МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра інформаційних систем та технологій

Звіт

з комп'ютерного практикуму 4

«Методи розробки алгоритмів»

з дисципліни

«Теорія алгоритмів»

Бригада – 10

Варіант № 5

Перевірила:

ст. вик. Солдатова М. О.

Виконали:

Бойко Катерина,

Гоголь Софія,

Павлова Софія,

Хіврич Володимир

Київ 2022

Комп’ютерний практикум 4

Тема: Абстрактні типи даних.

Мета роботи: порівняння алгоритмів розв’язку задачі, побудованих різними методами.

Завдання

Постановка задачі:

Варіант 5:

Викладач проводив за час сесії іспити в декількох групах. Коли він перевіряв роботи він складав їх в стопки в порядку перевірки. Після перевірки він заносить бали у свою звітну таблицю. Заради аналізу успішності студентів за останні 6 років він порівнює результат трьох підряд років і досліджує динаміку, для цього йому необхідно зробити певний перелік. Допоможіть викладачу раціонально організувати процес аналізу успішності.

Мета задачі – допомогти викладачу раціонально організувати процес аналізу успішності студентів за останні 6 років.

Використані структури даних: стек, покажчики, цілочисельний тип для збереження кількості елементів та пам'яті.

Вхідні та вихідні дані:

- У якості вхідних даних беремо кількість елементів стеку.
- У якості вихідних даних – виводимо стек та видалені елементи зі стеку.

Обрана структура даних

Стек — різновид лінійного списку, структура даних, яка працює за принципом «останнім прийшов — першим пішов» (LIFO, англ. last in, first out). Всі операції в стеку можна проводити тільки з одним елементом, який міститься на верхівці стека та був уведений в стек останнім.

Опис операцій зі стеком

push («заштовхнути елемент»): елемент додається в стек та розміщується в його верхівці. Розмір стека збільшується на одиницю. При перевищенні граничної величини розміру стека, відбувається переповнення стека.

pop («виштовхнути елемент»): повертає елемент з верхівки стека. При цьому він видаляється зі стека і його місце у верхівці стека займає наступний за ним відповідно до правила LIFO, а розмір стека зменшується на одиницю. При намаганні «виштовхнути» елемент зі вже пустого стека, відбувається ситуація «незаповненість» стека

Кожна з цих операцій зі стеком виконується за фіксований час O(1) і не залежить від розміру стека.

Модель:

Основні величини:

Назва змінної	Тип змінної	Значення змінної
size	unsigned int	кількість елементів, на які виділена пам'ять
n	unsigned int	кількість елементів, що зберігає стек
a	Data*	покажчик на динамічно виділену пам'ять під елементи стеку

Допоміжні величини:

Назва змінної	Тип змінної	Значення змінної
num	int	кількість елементів стеку
st	struct Stack	стек

Табл 1. Таблиця величин

Блок-схеми:

