Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії
Кафедра ПЗ

Лабораторна робота №3, варіант № 19
з дисципліни «Основи програмування»

Виконала: ст. 1ПІ-25Б Семенов В.О. .
Перевірив: доцент Решетнік О.О.

Вінниця – 2025

Тема: Робота з одномірними масивами

Мета: Ознайомитись із простими операціями над масивами: ініціалізація масивів, виконання обчислень з масивами, сортування та пошук.

Завдання 19. Дано дійсний масив X[n]. Знайти елемент масиву, значення якого найбільш близьке до якого-небудь цілого числа.

Декомпозиція

1. Ініціалізація масиву

Створення динамічного масиву для зберігання дійсних чисел.

2. Введення даних

Користувач вводить кількість елементів і самі числа, які зберігаються в масиві.

3. Пошук елемента, найближчого до цілого

Обчислення відстані кожного числа до найближчого цілого (|x - round(x)|).

Вибір числа з мінімальною відстанню.

4. Вивід результату

Виведення знайденого числа та його відстані до найближчого цілого.

5. Звільнення пам’яті

Очищення динамічного масиву для уникнення витоків пам’яті.

Код програми показано на лістингу 1 а блок-схему на рисунку 1.

Лістинг 1 – Програма для знаходження найбільш близького елементу масиву до цілого числа

```
#include <cstdio>
#include <math.h>

void inputArray(DynamicArray* array)
{
    size_t n;
    printf("Enter number of elements: ");
    scanf("%zu", &n);

    for(size_t i = 0; i < n; i++)
    {
```

Продовження лістингу 1

```
        double num;
        printf("array[%zu] = ", i);
        scanf("%lf", &num);
        push_back(array, &num);
    }
}

double findClosestToInteger(const DynamicArray* array, double*
outMinDiff)
{
    double closest = *(double*)at((DynamicArray*)array, 0);
    double minDiff = fabs(closest - round(closest));

    for(size_t i = 1; i < getSize(array); i++)
    {
        double current = *(double*)at((DynamicArray*)array, i);
        double diff = fabs(current - round(current));
        if(diff < minDiff)
        {
            minDiff = diff;
            closest = current;
        }
    }

    if(outMinDiff)
    {
        *outMinDiff = minDiff;
    }


    return closest;
}

int main(void) {
    DynamicArray array;
    initDynamicArray(&array, sizeof(double));

    inputArray(&array);

    double minDiff;
    double closest = findClosestToInteger(&array, &minDiff);

    printf("\nElement closest to an integer: %.4f\n", closest);
    printf("Distance to nearest integer: %.4f\n", minDiff);

    DelDynamicArray(&array);

    return 0;
}
```

