МІНІСТРЕСТВО ОСВІТИ Й НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“Харківський Політехнічний Інститут”

Кафедра управління проєктами в інформаційних технологіях

Звіт з лабораторної роботи №2

“Однозв’язні та двозв’язні списки й робота з ними”

з дисципліни

“Алгоритми та структури даних”

Варіант №2

Перевірив: ст. викл. каф. УПІТ Мошко Є.О.

Виконав: ст. гр. КН-1223г Ставицький А.А.

Харків – 2024

**Мета**: ознайомитися із основними способами організації стеків, черг, деків та особливостями їх програмної реалізації. Набути практичних навичок роботи зі стеками, чергами та деками.

**Стек** — це структура даних, яка працює за принципом “останній прийшов — перший пішов” (LIFO). Це означає, що елемент, який був доданий останнім, буде видалений першим. Уявіть це як стопку тарілок, де нова тарілка завжди кладеться зверху, а коли потрібно взяти тарілку, ви завжди берете ту, що зверху.

Стек підтримує основні операції: додавання елемента на вершину стека (push), видалення та повернення елемента з вершини стека (pop), перегляд верхнього елемента без його видалення (peek або top), а також перевірку, чи є стек порожнім (isEmpty) або повним (isFull).

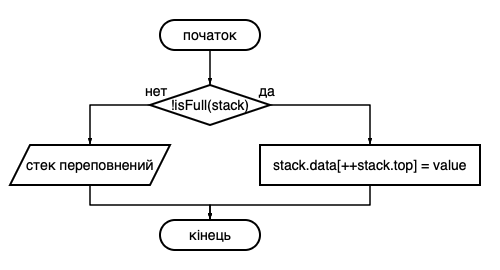


Рис 1. Блок-схема функції push()

if (!isFull(stack)) {

stack.data[++stack.top] = value;

} else {

cout << RED << "Стек переповнений!" << RESET << endl;

}

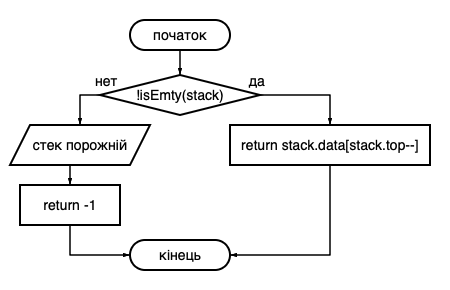


Рис. 2 Блок-схема функції pop()

if (!isEmty(stack)) {

*return* stack.data[stack.top--];

} else {

cout << RED << "Стек порожній!" << RESET << endl;

*return* -1;

}

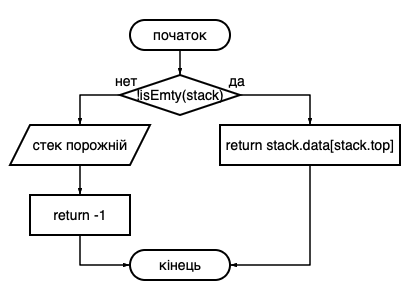


Рис. 3 Блок-схема функції peek()

if(!isEmty(stack)) {

*return* stack.data[stack.top];

} else {

cout << RED << "Стек порожній" << RESET << endl;

*return* -1;

}

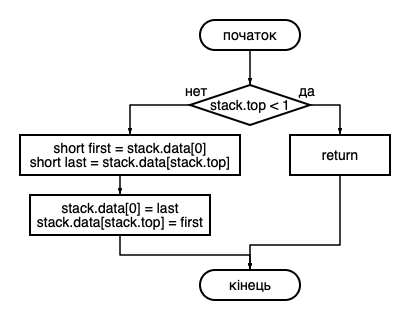


Рис. 4 Блок-схема функції swap\_first\_last()

if (stack.top < 1) *return*;

short first = stack.data[0];

short last = stack.data[stack.top];

stack.data[0] = last;

stack.data[stack.top] = first;