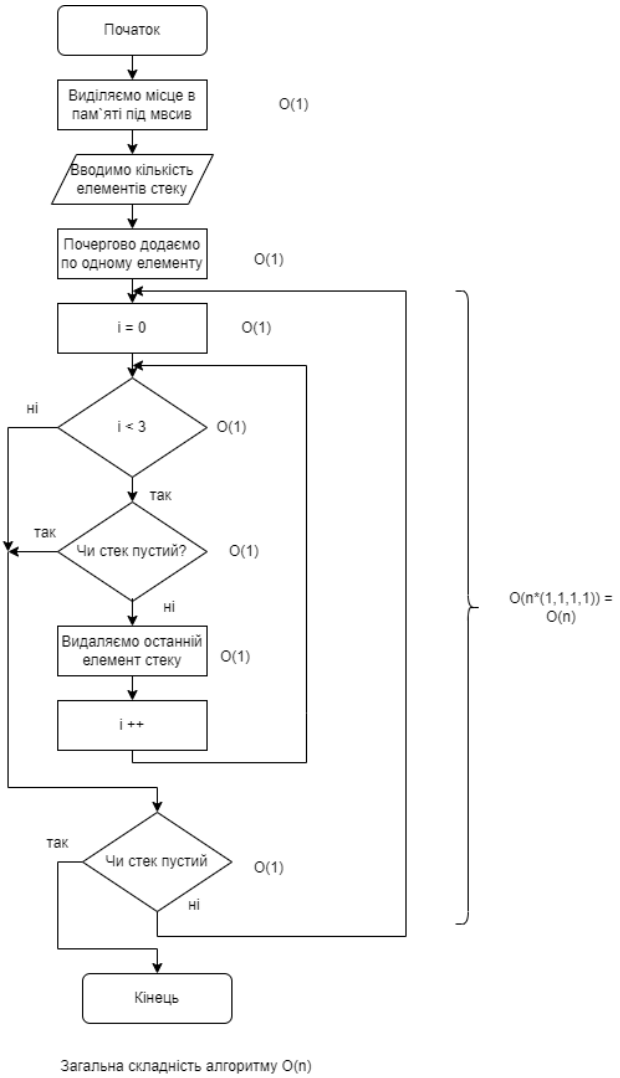


Рис. 1. Блок схема алгоритму програми зі статичним масивом та оцінкою складності

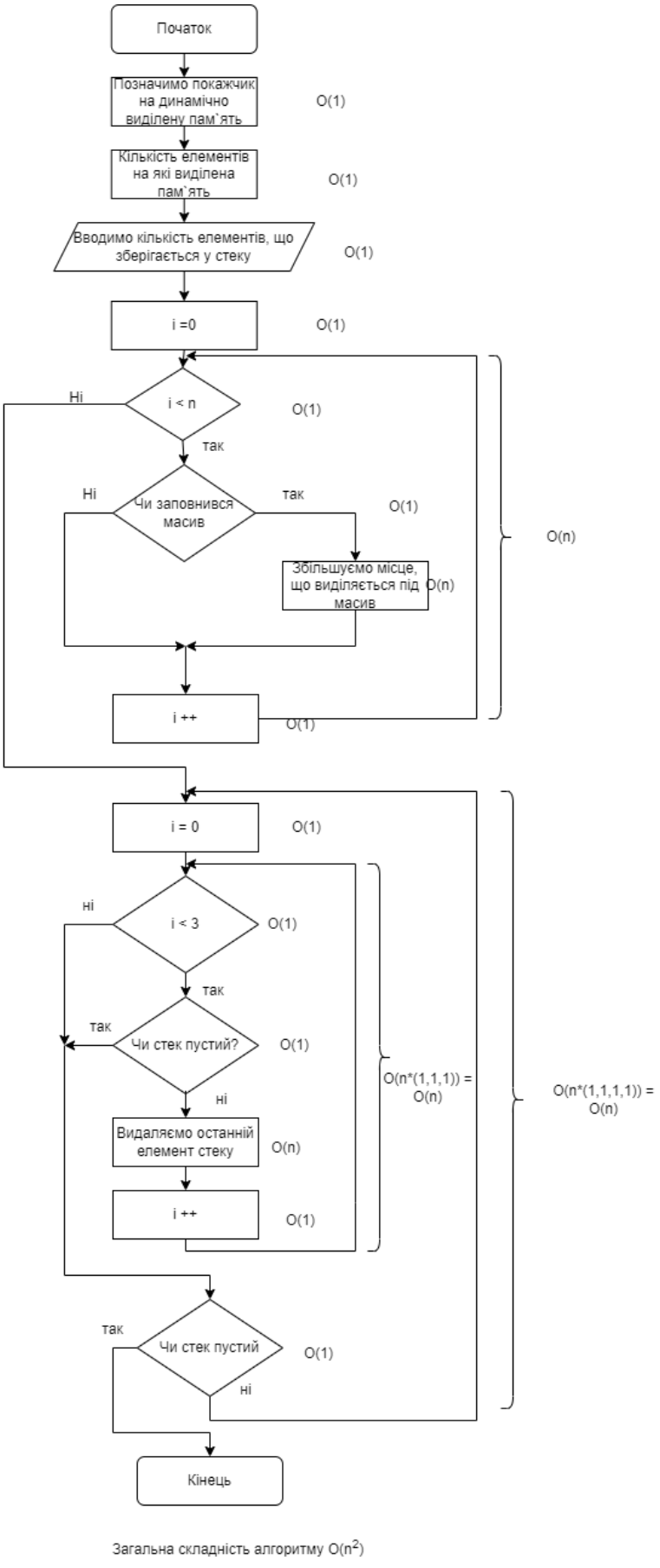


Рис. 2. Блок схема алгоритму програми з динамічним масивом та оцінкою складності

Оцінка теоретичної складності алгоритмів:

Статичний массив

На кожному кроці алгоритм збільшує потік хоча б на одну одиницю, отже він закінчить роботу не більше, ніж за O(n) кроків.

