

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення**



ЗВІТ

**до лабораторної роботи №8
на тему: «Лінійні структури даних»
з дисципліни: «Алгоритми і структури даних»**

Лектор:

доц. кафедри ПЗ
Коротеева Т. О.

Виконав:

ст. гр. ПЗ-22
Чаус О. М.

Прийняв:

асист. кафедри ПЗ
Франко А. В.

« ____ » _____ 2022 р.

Σ = ____

Тема роботи: Лінійні структури даних.

Мета роботи: познайомитися з лінійними структурами даних (стек, черга, дек, список) та отримати навички програмування алгоритмів, що їх обробляють.

Теоретичні відомості

Стек, черга, дек, список відносяться до класу лінійних динамічних структур.

Зі стеку (stack) можна видалити тільки той елемент, який був у нього доданий останнім: стек працює за принципом «останнім прийшов – першим пішов» (last-in, first-out – LIFO).

З черги (queue), навпаки, можна видалити тільки той елемент, який знаходився в черзі довше за всіх: працює принцип «першим прийшов – першим пішов» (first-in, first-out – FIFO).

Дек - це впорядкована лінійна динамічно змінювана послідовність елементів, у якій виконуються такі умови: 1) новий елемент може приєднуватися з обох боків послідовності; 2) вибірка елементів можлива також з обох боків послідовності. Дек називають реверсивною чергою або чергою з двома боками.

У зв'язаному списку (або просто списку; linked list) елементи лінійно впорядковані, але порядок визначається не номерами, як у масиві, а вказівниками, що входять до складу елементів списку. Списки є зручним способом реалізації динамічних множин.

Елемент двобічно зв'язаного списку (doubly linked list) – це запис, що містить три поля: key (ключ) і два вказівники next (наступний) і prev (попередній). Крім цього, елементи списку можуть містити додаткові дані.

У кільцевому списку (circular list) поле prev голови списку вказує на хвіст списку, а поле next хвоста списку вказує на голову списку.

Індивідуальне завдання

Розробити програму, яка читає з клавіатури послідовність даних, жодне з яких не повторюється, зберігає їх до структури даних (згідно з варіантом) та видає на екран такі характеристики:

- кількість елементів;
- мінімальний та максимальний елемент (для символів за кодом)
- третій елемент з початку послідовності та другий з кінця послідовності;
- елемент, що стоїть перед мінімальним елементом та елемент, що стоїть після максимального;
- знайти позицію елемента, значення якого задається з клавіатури;
- об'єднати дві структури в одну.

Всі характеристики потрібно визначити із заповненої структури даних.

Варіант 1: черга цілих.

Код програми

Файл **myqueue.h**

```
#ifndef MYQUEUE_H
#define MYQUEUE_H
#include <cstdint>
#include <vector>

#define QUEUE_CAPACITY 8
class MyQueue
{
public:
    MyQueue();
    MyQueue(const MyQueue &second);
    std::size_t size() const;
    bool empty() const;
    int front() const;
    int back() const;
    void push(int value);
    int pop();
    int min() const;
    int max() const;
    int third_from_start() const;
    int second_from_end() const;
    int before_min() const;
    int after_max() const;
    int index_of(int value) const;
    MyQueue operator+(const MyQueue& second) const;
    void operator=(const MyQueue& second);
    std::vector<int> to_vector() const;
    ~MyQueue();
private:
    int* array;
    int queue_size;
    int queue_capacity = QUEUE_CAPACITY;
    void reallocate_memory();
};

#endif // MYQUEUE_H
```

Файл **myqueue.cpp**

```
#include "myqueue.h"
#include <stdexcept>

MyQueue::MyQueue() {
    this->queue_size = 0;
    this->array = new int[this->queue_capacity];
}

MyQueue::MyQueue(const MyQueue &second) {
    this->queue_size = second.queue_size;
    this->queue_capacity = second.queue_capacity;
    this->array = new int[this->queue_capacity];
    for(int i = 0; i < this->queue_size; i++) {
        this->array[i] = second.array[i];
    }
}

std::size_t MyQueue::size() const {
    return this->queue_size;
}

int MyQueue::front() const {
    if (this->queue_size > 0) {
        return this->array[0];
    }
    return 0;
}
```

```

int MyQueue::back() const {
    if (this->queue_size > 0) {
        return this->array[this->queue_size - 1];
    }
    return 0;
}

void MyQueue::push(int value) {
    this->queue_size++;
    this->reallocate_memory();
    array[queue_size - 1] = value;
}

int MyQueue::pop() {
    if (queue_size > 0) {
        int item = this->array[0];
        for(int i = 1; i < this->queue_size; i++)
            this->array[i-1] = this->array[i];
        this->queue_size--;
        this->reallocate_memory();
        return item;
    }
    throw std::out_of_range("Queue is empty\n");
}

std::vector<int> MyQueue::to_vector() const {
    std::vector<int> values;
    int size = queue_size;
    for (int i = 0; i < size; i++) {
        values.push_back(this->array[i]);
    }
    return values;
}

int MyQueue::min() const {
    int min = 0;
    if (this->queue_size > 0) {
        min = this->array[0];
        for (int i = 0; i < this->queue_size; i++) {
            if (array[i] < min)
                min = array[i];
        }
    }
    return min;
}

int MyQueue::max() const {
    int max = 0;
    if (this->queue_size > 0) {
        max = this->array[0];
        for (int i = 0; i < this->queue_size; i++) {
            if (array[i] > max)
                max = array[i];
        }
    }
    return max;
}

int MyQueue::third_from_start() const {
    if (this->queue_size > 2)
        return this->array[2];
    return 0;
}

