Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет инженерно-экономический

Кафедра экономической информатики

**Отчет к лабораторной работе №8**

**по теме**

**«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ И ПРОГРАММИРОВАНИЕ.АДАПТЕРЫ КОНТЕЙНЕРОВ»**

|  |  |
| --- | --- |
| Выполнила: | Вдовенко Н. Д.  студент группы 272303 |
| Проверила: | Салапура М. Н.  ст. преподаватель |

Минск 2023

Общая постановка задачи

**Цель работы:** изучить ассоциативные адаптеры библиотеки STL.

**Индивидуальное задание:**

Необходимо создать контейнеры, которые будут хранить объекты классов по предметной области «банковские сотрудники». Для контейнера реализовать добавление, удаление, редактирование, вывод содержимого контейнера на экран и в файл, поиск и сортировку элементов. Необходимо создать удобное пользовательское меню. Необходимо реализовать работу с контейнерами queue, stack, priority\_queue.

**Краткие теоретические сведения:**

Контейнеры — это объекты, которые содержат в себе набор (совокупность) других объектов.

Библиотека STL содержит три контейнера-адаптера: 1. Контейнер queue соответствует принципу обработки элементов FIFO (First In – First Out, первым вошел — первым вышел). Первый элемент, помещенный в очередь, должен первым извлекается из очереди. Очередь в проектировании и программировании используется, когда нужно совершить какие-то действия в порядке их поступления, выполнив их последовательно. Примером может служить организация событий в операционной системе Windows. 2. Контейнер priority\_queue соответствует принципу обработки элементов FIFO (first in – first out, первым вошел — первым вышел), но первым в очереди всегда оказывается элемент с наибольшим значением определенной характеристики (priority, приоритетом). В качестве примера очереди с приоритетом можно рассмотреть список задач работника. Когда работник заканчивает одну задачу, он переходит к следующей самой приоритетной. Начальник добавляет задачи в список, указывая их приоритет. 3. Контейнер stack соответствует принципу обработки элементов LIFO (last in – first out, последним вошел — первым вышел). Последний элемент, помещенный в стек, становится первым извлекаемым из стека элементом. Поскольку контейнеры-адаптеры не поддерживают итераторы, их нельзя использовать в алгоритмах STL. Адаптер контейнера получает контейнер существующего типа и заставляет его действовать как другой. Например, адаптер stack получает любой из последовательных контейнеров (array или forward list) и заставляет его работать подобно стеку.

**Диаграмма классов:**

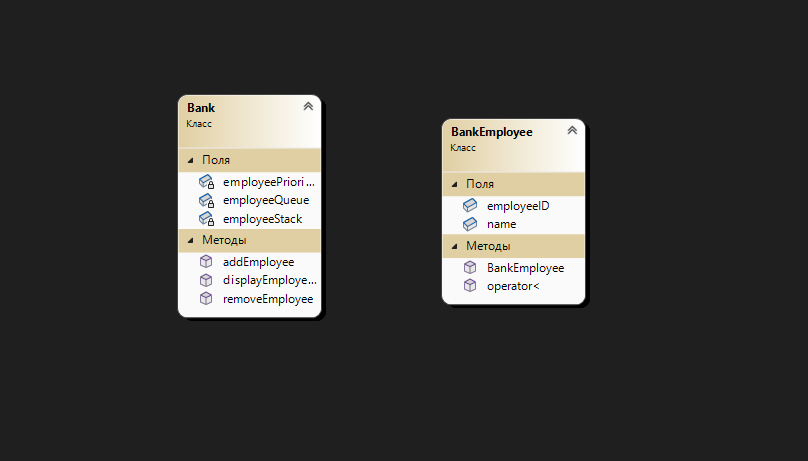


Рисунок 1 – Диаграмма классов

**Контрольные вопросы:**

1. Последовательные контейнеры хранят данные в линейном порядке (например, std::vector), ассоциативные используются для хранения данных в виде уникальных ключей и значений (например, std::map), адаптеры предоставляют интерфейс к другим контейнерам (например, std::stack, std::queue). Примеры: Использование std::vector для хранения элементов вектора, std::map для хранения пары ключ-значение, std::stack для управления стеком.
2. LIFO (Last-In-First-Out) структура данных. Методы включают push, pop, top.

Пример использования:

std::stack<int> myStack;

myStack.push(5);

int x = myStack.top();

myStack.pop();

1. FIFO (First-In-First-Out) структура данных. Методы включают push, pop, front, back.

Пример использования:

std::queue<int> myQueue;

myQueue.push(5);

int x = myQueue.front();

myQueue.pop();

1. Структура данных очередь с приоритетом. Методы включают push, pop, top.

