Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Стратегічного управління в сфері інформаційних технологій"

|  |  |
| --- | --- |
|  |  |

**Звітів до лабораторної роботи №2**

з дисципліни "Алгоритми на структури даних"

Виконав: студент групи КН-321А

Бородай Д. А.

Перевірив: старший викладач

Мошко Є.О.

Харків 2022

**Тема лабораторної роботи.** Стеки, черги і деки та робота з ними.

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

**Порядок виконання роботи**:

1. Написати програму, яка демонструє основні дії по роботі із стеком пропонує користувачу вибір дії:  
−поміняти місцями перший і останній елементи стека;  
−розгорнути стек, тобто зробити "дно" стека вершиною, а вершину -"дном";  
−видалити кожен другий елемент стека;  
−знайти максимальний елемент і вставити після нього 0;  
−видалити мінімальний елемент;−видалити всі елементи, крім першого;  
−видалити всі елементи, крім останнього.

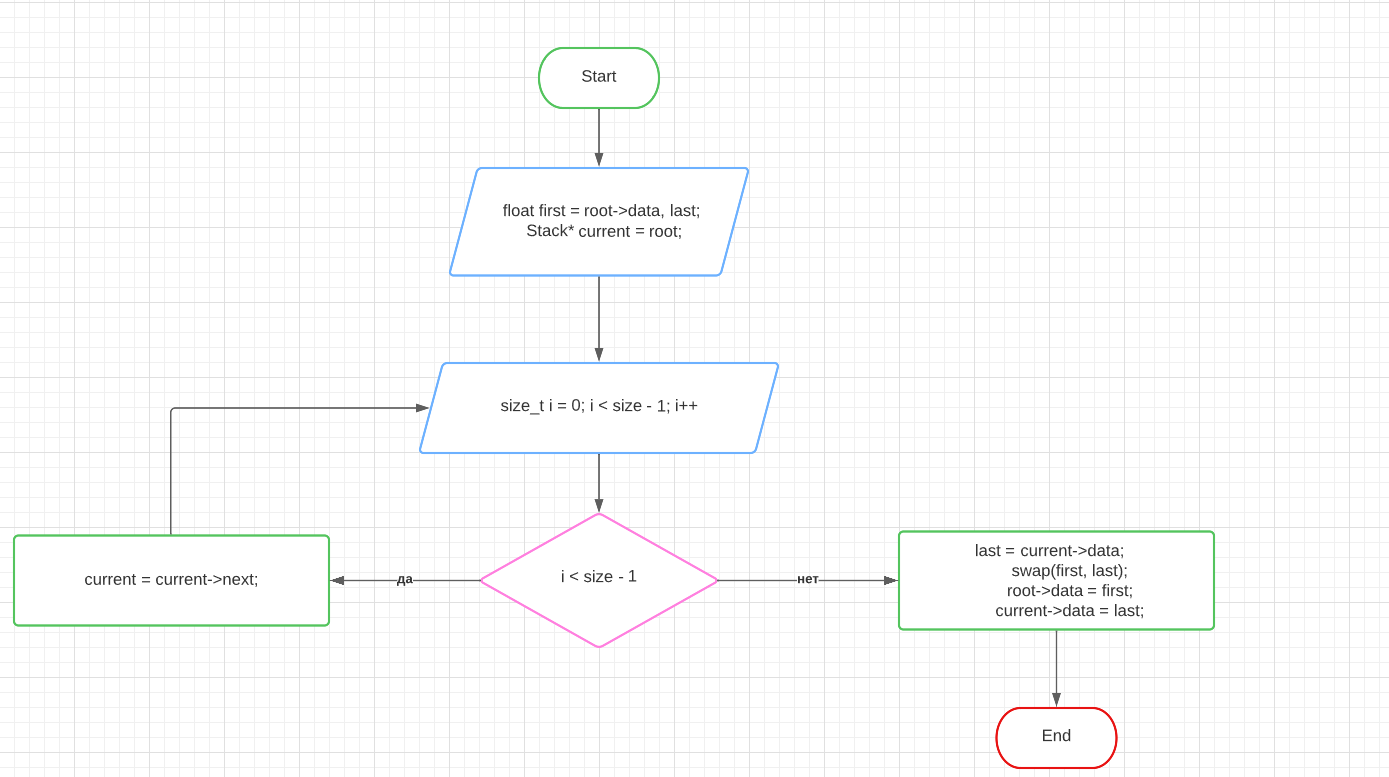
2. Написати програму, яка демонструє основні дії по роботі із чергою пропонує користувачу вибір дії:  
−знайти та вивести кількість елементів черги;  
−знайти та вивести середнє арифметичне збережених елементів;  
 −знайти та вивести мінімальний та максимальний елемент;  
 −знайти та вивести елемент, що йде перед мінімальним елементом.

3. Написати програму, яка демонструє основні дії по роботі із деком пропонує користувачу вибір дії:  
−створити дек;  
−перевірити на порожнечу;  
−додати елемент в початок;  
−додати елемент в кінець;  
−прочитати останній елемент.

# Завдання 1. Робота зі стеком

Згідно другого варіанту, поданого у завданнях, типом даних є float.

**Алгоритм зміни першого та останнього елементу:**

****

Блок-схема 1.1 - Алгоритм перестановки першого та останнього елементу стека

void Stack::replaceFirstandLast()

{

float first = root->data, last;

Stack\* current = root;

for (size\_t i = 0; i < size - 1; i++)

current = current->next;

last = current->data;

swap(first, last);

root->data = first;

current->data = last;