Приклади роботи програми можна побачити на рисунках 1, 2 та 3

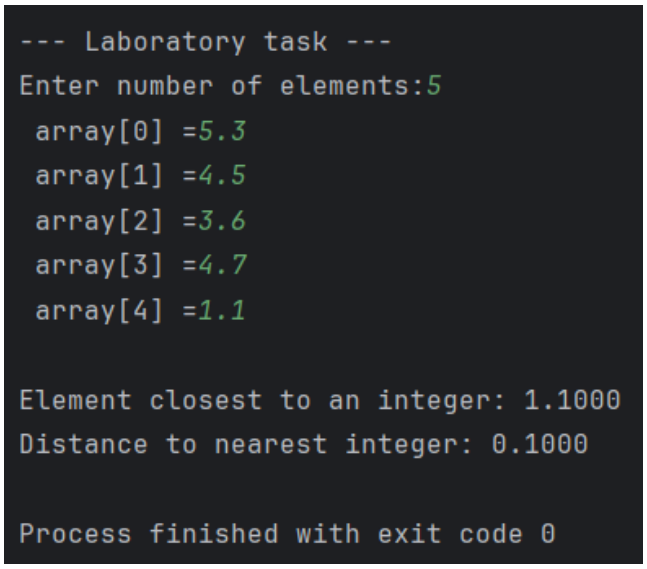

Рисунок 1 – приклад роботи програми

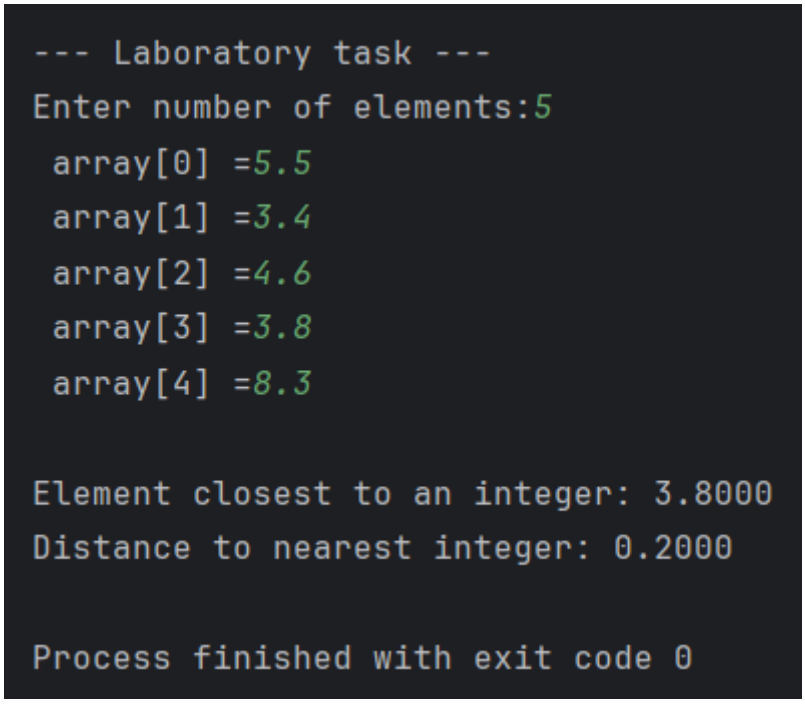

Рисунок 2 – приклад роботи програми

```
--- Laboratory task ---
Enter number of elements:3
 array[0] =3
 array[1] =4.1
 array[2] =9.9

Element closest to an integer: 3.0000
Distance to nearest integer: 0.0000

Process finished with exit code 0
```