Динамічний масив

Збільшуючи потік у n разів, ми “копіюємо” ті операції, що виконувались до цього, тому загальною складністю алгоритму з використанням динамічного масиву є O(n^2)

Висновки про доцільність кожної з реалізацій:

Стек може бути реалізований на основі статичного масиву або на основі динамічного. Реалізація на основі статичного більш проста, але має ряд недоліків: обмежену кількість елементів, неефективне використання пам’яті та інші. Реалізація на основі динамічного більш складна, але позбавлена вказаних недоліків.

Лістінг програм:

Реалізація з фіксованими масивами:

```
#include <iostream>
#include <ctime>
#include <windows.h>
using namespace std;

typedef int Data;
#define N 10
Data a[N];

struct Stack
{
   Data a[N];          // елементи стеку
   int n;              // їх кількість
};

bool stack_empy(struct Stack* s)
{
   if (s->n == 0) return true;
   else return false;
```

```
}

void stack_print(struct Stack* s)
{
   cout << "\n----------Стек----------" << endl;
   if (stack_empy(s)) cout << "Стек пустий!";
   for (int i = 0; i < s->n; i++)
      cout << s->a[i] << " ";
   cout << endl;
}

void stack_push(struct Stack* s, Data data)
{
   s->a[s->n] = data;
   s->n++;
}

void stack_pop(struct Stack* s)
{
   cout << "\n---Видалені елементи----" << endl;
   for (int i = 0; i < 3; i++)
   {
      if (stack_empy(s)) break;
      cout << s->a[s->n - 1] << " ";
      s->n--;
   }
   cout << endl;
   stack_print(s);
}

void stack_init(struct Stack* s)
{
   s->n = 0;
}

int main()
{
   SetConsoleCP(1251);
   SetConsoleOutputCP(1251);

   int num;
   struct Stack st;
```

```
    stack_init(&st);

    cout << "\t * -------- * Комп'ютерний практикум 3 * -------- *\n";
    cout << "\t\t\t Бригада 10. Варіант 5  \n";
    cout << "\t Реалізація АТД Стеку за допомогою статичних масивів\n\n";

    cout << "Уведіть кількість елементів стеку: ";
    cin >> num;

    for (int i = 0; i < num; i++)
        stack_push(&st, rand() % 100);

    stack_print(&st);
    while (!stack_empy(&st))
        stack_pop(&st);

    cout << endl;
    return 0;
}
```