}



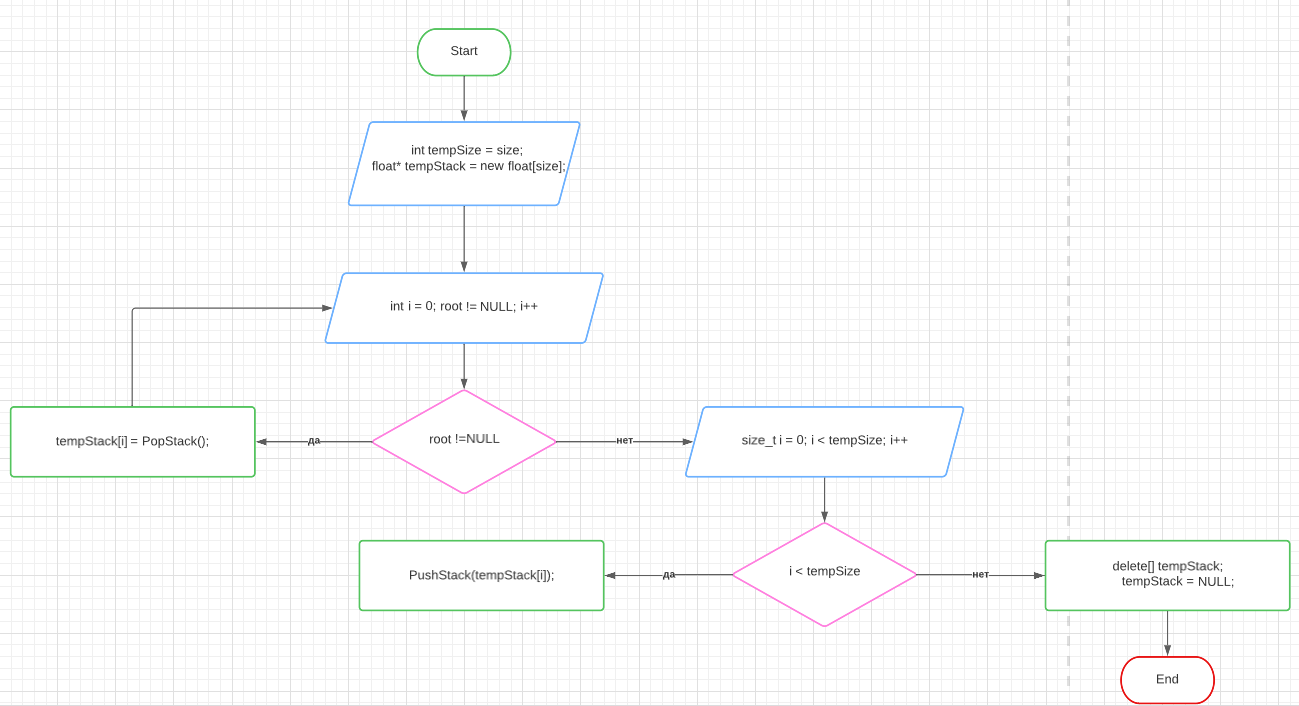
Рисунок 1.1 - Стек до виконання алгоритму



Рисунок 1.2 - Стек після виконання алгоритму

Повний текст програми наведено у додатку А.

**Алгоритм розгортки стеку:**

****

Блок-схема 1.2 – Алгоритм розгортки

void Stack::reverse()

{

int tempSize = size;

float\* tempStack = new float[size];

for ( int i = 0; root != NULL; i++)

tempStack[i] = PopStack();

for (size\_t i = 0; i < tempSize; i++)

PushStack(tempStack[i]);

delete[] tempStack;

tempStack = NULL;

}



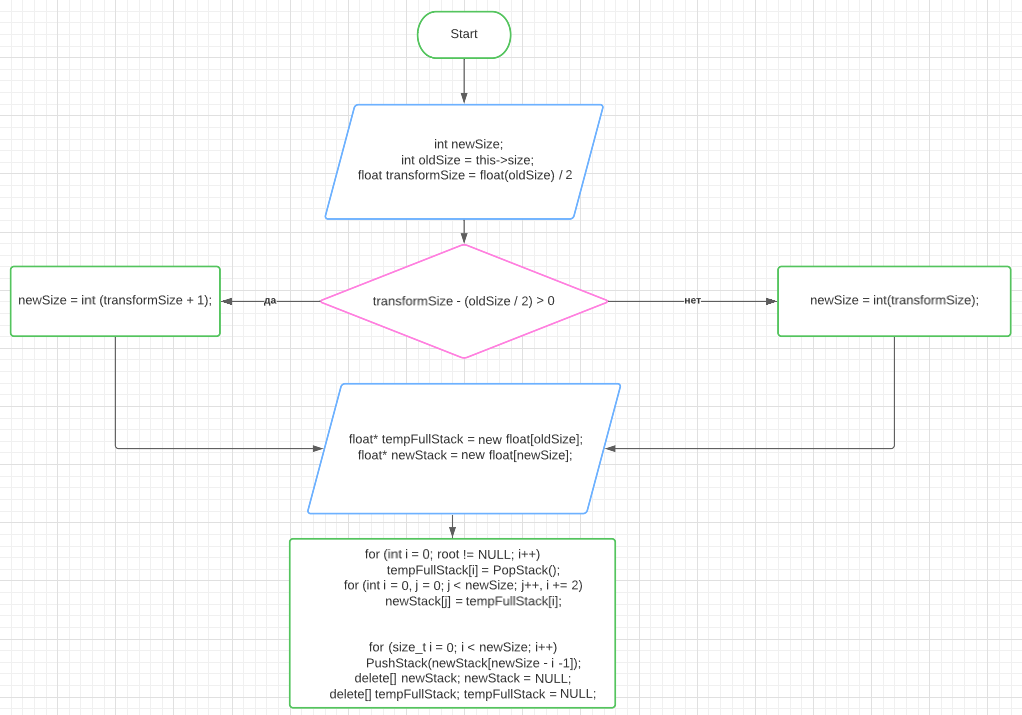
Рисунок 1.3 - Стек до виконання алгоритму



Рисунок 1.4 - Стек після виконання алгоритму

Повний текст програми наведено у додатку А.