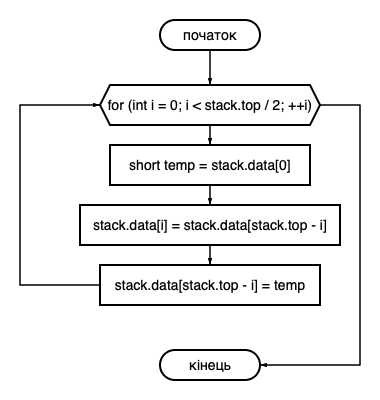


Рис. 5 Блок-схема функції reverse()

for (int i = 0; i < stack.top / 2; ++i) {

short temp = stack.data[i];

stack.data[i] = stack.data[stack.top - i];

stack.data[stack.top - i] = temp;

}

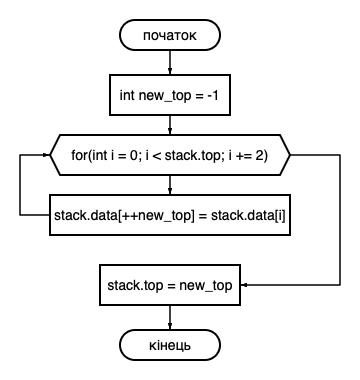


Рис. 6 Блок-схема функції delete\_every\_second()

int new\_top = -1;

for(int i = 0; i < stack.top; i += 2) {

stack.data[++new\_top] = stack.data[i];

}

stack.top = new\_top;

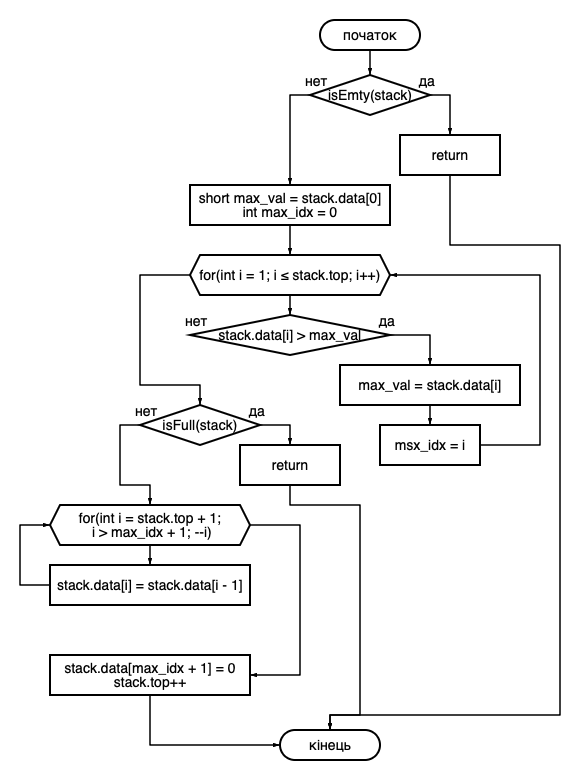


Рис 7. Блок-схема функції insert\_after\_max()

if(isEmty(stack)) *return*;

short max\_val = stack.data[0];

int max\_idx = 0;

for(int i = 1; i <= stack.top; i++) {

if (stack.data[i] > max\_val) {

max\_val = stack.data[i];

max\_idx = i;

}

}

if (isFull(stack)) *return*;

for (int i = stack.top + 1; i > max\_idx + 1; --i) {

stack.data[i] = stack.data[i - 1];

}

stack.data[max\_idx + 1] = 0;

stack.top++;