Рисунок 3 – приклад роботи програми

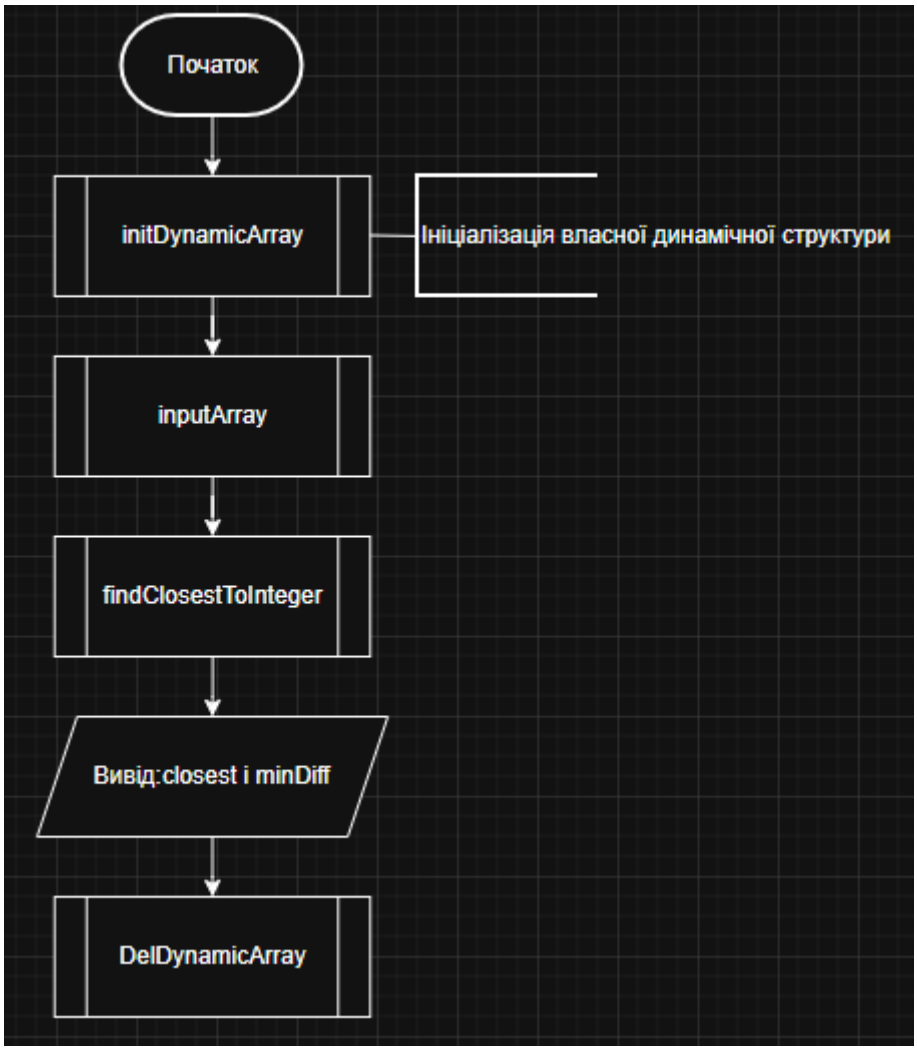

Рисунок 4 – Блок-схема main

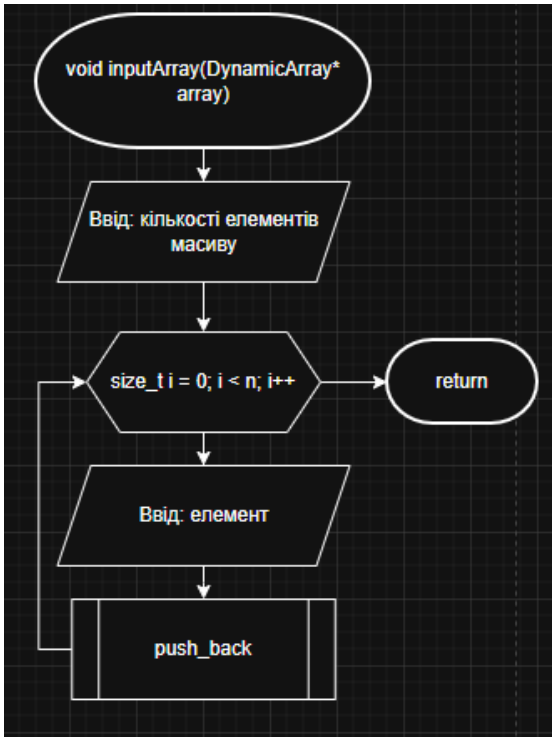

Рисунок 5 – Блок-схема inputArray

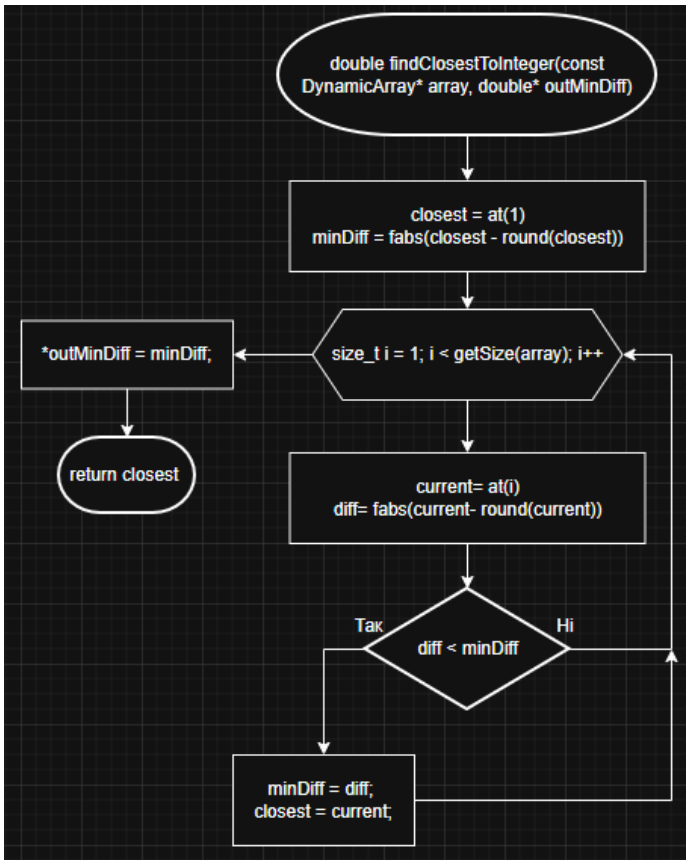

Рисунок 6 – Блок-схема findClosestToInteger

Контрольні запитання:

19. Що таке операція взяття адреси?