**Алгоритм видалення кожного другого:**

****

Блок-схема 1.3 – Алгоритм видалення кожного другого

void Stack::deleteEverySecond()

{

int newSize;

int oldSize = this->size;

float transformSize = float(oldSize) / 2;

if (transformSize - (oldSize / 2) > 0)

newSize = int (transformSize + 1);

else

newSize = int(transformSize);

float\* tempFullStack = new float[oldSize];

float\* newStack = new float[newSize];

for (int i = 0; root != NULL; i++)

tempFullStack[i] = PopStack();

for (int i = 0, j = 0; j < newSize; j++, i += 2)

newStack[j] = tempFullStack[i];

for (size\_t i = 0; i < newSize; i++)

PushStack(newStack[newSize - i -1]);

delete[] newStack; newStack = NULL;

delete[] tempFullStack; tempFullStack = NULL;

}



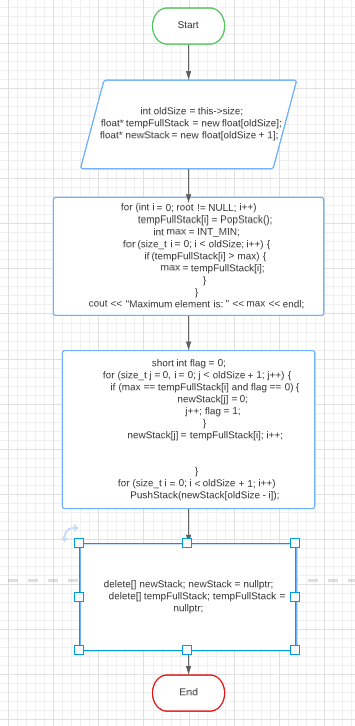
Рисунок 1.5 - Стек до виконання алгоритму



Рисунок 1.6 - Стек після виконання алгоритму

Повний текст програми наведено у додатку А.

**Алгоритм знаходження максимального:**

****

Блок-схема 1.4 – Алгоритм знаходження максимального і мінімального

****

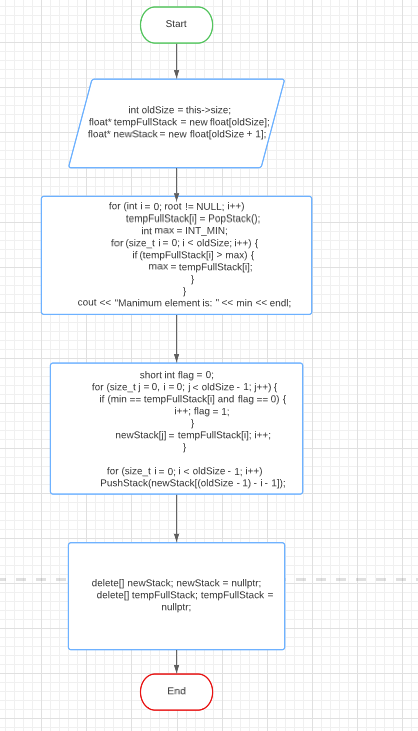
Рисунок 1.7 - Стек до виконання алгоритму



Рисунок 1.8 - Стек після виконання алгоритму

Повний текст програми наведено у додатку А.

**Алгоритм видалення мінімального:**

****

Блок-схема 1.3 – Алгоритм видалення мінімального

void Stack::deleteMin()

{

int oldSize = this->size;

float\* tempFullStack = new float[oldSize];

float\* newStack = new float[oldSize - 1];

for (int i = 0; root != NULL; i++)

tempFullStack[i] = PopStack();

int min = INT\_MAX;

for (size\_t i = 0; i < oldSize; i++) {

if (tempFullStack[i] < min) {

min = tempFullStack[i];

}

}

cout << "Manimum element is: " << min << endl;

short int flag = 0;

for (size\_t j = 0, i = 0; j < oldSize - 1; j++) {

if (min == tempFullStack[i] and flag == 0) {

i++; flag = 1;

}

newStack[j] = tempFullStack[i]; i++;

}

for (size\_t i = 0; i < oldSize - 1; i++)

PushStack(newStack[(oldSize - 1) - i - 1]);

delete[] newStack; newStack = nullptr;

delete[] tempFullStack; tempFullStack = nullptr;

}



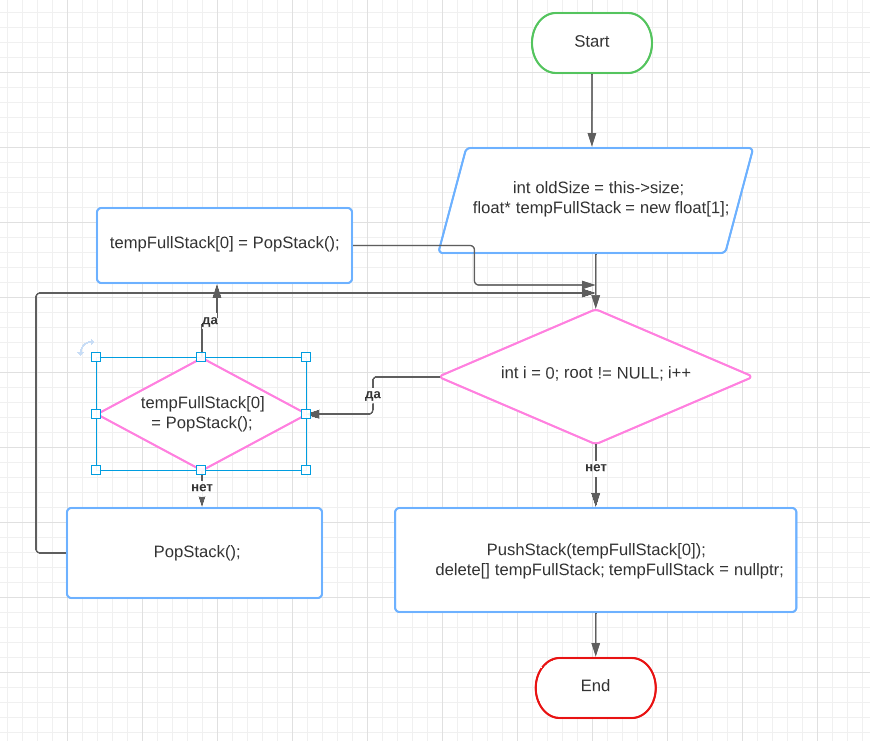
Рисунок 1.9 - Стек до виконання алгоритму



Рисунок 1.10 - Стек після виконання алгоритму

Повний текст програми наведено у додатку А.

