

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 № 7 з
дисципліни «Алгоритми та структури
даних-1. Основи алгоритмізації»
«Дослідження лінійного пошуку в
послідовностях»
Варіант 2

Виконав студент ПІ-12, Басараб Олег Андрійович
(шифр, прізвище, ім'я, по батькові)

Перевірів _____
(прізвище, ім'я, по батькові)

Лабораторна робота № 7 “Дослідження лінійного пошуку в послідовностях”

Варіант 2

Мета – дослідити методи послідовного пошуку у впорядкованих і неупорядкованих послідовностях та набути практичних навичок їх використання під час складання програмних специфікацій.

Задача 2. Розробити алгоритм та написати програму, яка складається з наступних дій:

1. Опису трьох змінних індексованого типу з 10 символьних значень.
2. Ініціювання двох змінних виразами згідно з варіантом ($arr1[i] = 5 * i + 30$, $arr2[i] = 60 - 5 * i$).
3. Ініціювання третьої змінної рівними значеннями двох попередніх змінних.
4. Обробка третьої змінної згідно з варіантом (Знайти добуток елементів, коди яких менше 40).

Розв'язування.

Постановка задачі. Результатом розв'язку є дійсна змінна `satisfyingElementsProduct`. Для його знаходження вхідні дані не потрібні. Початковими даними є цілочисельна константа `ARR_SIZE = 10`.

Побудова математичної моделі. Складемо таблицю імен змінних основної програми.

Змінна	Тип	Ім'я	Призначення
Розмір змінних індексованого типу	Цілий (константа)	ARR_SIZE	Початкове дане
Перша змінна індексованого типу	Символьний масив	arr1	Поточне дане
Друга змінна індексованого типу	Символьний масив	arr2	Поточне дане

Третя змінна індексованого типу	Символьний масив	arr3	Поточне дане
Добуток елементів, коди яких менше 40	Символьний	satisfyingElementsProduct	Результат

Складемо таблицю імен змінних підпрограми arr1Assignment для задання елементів першого символьного масиву.

Змінна	Тип	Ім'я	Призначення
Масив, елементи якого буду задаватися через підпрограму	Символьний масив	arr1	Вхідне дане
Розмір масиву arr1	Цілий (константа)	ARR_1_SIZE	Вхідне дане
Лічильник арифметичного циклу для знаходження елементів масиву arr1	Цілий	i	Поточне дане

Складемо таблицю імен змінних підпрограми arr2Assignment для задання елементів другого символьного масиву.

Змінна	Тип	Ім'я	Призначення
Масив, елементи якого буду задаватися через підпрограму	Символьний масив	arr2	Вхідне дане
Розмір масиву arr2	Цілий (константа)	ARR_2_SIZE	Вхідне дане
Лічильник арифметичного циклу для знаходження елементів масиву arr2	Цілий	i	Поточне дане

Складемо таблицю імен змінних підпрограми arr3Assignment для задання елементів третього символьного масиву.

Змінна	Тип	Ім'я	Призначення
Перший символьний масив	Символьний масив	arr1	Вхідне дане
Другий символьний масив	Символьний масив	arr2	Вхідне дане
Третій символьний масив	Символьний масив	arr3	Вхідне дане
Розмір символьних масивів	Цілий (константа)	ARR_SIZE	Вхідне дане
Лічильник арифметичного циклу для знаходження спільних елементів arr1 та arr2	Цілий	i	Поточне дане
Лічильник арифметичного циклу для знаходження спільних елементів arr1 та arr2	Цілий	j	Поточне дане
Лічильник арифметичного циклу для перебору	Цілий	k	Поточне дане

елементів масиву arr3			
Змінна для перевірки наявності елемента в масиві arr3	Булевий	elementIsAlreadyInArray3	Поточне дане
Індекс поточного елемента масиву arr3	Цілий	arr3Index	Поточне дане

Складемо таблицю імен змінних підпрограми arr3Processing для обробки елементів третього символьного масиву.