Операція & повертає адресу змінної в пам’яті.

Приклад:

```
int x = 5;
int* p = &x; // p містить адресу x
```

20. Що таке константний покажчик? Приклад.

Константний покажчик – покажчик, який не можна переназначити на іншу адресу після ініціалізації.

Приклад:

```
int a = 10, b = 20;
int* const p = &a; // p завжди вказує на a
*p = 15;           // можна змінювати значення a через p
p = &b;          // помилка, змінити адресу не можна
```

21. Які арифметичні дії можна проводити над покажчиками і для чого?

Додавання/віднімання цілого числа: p + 1, p - 2 – пересування по елементах масиву.

Віднімання покажчиків: p2 - p1 – кількість елементів між двома адресами.

Порівняння: p1 < p2, p1 == p2.

Використовуються для переміщення по масиву, ітерацій та обчислення відстані між елементами.

Висновок:

У лабораторній роботі було ознайомлено з простими операціями над масивами: виконано ініціалізацію масивів, обчислення з їх елементами та пошук значень за заданим критерієм. Отримані навички закріплюють базові знання роботи з масивами у мові С.

Додаток А – Реалізація аналогу вектора по ініціативі студента

Лістинг А - реалізація аналогу вектора і базовий приклад роботи

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

typedef struct
{
    void* array;
    size_t size;
    size_t capacity;
    size_t elemSize;
} DynamicArray;

void initDynamicArray(DynamicArray* array, size_t elemSize)
{
    array->array = NULL;
    array->size = 0;
    array->capacity = 0;
    array->elemSize = elemSize;
}

void initDynamicArrayWithSize(DynamicArray* array, size_t
elemSize, size_t size)
{
    array->array = calloc(size, elemSize);
    array->size = size;
    array->capacity = size;
    array->elemSize = elemSize;
}

void DelDynamicArray(DynamicArray* array)
{
    free(array->array);
    array->array = NULL;
    array->size = 0;

  array->capacity = 0;
}

void assign(const DynamicArray* src, DynamicArray* dest)
{
    if(src == dest) return;

    DelDynamicArray(dest);
    dest->size = src->size;
    dest->capacity = src->capacity;
    dest->elemSize = src->elemSize;
    dest->array = malloc(src->capacity * src->elemSize);
```

Продовження лістингу А

```
    memcpy(dest->array, src->array, src->size * src->elemSize);
}

void* at(DynamicArray* array, size_t index)
{
    if(index >= array->size)
    {
        fprintf(stderr, "Index out of range\n");
        exit(EXIT_FAILURE);
    }
    return (char*)array->array + index * array->elemSize;
}

void reserve(DynamicArray* array, size_t newCapacity)
{
    if(newCapacity <= array->capacity) return;

    void* newArray = calloc(newCapacity, array->elemSize);
    if(array->array)
    {
        memcpy(newArray, array->array, array->size * array-
>elemSize);
        free(array->array);
    }
    array->array = newArray;
    array->capacity = newCapacity;
}
void setSize(DynamicArray* array, size_t newSize)
{
    if (newSize > array->capacity)
    {
        reserve(array, newSize);
    }

    if(newSize < array->size)
    {
        memset((char*)array->array + newSize * array->elemSize,
0, (array->size - newSize) * array->elemSize);
    }

    array->size = newSize;
}

size_t getSize(const DynamicArray* array)
{
    return array->size;
}

size_t getCapacity(const DynamicArray* array)
```

Продовження лістингу А

```
{
    return array->capacity;
}

void clear(DynamicArray* array)
{
    if(array->array)
    {
        memset(array->array, 0, array->size * array->elemSize);
    }
    array->size = 0;
}

void push_back(DynamicArray* array, const void* element)
{
    if(array->size >= array->capacity)
    {
        reserve(array, array->capacity == 0 ? 1 : array-
>capacity*2);
    }