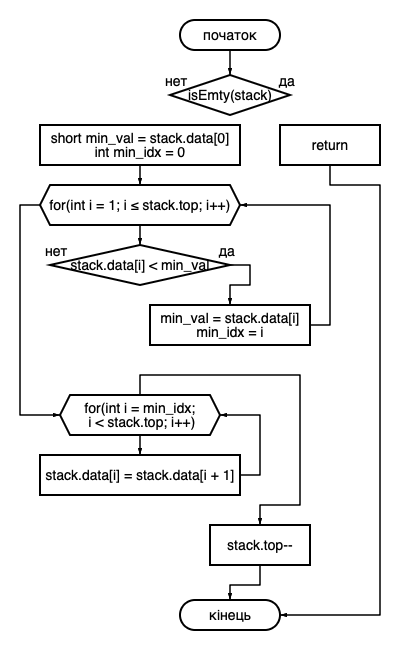




Рис. 8 Блок-схема до функції delete\_min()

if (isEmty(stack)) *return*;

short min\_val = stack.data[0];

int min\_idx = 0;

for (int i = 1; i <= stack.top; i++) {

if (stack.data[i] < min\_val) {

min\_val = stack.data[i];

min\_idx = i;

}

}

for (int i = min\_idx; i < stack.top; i++) {

stack.data[i] = stack.data[i + 1];

}

stack.top--;

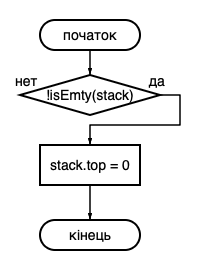


Рис. 9 Блок-схема до функції delete\_all\_except\_first()

if (!isEmty(stack)) {

stack.top = 0;

}

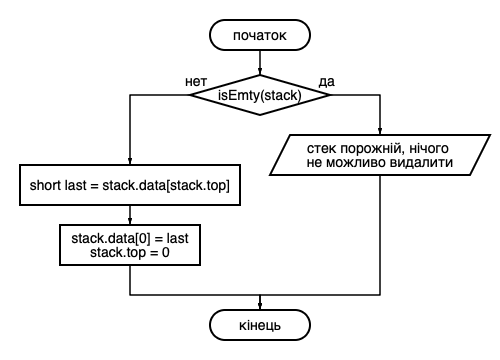


Рис 10. Блок-схема до функції delete\_all\_except\_last()

if (isEmty(stack)) {

cout << RED << "Стек порожній, нічого не можна видалити!" << RESET << endl;

}

short last = stack.data[stack.top];

stack.data[0] = last;

stack.top = 0;

**Черга** — це структура даних, яка працює за принципом “перший прийшов — перший пішов” (FIFO). Це означає, що елемент, який був доданий першим, буде видалений першим.

Черга підтримує основні операції: додавання елемента в кінець черги (enqueue) і видалення та повернення елемента з початку черги (dequeue). Також можлива перевірка, чи є черга порожньою (isEmpty) або повною (isFull).

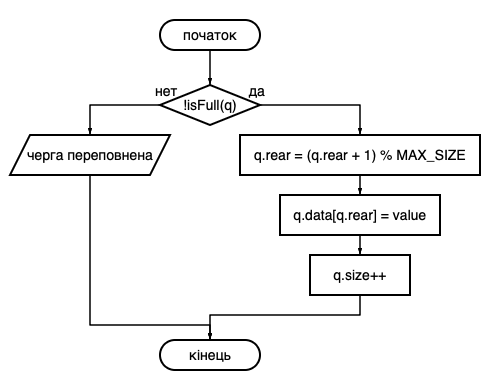


Рис. 11 Блок-схема до функції enqueue()

if (!isFull(q)) {

q.rear = (q.rear + 1) % MAX\_SIZE;

q.data[q.rear] = value;

q.size++;

} else {

cout << RED << "Черга переповнена!" << RESET << endl;

}

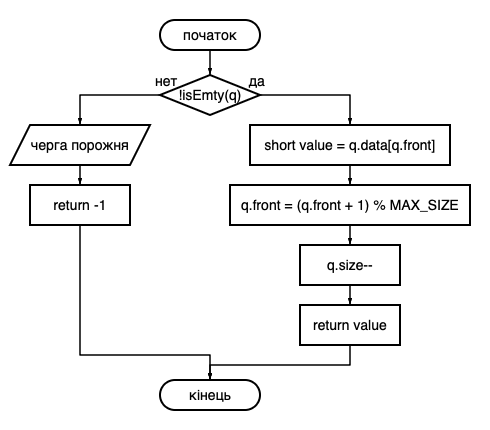


Рис. 12 Блок-схема до функції dequeu()

if (!isEmty(q)) {

short value = q.data[q.front];

q.front = (q.front + 1) % MAX\_SIZE;

q.size--;