Змінна	Тип	Ім'я	Призначення
Масив, елементи якого будуть оброблятися	Символьний масив	arr3	Вхідне дане
Розмір масиву arr3	Цілий (константа)	ARR_3_SIZE	Вхідне дане
Лічильник арифметичного циклу для перебору елементів масиву arr3	Цілий	i	Поточне дане
Добуток елементів arr3, коди яких менше 40	Символьний	satisfyingElementsProduct	Результат

Таким чином, формулювання завдання зводиться до:

- 1. Опису підпрограми `arr1Assignment(char arr1[], const int ARR_1_SIZE)` для задання елементів першого символьного масиву.** Для обробки всіх елементів масиву використовується арифметичний цикл з лічильником i , початкове значення якого 0, умовою невиходу ($i < \text{ARR_1_SIZE}$) та кроком 1. В тілі циклу знаходимо i -ий елемент масиву `arr1` за формулою з умови завдання ($\text{arr1}[i+1] = 5 * (i + 1) + 30$). Результатом виконання підпрограми є заповнений масив `arr1`.
- 2. Опису підпрограми `arr2Assignment(char arr2[], const int ARR_2_SIZE)` для задання елементів другого символьного масиву.** Для обробки всіх елементів масиву використовується арифметичний цикл з лічильником i , початкове значення якого 0, умовою невиходу ($i < \text{ARR_1_SIZE}$) та кроком 1. В тілі циклу знаходимо i -ий елемент масиву `arr2` за формулою з умови завдання ($\text{arr2}[i+1] = 60 - 5 * (i + 1)$). Результатом виконання підпрограми є заповнений масив `arr2`.
- 3. Опису підпрограми `arr3Assignment(char arr1[], char arr2[], char arr3[], const int ARR_SIZE)` для задання елементів третього символьного масиву.** Задаємо початкове значення змінної `arr3Index = 0`. Для знаходження спільних елементів масивів `arr1` та `arr2` використовується система з двох арифметичних циклів, один з яких вкладений. Зовнішній цикл з лічильником i , початкове значення якого 0, умовою невиходу ($i < \text{ARR_SIZE}$) та кроком 1. Внутрішній цикл з лічильником j , початкове значення якого 0, умовою невиходу ($j < \text{ARR_SIZE}$) та кроком 1. У внутрішньому циклі по j перевіряємо, чи `arr1[i + 1]` дорівнює `arr2[j + 1]`. Якщо так, виконуємо наступні дії: `elementIsAlreadyInArray3 = false`; у циклі з лічильником k , початкове значення якого 0, умовою невиходу ($k < \text{arr3Index}$) та кроком 1 перевіряємо, чи `arr1[i + 1]` дорівнює `arr3[k + 1]` і, якщо так, `elementIsAlreadyInArray3 = true`, `break`; якщо виконується умова

(!elementIsAlreadyInArray3), то $\text{arr3}[\text{arr3Index} + 1] = \text{arr1}[i + 1]$ та $\text{arr3Index} += 1$.

4. Опису підпрограми `arr3Processing(char arr3[], const int ARR_3_SIZE)` для обробки елементів третього символьного масиву. Задаємо початкове значення `satisfyingElementsProduct = 1`. Для обробки всіх елементів масиву використовується арифметичний цикл з лічильником `i`, початкове значення якого 0, умовою невиходу ($i < \text{ARR_3_SIZE}$) та кроком 1. В тілі циклу, якщо $\text{arr3}[i + 1] \neq 0$ та $\text{arr3}[i + 1] < 40$, то $\text{satisfyingElementsProduct} *= \text{arr3}[i + 1]$.

5. Опису основного алгоритму. Вважаємо, що $\text{ARR_SIZE} = 10$ відповідно до умови. Задаємо три символьні масиви за допомогою підпрограм `arr1Assignment`, `arr2Assignment`, `arr3Assignment`. Ініціалізуємо елементи масивів нулями. Знаходження добутку елементів третього символьного масиву, коди яких менші 40, за допомогою підпрограми `arr3Processing`.

