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 – char.

Алгоритм розв’язання задачі:

При створенні нового стеку, покажчик на його верхній елемент має значення NULL.

Далі, заповнюємо стек значеннями. Для цього викликаємо метод Push і послідовно додаємо елементи до стека.

void Stack::Push(Element\_stack\* temp) {

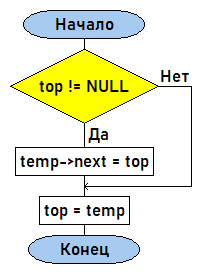
if (top != NULL) {

temp->next = top;

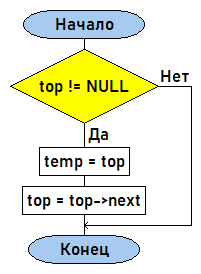
}

top = temp;

}



Щоб вийняти елемент зі стека, використовується метод Pop. Він дозволяє вийняти зі стека верхній елемент, тобто той, що було додано останнім.



void Stack::Pop() {

if (top != NULL) {

Element\_stack\* temp = top;

top = top->next;

delete temp;

}

}

Метод Out виводить всі елементи стека на екран та до файлу. Якщо стек ще пустий, видає повідомлення про це.

void Stack::Out() {

ofstream out("out\_stack.txt");

cout << "Стек:\n";

out << "Стек:\n";

if (top == NULL) {

cout << "Стек пустой...\n";

out << "Стек пустой...\n";

out.close();

return;

}

Stack\* temp = new Stack;

while (this->top != NULL) {

temp->Push(new Element\_stack(this->top->data));

cout << setw(5) << top->data << endl;

out << setw(5) << top->data << endl;

Pop();

}

while (temp->top != NULL) {

Push(new Element\_stack(temp->top->data));

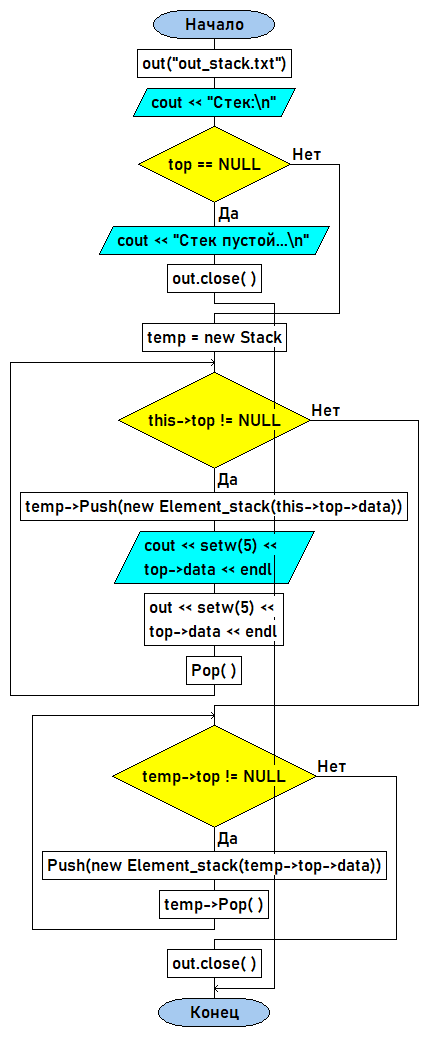
temp->Pop();

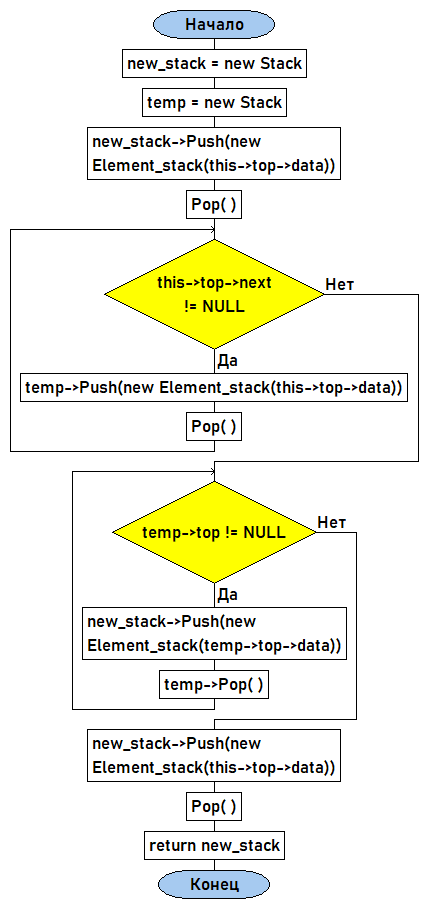
}

out.close();

delete temp;

}



Метод Swap дозволяє поміняти місцями перший та останній елементи. Для цього створюється новий стек, в який методом Push копіюються елементи заданого стека. При цьому всі елементи, крім останнього, потрібно “перевернути”, бо після копіювання вони знаходяться у зворотньому порядку.

