Міністерство освіти і науки України

Національний технічний університет

«Харківський політехнічний інститут»

Навчально-науковий інститут комп’ютерних наук та інформаційних технологій

ЗВІТ

про виконання лабораторної роботи №2

Варіант 7

за дисципліною

«Алгоритми і структури даних»

Виконав:

студент групи КН-320Б

Миргород В.І.

Перевірила:

старший викладач

Мошко Є.О.

Харків 2022

**Лабораторна робота №2**  
Тема лабораторної роботи: Стеки, черги і декі та робота з ними  
Мета: ознайомитися із основними способами організації стеків, черг, деків та  
особливостями їх програмної реалізації. Набути практичних навичок по роботі з стеком, чергою, деком.

Завдання за варіантом:

Стек, черга та дек з елементами типу char.

Завдання:

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;  
− вставити символ '\*' в середину стека, якщо число елементів парне, або  
після середнього елемента, якщо число елементів непарне;  
− 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 шостий елемент;  
− прочитати останній елемент;  
− перевірити на порожнечу

Аналіз завдання:

Стек складається з елементів, кожний з яких містить покажчик на елемент, що знаходиться нижче за нього у структурі, а також поле даних. Стек містить покажчик на верхній елемент (top). Доступ до елементів відбувається по принципу LIFO, тобто елемент, що додано останнім, буде зчитано першим. Користувач може зчитувати лише верхній елемент, виймаючи його зі стеку, і таким чином відкриваючи елемент, що розташований нижче.

Структура вхідних та вихідних даних:

Дані вводяться користувачем з консолі або зчитуються з файлу. Результат виводиться до консолі та у файл. Тип даних – double.

Алгоритм розв’язання задачі:

Спочатку у функції main створюємо список. Конструктор за умовою встановлює значення NULL для голови та хвоста списку. Кількість вузлів дорівнює 0.

class List {

public:

Node\* head;

Node\* tail;

int count;

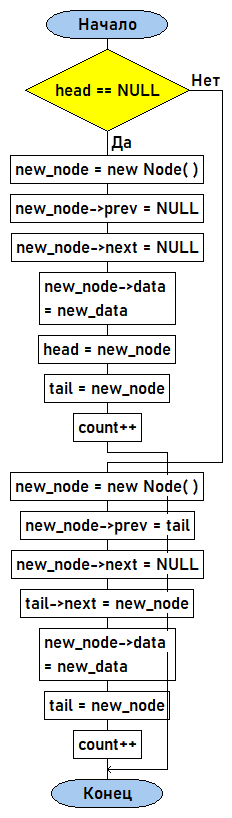
List() {

head = NULL;

tail = NULL;

count = 0;

}

}

Щоб додати вузли до списку, викорустовується метод AddEnd. У якості параметрів метод отримує дані типу double. Метод додає новий вузол в кінець списку та створює нові зв’язки між попередніми та наступними вузлами за допомогою покажчиків на вузел.

void List::AddEnd(Data\* new\_data) {

if (head == NULL) {

Node\* new\_node = new Node();

new\_node->prev = NULL;

new\_node->next = NULL;

new\_node->data = new\_data;

head = new\_node;

tail = new\_node;

count++;

return;

}

Node\* new\_node = new Node();

new\_node->prev = tail;

new\_node->next = NULL;

tail->next = new\_node;

new\_node->data = new\_data;

tail = new\_node;

count++;

}

Метод Delete дозволяє видалити один елемент зі списку. У якості параметра отримує номер елемента, який потрібно видалити. Також відбувається перевірка на можливість видалення. У разі наявності такого елемента у списку видаляє дані, що зберігалися у вузлі, та змінює зв’язки між покажчиками на попередній та наступний елементи.

void List::Delete(int num) {

Node\* cur = head;

if (num > count)

return;

for (int i = 1; i < num; i++)

cur = cur->next;

if ((cur == head) && (cur == tail)) {

head = NULL;

tail = NULL;