```

```

int MyQueue::second_from_end() const {
    if (this->queue_size > 1)
        return this->array[this->queue_size - 2];
    return 0;
}

int MyQueue::index_of(int value) const {
    if (this->queue_size > 0)
        for (int i = 0; i < this->queue_size; i++)
            if (this->array[i] == value)
                return i;
    return -1;
}

int MyQueue::before_min() const {
    int min = 0;
    if (this->queue_size > 0) {
        for (int i = 0; i < this->queue_size; i++) {
            if (array[i] < array[min])
                min = i;
        }
    }
    if (min > 0)
        return this->array[min - 1];
    else return 0;
}

int MyQueue::after_max() const {
    int max = 0;
    if (this->queue_size > 0) {
        for (int i = 0; i < this->queue_size; i++) {
            if (array[i] > array[max])
                max = i;
        }
    }
    if (max < this->queue_size)
        return this->array[max + 1];
    else return 0;
}

MyQueue MyQueue::operator+(const MyQueue& second) const {
    MyQueue new_queue;
    for (int i = 0; i < this->size(); i++)
        new_queue.push(this->array[i]);
    for (int i = 0; i < second.size(); i++)
        new_queue.push(second.array[i]);
    return new_queue;
}

void MyQueue::reallocate_memory() {
    int old_capacity = this->queue_capacity;
    this->queue_capacity = QUEUE_CAPACITY;
    while (this->queue_size >= this->queue_capacity)
        queue_capacity += QUEUE_CAPACITY;
    if (!(old_capacity == this->queue_capacity)) {
        int* new_array = new int[this->queue_capacity];
        int i = 0;
        while (i < this->queue_size) {
            new_array[i] = this->array[i];
            i++;
        }
        delete[] this->array;
        this->array = new_array;
    }
}

```

```

void MyQueue::operator=(const MyQueue& second) {
    delete[] this->array;
    this->queue_size = second.queue_size;
    this->queue_capacity = second.queue_capacity;
    this->array = new int[this->queue_capacity];
    for(int i = 0; i < this->queue_size; i++) {
        this->array[i] = second.array[i];
    }
}

MyQueue::~MyQueue() {
    delete[] this->array;
}

```

Зображення програми

The image displays two screenshots of a C++ application interface for managing queues. The interface is titled "MainWindow" and features two main panels, each representing a queue.

Top Screenshot:

- Left Queue:** Contains the elements "1, 12, 123, 1234, 1, -12345". The "Push" button is labeled with "-12345". The "Pop" button is labeled with "1".
- Right Queue:** Contains the elements "1, 5, 4, 6, 4". The "Push" button is labeled with "4". The "Pop" button is labeled with "1".
- Info Section:** Displays various statistics for each queue, including "Queue size", "Front", "Back", "Min element", "Max element", "Third element from start", and "Second element from end".
- Merge Button:** A button labeled "Merge" is located at the bottom left.

Bottom Screenshot:

- Left Queue:** After the merge operation, the left queue now contains all elements from both previous queues: "1, 12, 123, 1234, 1, -12345, 1, 5, 4, 6, 4". The "Push" button is labeled with "-12345". The "Pop" button is labeled with "4".
- Right Queue:** Remains unchanged with elements "1, 5, 4, 6, 4". The "Push" button is labeled with "4". The "Pop" button is labeled with "1".
- Info Section:** The statistics for the left queue are updated to reflect the merged state, showing a "Queue size" of 11.
- Merge Button:** The "Merge" button is still present at the bottom left.

MainWindow

6, 4, 3

1, 5, 4, 6, 4

Push

3

Pop

Push

4

Pop

Info:

Find index

Info:

Find index

Queue size: 3

Front: 6

Queue size: 5

Front: 1

Min element: 3

Before min: 4

Back: 3

Min element: 1

Before min: 0

Back: 4

Max element: 6

After max: 4

Max element: 6

After max: 4

Third element from start: 3

Third element from start: 4

Second element from end: 4

Second element from end: 6

Merge

Висновок:

познайомився з лінійними структурами даних (стек, черга, дек, список) та отримав навички програмування алгоритмів, що їх обробляють.