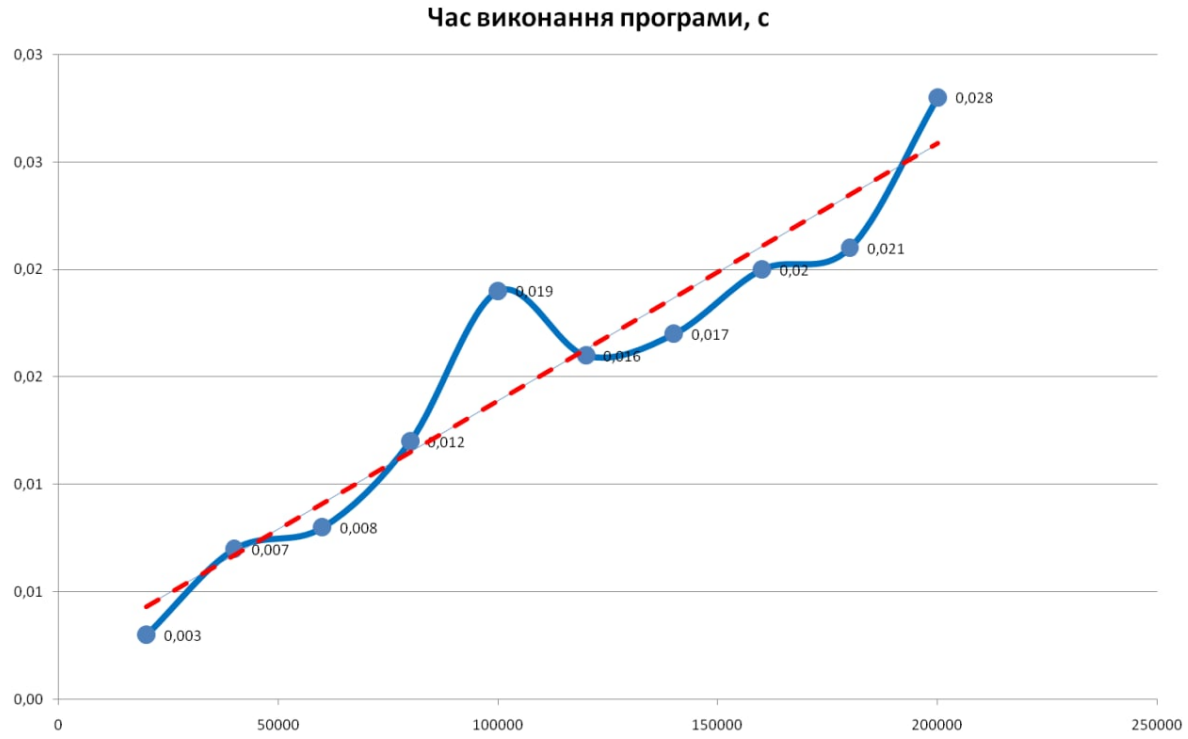
Реалізація з динамічними масивами:

```
#include <iostream>
#include <ctime>
#include <windows.h>
using namespace std;

typedef int Data;
#define N 10

struct Stack
{
    Data* a;                    // покажчик на динамічно виділену пам'ять під
елементи стеку
    unsigned int size;          // кількість елементів, на які виділена пам'ять
    unsigned int n;             // кількість елементів, що зберігає стек
};

void stack_init(struct Stack* s)
{
    s->n = 0;                   // пустий стек
    s->size = N;                // об'ємом N елементів
```