**Алгоритм видалення видалення усіх елементів окрім першого:**

****

Блок-схема 1.4 – Алгоритм видалення усіх окрім першого

void Stack::deletAllExpFrist()

{

int oldSize = this->size;

float\* tempFullStack = new float[1];

for (int i = 0; root != NULL; i++) {

if (i == (oldSize - 1)) {

tempFullStack[0] = PopStack();

}

else PopStack();

}

PushStack(tempFullStack[0]);

delete[] tempFullStack; tempFullStack = nullptr;

}



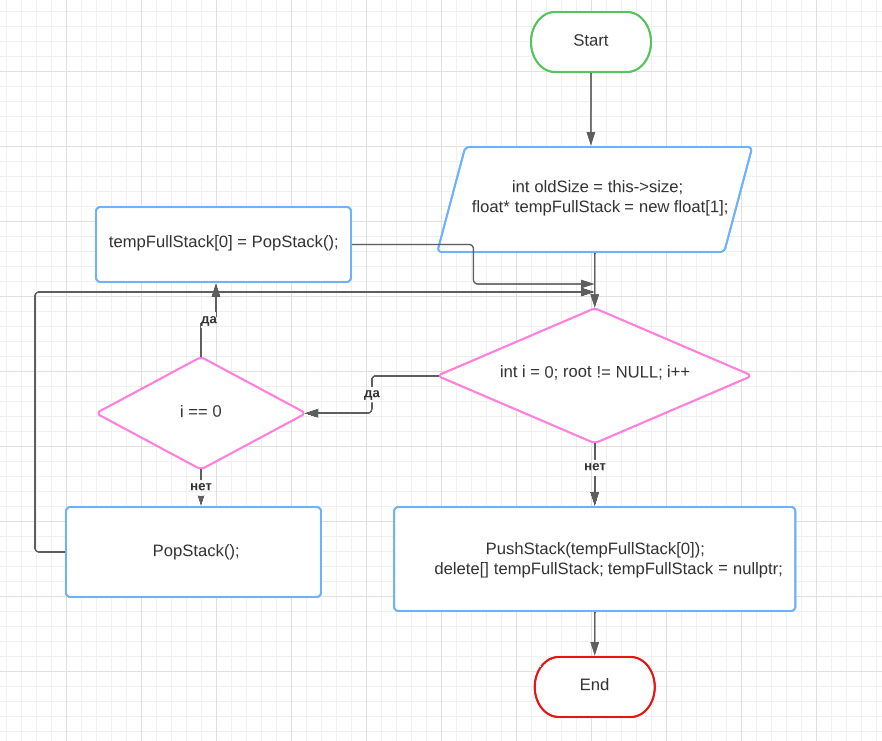
Рисунок 1.11 - Стек до виконання алгоритму



Рисунок 1.12 - Стек після виконання алгоритму

Повний текст програми наведено у додатку А.

**Алгоритм видалення видалення усіх елементів окрім першого:**

****

Блок-схема 1.5 – Алгоритм видалення усіх окрім першого

void Stack::deletAllExpLast()

{

int oldSize = this->size;

float\* tempFullStack = new float[1];

for (int i = 0; root != NULL; i++) {

if (i == 0) {

tempFullStack[0] = PopStack();

}

else PopStack();

}

PushStack(tempFullStack[0]);

delete[] tempFullStack; tempFullStack = nullptr;

}



Рисунок 1.13 - Стек до виконання алгоритму



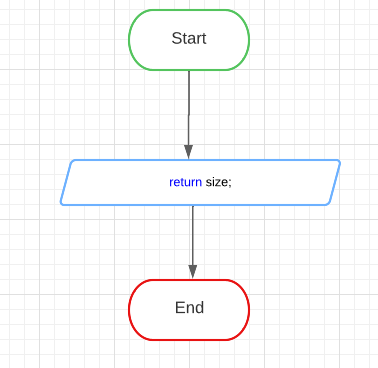
Рисунок 1.14 - Стек після виконання алгоритму

Повний текст програми наведено у додатку А.

# Завдання 2. Робота зі чергою

Згідно другого варіанту, поданого у завданнях, типом даних є float.

**Алгоритм підрахунку кількості елементів черги:**

****

Блок-схема 2.1 – Алгоритм підрахунку кількості елементів черги

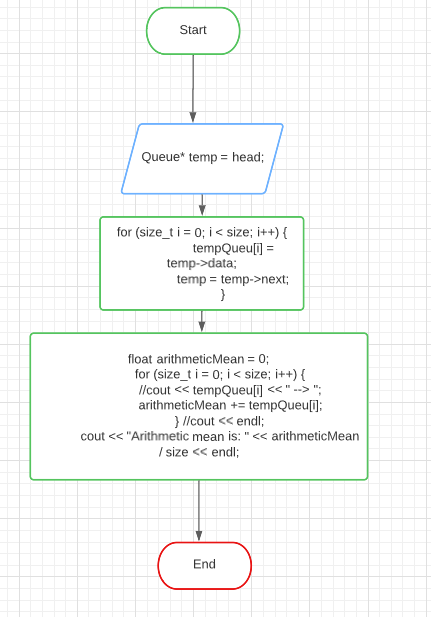
int amount() { return size; }

****

Рисунок 2.1 – Результат алгоритму

Повний текст програми наведено у додатку Б.