Stack\* Stack::SwapFirstLast() {

Stack\* new\_stack = new Stack; //новый стек, тут будет результат

Stack\* temp = new Stack; //временный стек, тут хранятся элементы между первым и последним

new\_stack->Push(new Element\_stack(this->top->data)); //верхний элемент будет в самом низу

Pop(); //убираем верхний элемент

while (this->top->next != NULL) { //записали элементы во временный стек

temp->Push(new Element\_stack(this->top->data));

Pop();

}

while (temp->top != NULL) { //переписали все элементы в новый стек, кроме последнего

new\_stack->Push(new Element\_stack(temp->top->data));

temp->Pop();

}

new\_stack->Push(new Element\_stack(this->top->data)); //добавили последний элемент

Pop();

delete this;

delete temp;

return new\_stack;

}

Наступний вид структури – Черга (Queue). Вона працює за принципом FIFO, тобто елемент, що додано першим, вийде зі структури також першим. Тому структура зберігає покажчики на перший та останній елементи, щоб знати. Потрібно реалізувати методи додавання та виймання елементів, підрахунок кількості елементів, знаходження середнього арифметичного, мінімального та максимального елементів.

Для додавання елемента викликається метод Push. Він додає новий елемент у кінець черги, а покажчик tail тепер показує на цей елемент.

void Queue::Push(Element\_queue\* temp) {

if (head == NULL) {

head = temp;

}

else {

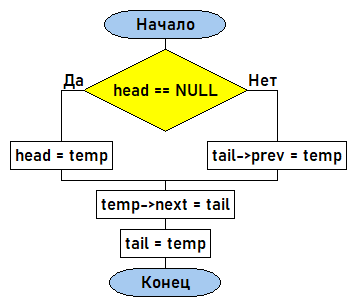
tail->prev = temp;

}

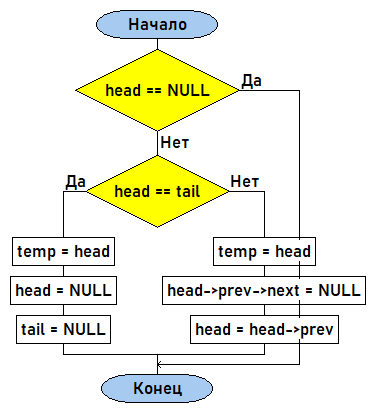
temp->next = tail;

tail = temp;

}



“Виймання” елемента з черги виконується за допомогою метода Pop. При цьому виймається завжди перший елемент, на який показує покажчик head. Також виконується перевірка, чи не є черга пустою.

void Queue::Pop() {

if (head == NULL)

return;

if (head == tail) {

Element\_queue\* temp = head;

head = NULL;

tail = NULL;

delete temp;

}

else {

Element\_queue\* temp = head;

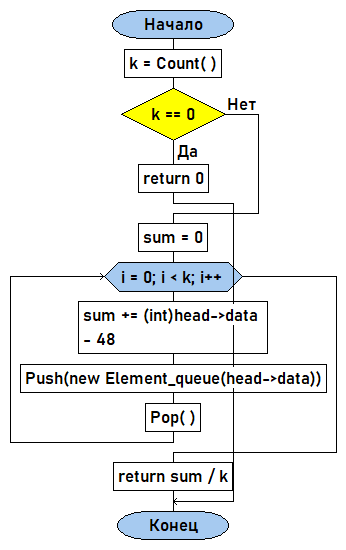
head->prev->next = NULL;

head = head->prev;

delete temp;

}

}



Для знаходження середнього арифметичного застосовується метод Middle.

double Queue::Middle() {

int k = Count();

if (k == 0)

return 0;

double sum = 0;

for (int i = 0; i < k; i++) {

sum += (int)head->data - 48;

Push(new Element\_queue(head->data));

Pop();

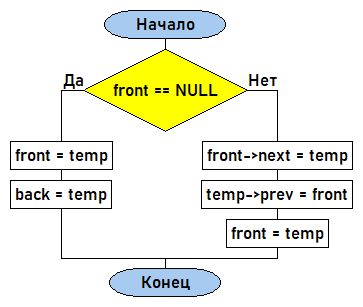
}

return sum / k;

}

Остання структура, розроблена протягом виконання лабораторної роботи – дек (Deque). Ця структура схожа на чергу. Але, на відміну від черги, в дек можна додавати елементи як у початок, так і в кінець. Виймання елементів також можна здійснювати з обох боків дека.