Для підпрограм `arr1Assignment`, `arr2Assignment`, `arr3Assignment` вважаємо, що масиви `arr1`, `arr2`, `arr3` передаються до них за посиланням.

Розіб'ємо алгоритм роботи основної програми на кроки:

Крок 1. Визначимо основні дії.

Крок 2. Деталізуємо дію задання елементів масивів `arr1`, `arr2`, `arr3` за допомогою власних підпрограм.

Крок 3. Деталізуємо дію обробки елементів масиву `arr3` відповідно до умови.

Розіб'ємо алгоритм роботи підпрограми `arr1Assignment(char arr1[], const int ARR_1_SIZE)` для задання елементів першого символьного масиву:

Крок 1. Визначимо основні дії.

Крок 2. Деталізуємо дію знаходження `arr1[i]` в тілі арифметичного циклу.

Розіб'ємо алгоритм роботи підпрограми `arr2Assignment(char arr2[], const int ARR_2_SIZE)` для задання елементів другого символьного масиву:

Крок 1. Визначимо основні дії.

Крок 2. Деталізуємо дію знаходження `arr2[i]` в тілі арифметичного циклу.

Розіб'ємо алгоритм роботи підпрограми `arr3Assignment(char arr1[], char arr2[], char arr3[], const int ARR_SIZE)` для задання елементів третього символьного масиву:

Крок 1. Визначимо основні дії.

Крок 2. Деталізуємо дію знаходження спільних елементів масивів `arr1` та `arr2` за допомогою системи з двох циклів, один з яких вкладений.

Крок 3. Деталізуємо дію перевірки наявності спільного елементу в масиві `arr3` (якщо він не присутній, то додаємо його до масиву `arr3`).

Розіб'ємо алгоритм роботи підпрограми `arr3Processing(char arr3[], const int ARR_3_SIZE)` для обробки елементів третього символьного масиву:

Крок 1. Визначимо основні дії.

Крок 2. Деталізуємо дію обробки елементів масиву `arr3` відповідно до умови.

Програмні специфікації запишемо у псевдокоді та в графічній формі в блок-схемі алгоритму.

Псевдокод.

Основна програма:

Крок 1

початок

задання елементів масивів arr1,
arr2, arr3 за допомогою власних
підпрограм

обробка елементів масиву arr3
відповідно до умови
вивід satisfyingElementsProduct

кінець

Крок 2

початок

arr1Assignment(arr1, ARR_SIZE)
arr2Assignment(arr2, ARR_SIZE)
arr3Assignment(arr1, arr2, arr3,
ARR_SIZE)

обробка елементів масиву arr3
відповідно до умови
вивід satisfyingElementsProduct

кінець

Крок 3

початок

arr1Assignment(arr1, ARR_SIZE)
arr2Assignment(arr2, ARR_SIZE)
arr3Assignment(arr1, arr2, arr3, ARR_SIZE)
satisfyingElementsProduct = arr3Processing(arr3, ARR_SIZE)
вивід satisfyingElementsProduct

кінець

Підпрограми arr1Assignment(char arr1[], const int ARR_1_SIZE):

Крок 1

початок

знаходження arr1[i] в тілі

арифметичного циклу

кінець

Крок 2

початок

повторити

для i від 0 до ARR_1_SIZE

$arr1[i + 1] = 5 * (i + 1) + 30$

все повторити

кінець

Підпрограми arr2Assignment(char arr2[], const int ARR_2_SIZE):

Крок 1

початок

знаходження arr2[i] в тілі

арифметичного циклу

кінець

Крок 2

початок

повторити

для i від 0 до `ARR_2_SIZE`

$\text{arr2}[i + 1] = 60 - 5 * (i + 1)$

все повторити

кінець

Підпрограма `arr3Assignment(char arr1[], char arr2[], char arr3[], const int ARR_SIZE)`:

Крок 1

початок

знаходження спільних елементів
масивів `arr1` та `arr2` за допомогою
системи з двох циклів, один з яких
вкладений

перевірки наявності спільного
елементу в масиві `arr3` (якщо він не
присутній, то додаємо його до
масиву `arr3`)

