

Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №5 з дисципліни
«Алгоритми та структури даних-1.
Основи алгоритмізації»

«Лінійні та нелінійні структури даних»

Варіант 13

Виконав студент ІП-43 Дутов Іван Андрійович

Перевірила Вітковська Ірина Іванівна

Київ 2024

Лабораторна робота №5

ЛІНІЙНІ ТА НЕЛІНІЙНІ СТРУКТУРИ ДАНИХ

Мета. Вивчити можливості та особливості створення та обробки лінійних та нелінійних структур даних.

Задача. Створити стек у векторному поданні довжиною $N \leq 15$ дійсного типу. Видалити елементи, які розташовані до мінімального елементу.

Постановка задачі

Створення стеку у векторному форматі передбачає виділення масиву певної максимальної довжини. У випадку нашої задачі достатньо розміру стеку **STACK_SIZE = 15**. Додатково обмежимо ввід N користувачем від 0 до 15, застосовуючи альтернативну форму оператора вибору після вводу. Верх стеку позначатиме **stack.top**, що буде індексом масиву **stack.values**.

Важливо розуміти, що стек — це структура, у якій доступний лише один елемент (на вершині стеку), тому в алгоритмах виведення та знаходження мінімального, що передбачають прохід по всіх елементах стеку, необхідно зберігати індекс початкової вершини стеку у змінній у підпрограмах **initial_top**.

Для видалення елементів, розташованих до мінімального елементу, знайдемо індекс мінімального елемента, застосовуючи підпрограму **find_min_element_index_in_stack**. Принцип її роботи полягає в проходженні масиву всередині стеку, починаючи з верху стеку, циклом з передумовою, на кожній ітерації якого, якщо щойно знайдене значення менше за мінімальне, що вже було знайдене, то мінімальне значення оновлюється, як і відповідний індекс.

Для додавання елементів у стек використаємо підпрограму **stack_push**, для виведення стека користувачу — **print_stack**.

Табл. 1: Таблиця імен змінних

Програма			
Змінна	Тип	Ім'я	Призначення
Максимальний розмір стека	цілий	STACK_SIZE	Глобальна константа
Масив можливих елементів	масив дійсних	elements	Глобальна константа
Обраний розмір стеку	цілий	actual_size	Ввід
Стек	Stack	stack	Структура даних, змінна
Підпрограма find_min_element_index_in_stack			
Змінна	Тип	Ім'я	Призначення
Вказівник на стек	Вказівник на Stack	stack	Формальний параметр
Мінімальний елемент стеку	дійсний	min	змінна
Індекс мінімального елемента	цілий	min_index	змінна
Початковий верх стека	цілий	initial_stack	змінна

Крок 1. Визначимо основні дії.

Крок 2. Уточнимо дію ініціалізації змінних.

Крок 3. Уточнимо дію перевірки **actual_size**

Крок 4. Уточнимо дію додавання елементів до стеку.

Крок 5. Уточнимо дію видалення елементів у стеку після мінімального.

Псевдокод (програма)

Крок 1

1. **Початок.**
2. Ініціалізація змінних.
3. **Ввід actual_size**
4. Перевірка **actual_size**
5. Додати елементи в стек.
6. **Вивід** стеку.
7. Видалити елементи у стеку до мінімального елемента.
8. **Вивід** стеку.
9. **Кінець.**

Крок 2

1. **Початок.**
2. Ініціалізація змінних.
 - 2.1 **STACK_SIZE := 15**
 - 2.2 **elements := [...]**
 - 2.3 **stack := struct Stack**
 - 2.4 **stack.top := -1**
3. **Ввід actual_size**
4. Перевірка **actual_size**
5. Додати елементи в стек.
6. **Вивід** стеку.
7. Видалити елементи у стеку до мінімального елемента.
8. **Вивід** стеку.
9. **Кінець.**

Крок 3

1. **Початок.**
2. Ініціалізація змінних.
 - 2.1 **STACK_SIZE := 15**
 - 2.2 **elements := [...]**
 - 2.3 **stack := struct Stack**
 - 2.4 **stack.top := -1**

3. **Ввід** `actual_size`
4. **Якщо** `actual_size > STACK_SIZE`,
то

 4.1 **Вивід** помилки.