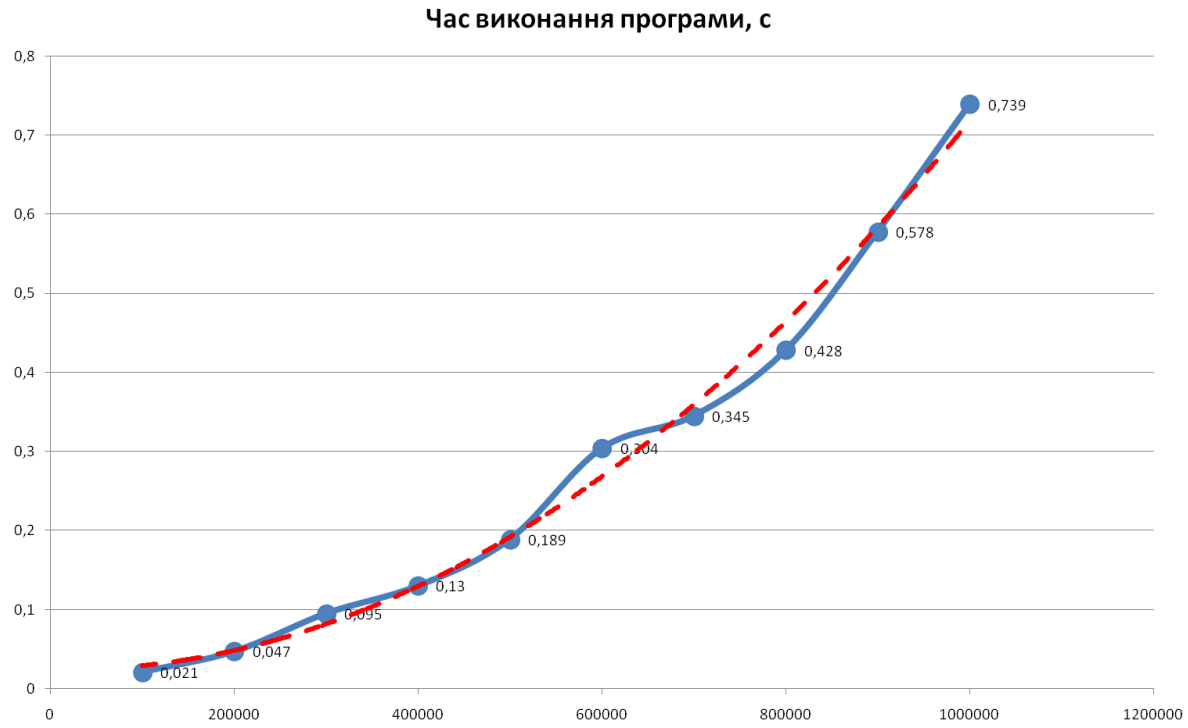
```
    s->a = (Data*)malloc(s->size * sizeof(Data));
}

void stack_print(struct Stack* s)
{
    cout << "\n----------Стек----------" << endl;
    if (s->n == 0) cout << "Стек пустий!";
    for (int i = 0; i < s->n; i++)
        cout << s->a[i] << " ";
    cout << endl;
}

void stack_push(struct Stack* s, Data data)
{
    // якщо стек повний, потрібно виділити пам'ять на 10 елементів
    if (s->n == s->size) {
        s->size += N;
        s->a = (Data*)realloc(s->a, s->size * sizeof(Data));
    }
    s->a[s->n] = data;
    s->n++;
}

void stack_pop(struct Stack* s)
{
    cout << "\n---Видалені елементи----" << endl;
    for (int i = 0; i < 3; i++)
    {
        if (s->n == 0) break;
        cout << s->a[s->n - 1] << " ";
        s->n--;
    }
    cout << endl;
    stack_print(s);
}

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    int num;
```

```
    struct Stack st;
    stack_init(&st);

    cout << "\t * -------- * Комп'ютерний практикум 3 * -------- *\n";
    cout << "\t\t\t Бригада 10. Варіант 5  \n";
    cout << "\t Реалізація АТД Стеку за допомогою динамічних масивів\n\n";

    cout << "Уведіть кількість елементів стеку: ";
    cin >> num;

    for (int i = 0; i < num; i++)
        stack_push(&st, rand() % 100);

    stack_print(&st);
    while (st.n != 0)
        stack_pop(&st);

    cout << endl;
    return 0;
}
```

Перевірка обчислювальної складності програми:



Діагр. 1. Діаграма часу виконання програми (реалізація статичного масиву)



Діагр. 2. Діаграма часу виконання програми (реалізація динамічного масиву)

Червоний пунктирний графік – теоретична складність.
Синій графік – реальна складність.

На великих значеннях > 200 000 елементів стеку, алгоритм зі статичними масивами видає помилку. Через це для оцінки складності роботи програми було взято діапазон вимірювання часу роботи програми від 20 000 до 200 000 елементів.

Скріншоти роботи програми:

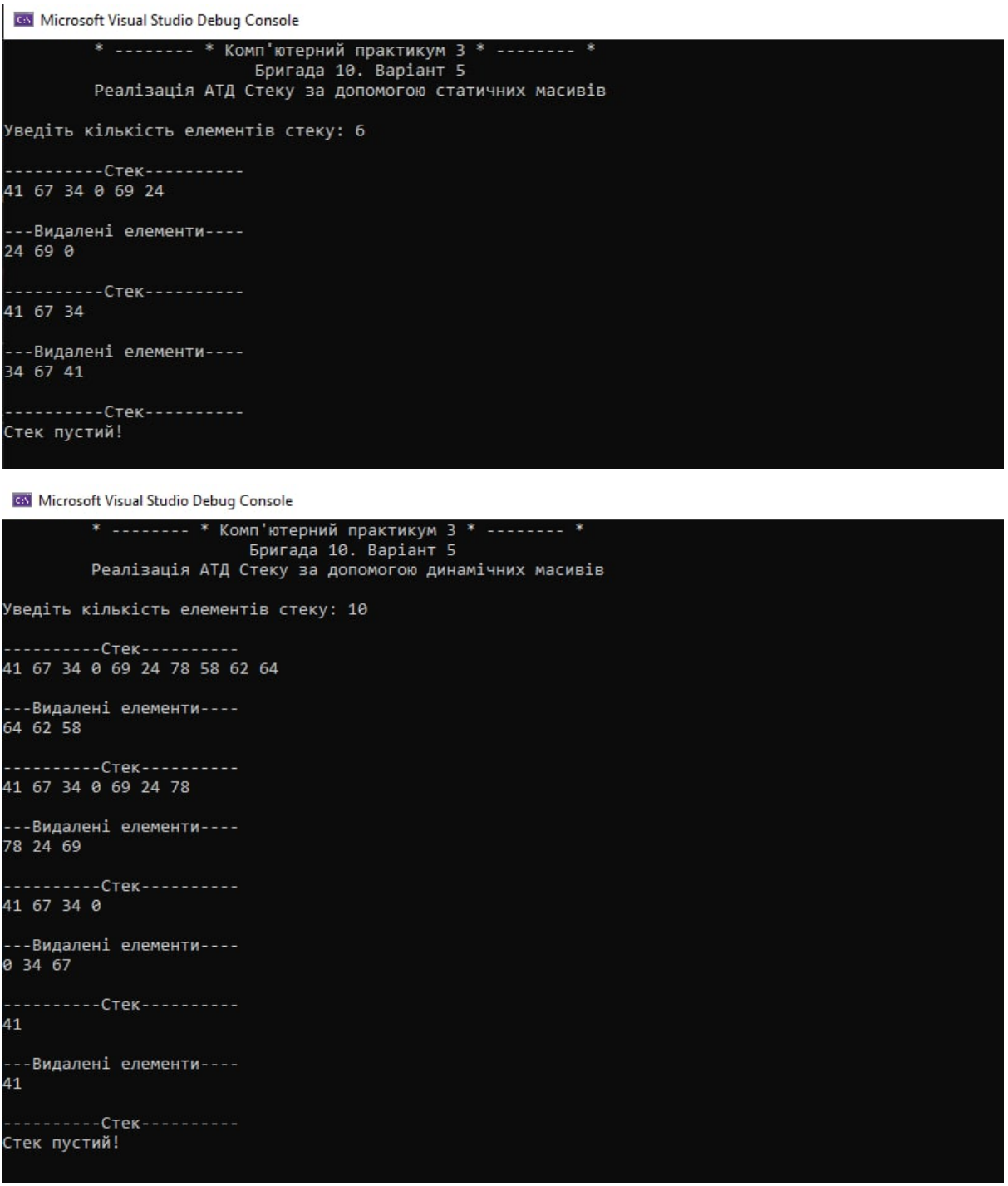


Рис. 3. Скріншоти виконання програм

Перевірка правильності програми:

Вхідні дані	Результат	Призначення тесту
num		
0	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
-1	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
1.7	Відомий	Перевірка реакції на некоректні вхідні дані (дробова частина буде відкинута, результат буде -> 1)
b	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
*	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)

Табл. 2. Таблиця тестування програми

Висновок

Під час виконання лабораторної роботи ми ознайомились з різними видами абстрактних типів даних. Для виконання варіанту завдання ми обрали стек та реалізували його двома варінтами – статичним та динамічним масивами. Були виконані перевірки обчислювальної складності та правильності виконання програми.

Відповіді на контрольні запитання:

1. Поясніть різницю між термінами «тип даних», «абстрактний тип даних» «структура даних». Для чого вони застосовується?

Тип даних — характеристика, яку явно чи неявно надано об'єкту (змінній, функції, полю запису, константі, масиву тощо). Тип даних визначає множину припустимих значень, формат їхнього збереження, розмір виділеної пам'яті та набір операцій, які можна робити над даними.

Абстрактний тип даних — це математична модель для типів даних, де тип даних визначається поведінкою (семантикою) з точки зору користувача даних, а саме в термінах можливих значень, можливих операцій над даними цього типу і поведінки цих операцій. Вся внутрішня структура такого типу захована від розробника програмного забезпечення — в цьому і полягає суть абстракції.