*return* value;

} else {

cout << RED << "Черга порожня!" << RESET << endl;

*return* -1;

}

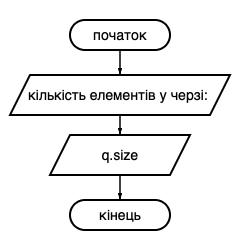


Рис. 12 Блок-схема до функції printSize()

cout << "Кількість елементів у черзі: " << q.size << endl;

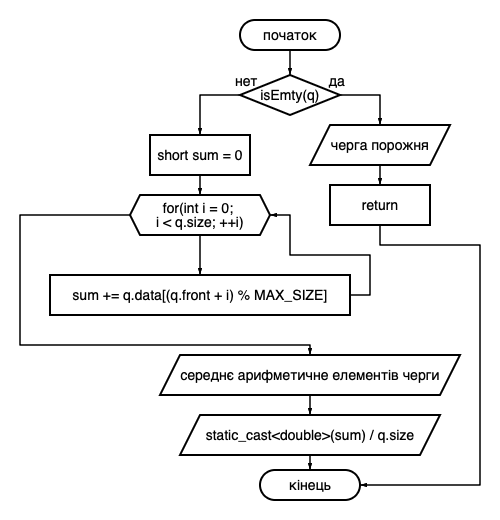


Рис. 13 Блок-схема до функції printAvarage()

if (isEmty(q)) {

cout << RED << "Черга порожня" << RESET << endl;

*return*;

}

short sum = 0;

for (int i = 0; i < q.size; ++i) {

sum += q.data[(q.front + i) % MAX\_SIZE];

}

cout << GREEN << "Середнє арифметичне елементів черги: " << static\_cast<double>(sum) / q.size << RESET <<endl;

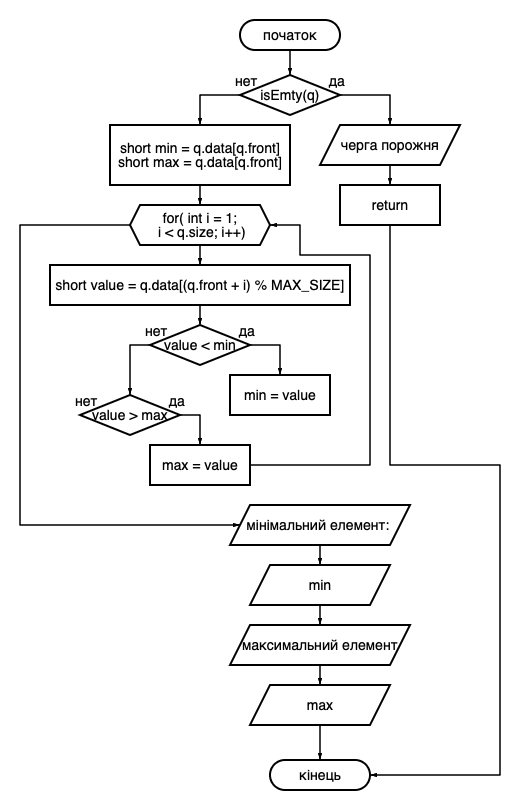


Рис. 14 Блок-схема до функції printMinMax()

if (isEmty(q)) {

cout << RED << "Черга порожня" << RESET << endl;

*return*;

}

short min = q.data[q.front];

short max = q.data[q.front];

for (int i = 1; i < q.size; i++) {

short value = q.data[(q.front + i) % MAX\_SIZE];

if (value < min) min = value;

if (value > max) max = value;

}

cout << "Мінімальний елемент: " << min << endl;

cout << "Максимальний елемент: " << max << endl;