інакше

5. Додати елементи в стек.
6. **Вивід** стеку.
7. Видалити елементи у стеку до мінімального елемента.
8. **Вивід** стеку.
9. **Кінець.**

Крок 4

1. **Початок.**

2. Ініціалізація змінних.

```
2.1 STACK_SIZE := 15  
2.2 elements := [...]  
2.3 stack := struct Stack  
2.4 stack.top := -1
```

3. **Ввід** `actual_size`

4. **Якщо** `actual_size > STACK_SIZE`,
то

 4.1 **Вивід** помилки.

інакше

5. **Повторити для** `i := 0` **до** `actual_size - 1`

 5.1 `stack_push(&stack, elements[i])`

все повторити

6. **Вивід** стеку.

7. Видалити елементи у стеку до мінімального елемента.

8. **Вивід** стеку.

9. **Кінець.**

Крок 5

1. **Початок.**

2. Ініціалізація змінних.

- 2.1 `STACK_SIZE := 15`
- 2.2 `elements := [...]`
- 2.3 `stack := struct Stack`
- 2.4 `stack.top := -1`
3. **Ввід** `actual_size`
4. **Якщо** `actual_size > STACK_SIZE`,
то
 - 4.1 **Вивід** помилки.
 - інакше**
5. **Повторити для** `i := 0` до `actual_size - 1`
 - 5.1 `stack_push(&stack, elements[i])`**все повторити**
6. `print_stack(&stack)`
7. Видалити елементи у стеку до мінімального елемента.
8. `print_stack(&stack)`
9. **Кінець.**

Крок 6

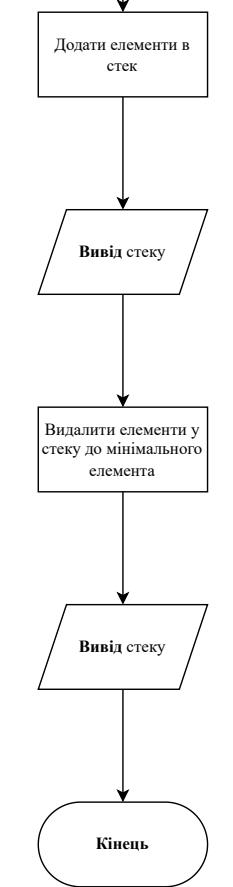
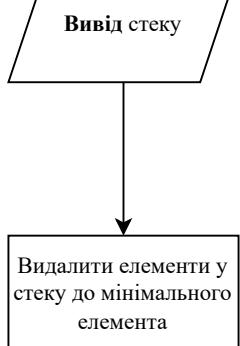
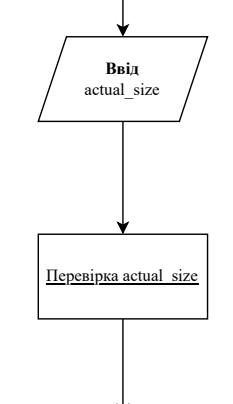
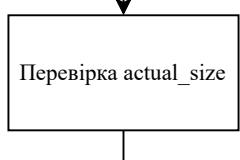
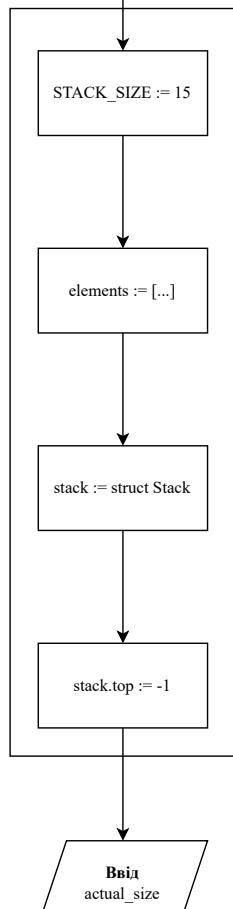
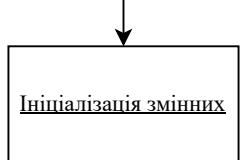
1. **Початок**.
2. Ініціалізація змінних.
 - 2.1 `STACK_SIZE := 15`
 - 2.2 `elements := [...]`
 - 2.3 `stack := struct Stack`
 - 2.4 `stack.top := -1`
3. **Ввід** `actual_size`
4. **Якщо** `actual_size > STACK_SIZE`,
то
 - 4.1 **Вивід** помилки.
 - інакше**
5. **Повторити для** `i := 0` до `actual_size - 1`
 - 5.1 `stack_push(&stack, elements[i])`**все повторити**
6. **Вивід** стеку.
7. `stack.top := find_min_element_in_stack(&stack)`
8. **Вивід** стеку.
9. **Кінець.**