}

else if (cur == head) {

cur->next->prev = NULL;

head = cur->next;

}

else if (cur == tail) {

cur->prev->next = NULL;

tail = cur->prev;

}

else {

cur->prev->next = cur->next;

cur->next->prev = cur->prev;

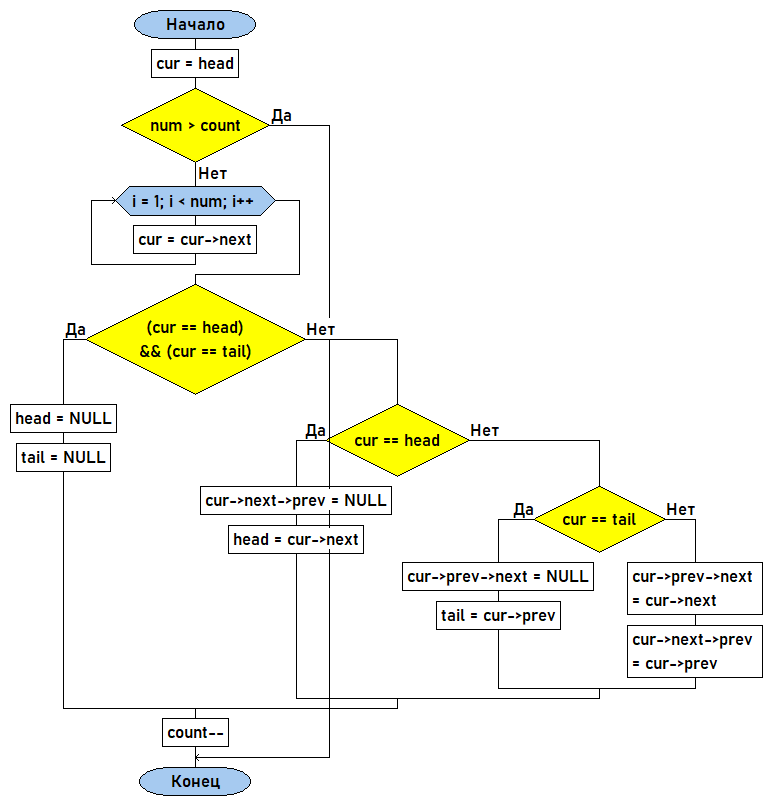
}

delete cur->data;

delete cur;

count--;

}



Метод Swap дозволяє поміняти місцями 2 сусідні елементи. У якості параметрів отримує 2 цілих числа – номери елементів, що потрібно поміняти місцями. Також перевіряється правильність введення номерів елементів.

void List::Swap(int num1, int num2) {

if ((num1 != num2 + 1) && (num2 != num1 + 1)) {

cout << "Это не соседние элементы!\n";

return;

}

if ((num1 > count) || (num2 > count)) {

cout << "Неправильно введен номер элемента!\n";

return;

}

if (num1 == num2)

return;

Node\* p1 = head;

for (int i = 1; i < num1; i++) {

p1 = p1->next;