**Алгоритм підрахунку середнього арефметичного елементів черги:**

****

Блок-схема 2.1 – Алгоритм підрахунку кількості елементів черги

void Queue::ArithmeticMean()

{

float\* tempQueu = new float[size];

if (size != 0) {

Queue\* temp = head;

for (size\_t i = 0; i < size; i++) {

tempQueu[i] = temp->data;

temp = temp->next;

}

float arithmeticMean = 0;

for (size\_t i = 0; i < size; i++) {

//cout << tempQueu[i] << " --> ";

arithmeticMean += tempQueu[i];

} //cout << endl;

cout << "Arithmetic mean is: " << arithmeticMean / size << endl;

}

else

cout << "\tQueue dose not exist!\n";

delete[] tempQueu; tempQueu = NULL;

}



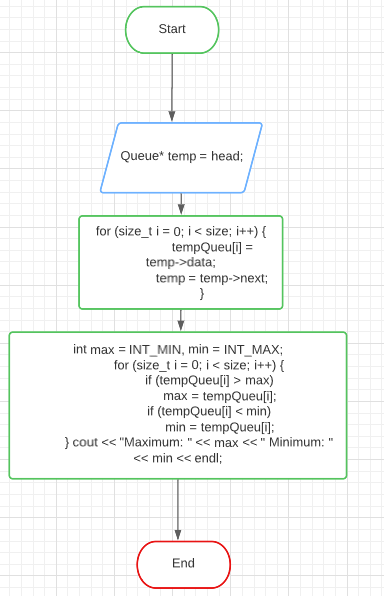
Рисунок 2.2- Черга до виконання алгоритму



Рисунок 2.3- Результат виконання алгоритму

Повний текст програми наведено у додатку Б.

**Алгоритм знаходження максимального і мінімального:**



Блок-схема 2.2 – Алгоритм знаходження максимального і мінімального

void Queue::MinandMax()

{

float\* tempQueu = new float[size];

if (size != 0) {

Queue\* temp = head;

for (size\_t i = 0; i < size; i++) {

tempQueu[i] = temp->data;

temp = temp->next;

}

int max = INT\_MIN, min = INT\_MAX;

for (size\_t i = 0; i < size; i++) {

if (tempQueu[i] > max)

max = tempQueu[i];

if (tempQueu[i] < min)

min = tempQueu[i];

} cout << "Maximum: " << max << " Minimum: " << min << endl;

}

else

cout << "\tQueue dose not exist!\n";

delete[] tempQueu; tempQueu = NULL;

}



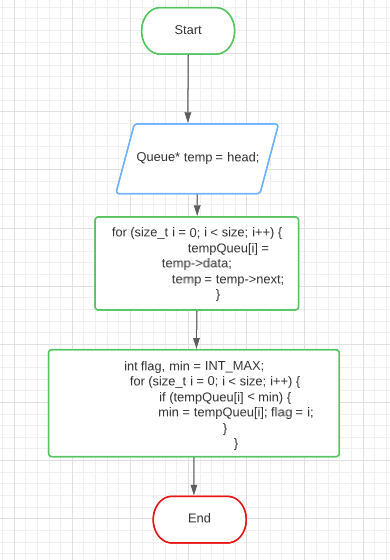
Рисунок 2.4- Черга до виконання алгоритму



Рисунок 2.5- Результат виконання алгоритму

Повний текст програми наведено у додатку Б.

**Алгоритм знаходження елементу перед мінімальним:**



Блок-схема 2.3 – Алгоритм знаходження елементу перед минімальним

void Queue::beforeMIN()

{

float\* tempQueu = new float[size];

if (size != 0) {

Queue\* temp = head;

for (size\_t i = 0; i < size; i++) {

tempQueu[i] = temp->data;

temp = temp->next;

}

int flag, min = INT\_MAX;

for (size\_t i = 0; i < size; i++) {

if (tempQueu[i] < min) {

min = tempQueu[i]; flag = i;

}

}

if ( flag == 0)

cout << "This element dose not exit yet.\n";

else

cout << "Next to min element: " << tempQueu[flag - 1] << endl;

}

else

cout << "\tQueue dose not exist!\n";

delete[] tempQueu; tempQueu = NULL;

}



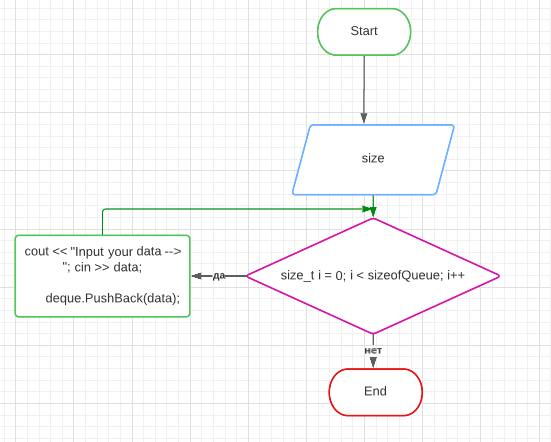
Рисунок 2.5- Результат виконання алгоритму

Повний текст програми наведено у додатку Б.

# Завдання 3. Робота зі деком

Згідно другого варіанту, поданого у завданнях, типом даних є float.

**Алгоритм створення деку:**

****

Блок-схема 3.2 – Алгоритм створення деку

if (deque.isNULL()) {

cout << "Size of deque: "; cin >> sizeofQueue;

for (size\_t i = 0; i < sizeofQueue; i++) {

cout << "Input your data --> "; cin >> data;

deque.PushBack(data);

}

}

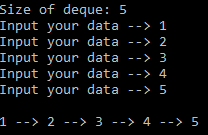
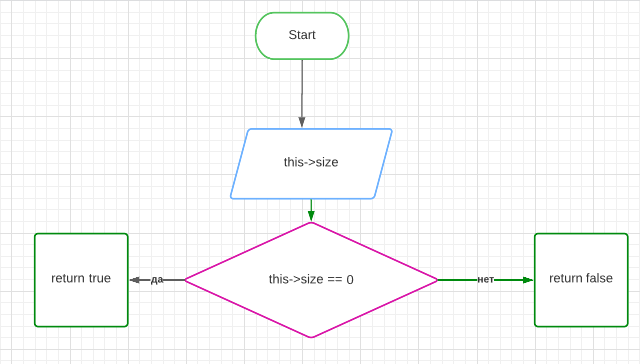


Рисунок 3.1 – Результат роботи алгоритму

Повний текст програми наведено у додатку В.