    memcpy((char *)array->array + array->size * array->elemSize,
element, array->elemSize);
    array->size++;
}

void pop_back(DynamicArray* array)
{
    if (array->size == 0)
    {
        fprintf(stderr, "Array is empty\n");
        exit(EXIT_FAILURE);
    }
    array->size--;
    memset((char*)array->array + array->size * array->elemSize,
0, array->elemSize);
}

void* back(DynamicArray* array)
{
    if(array->size == 0)
    {
        fprintf(stderr, "Array is empty\n");
        exit(EXIT_FAILURE);
    }
    return at(array, array->size - 1);
}

void shringToFit(DynamicArray* array)
{
```

Продовження лістингу А

```
    if(array->size < array->capacity)
    {
        void* newArray = malloc(array->size * array->elemSize);
        memcpy(newArray, array->array, array->size * array-
>elemSize);
        free(array->array);
        array->array = newArray;
        array->capacity = array->size;
    }
}

int main(void) {
    DynamicArray arr;
    initDynamicArray(&arr, sizeof(int));
    printf("Initial size: %zu, capacity: %zu\n", getSize(&arr),
getCapacity(&arr));

    // push_back
    for(int i = 1; i <= 5; i++) {
        push_back(&arr, &i);
        printf("After push_back %d: size = %zu, capacity =
%zu\n", i, getSize(&arr), getCapacity(&arr));
    }

    // access elements with at
    printf("Elements using at(): ");
    for(size_t i = 0; i < getSize(&arr); i++) {
        printf("%d ", *(int*)at(&arr, i));
    }
    printf("\n");

    // back
    printf("Last element (back): %d\n", *(int*)back(&arr));

    // pop_back
    pop_back(&arr);
    printf("After pop_back: size = %zu, last element = %d\n",
getSize(&arr), *(int*)back(&arr));

    // setSize larger
    setSize(&arr, 8);
    printf("After setSize(8): size = %zu, capacity = %zu\n",
getSize(&arr), getCapacity(&arr));

    // setSize smaller
    setSize(&arr, 3);
    printf("After setSize(3): size = %zu, capacity = %zu\n",
getSize(&arr), getCapacity(&arr));

    // clear
```

Продовження лістингу А

```
    clear(&arr);
    printf("After clear: size = %zu, capacity = %zu\n",
getSize(&arr), getCapacity(&arr));

    // reserve
    reserve(&arr, 10);

    printf("After reserve(10): size = %zu, capacity = %zu\n",
getSize(&arr), getCapacity(&arr));

    // push_back again
    int val = 42;
    push_back(&arr, &val);
    printf("After push_back 42: size = %zu, last element =
%d\n", getSize(&arr), *(int*)back(&arr));

    // shrinkToFit
    shringToFit(&arr);
    printf("After shrinkToFit: size = %zu, capacity = %zu\n",
getSize(&arr), getCapacity(&arr));

    // assign
    DynamicArray arr2;
    initDynamicArray(&arr2, sizeof(int));
    assign(&arr, &arr2);
    printf("After assign: arr2 size = %zu, last element = %d\n",
getSize(&arr2), *(int*)back(&arr2));

    // cleanup
    DelDynamicArray(&arr);
    DelDynamicArray(&arr2);