}

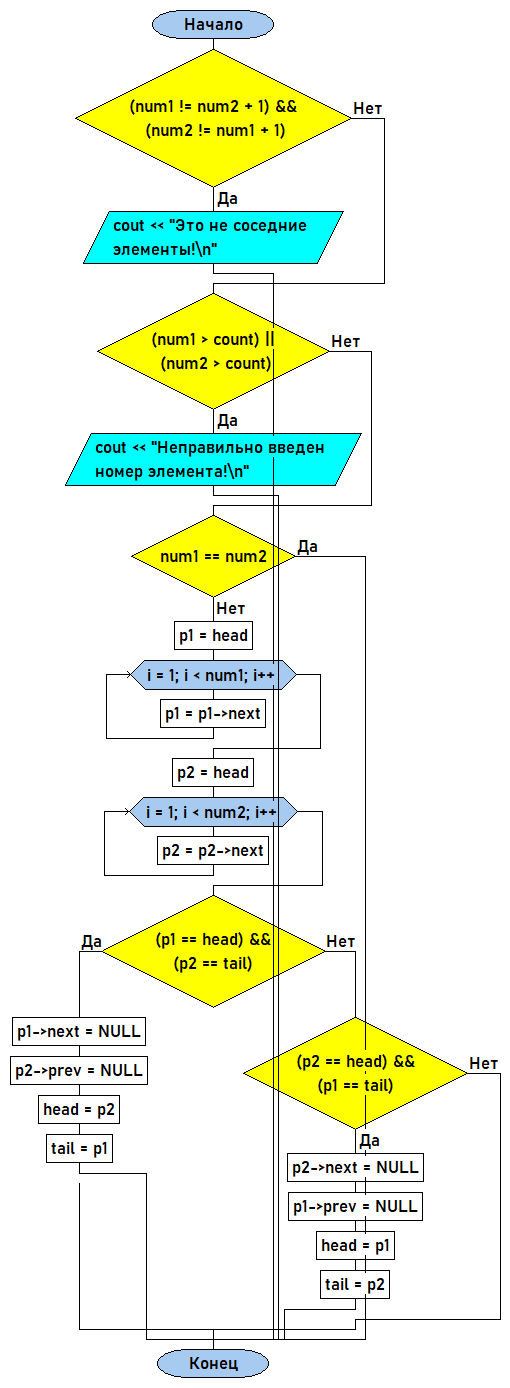
Node\* p2 = head;

for (int i = 1; i < num2; i++) {

p2 = p2->next;

}

}

 if ((p1 == head) && (p2 == tail)) {

p1->next = NULL;

p1->prev = p2;

p2->next = p1;

p2->prev = NULL;

head = p2;

tail = p1;

return;

}

else if ((p2 == head) && (p1 == tail)) {

p2->next = NULL;

p2->prev = p1;

p1->next = p2;

p1->prev = NULL;

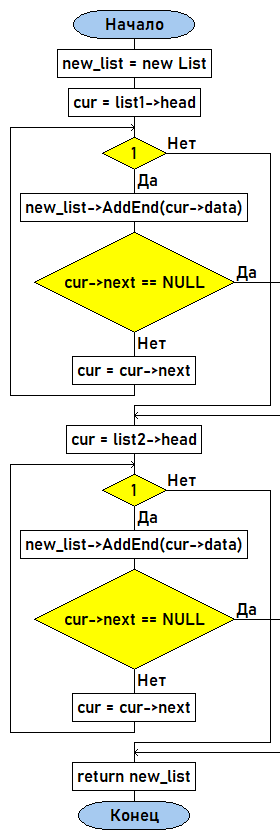
head = p1;

tail = p2;

return;

}

}

Метод AddList створює новий список, який складається з двох списків: початковий список, з яким проводились дії, та новий список, який дописується в кінець основного списку. Метод повертає список покажчик на новий список, й надалі він використовується як основний.

List\* List::AddList(List\* list1, List\* list2) {

List\* new\_list = new List;

Node\* cur = list1->head;

while (1) {

new\_list->AddEnd(cur->data);

if (cur->next == NULL)

break;

cur = cur->next;

}

cur = list2->head;

while (1) {

new\_list->AddEnd(cur->data);

if (cur->next == NULL)

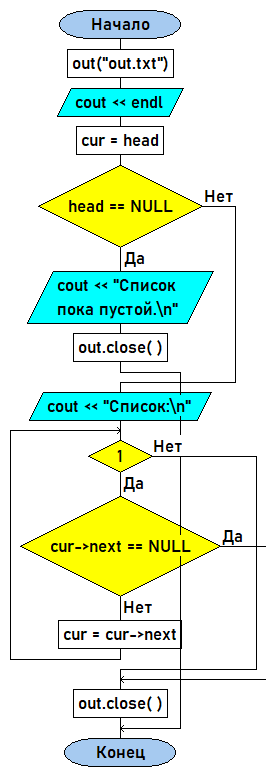
break;

cur = cur->next;