Пример использования:

std::priority\_queue<int> myPriorityQueue;

myPriorityQueue.push(5);

int x = myPriorityQueue.top();

myPriorityQueue.pop();

1. push() - добавление элемента в стек. pop() - удаление верхнего элемента из стека. top() - доступ к верхнему элементу стека.
2. push() - добавление элемента в очереди. pop() - удаление первого элемента из очереди. front() - доступ к первому элементу в очереди. back() - доступ к последнему элементу в очереди (если поддерживается).
3. push() - добавление элемента в очередь с приоритетом. pop() - удаление верхнего элемента из очереди с приоритетом. top() - доступ к верхнему элементу очереди с приоритетом.
4. std::vector: динамический массив с быстрым доступом к элементам, медленные вставки/удаления в середине. std::list: двусвязный список с быстрыми вставками/удалениями в середине, медленный доступ к элементам.
5. std::array - статический массив фиксированного размера. std::vector - динамический массив с изменяемым размером.
6. std::queue: FIFO структура. std::stack: LIFO структура. std::priority\_queue: очередь с приоритетом, извлечение элементов по заданному приоритету.

**Листинг кода:**

#include <iostream>

#include <iomanip>

#include <queue>

#include <stack>

#include <vector>

#include <Windows.h>

class BankEmployee {

public:

std::string name;

int employeeID;

BankEmployee(std::string n, int id) : name(n), employeeID(id) {}

bool operator<(const BankEmployee& other) const {

return employeeID < other.employeeID;

}

};

class Bank {

private:

std::queue<BankEmployee> employeeQueue;

std::stack<BankEmployee> employeeStack;

std::priority\_queue<BankEmployee> employeePriorityQueue;

public:

void addEmployee(const BankEmployee& employee) {

employeeQueue.push(employee);

employeeStack.push(employee);

employeePriorityQueue.push(employee);

}

void removeEmployee() {

if (!employeeQueue.empty()) {

employeeQueue.pop();

}

if (!employeeStack.empty()) {

employeeStack.pop();

}

if (!employeePriorityQueue.empty()) {

employeePriorityQueue.pop();

}

}

void displayEmployees() const {

std::cout << "\nEmployees in the queue:" << std::endl;

std::cout << std::setw(20) << "Employee Name" << std::setw(15) << "Employee ID" << std::endl;

std::cout << std::setw(35) << std::setfill('-') << "" << std::setfill(' ') << std::endl;

std::queue<BankEmployee> tempQueue = employeeQueue;

while (!tempQueue.empty()) {

std::cout << std::setw(20) << tempQueue.front().name << std::setw(15) << tempQueue.front().employeeID << std::endl;

tempQueue.pop();

}

std::cout << "\nEmployees in the stack:" << std::endl;

std::cout << std::setw(20) << "Employee Name" << std::setw(15) << "Employee ID" << std::endl;

std::cout << std::setw(35) << std::setfill('-') << "" << std::setfill(' ') << std::endl;

std::stack<BankEmployee> tempStack = employeeStack;

while (!tempStack.empty()) {

std::cout << std::setw(20) << tempStack.top().name << std::setw(15) << tempStack.top().employeeID << std::endl;

tempStack.pop();

}

std::cout << "\nEmployees in the priority queue:" << std::endl;

std::cout << std::setw(20) << "Employee Name" << std::setw(15) << "Employee ID" << std::endl;

std::cout << std::setw(35) << std::setfill('-') << "" << std::setfill(' ') << std::endl;

std::priority\_queue<BankEmployee> tempPriorityQueue = employeePriorityQueue;

while (!tempPriorityQueue.empty()) {

std::cout << std::setw(20) << tempPriorityQueue.top().name << std::setw(15) << tempPriorityQueue.top().employeeID << std::endl;

tempPriorityQueue.pop();

}

}

};

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Bank bank;

while (true) {

std::cout << "\nBank Employee Management System\n";

std::cout << "1. Add Employee\n";

std::cout << "2. Remove Employee\n";

std::cout << "3. Display Employees\n";

std::cout << "4. Exit\n";

std::cout << "Enter your choice: ";

int choice;

std::cin >> choice;

switch (choice) {

case 1: {

std::string name;

int id;

std::cout << "Enter employee name: ";

std::cin >> name;

std::cout << "Enter employee ID: ";

std::cin >> id;

bank.addEmployee(BankEmployee(name, id));

break;

}

case 2:

bank.removeEmployee();

std::cout << "Employee removed.\n";

break;

case 3:

bank.displayEmployees();

break;

case 4:

std::cout << "Exiting program.\n";

return 0;

default:

std::cout << "Invalid choice. Please try again.\n";

}

}

return 0;

}

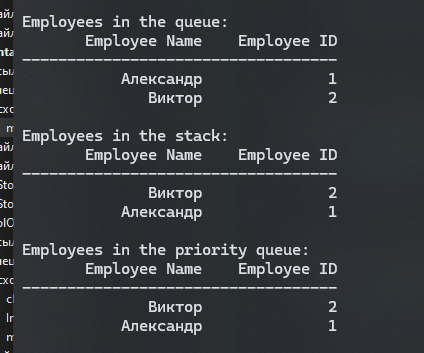


Рисунок 2 – Просмотр информации

**Вывод**: в ходе выполнения данной лабораторной работы были изучены и применены основные принципы работы с адаптерами.