МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 8
На тему: "*ЛІНІЙНІ СТРУКТУРИ ДАНИХ* "
З дисципліни: "Алгоритми та структури даних"

Лектор:
доцент кафедри ПЗ
Коротєєва Т.О.

Виконав:
студент групи ПЗ-22
Коваленко Д.М.

Прийняв:
асистент кафедри ПЗ
Франко А.В.

«_____» _____ 2022 р.
$\sum$ = _____.....

Львів — 2022

Тема. ЛІНІЙНІ СТРУКТУРИ ДАНИХ .

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

Лабораторне 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:

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

Варіант 2: стек дійсних.

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.

Хід роботи

```
use fake::{Faker, Fake};

use std::rc::Rc;

#[derive(Debug)]
pub struct Stack<T> {
    head: Link<T>,
}

type Link<T> = Option<Rc<Node<T>>>;

#[derive(Debug)]
struct Node<T> {
    elem: T,
    next: Link<T>,
}
```

```
impl<T> Stack<T> {
    pub fn new() -> Self {
        Stack { head: None }
    }

    pub fn push(&self, elem: T) -> Stack<T> {
        Stack { head: Some(Rc::new(Node {
                elem,
                next: self.head.clone(),
            }))}
    }

    pub fn pop(&self) -> Stack<T> {
        Stack { head: self.head.as_ref().and_then(|node| node.next.clone()) }
    }

    pub fn head(&self) -> Option<&T> {
        self.head.as_ref().map(|node| &node.elem)
    }

    pub fn iter(&self) -> Iter<'_, T> {
        Iter { next: self.head.as_deref() }
    }
}

impl Stack<f32> {
    pub fn new_random() -> Self {
        let stack = Stack::new();
        let stack = stack.push((f32::trunc(Faker.fake::<f32>() * 100.0)) / 100.0);
        let stack = stack.push((f32::trunc(Faker.fake::<f32>() * 100.0)) / 100.0);
        let stack = stack.push((f32::trunc(Faker.fake::<f32>() * 100.0)) / 100.0);
        let stack = stack.push((f32::trunc(Faker.fake::<f32>() * 100.0)) / 100.0);
        stack
    }
}

impl<T> Drop for Stack<T> {
    fn drop(&mut self) {
        let mut head = self.head.take();
        while let Some(node) = head {
            if let Ok(mut node) = Rc::try_unwrap(node) {
                head = node.next.take();
            } else {
                break;
            }
        }
    }
}

pub struct Iter<'a, T> {
    next: Option<&'a Node<T>>,
}

impl<'a, T> Iterator for Iter<'a, T> {
    type Item = &'a T;

    fn next(&mut self) -> Option<Self::Item> {
        self.next.map(|node| {
            self.next = node.next.as_deref();
            &node.elem
        })
```

```
        }
    }
```

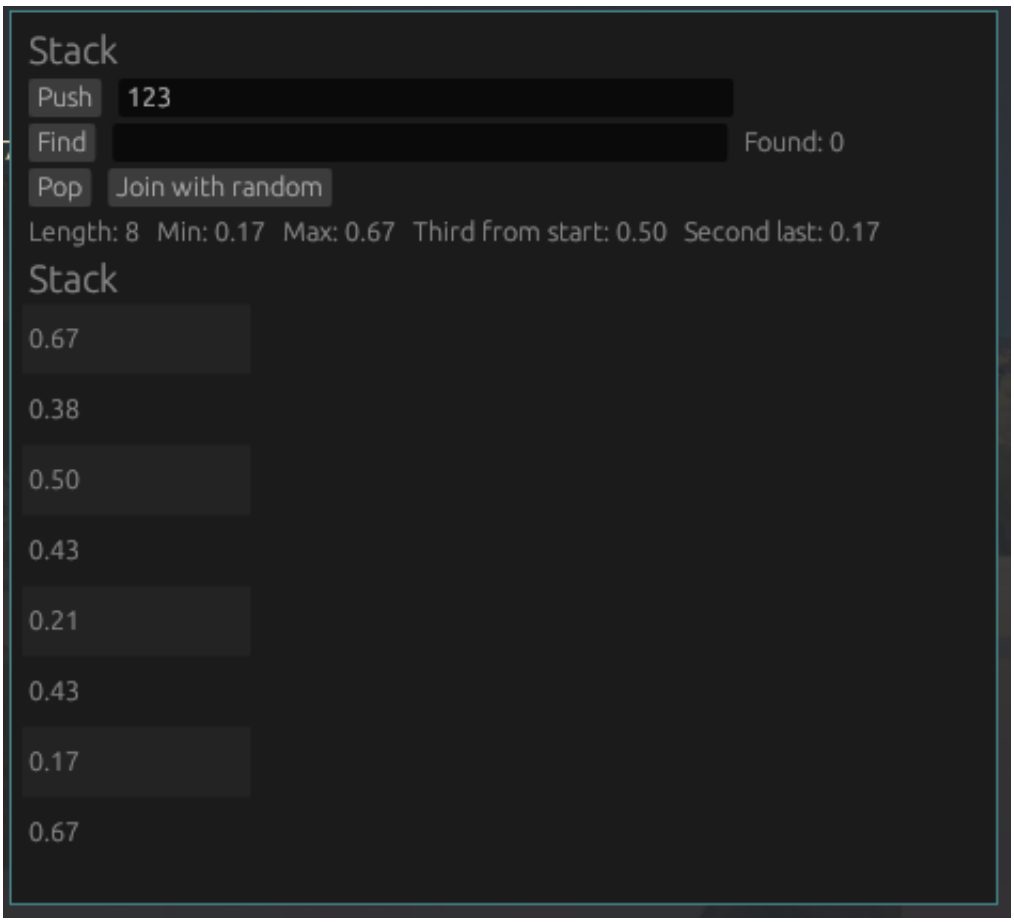

Рис. 1: Виконання програми

Висновоки

Під час виконання лабораторної роботи я познайомився з лінійними структурами даних (стек, черга, дек, список) та отримав навички програмування алгоритмів, що їх обробляють