}

return new\_list;

}

Метод Show дозволяє вивести в консоль список, а також виводить у файл останній вигляд списку. Нічого не отримую у параметрах, та нічого не повертає. У разі, якщо список ще не має елементів, виводить відповідне повідомлення.

void List::Show() {

ofstream out("out.txt");

cout << endl;

Node\* cur = head;

if (head == NULL) {

cout << "Список пока пустой.\n";

out.close();

return;

}

cout << "Список:\n";

while (1) {

cout << cur->data->v << " ";

out << cur->data->v << " ";

if (cur->next == NULL)

break;

cur = cur->next;

}

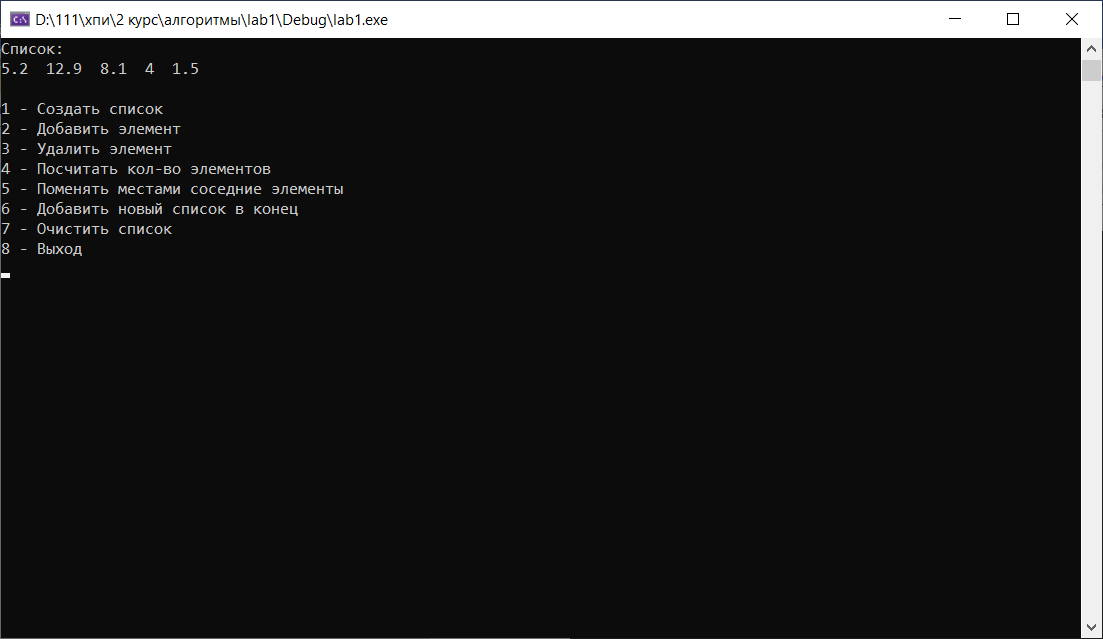
out.close();