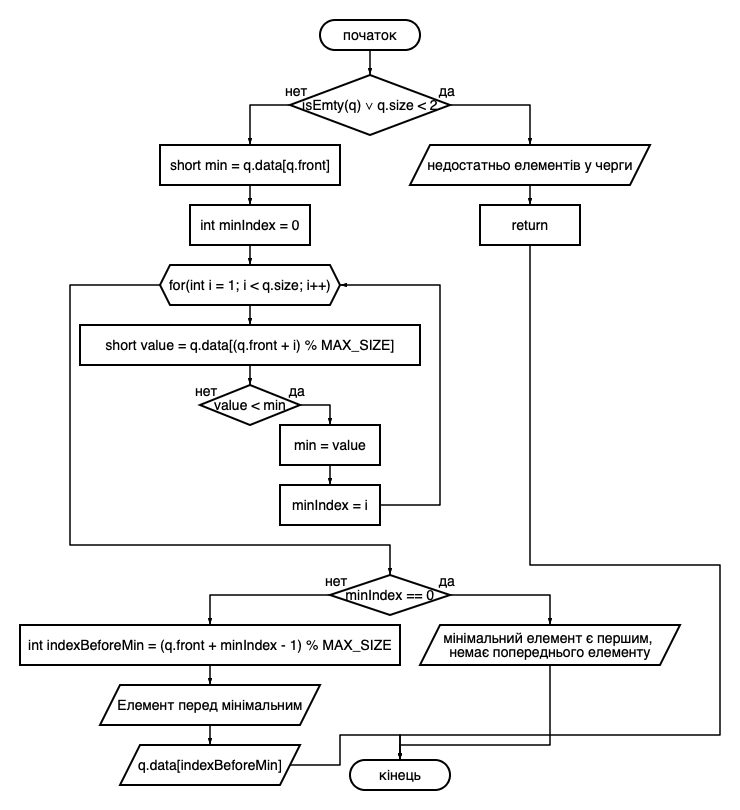


Рис. 15 Блок-схема до функції printElementBeforeMin()

if (isEmty(q) || q.size < 2) {

cout << RED << "Недостатньо елементів у черзі!" << RESET << endl;

*return*;

}

short min = q.data[q.front];

int minIndex = 0;

for (int i = 1; i < q.size; ++i) {

short value = q.data[(q.front + i) % MAX\_SIZE];

if (value < min) {

min = value;

minIndex = i;

}

}

if (minIndex == 0) {

cout << RED << "Мінімальний елемент є першим, немає попереднього елементу!" << RESET << endl;

} else {

int indexBeforeMin = (q.front + minIndex - 1) % MAX\_SIZE;

cout << "Елемент перед мінімальним: " << q.data[indexBeforeMin] << endl;