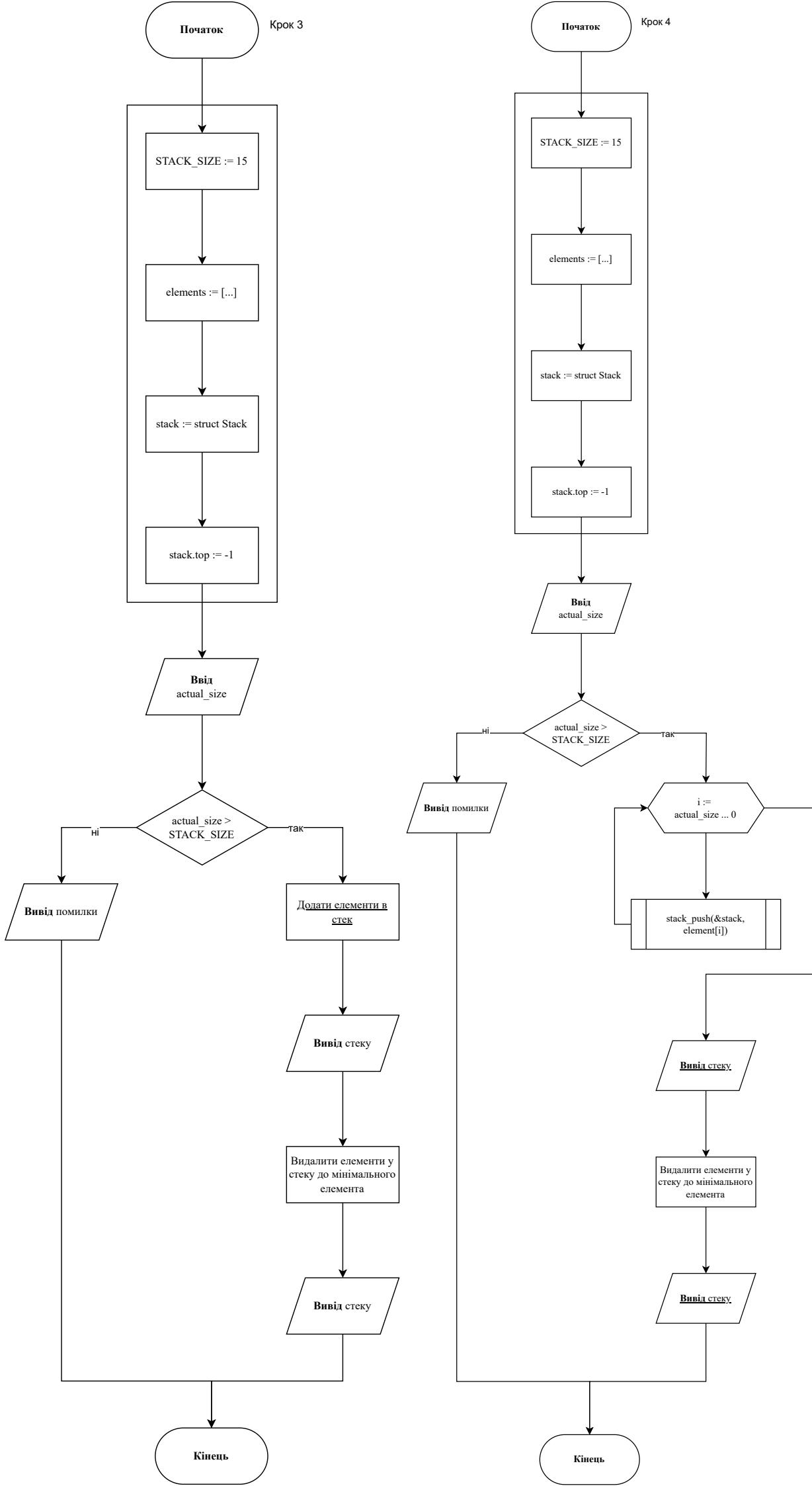
Початок

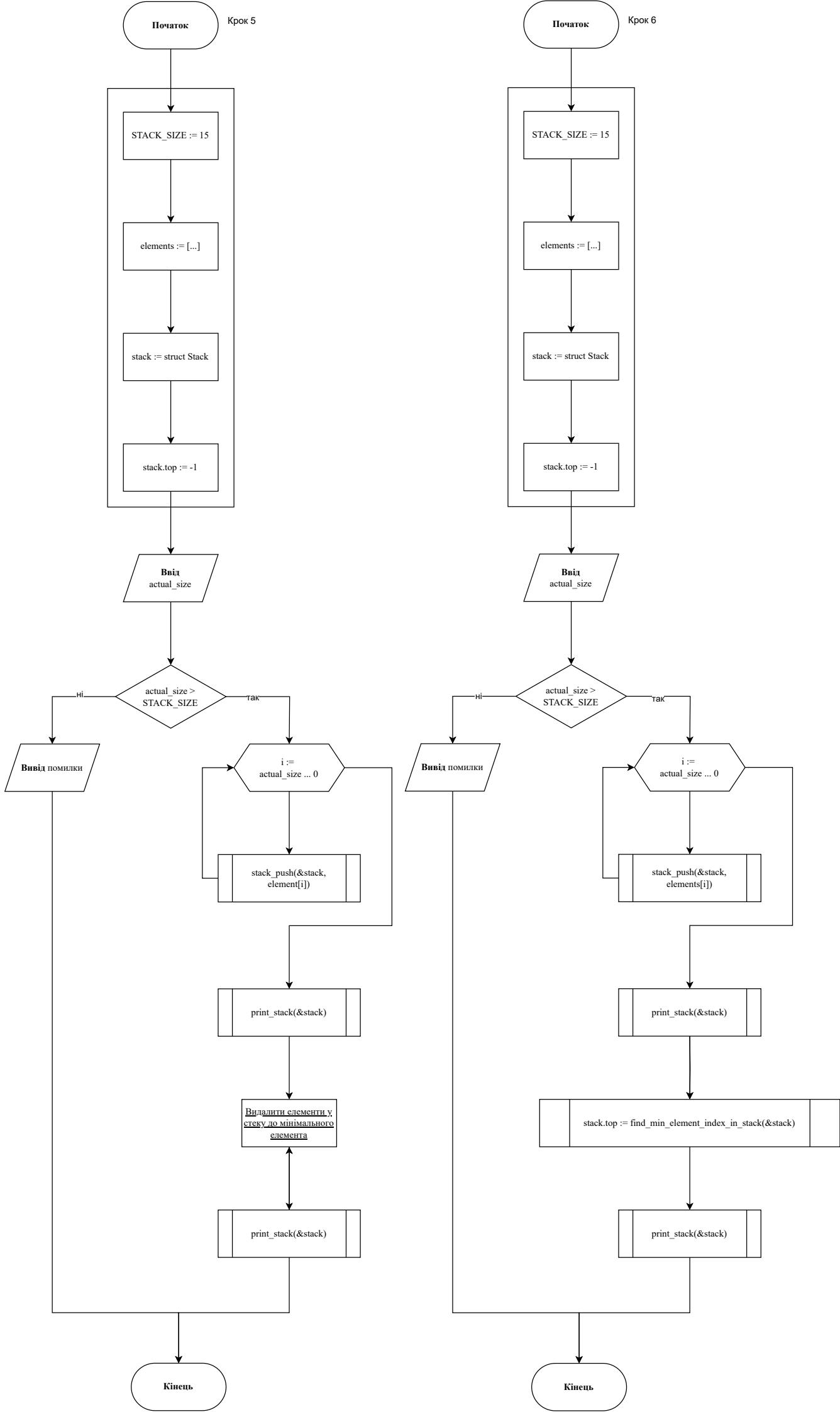
Крок 1

Початок

Крок 2







Псевдокод (підпрограма stack_push)

