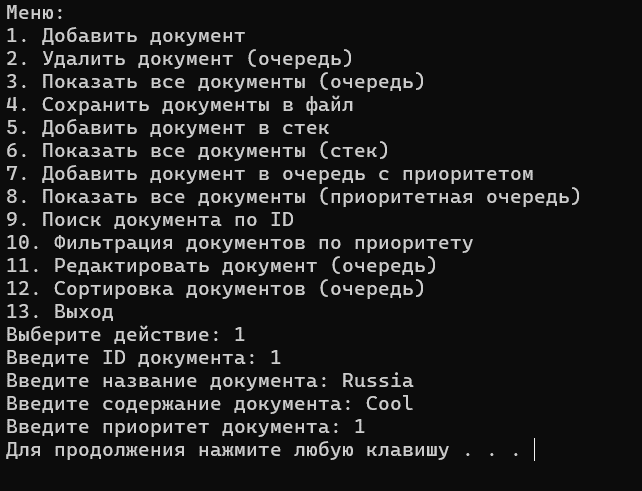
} while (choice != 13);

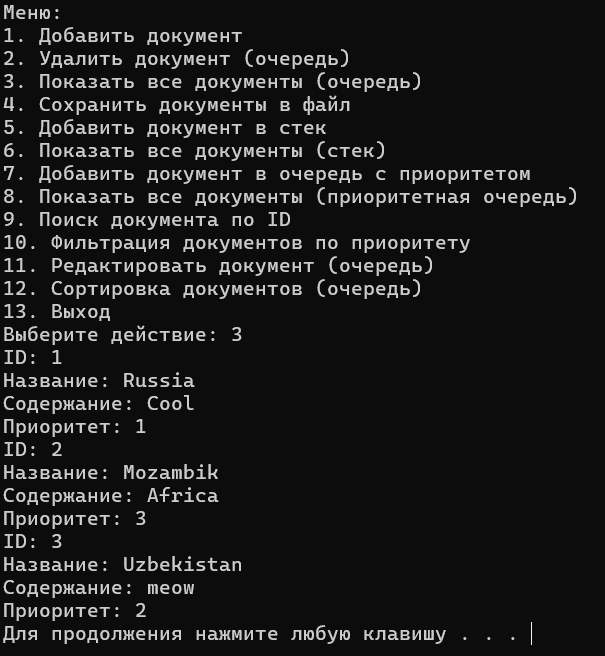
return 0;

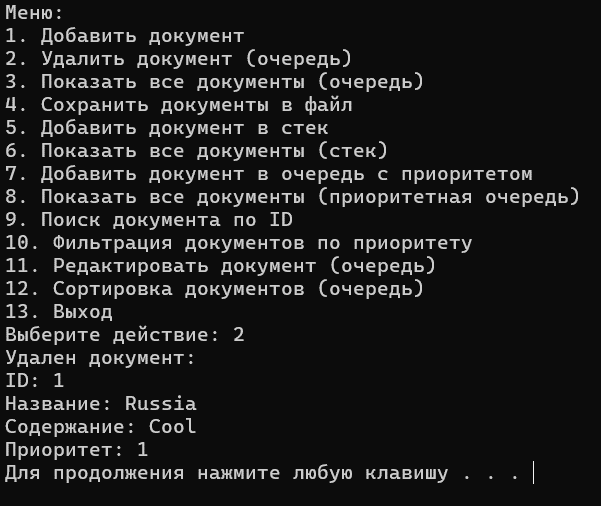
}

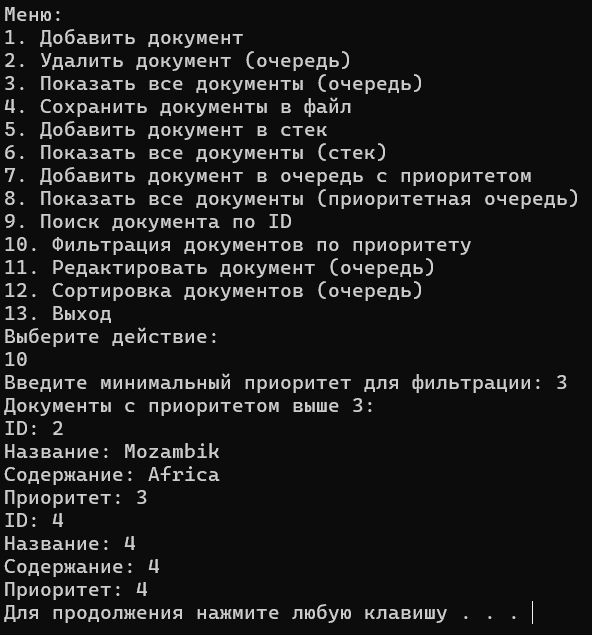
**5 Скриншоты выполнения индивидуального задания**