}

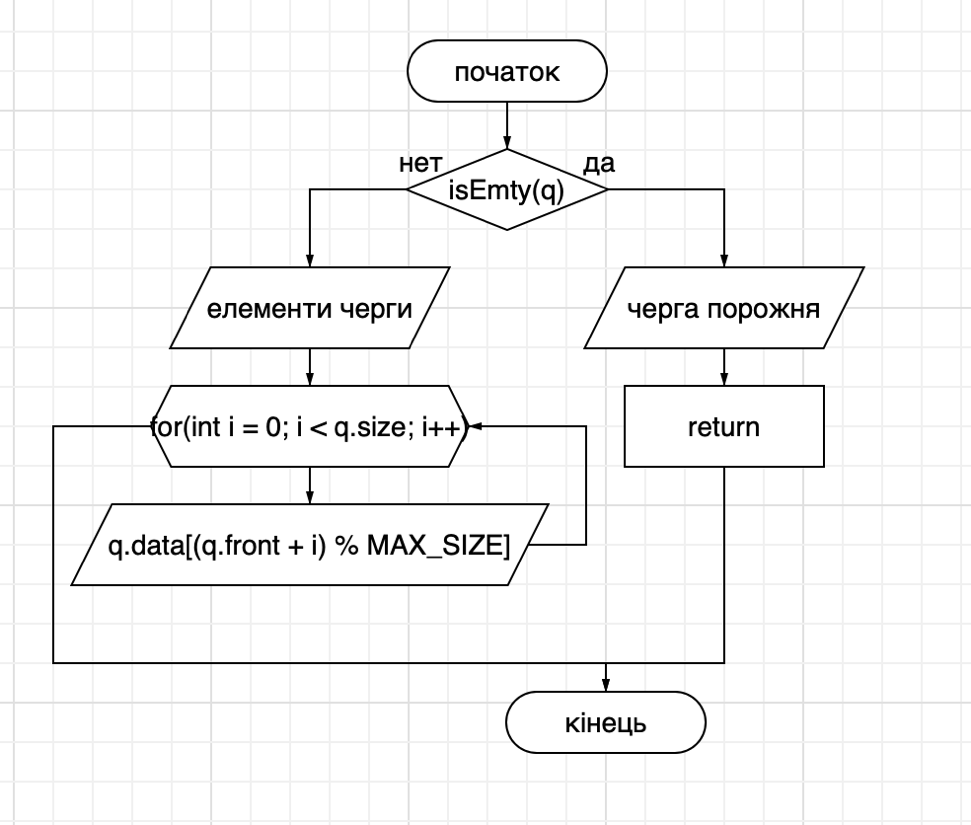


Рис. 16 Блок-схема до функції printQueue()

if (isEmty(q)) {

cout << "Черга порожня!" << endl;

*return*;

}

cout << "Елементи черги: ";

for (int i = 0; i < q.size; ++i) {

cout << q.data[(q.front + i) % MAX\_SIZE] << " ";

}

cout << endl;

**Дек** (двостороння черга) – це структура даних, яка дозволяє додавати і видаляти елементи як з початку, так і з кінця черги. Завдяки цьому можна працювати з обох кінців, що робить дек більш гнучким, ніж звичайна черга або стек.

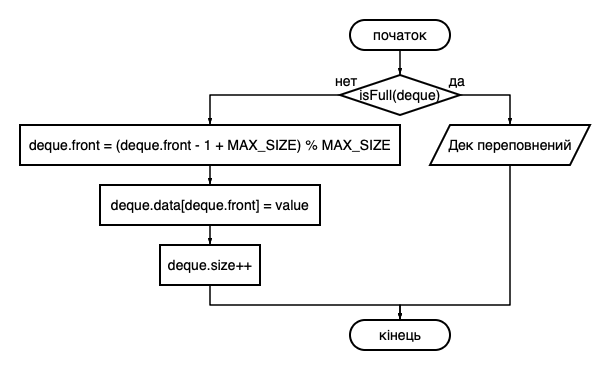


Рис. 17 Блок-схема до функції addFront()

if (isFull(deque)) {

cout << RED << "Дек переповнений" << RESET << endl;

}

deque.front = (deque.front - 1 + MAX\_SIZE) % MAX\_SIZE;

deque.data[deque.front] = value;

deque.size++;

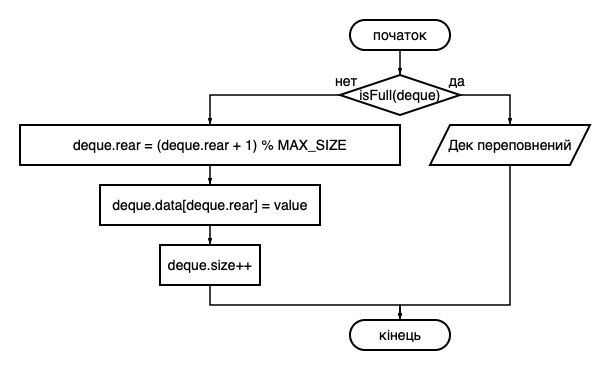


Рис. 18 Блок-схема до функції addRear()

if (isFull(deque)) {

cout << RED << "Дек переповнений" << RESET << endl;