**Алгоритм перевірки на порожнечу:**



Блок-схема 3.2 – Алгоритм перевірки на порожнечу

bool isNULL() {

if (size == 0)

return true;

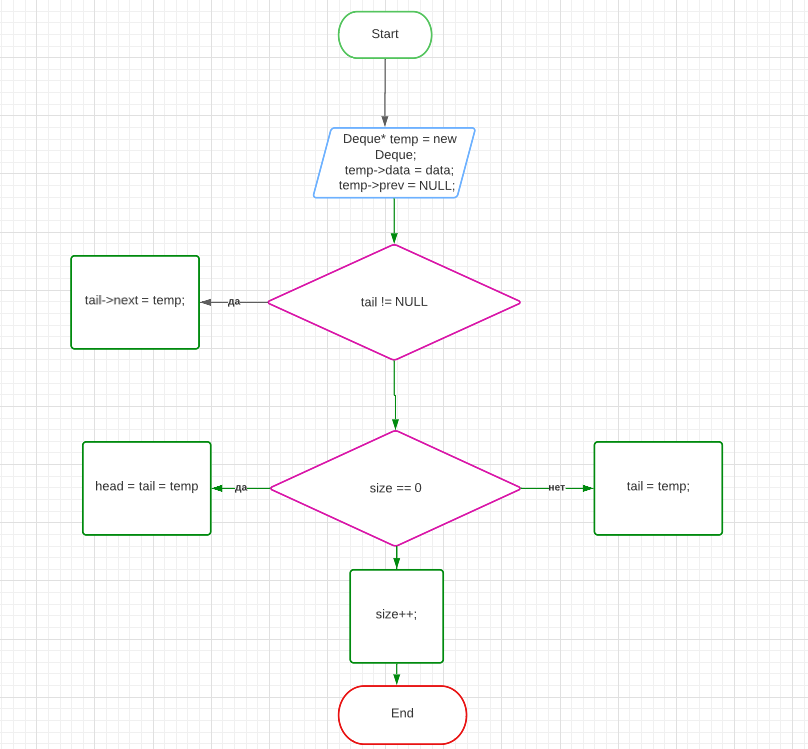
else

return false;

}

Повний текст програми наведено у додатку В.

**Алгоритм додавання елементу у кінець:**



Блок-схема 3.2 – Алгоритм додавання у кінець

void Deque::PushBack(const float& data)

{

Deque\* temp = new Deque;

temp->next = NULL;

temp->data = data;

temp->prev = tail;

if (tail != NULL)

tail->next = temp;

if (size == 0)

head = tail = temp;

else

tail = temp;

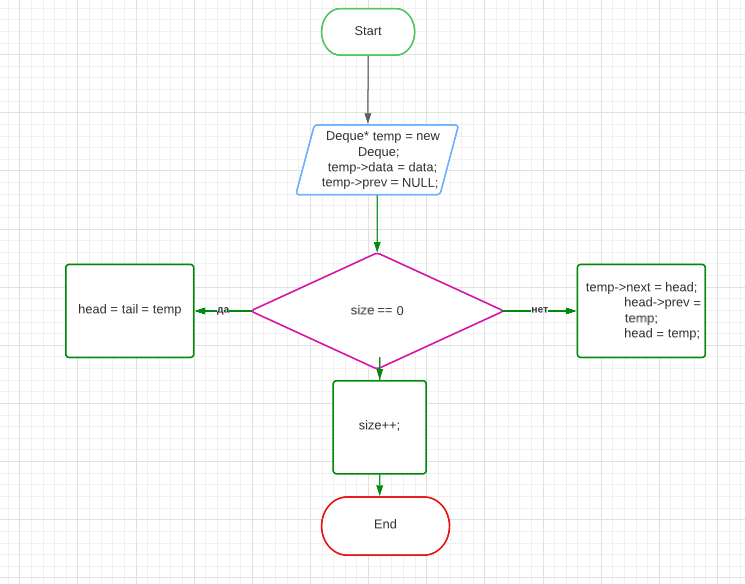
size++; }



Рисунок 3.2 – Результат роботи алгоритму

Повний текст програми наведено у додатку В.

**Алгоритм додавання у початок:**



Блок-схема 3.2 – Алгоритм додавання у кінець

void Deque::PushFront(const float& data)

{

Deque\* temp = new Deque;

temp->data = data;

temp->prev = NULL;

if (size == 0)

head = tail = temp;

else {

temp->next = head;

head->prev = temp;

head = temp;

}

size++;

}



Рисунок 3.3 – Результат роботи алгоритму

Повний текст програми наведено у додатку В.

# Висновок

Під час виконання лабораторної роботи було засвоєно навички по роботі з стеками, чергами та деками.

Стек – це структура даних, у якій новий елемент елемент завжди записується на її початок (вершину) і ще один елемент вибирається з її початку. У стеках використовується метод доступу LIFO – останнім прийшов – першим вийшов.

Основними функціями по роботі зі стеком є створення стеку, друк стеку, додавання до вершини, вилучення з вершини елементу, перевірка на порожнечу, очищення стеку, тощо.

Окрім рекурсії, організувати стек можна за допомогою лінійного однозв’язного списку.

Черга – це структура даних, що є послідовністю елементів, створена порядку їх надходження. Кожний новий елемент розміщується наприкінці черги. У черзі використовується метод доступу FIFO – першим прийшов – першим вийшов.

Основною відмінністю черги від стеку є метод доступу FIFO невідмінну від LIFO.

Основними структурними елементами стеку є вершина та покажчик на наступний елемент, коли у черги початок, кінець і покажчик на наступний.

Дек – це особливий вид черги. Вилучення елементу може також відбуватися з кінця.