    return 0;
}
```

```
Initial size: 0, capacity: 0
After push_back 1: size = 1, capacity = 1
After push_back 2: size = 2, capacity = 2
After push_back 3: size = 3, capacity = 4
After push_back 4: size = 4, capacity = 4
After push_back 5: size = 5, capacity = 8
Elements using at(): 1 2 3 4 5
Last element (back): 5
After pop_back: size = 4, last element = 4
After setSize(8): size = 8, capacity = 8
After setSize(3): size = 3, capacity = 8
After clear: size = 0, capacity = 8
After reserve(10): size = 0, capacity = 10
After push_back 42: size = 1, last element = 42
After shrinkToFit: size = 1, capacity = 1
After assign: arr2 size = 1, last element = 42
```

Рисунок А.1 – приклад роботи аналогу вектора

Блок схеми:

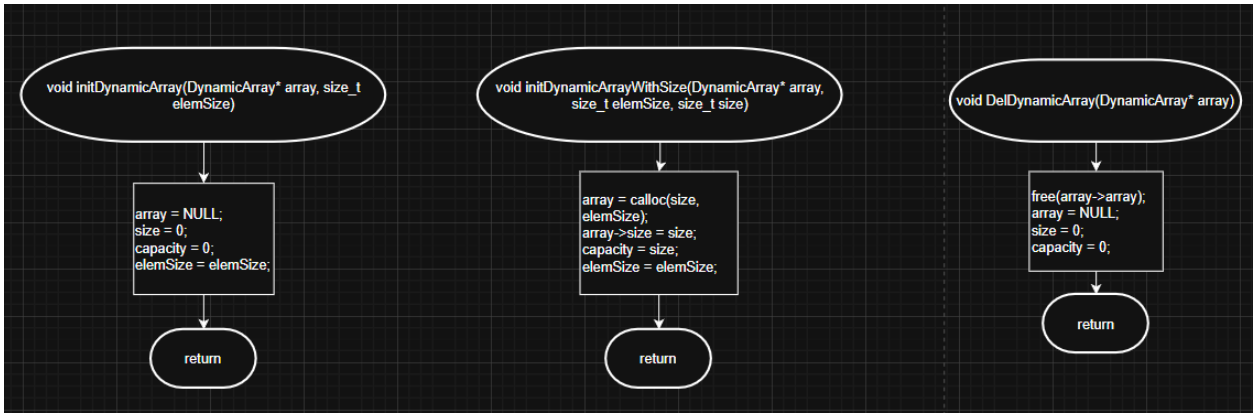

Рисунок А.2 – функції для ініціаліхації та видалення пам’яті

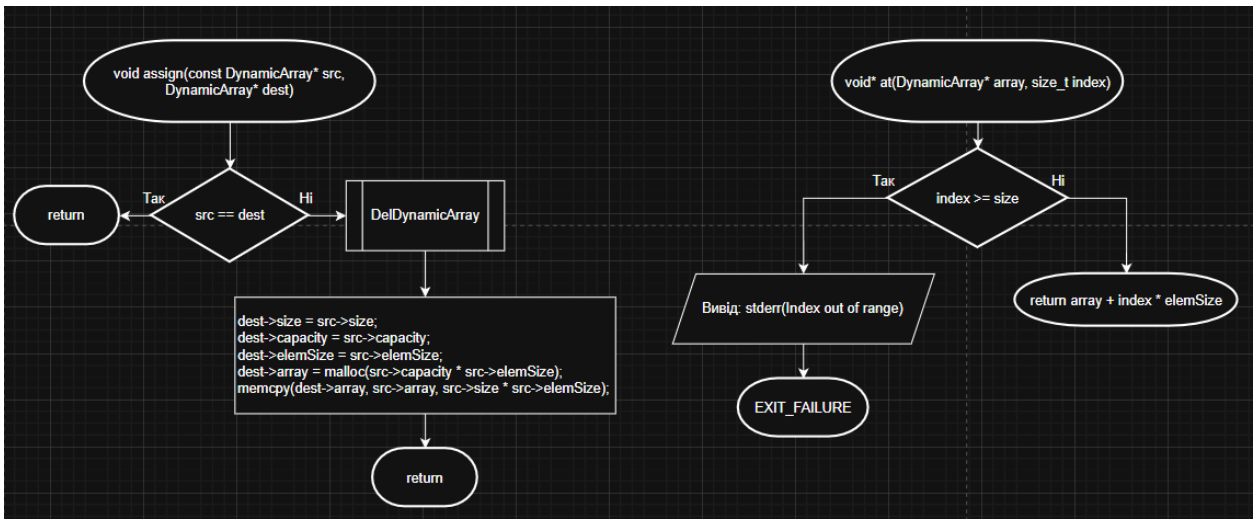