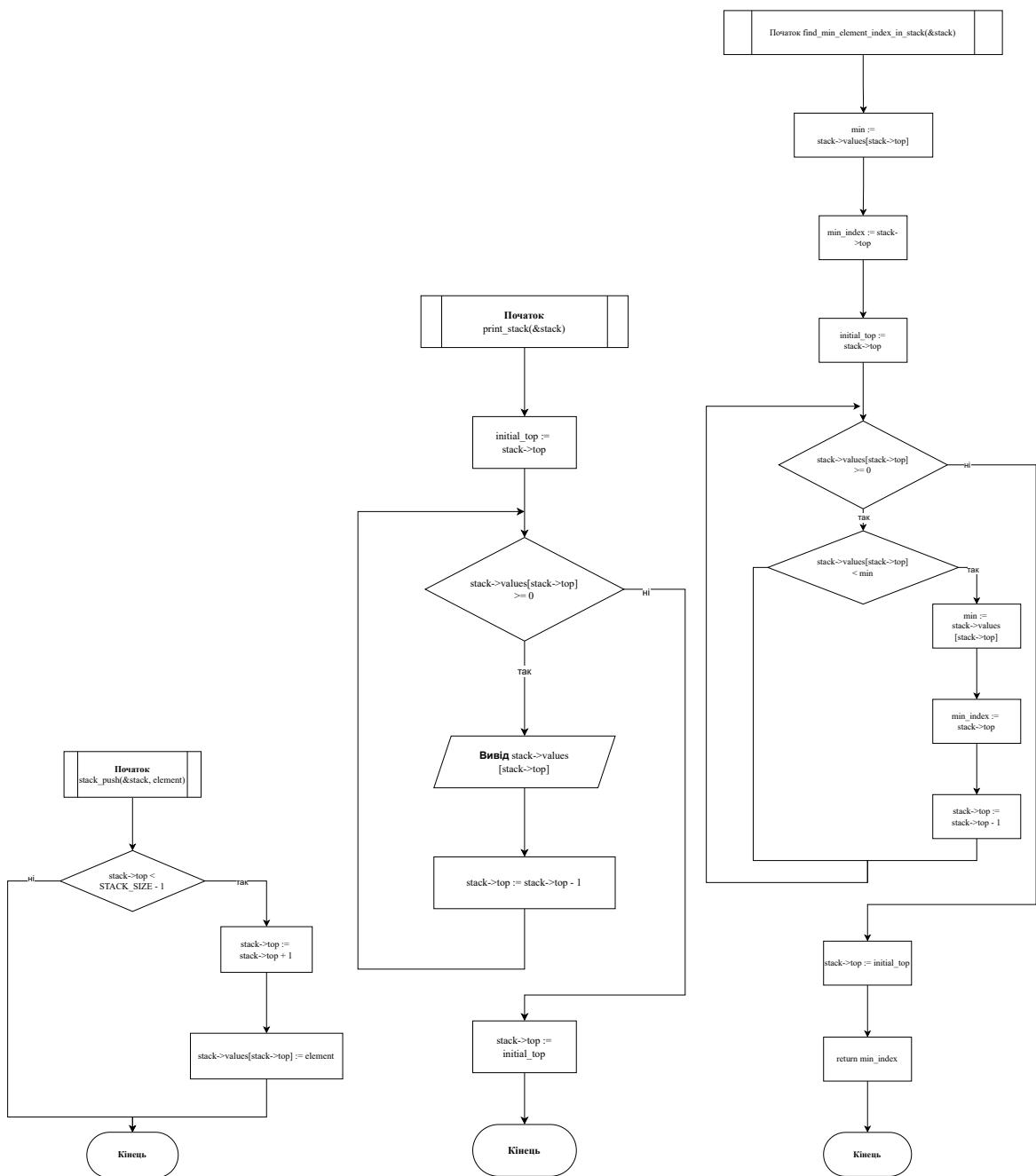
1. **Початок** stack_push(&stack, element)
2. **Якщо** stack->top < STACK_SIZE - 1,
то
 - 2.1 stack->top := stack->top + 1
 - 2.2 stack->values[stack->top] := element
3. **Кінець.**

Псевдокод (підпрограма print_stack)

1. **Початок** print_stack(&stack)
2. initial_top := stack->top
3. **Поки** stack->top >= 0
повторити
 - 3.1 **Вивід** stack->values[stack->top]
 - 3.2 stack->top := stack->top - 1
4. stack->top := initial_top
5. **Кінець**

Псевдокод (підпрограма find_min_element_index_in_stack)

1. **Початок** find_min_element_index_in_stack(&stack)
2. min := stack->values[stack->top]
3. min_index := stack->top
4. initial_top := stack->top
5. **Поки** stack->top >= 0
повторити
 - 5.1 **Якщо** stack->values[stack->top] < min,
то
 - 5.1.1 min := stack->values[stack->top]
 - 5.1.2 min_index := stack->top
 - 5.1.3 stack->top := stack->top - 1
6. stack->top := initial_top
7. **return** min_index
8. **Кінець**



Перевірка правильності алгоритму для підпрограми `find_min_element_index_in_stack`

Нехай стек задано так:

```

1  stack.values = {
2      -7.498132696263866, -87.28780001761642, -4.682371098571011,
3      -12.359740394074862, -17.29508717466615, -86.13095629379185,
4      -31.666378488447194, 25.189350864618376, 88.08361577282281,
5      62.193763582828694 ...
6  };
7  stack.top = 9;