}

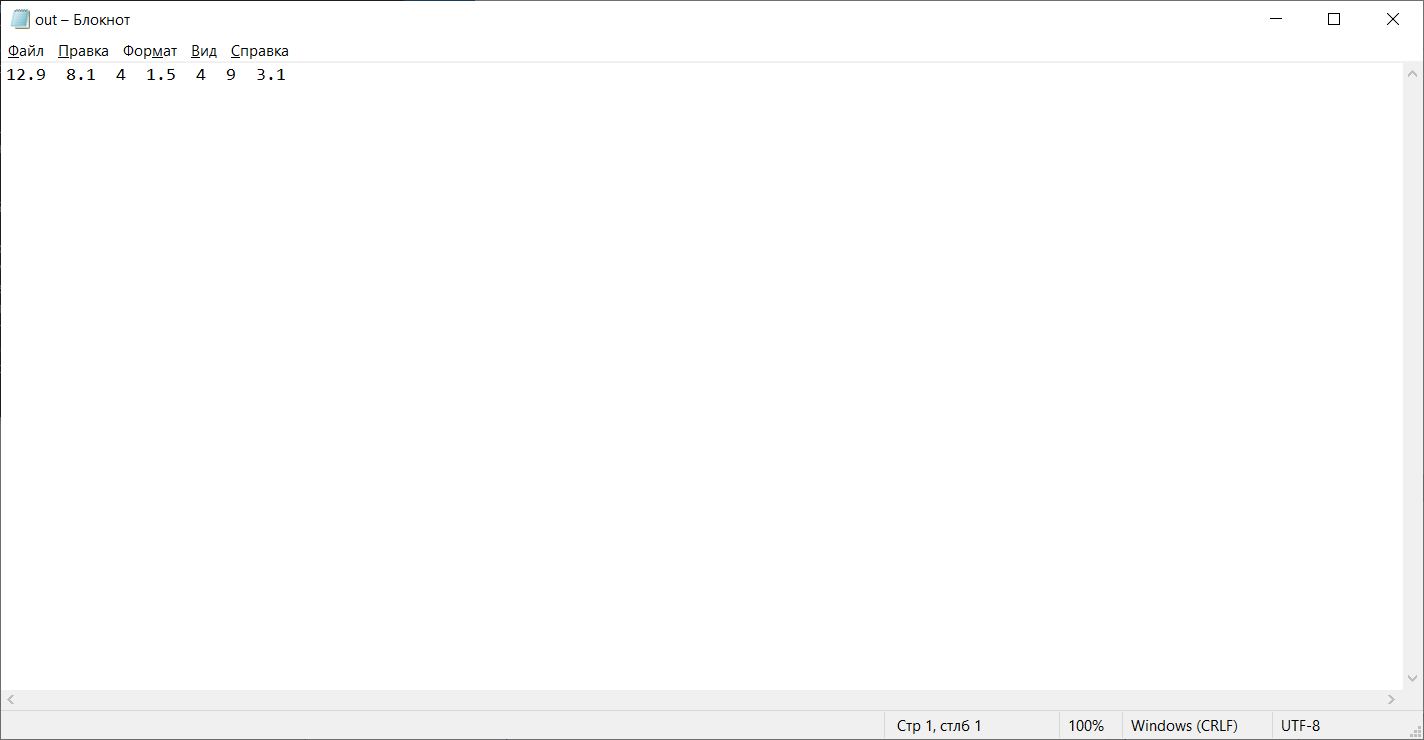
При завершенні програми всі динамічні структури видаляються з пам’яті. В файлі залишається остання збережена копія списку.

Результати роботи програми:

Консольне вікно



Файл із списком



Висновки:

Двозв’язний список – динамічна лінійна структура даних з послідовним доступом до елементів. Двозв’язний список складається з вузлів, що мають покажчики на наступний та попередній елемент, а також зберігають дані.

Двозв’язний кільцевий список відрізняється тим, що останній елемент вказую на перший, й навпаки. Кільцевий список дозволяє виконувати всі ті ж дії, що й звичайний двозв’язний список, але під час роботи з першим та останнім елементами списку потрібно пам’ятати про покажчики. В будь-якому непустому кільцевому списку покажчик на попередній та наступний елемент зберігає адресу вузла.

Зв’язні списки дозволяють працювати з елементами, надаючи майже всі функції обробки звичайного масиву, але деякі дії для них виконуються швидше та зручніше, наприклад, додавання чи видалення елементів.