Структура даних — це спосіб організації даних в комп'ютерах. Часто разом зі структурою даних пов'язується і специфічний перелік операцій, що можуть бути виконаними над даними, організованими в таку структуру.

Усі ці три поняття відрізняються один від одних, вказують на різне походження та описують різні способи організації інформації.

Тип даних

Структура даних

Абстрактний вид даних

2. Проаналізуйте АТД «зв'язний лінійний список»; перерахуйте, які різновиди списків бувають.

Зв'язний список - це структура даних, в якій елементи лінійно впорядковані, але порядок визначається не номерами елементів (як в масивах), а вказівниками, що входять у склад елементів списку та вказують на наступний елемент.

Однозв'язні списки:

- Стек
- Черга
- Однозв'язний лінійний список
- Однозв'язний циклічний список

Двозв'язні списки:

- Двозв'язний лінійний список
- Двозв'язний циклічний список

3. Проаналізуйте АТД «стек»; перерахуйте, які операції можна виконувати з цим АТД.

Стек — різновид лінійного списку, структура даних, яка працює за принципом «останнім прийшов — першим пішов» (LIFO, англ. last in, first out). Всі операції в стеку можна проводити тільки з одним елементом, який міститься на верхівці стека та був уведений в стек останнім.

Опис операцій зі стеком

push («заштовхнути елемент»): елемент додається в стек та розміщується в його верхівці. Розмір стека збільшується на одиницю. При перевищенні граничної величини розміру стека, відбувається переповнення стека.

pop («виштовхнути елемент»): повертає елемент з верхівки стека. При цьому він видаляється зі стека і його місце у верхівці стека займає наступний за ним відповідно до правила LIFO, а розмір стека зменшується на одиницю. При намаганні «виштовхнути» елемент зі вже пустого стека, відбувається ситуація «незаповненість» стека

Кожна з цих операцій зі стеком виконується за фіксований час O(1) і не залежить від розміру стека.

4. Алоритм вставки елемента до стеку

1. Виділити пам'ять для нового елемента стеку
2. Ввести дані до нового елемента
3. Зв'язати новий елемент із вершиною
4. Встановити вершину стеку на новостворений елемент

5. Алгоритм видалення елемента зі стеку

1. Створити копію покажчика на вершину стеку
2. Перемістити покажчик на вершину стеку на наступний елемент
3. Звільнити пам’ять із-під колишньої вершини стеку
4. Для очищення всього стеку слід повторювати кроки 1-3 доти, доки покажчик head(вершина стеку) не дорівнюватиме null.

6. Проаналізуйте АТД «черга»; перерахуйте, які операції можна виконувати з цим АТД.

Черга — це однозв'язний лінійний список, в якому компоненти додаються в кінець списку, а видаляються з вершини, тобто з початку списку.

Опис операцій із стеком

enqueue ("поставити в чергу") – Операція додавання елемента в "хвіст" черги. При цьому довжина черги збільшується на одиницю. Якщо відбувається намагання додати елемент у вже заповнену чергу, відбувається її переповнення.

dequeue ("отримання з черги") – Операція, яка повертає елемент з голови та видаляє його з черги, таким чином встановлюючи голову на наступний за видаленим елемент та зменшуючи довжину на одиницю. При намаганні видалити елемент з пустої черги, виникає ситуація "незаповненість".

7. Алгоритм вставки елемента до порожньої черги

1. Виділити пам’ять для нового елемента черги
2. Ввести дані до нового елемента
3. Вважати новий елемент останнім у черзі
4. Якщо черга порожня, то вважати новий елемент вершиною черги

8. Проаналізуйте АТД «однозв’язний лінійний список»; перерахуйте, які операції можна виконувати з цим АТД.

Однозв'язний лінійний список — це список, в якому попередній компонент посилається на наступний.

Операція *додавання* елемента в кінець списку виконується за алгоритмом додавання елемента до черги, а на початок списку — за алгоритмом додавання елемента до стеку.

Операція *видалення* елемента з початку списку здійснюється за алгоритмом видалення елемента зі стеку або з черги.