Розглянемо додавання та видалення елементів із початку дека. Методи додавання та видалення кінцевого елемента відбуваються так само, але змінюється покажчик кінцевого елемента (back).



void Deque::Push\_front(Element\_deque\* temp) {

if (front == NULL) {

front = temp;

back = temp;

}

else {

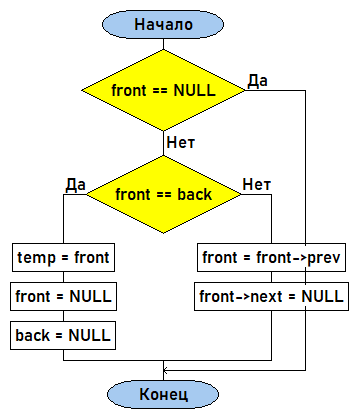
front->next = temp;

temp->prev = front;

front = temp;

}

}



void Deque::Pop\_front() {

if (front == NULL)

return;

if (front == back) {

Element\_deque\* temp = front;

front = NULL;

back = NULL;

delete temp;

}

else {

front = front->prev;

delete front->next;

front->next = NULL;

}

}

Метод Count рахує кількість елементів дека. При цьому потрібно створювати новий дек та пересипати елементи до нього. Надалі, знаючи кількість елементів, можна не створювати новий дек, а одразу виймати з одного боку, та додавати з іншого. Таким чином дек не зміниться після виконання дій.

int Deque::Count() {

Deque\* temp = new Deque;

int k = 0;

while (front != NULL) {

temp->Push\_back(new Element\_deque(front->data));

Pop\_front();

k++;

}

while (temp->front != NULL) {

Push\_back(new Element\_deque(temp->front->data));

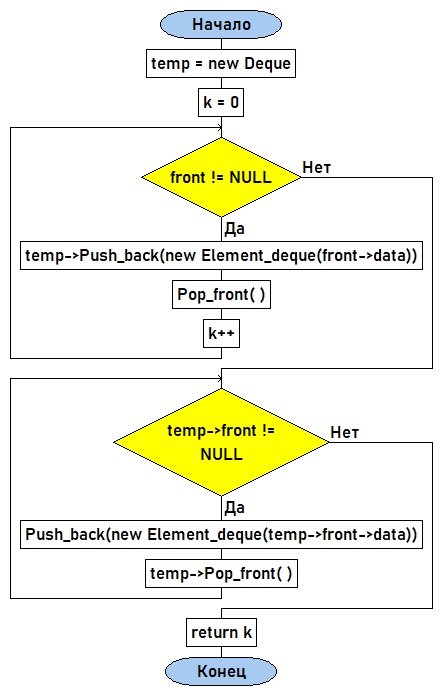
temp->Pop\_front();

}

delete temp;

return k;