кінець

Крок 2

початок

arr3Index = 0

повторити

для і від 0 до ARR_SIZE

повторити

для j від 0 до ARR_SIZE

перевірки наявності спільного елементу в масиві arr3

(якщо він не присутній, то додаємо його до масиву arr3)

все потворити

все повторити

кінець

Крок 3

початок

arr3Index = 0

повторити

для і від 0 до ARR_SIZE

повторити

для j від 0 до ARR_SIZE

якщо arr1[i + 1] == arr2[j + 1]

то

elementIsAlreadyInArray3 = false

повторити

для k від 0 до arr3Index

```

        якщо arr1[i + 1] == arr3[k + 1]
            то
                elementIsAlreadyInArray3 =
                    true
                break
        все якщо
    все повторити
    якщо !elementIsAlreadyInArray3
        то
            arr3[arr3Index + 1] = arr1[i + 1]
            arr3Index += 1
        все якщо
    все якщо
все потворити
все повторити
кінець

```

Підпрограма arr3Processing(char arr3[], const int ARR_3_SIZE):

Крок 1

початок

обробка елементів масиву arr3

відповідно до умови

повернути satisfyingElementsProduct

кінець

Крок 2

початок

satisfyingElementsProduct = 1

повторити

для i від 0 до ARR_3_SIZE

якщо ($\text{arr3}[i + 1] \neq 0 \ \&\& \ \text{arr3}[i + 1] < 40$)

то

satisfyingElementsProduct *= $\text{arr3}[i + 1]$

все якщо

все повторити

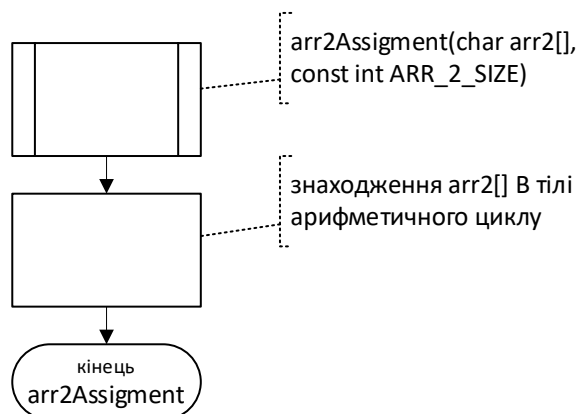
повернути satisfyingElementsProduct

кінець

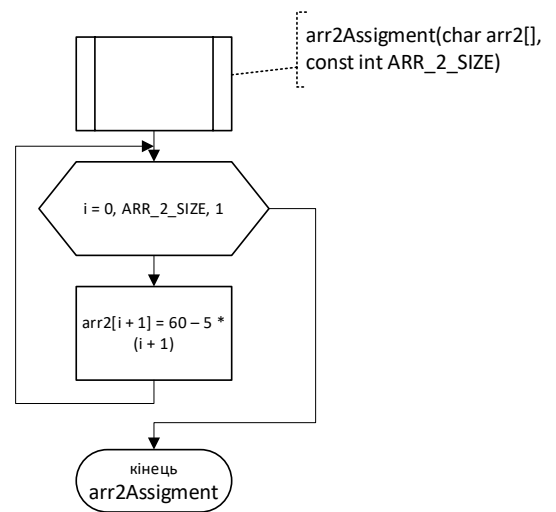
Блок-схема алгоритму.