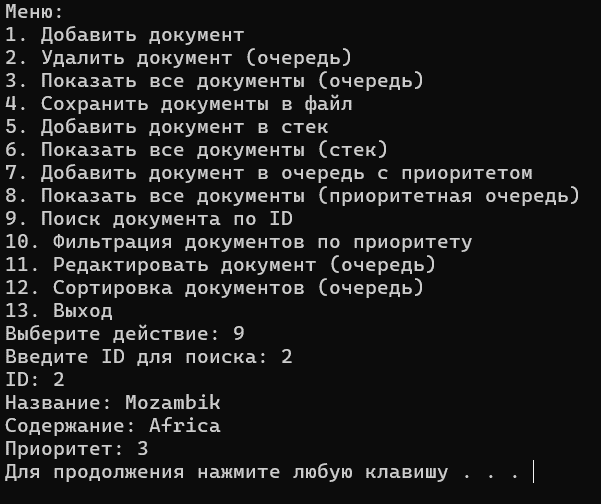


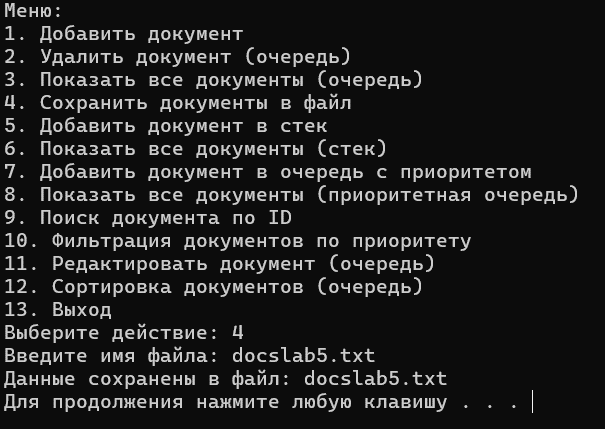


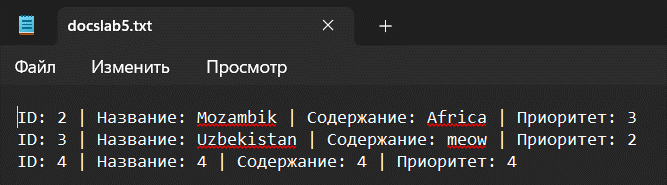


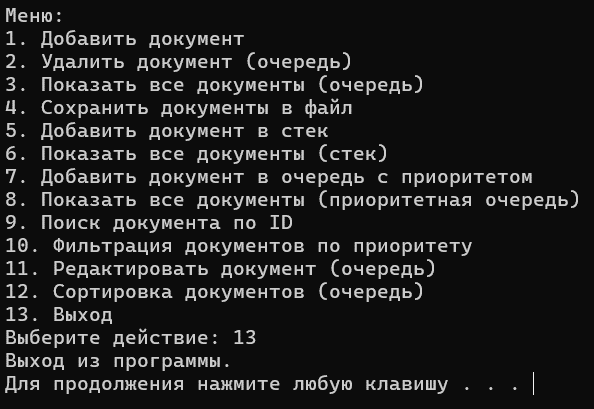












**6 Ответы на контрольные вопросы**

1. В чем различие между последовательными, ассоциативными и адаптерами контейнерами? Приведите примеры использования каждой категории контейнеров в проекте.

- Последовательные контейнеры: эти контейнеры хранят элементы в определённом порядке, который зависит от порядка их добавления. Примеры: vector, deque, list. Эти контейнеры обеспечивают возможность доступа к элементам по индексу и могут хранить дубликаты. Пример использования: vector часто используется для хранения динамических массивов данных в проектах, где важен произвольный доступ к элементам по индексу, например, для хранения списка студентов;

- Ассоциативные контейнеры: эти контейнеры хранят элементы в виде ключ-значение, а порядок элементов определяется не их порядком вставки, а значениями ключей (в порядке возрастания). Примеры: map, set. Ассоциативные контейнеры не допускают дубликаты ключей. Пример использования: map используется для хранения пар значений, например, имени пользователя и его идентификатора в системе, когда важна уникальность ключей и быстрый доступ по ним;

- Адаптеры контейнеров: это контейнеры, которые реализуют определенные структуры данных (например, стек, очередь), используя другие контейнеры как основу. Примеры: stack, queue, priority\_queue. Они ограничивают интерфейс базового контейнера, предоставляя только нужные операции. Пример использования: stack можно использовать для реализации обратной обработки данных, например, выполнения рекурсивных функций или отката изменений.

2. Опишите основные характеристики и пример использования контейнера stack.

Основные характеристики:

- Контейнер stack реализует структуру данных по принципу LIFO (Last In, First Out — последним пришел, первым вышел);

- Он поддерживает операции добавления элемента в вершину стека, удаления верхнего элемента и доступа к верхнему элементу без его удаления;

- По умолчанию реализован на основе контейнера deque.

Пример использования: стек можно использовать для организации обработки вызовов функций, хранения состояния для возврата назад при работе с рекурсией или откатах (undo).

Пример:

#include <stack>

#include <iostream>

using namespace std;

int main() {

stack<int> stk;

stk.push(10);

stk.push(20);

cout << "Top element: " << stk.top() << endl; // Выведет 20

stk.pop();

cout << "Top element after pop: " << stk.top() << endl; // Выведет 10

return 0;

}

3. Опишите основные характеристики и пример использования контейнера queue.