}

deque.rear = (deque.rear + 1) % MAX\_SIZE;

deque.data[deque.rear] = value;

deque.size++

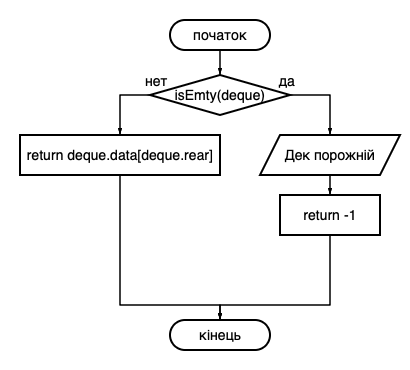


Рис. 19 Блок-схема до функції getRear()

if (isEmty(deque)) {

cout << RED << "Дек порожній" << RESET << endl;

*return* -1;

}

*return* deque.data[deque.rear];

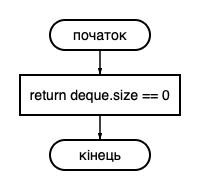


Рис. 20 Блок-схема до функції isEmpty()

*return* deque.size == 0;

**Висновки**: Під час виконання лабораторної роботи я ознайомився з основними операціями над стеком, чергою та деком і розробив відповідні функції для кожної структури даних. Стек продемонстрував принцип роботи за схемою LIFO (Last In, First Out), коли останній доданий елемент стає першим, що вилучається. Для цієї структури я реалізував функції push, яка додає елемент на вершину стека, pop для вилучення елемента з вершини, та peek, яка дозволяє переглядати елемент на вершині без його вилучення. Крім того, я розробив додаткові функції, такі як reverse для реверсу стека, swap\_first\_last для обміну першого і останнього елементів, а також функцію для видалення кожного другого елемента.

Черга працює за принципом FIFO (First In, First Out), де перший доданий елемент стає першим, що вилучається. Я розробив функції enqueue для додавання елементів у кінець черги і dequeue для вилучення елемента з початку. Також реалізував функції для обчислення середнього значення елементів черги (printAvarage), пошуку мінімуму та максимуму (printMinMax), та виведення елемента перед мінімальним (printElementBeforeMin).

Дек (двостороння черга) виявився більш універсальною структурою даних, оскільки дозволяє виконувати операції з обох кінців: додавати і видаляти елементи як з початку, так і з кінця черги. Я реалізував функції для додавання елементів на початок та кінець дека, а також для вилучення елементів з обох сторін. Для дека було розроблено функції pushFront та pushBack для додавання елементів з двох сторін, а також функції для перевірки на порожнечу і відображення вмісту дека.

**Порівняння стеку, черги та дека:**

**Стек** працює за принципом LIFO, що робить його придатним для задач, де важливо зберегти останній доданий елемент до останнього моменту. Операції над стеком прості, оскільки додавання і вилучення відбуваються лише з одного кінця (вершини). Основна перевага стека — ефективність при роботі з останнім елементом, але його недоліком є те, що до інших елементів доступ обмежений.

**Черга** функціонує за схемою FIFO, що робить її зручною для організації процесів, де важливо, щоб найперший доданий елемент оброблявся першим. Це ідеально підходить для задач на обробку даних у порядку надходження, як у реальних чергах. Черга має обмеженість у тому, що всі операції відбуваються з різних кінців: додавання в кінець, а вилучення з початку, і неможливо напряму обробити середні елементи.

**Дек** поєднує в собі властивості як стеку, так і черги, дозволяючи виконувати операції з обох кінців. Це робить дек найбільш гнучкою структурою даних серед розглянутих, оскільки він дозволяє не лише додавати та вилучати елементи з обох сторін, але й змінювати порядок обробки елементів. Дек може працювати як стек або як черга залежно від потреб, що робить його універсальним інструментом для задач, де потрібен динамічний доступ до обох кінців структури.

Таким чином, лабораторна робота дала мені змогу практично застосувати та поглибити свої знання з роботи з такими структурами даних, як стек, черга та дек. Завдяки розробці функцій для кожної з цих структур я зрозумів їхні основні принципи і переваги у різних сценаріях використання.