Підпрограми arr2Assignment(char arr2[], const int ARR_2_SIZE):

Крок 1

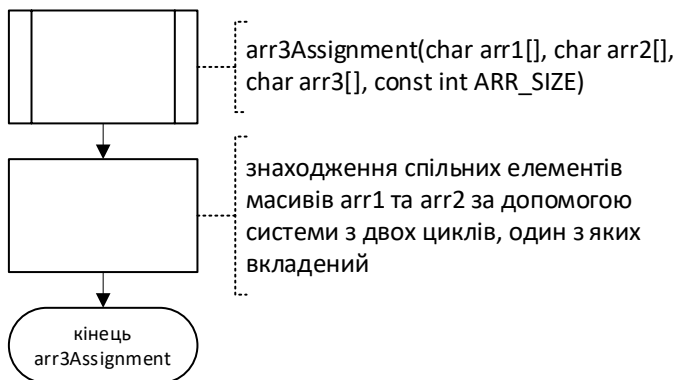


Крок 2

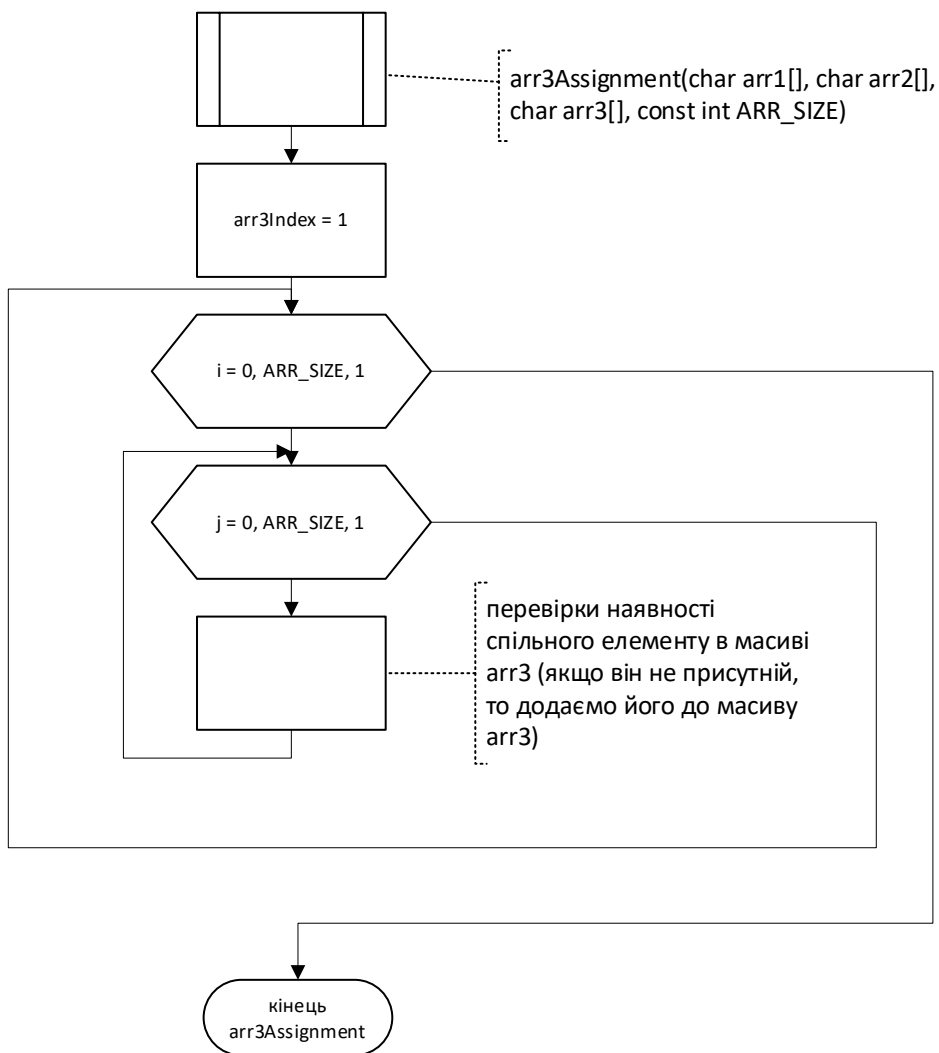


Підпрограма `arr3Assignment(char arr1[], char arr2[], char arr3[], const int ARR_SIZE):`