```

1. `min := values[9] = 62.193763582828694`
2. `min_index := 9`

3. `values[9] = 62.193763582828694 < min = 62.193763582828694`
— хиба, переходимо до п. 3
4. `values[8] = 88.08361577282281 < min = 62.193763582828694`
— хиба, переходимо до п. 4
5. `values[7] = 25.189350864618376 < min = 62.193763582828694`
— істина
6. `min := 25.189350864618376`
7. `min_index := 7`, переходимо до п. 5
8. `values[6] = -31.666378488447194 < min = 25.189350864618376`
— істина
9. `min := -31.666378488447194`
10. `min_index := 6`, переходимо до п. 8
11. `values[5] = -86.13095629379185 < min = -31.666378488447194`
— істина
12. `min := -86.13095629379185`
13. `min_index := 5`, переходимо до п. 11
14. `values[4] = -17.2950871746661 < min = -86.13095629379185`
— хиба, переходимо до п. 14
15. `values[3] = -12.359740394074862 < min = -86.13095629379185`
— хиба, переходимо до п. 15
16. `values[2] = -4.682371098571011 < min = -86.13095629379185`
— хиба, переходимо до п. 16
17. `values[1] = -87.28780001761642 < min = -86.13095629379185`
— істина
18. `min := -87.28780001761642`
19. `min_index := 1`, переходимо до п. 17
20. `values[4] = -7.498132696263866 < min = -86.13095629379185` — хиба,
переходимо до п. 20
21. Повертаємо значення `min_index = 1`

Kod програми мовою C

```

1 #include <stdio.h>
2
3 #define STACK_SIZE 15
4
5 struct Stack {
6     int top;
7     double values[STACK_SIZE];

```

```

8 };
9
10 void stack_push(struct Stack *stack, double element);
11 void print_stack(struct Stack *stack);
12 int find_min_element_index_in_stack(struct Stack *stack);
13
14 const double elements[STACK_SIZE] = {
15     1.6352494344081094, 4.249691555207235, 5.656416109730378,
16     -6.826495617321376, 8.534751155386815, -9.163836489545096,
17     4.062834525719474, -2.2544772247450284, 8.07443987998061,
18     -3.910143776435233, 6.684845233843628, 5.560324553666035,
19     2.055901936315042, 3.7965916102743194, 1.1942907658533386};
20
21 int main() {
22     printf("Як_багато_елементів_матиме_ваш_стек_від(_1_до_15):_");
23     int actual_size;
24     scanf("%U", &actual_size);
25
26     if (actual_size > STACK_SIZE) {
27         printf("ПОМИЛКА!_Стек_розміром_%d_не_може_мати_більше_%d_елементів\n",
28                STACK_SIZE, actual_size);
29         return 1;
30     }
31
32     struct Stack stack;
33     stack.top = -1;
34
35     for (int i = 0; i < actual_size; i++) {
36         stack_push(&stack, elements[i]);
37     }
38
39     printf("Стек_на_початку:\n");
40     print_stack(&stack);
41
42     stack.top = find_min_element_index_in_stack(&stack);
43
44     printf("Стек_у_кінці:\n");
45     print_stack(&stack);
46
47     return 0;
48 }
49
50 void stack_push(struct Stack *stack, double element) {
51     if (stack->top < STACK_SIZE - 1) {
52         stack->top++;
53         stack->values[stack->top] = element;
54     }
55 }
56
57 void print_stack(struct Stack *stack) {
58     int initial_top = stack->top;
59
60     while (stack->top >= 0) {
61         printf("%lf\n", stack->values[stack->top]);
62         stack->top--;
63     }
64
65     stack->top = initial_top;
66 }
67

```

```
68 int find_min_element_index_in_stack(struct Stack *stack) {  
69  
70     int min = stack->values[stack->top];  
71     int min_index = stack->top;  
72     int initial_top = stack->top;  
73  
74     while (stack->top >= 0) {  
75         if (stack->values[stack->top] < min) {  
76             min = stack->values[stack->top];  
77             min_index = stack->top;  
78         }  
79         stack->top--;  
80     }  
81  
82     stack->top = initial_top;  
83  
84     return min_index;  
85 }
```

Висновок

Ми вивчили можливості та особливості створення та обробки лінійних та нелінійних структур даних на прикладі лінійного статичного векторного стека, розробивши підпрограму для видалення елементів стеку з його вершини до мінімального елемента.