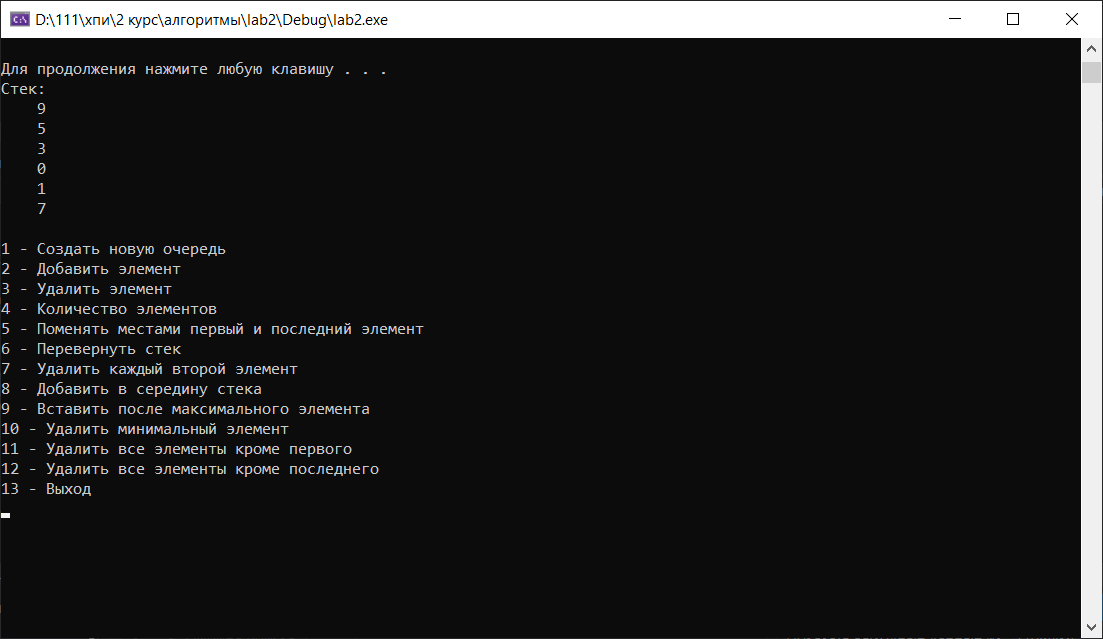
}



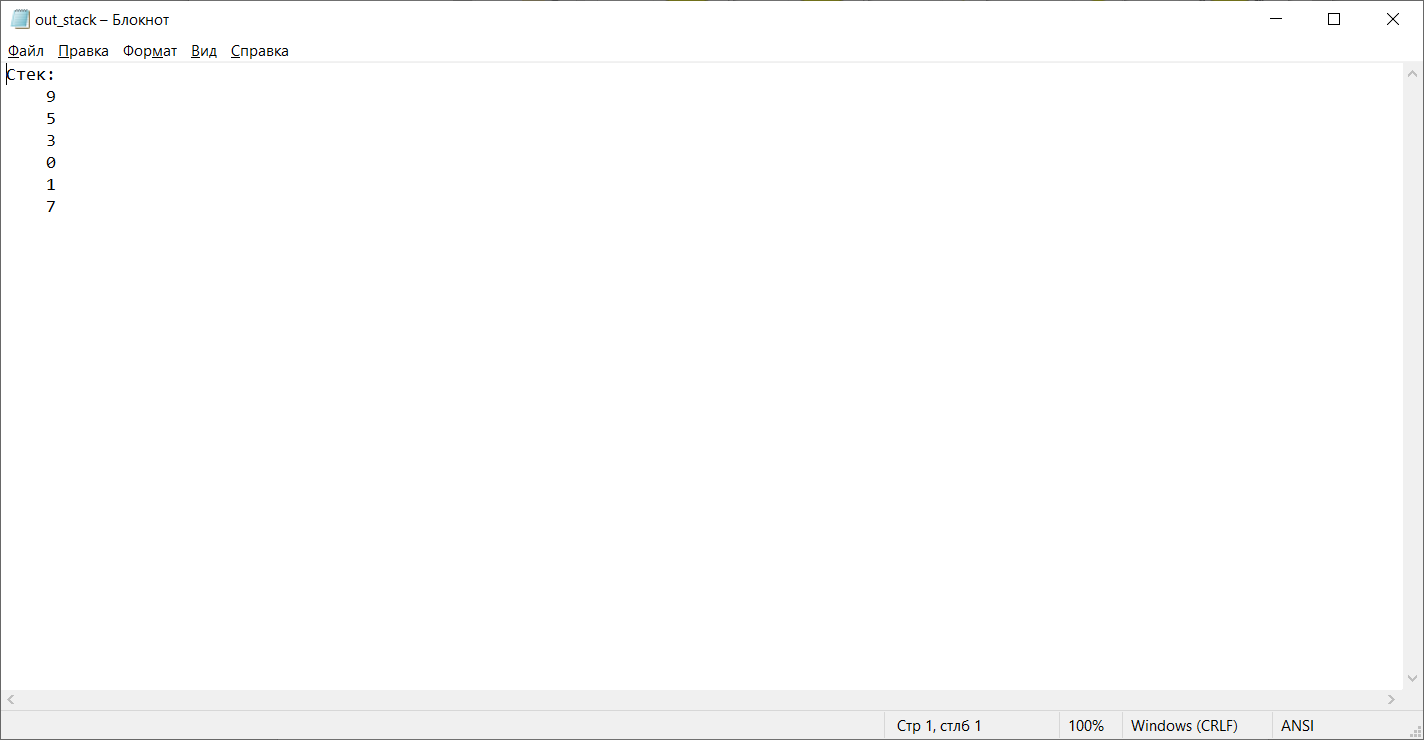
При завершенні програми всі динамічні структури видаляються з пам’яті. В файлах залишаються останні копія стека, черги та дека відповідно.

Результати роботи програми:

Консольне вікно під час роботи зі стеком.



Файл, в якому збережено останню копію стеку.



Висновки:

Стек, черга та дек дозволяють швидко додавати елемент, якщо не важливе його розташування. Також видалення елемента, на відміну від масиву, виконується досить швидко. Складнощі можуть бути, якщо елемент знаходиться в середині структури.

Взагалі, ці структури дозволяють виконувати всі дії, що й звичайний масив. Деякі дії виконуються швидше за інші – в першу чергу зміна розмірів структури.