Основные характеристики:

- Контейнер queue реализует структуру данных по принципу FIFO (First In, First Out — первым пришел, первым вышел);

- Он поддерживает добавление элемента в конец очереди и извлечение элемента из начала очереди;

- Очередь может быть реализована на основе deque или list.

Пример использования: очередь используется для обработки данных в порядке поступления, например, в системе обработки заявок пользователей, где важно, чтобы запросы обрабатывались в том порядке, в котором они поступили.

Пример:

#include <queue>

#include <iostream>

using namespace std;

int main() {

queue<int> q;

q.push(10);

q.push(20);

cout << "Front element: " << q.front() << endl; // Выведет 10

q.pop();

cout << "Front element after pop: " << q.front() << endl; // Выведет 20

return 0;

}

4. Опишите основные характеристики и пример использования контейнера priority\_queue.

Основные характеристики:

- Контейнер priority\_queue также реализует структуру данных по принципу FIFO, но с учетом приоритета элементов;

- Элемент с наивысшим приоритетом всегда оказывается первым;

- По умолчанию используется контейнер vector для хранения данных;

Пример использования: очередь с приоритетом может быть полезна при планировании задач, где задачи с наивысшим приоритетом должны выполняться первыми. Например, планировщик задач операционной системы.

Пример:

#include <queue>

#include <iostream>

using namespace std;

int main() {

priority\_queue<int> pq;

pq.push(10);

pq.push(20);

pq.push(15);

cout << "Top element: " << pq.top() << endl; // Выведет 20

pq.pop();

cout << "Top element after pop: " << pq.top() << endl; // Выведет 15

return 0;

}

5. Опишите и приведите примеры использования основных методов контейнера stack.

- empty() — проверяет, пуст ли стек.

- push(const Type& val) — добавляет элемент в стек.

- pop() — удаляет элемент с вершины стека.

- top() — возвращает элемент на вершине стека без его удаления.

- size() — возвращает количество элементов в стеке.

Пример использования:

stack<int> stk;

stk.push(10);

stk.push(20);

cout << "Top element: " << stk.top() << endl; // Выведет 20

stk.pop();

cout << "Stack size: " << stk.size() << endl; // Выведет 1

6. Опишите и приведите примеры использования основных методов контейнера queue.

- empty() — проверяет, пуста ли очередь.

- push(const Type& val) — добавляет элемент в конец очереди.

- pop() — удаляет элемент из начала очереди.

- front() — возвращает первый элемент без его удаления.

- back() — возвращает последний элемент без его удаления.

- size() — возвращает количество элементов в очереди.

Пример использования:

queue<int> q;

q.push(10);

q.push(20);

cout << "Front element: " << q.front() << endl; // Выведет 10

cout << "Back element: " << q.back() << endl; // Выведет 20

q.pop();

cout << "Queue size: " << q.size() << endl; // Выведет 1

7. Опишите и приведите примеры использования основных методов контейнера priority\_queue.

- empty() — проверяет, пуста ли очередь.

- push(const Type& val) — добавляет элемент в очередь с приоритетом.

- pop() — удаляет элемент с наивысшим приоритетом.

- top() — возвращает элемент с наивысшим приоритетом без его удаления.

- size() — возвращает количество элементов в очереди.

Пример использования:

priority\_queue<int> pq;

pq.push(10);

pq.push(20);

pq.push(15);

cout << "Top element: " << pq.top() << endl; // Выведет 20

pq.pop();

cout << "Priority queue size: " << pq.size() << endl; // Выведет 2

8. В чем различие между контейнерами vector и list?

vector:

- Хранит элементы в непрерывной области памяти;

- Обеспечивает быстрый произвольный доступ по индексу;

- Добавление или удаление элементов в середине или начале требует перемещения элементов, что может быть медленным.

list:

- Хранит элементы как двусвязный список;

- Не поддерживает доступ по индексу — требуется последовательный доступ;

- Добавление и удаление элементов в любой части списка происходит быстро, так как не требует перемещения элементов.

9. В чем различие между контейнерами array и vector?

array:

- Имеет фиксированный размер, который должен быть известен на этапе компиляции;

- Не поддерживает динамическое изменение размера.

vector:

- Имеет динамический размер, который можно изменять во время выполнения программы;

- Может автоматически расширяться по мере добавления новых элементов.

10. В чем различие между контейнерами queue, stack, priority\_queue?

queue: реализует принцип FIFO — элементы добавляются в конец и извлекаются из начала.

stack: реализует принцип LIFO — элементы добавляются и извлекаются с одного конца (вершины).

priority\_queue: реализует принцип FIFO, но с приоритетом элементов — элемент с наивысшим приоритетом извлекается первым.

**7 Выводы по лабораторной работе**

Вывод: я изучил адаптеры контейнеров библиотеки STL в языке C++, их особенности, функциональности и применения, освоил работу с контейнерами-адаптерами stack, queue и priority\_queue, их методы и возможности для решения задач, требующих специфической структуры данных.