# Додаток А

#pragma once

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Stack

{

private:

float data;

Stack\* next;

Stack\* root;

int size;

public:

void PushStack(float &newData);

float PopStack();

void CreateStack(int amount);

bool isNULL();

void show();

void deleteStack();

//task mathods

void replaceFirstandLast();

void reverse();

void deleteEverySecond();

void findMax0();

void deleteMin();

void deletAllExpFrist();

void deletAllExpLast();

//filework

void read();

void write();

Stack() { root = NULL; size = 0; }

};

#include "Header.h"

string pathData = "data.txt";

void Stack::PushStack(float& newData)

{

Stack\* newStack = new Stack;

newStack->data = newData;

newStack->next = this->root;

this->root = newStack;

size++;

}

float Stack::PopStack()

{

Stack\* deletedITM = this->root;

float tempData = deletedITM->data;

this->root = deletedITM->next;

delete deletedITM; deletedITM = NULL;

size--;

return tempData;

}

void Stack::CreateStack(int amount)

{

if (amount == 0) {

cout << "\tStack size can`t be 0!\n\t Please try again.\n";

return;

}

float tempData;

for (size\_t i = 0; i < amount; i++){

cout << "Input you`r data --> "; cin >> tempData;

PushStack(tempData);

}

}

bool Stack::isNULL()

{

if (size == 0)

return true;

else

return false;

}

void Stack::show()

{

if (size == 0)

cout << "\tStack haven`t found!\n";

else {

cout << "Stack: ";

Stack\* current = this->root;

while (current != NULL) {

cout << current->data << " -> ";

current = current->next;

} cout << endl;

}

}

void Stack::deleteStack()

{

while (root) PopStack();

size = 0;

cout << "\tStack have been deleted!\n";

}

void Stack::replaceFirstandLast()

{

float first = root->data, last;

Stack\* current = root;

for (size\_t i = 0; i < size - 1; i++)

current = current->next;

last = current->data;

swap(first, last);

root->data = first;

current->data = last;

}

void Stack::reverse()

{

int tempSize = size;

float\* tempStack = new float[size];

for ( int i = 0; root != NULL; i++)

tempStack[i] = PopStack();

for (size\_t i = 0; i < tempSize; i++)

PushStack(tempStack[i]);

delete[] tempStack;

tempStack = NULL;

}

void Stack::deleteEverySecond()

{

int newSize;

int oldSize = this->size;

float transformSize = float(oldSize) / 2;

if (transformSize - (oldSize / 2) > 0)

newSize = int (transformSize + 1);

else

newSize = int(transformSize);

float\* tempFullStack = new float[oldSize];

float\* newStack = new float[newSize];

for (int i = 0; root != NULL; i++)

tempFullStack[i] = PopStack();

for (int i = 0, j = 0; j < newSize; j++, i += 2)

newStack[j] = tempFullStack[i];

for (size\_t i = 0; i < newSize; i++)

PushStack(newStack[newSize - i -1]);

delete[] newStack; newStack = NULL;

delete[] tempFullStack; tempFullStack = NULL;

}

void Stack::findMax0()

{

int oldSize = this->size;

float\* tempFullStack = new float[oldSize];

float\* newStack = new float[oldSize + 1];

for (int i = 0; root != NULL; i++)

tempFullStack[i] = PopStack();

int max = INT\_MIN;

for (size\_t i = 0; i < oldSize; i++) {

if (tempFullStack[i] > max) {

max = tempFullStack[i];

}

}

cout << "Maximum element is: " << max << endl;

short int flag = 0;

for (size\_t j = 0, i = 0; j < oldSize + 1; j++) {

if (max == tempFullStack[i] and flag == 0) {

newStack[j] = 0;

j++; flag = 1;

}

newStack[j] = tempFullStack[i]; i++;

}

for (size\_t i = 0; i < oldSize + 1; i++)

PushStack(newStack[oldSize - i]);

delete[] newStack; newStack = nullptr;

delete[] tempFullStack; tempFullStack = nullptr;

}

void Stack::deleteMin()

{

int oldSize = this->size;

float\* tempFullStack = new float[oldSize];

float\* newStack = new float[oldSize - 1];

for (int i = 0; root != NULL; i++)

tempFullStack[i] = PopStack();

int min = INT\_MAX;

for (size\_t i = 0; i < oldSize; i++) {

if (tempFullStack[i] < min) {

min = tempFullStack[i];

}

}

cout << "Manimum element is: " << min << endl;

short int flag = 0;

for (size\_t j = 0, i = 0; j < oldSize - 1; j++) {

if (min == tempFullStack[i] and flag == 0) {

i++; flag = 1;

}

newStack[j] = tempFullStack[i]; i++;

}

for (size\_t i = 0; i < oldSize - 1; i++)

PushStack(newStack[(oldSize - 1) - i - 1]);

delete[] newStack; newStack = nullptr;

delete[] tempFullStack; tempFullStack = nullptr;

}

void Stack::deletAllExpFrist()

{

int oldSize = this->size;

float\* tempFullStack = new float[1];

for (int i = 0; root != NULL; i++) {

if (i == (oldSize - 1)) {

tempFullStack[0] = PopStack();

}

else PopStack();

}

PushStack(tempFullStack[0]);

delete[] tempFullStack; tempFullStack = nullptr;

}

void Stack::deletAllExpLast()

{

int oldSize = this->size;

float\* tempFullStack = new float[1];

for (int i = 0; root != NULL; i++) {

if (i == 0) {

tempFullStack[0] = PopStack();

}

else PopStack();

}

PushStack(tempFullStack[0]);

delete[] tempFullStack; tempFullStack = nullptr;

}

void Stack::read()

{

ifstream dataFile;

dataFile.open(pathData);

if (dataFile.is\_open()) {

cout << "\tReading from the file was successful!\n";

float temp;

while (!dataFile.eof()) {

dataFile >> temp;

PushStack(temp);

}

reverse();

}

else

cout << "\tReading from the file was not successful!\n";

dataFile.close();

}

void Stack::write()

{

ofstream dataFile;

dataFile.open(pathData);

if (size == 0) {

cout << "\tStack haven`t found!\n";

cout << "\tWriting to the file was not successful!\n";

}

else {

cout << "Stack: ";

Stack\* current = this->root;

while (current != NULL) {

dataFile << " " << current->data;

current = current->next;

}

if (dataFile.is\_open()) {

//cout << "\tWriting to the file was successful!\n";

}

else

cout << "\tWriting to the file was not successful!\n";

}

dataFile.close();

}

# Додаток Б

#pragma once

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Queue {

private:

int size;

float data;

Queue\* head;

Queue\* next;

Queue\* tail;

public:

void Push(const float& data);

void show();

int amount() { return size; }

void ArithmeticMean();

void MinandMax();

void beforeMIN();

void write();

void read();

void deleteq() {

while (head) {

Queue\* deletedITM = this->head;

float tempData = deletedITM->data;

this->head = deletedITM->next;

delete deletedITM; deletedITM = NULL;

size--;