Рисунок А.3 – аналоги операторів = та []

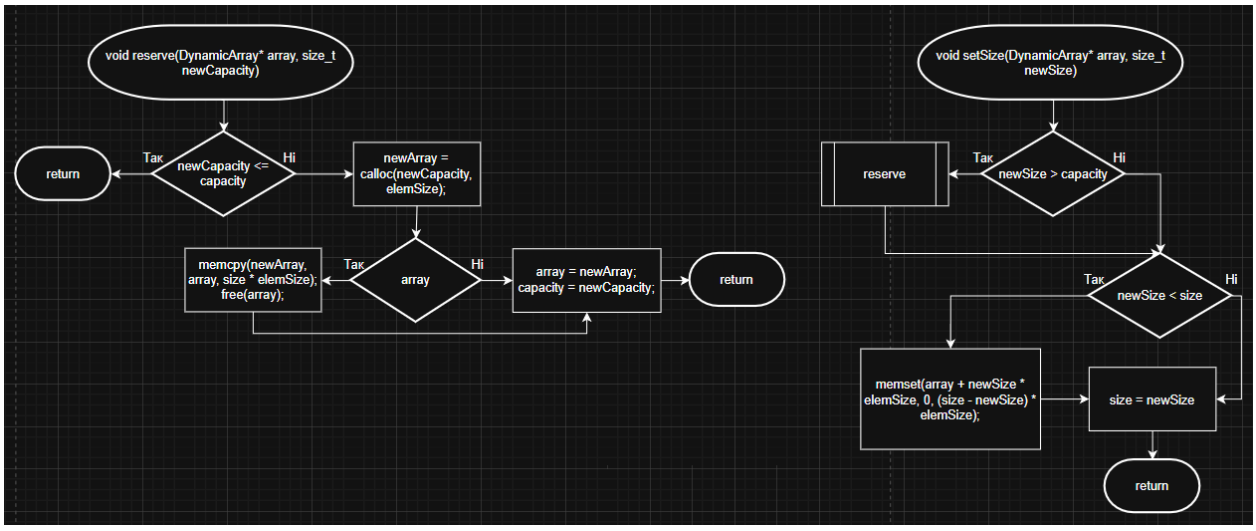

Рисунок А.4 – функції для зміни розміру і місткості

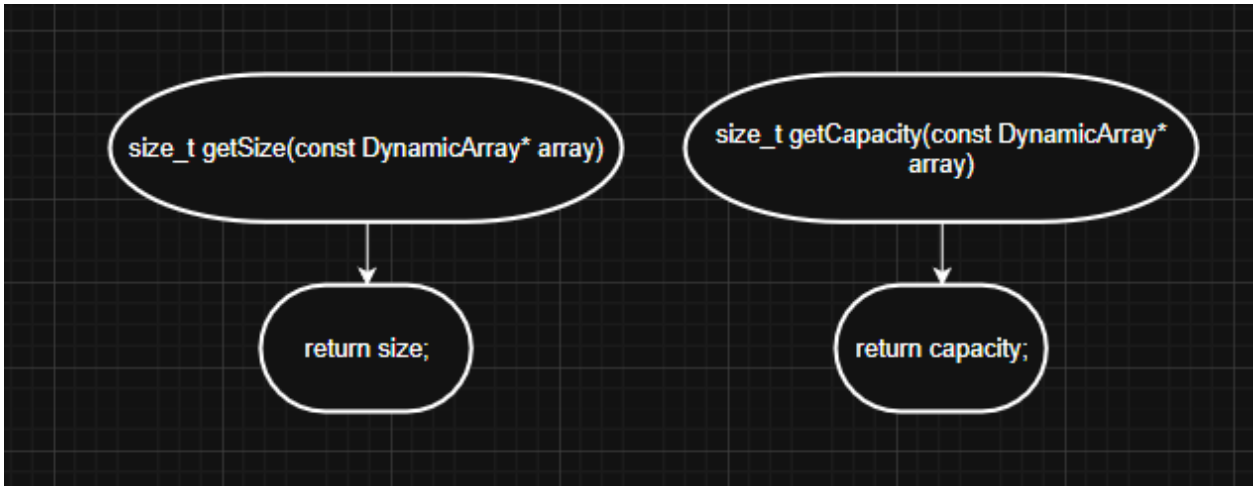

Рисунок А.5 – функції для отримання розміру і місткості

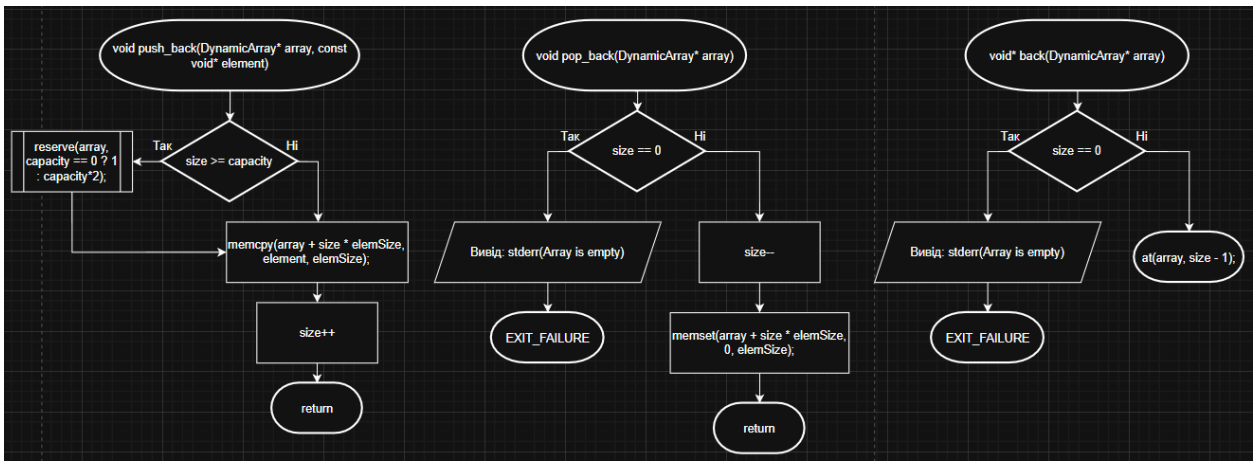

Рисунок А.6 – функції аналоги push_back, pop_back, back

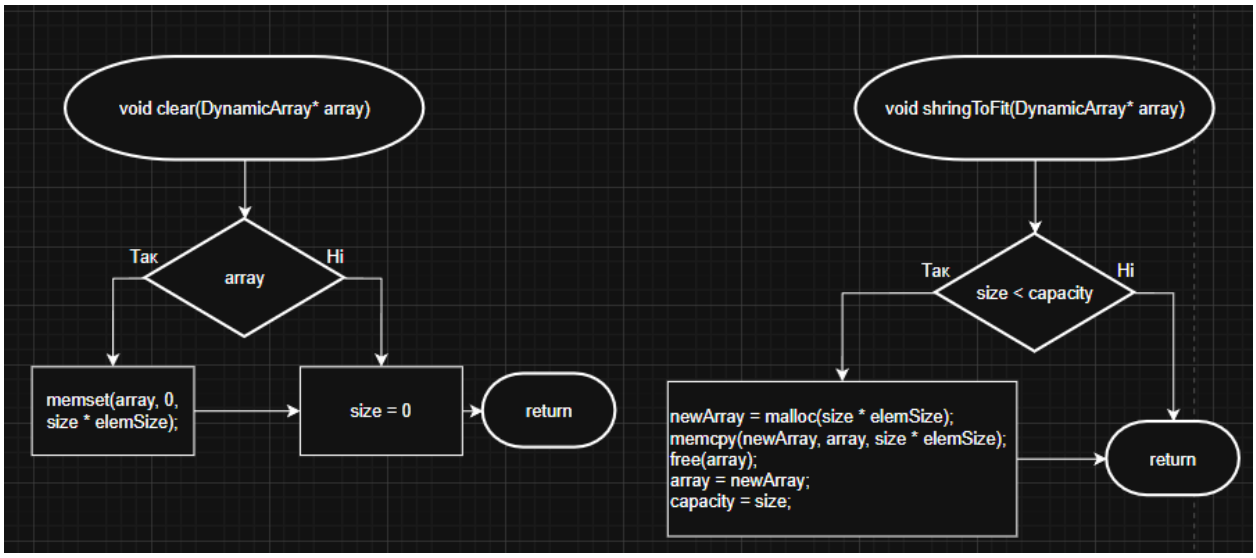

Рисунок А.7 –функції clear, shringToFit