}

cout << "\tQueue have been deleted!\n";

}

Queue() { head = NULL; tail = NULL; size = 0; }

};

#include "Header.h"

string dataPath = "data.txt";

void Queue::Push(const float& data)

{

Queue\* temp = new Queue;

temp->next = 0;

temp->data = data;

if (tail != NULL)

tail->next = temp;

if (size == 0)

head = tail = temp;

else

tail = temp;

size++;

}

void Queue::show()

{

if (size != 0) {

Queue\* temp = head;

while (temp->next != 0)

{

cout << temp->data << " --> ";

temp = temp->next;

}

cout << temp->data << endl;

}

else

cout << "\tQueue dose not exist!\n";

}

void Queue::ArithmeticMean()

{

float\* tempQueu = new float[size];

if (size != 0) {

Queue\* temp = head;

for (size\_t i = 0; i < size; i++) {

tempQueu[i] = temp->data;

temp = temp->next;

}

float arithmeticMean = 0;

for (size\_t i = 0; i < size; i++) {

//cout << tempQueu[i] << " --> ";

arithmeticMean += tempQueu[i];

} //cout << endl;

cout << "Arithmetic mean is: " << arithmeticMean / size << endl;

}

else

cout << "\tQueue dose not exist!\n";

delete[] tempQueu; tempQueu = NULL;

}

void Queue::MinandMax()

{

float\* tempQueu = new float[size];

if (size != 0) {

Queue\* temp = head;

for (size\_t i = 0; i < size; i++) {

tempQueu[i] = temp->data;

temp = temp->next;

}

int max = INT\_MIN, min = INT\_MAX;

for (size\_t i = 0; i < size; i++) {

if (tempQueu[i] > max)

max = tempQueu[i];

if (tempQueu[i] < min)

min = tempQueu[i];

} cout << "Maximum: " << max << " Minimum: " << min << endl;

}

else

cout << "\tQueue dose not exist!\n";

delete[] tempQueu; tempQueu = NULL;

}

void Queue::beforeMIN()

{

float\* tempQueu = new float[size];

if (size != 0) {

Queue\* temp = head;

for (size\_t i = 0; i < size; i++) {

tempQueu[i] = temp->data;

temp = temp->next;

}

int flag, min = INT\_MAX;

for (size\_t i = 0; i < size; i++) {

if (tempQueu[i] < min) {

min = tempQueu[i]; flag = i;

}

}

if ( flag == 0)

cout << "This element dose not exit yet.\n";

else

cout << "Next to min element: " << tempQueu[flag - 1] << endl;

}

else

cout << "\tQueue dose not exist!\n";

delete[] tempQueu; tempQueu = NULL;

}

void Queue::read()

{

ifstream dataFile;

dataFile.open(dataPath);

if (dataFile.is\_open()) {

cout << "\tReading from the file was successful!\n";

float temp;

while (!dataFile.eof()) {

dataFile >> temp;

Push(temp);

}

}

else

cout << "\tReading from the file was not successful!\n";

dataFile.close();

}

void Queue::write()

{

ofstream dataFile;

dataFile.open(dataPath);

if (size == 0) {

cout << "\tQueue haven`t found!\n";

cout << "\tWriting to the file was not successful!\n";

}

else {

Queue\* current = this->head;

while (current != NULL) {

dataFile << " ";

dataFile << current->data;

current = current->next;

}

//if (dataFile.is\_open())

// cout << "\tWriting to the file was successful!\n";

//else

// cout << "\tWriting to the file was not successful!\n";

}

dataFile.close();

# Додаток В

#pragma once

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Deque {

private:

int size;

float data;

Deque\* head;

Deque\* next;

Deque\* tail;

Deque\* prev;

public:

void PushFront(const float& data);

void PushBack(const float& data);

void show();

bool isNULL() {

if (size == 0)

return true;

else

return false;

}

float readLast(){ return tail->data;};

void read();

void write();

void deleted() {

while (head) {

Deque\* deletedITM = this->head;

float tempData = deletedITM->data;

this->head = deletedITM->next;

delete deletedITM; deletedITM = NULL;

size--;

}

cout << "\tDeque have been deleted!\n";

}

Deque() { head = NULL; tail = NULL; size = 0; }

};

#include "Header.h"

string dataPath = "data.txt";

void Deque::PushBack(const float& data)

{

Deque\* temp = new Deque;

temp->next = NULL;

temp->data = data;

temp->prev = tail;

if (tail != NULL)

tail->next = temp;

if (size == 0)

head = tail = temp;

else

tail = temp;

size++;

}

void Deque::PushFront(const float& data)

{

Deque\* temp = new Deque;

temp->data = data;

temp->prev = NULL;

if (size == 0)

head = tail = temp;

else {

temp->next = head;

head->prev = temp;

head = temp;

}

size++;

}

void Deque::show()

{

if (size != 0) {

Deque\* temp = head;

while (temp->next != 0)

{

cout << temp->data << " --> ";

temp = temp->next;

}

cout << temp->data << endl;

}

else

cout << "\tDack dose not exist!\n";

}

void Deque::read()

{

ifstream dataFile;

dataFile.open(dataPath);

if (dataFile.is\_open()) {

cout << "\tReading from the file was successful!\n";

float temp;

while (!dataFile.eof()) {

dataFile >> temp;

PushBack(temp);

}

}

else

cout << "\tReading from the file was not successful!\n";

dataFile.close();

}

void Deque::write()

{

ofstream dataFile;

dataFile.open(dataPath);

if (size == 0) {

cout << "\tDack haven`t found!\n";

cout << "\tWriting to the file was not successful!\n";

}

else {

Deque\* current = this->head;

while (current != NULL) {

dataFile << " ";

dataFile << current->data;

current = current->next;

}

/\*if (dataFile.is\_open())

cout << "\tWriting to the file was successful!\n";

else

cout << "\tWriting to the file was not successful!\n";\*/

}

dataFile.close();

}