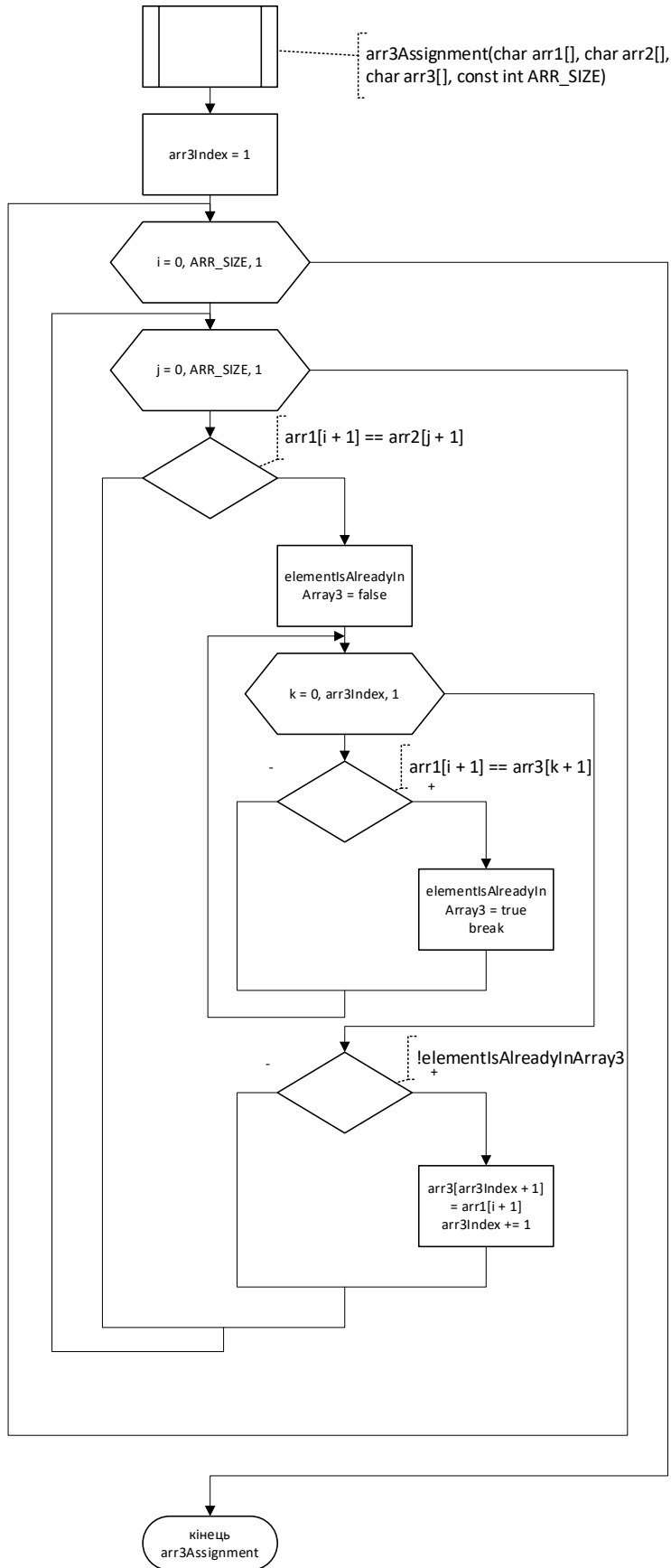
Крок 1



Крок 2

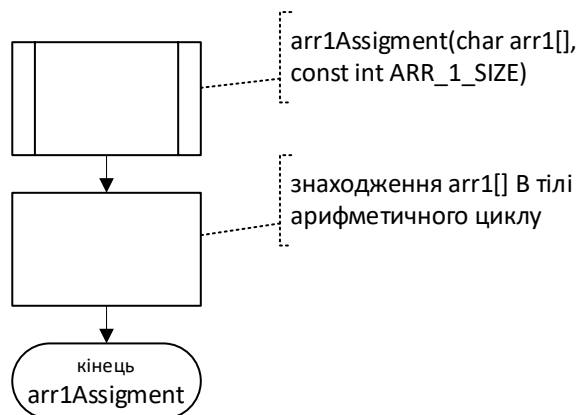


Крок 3

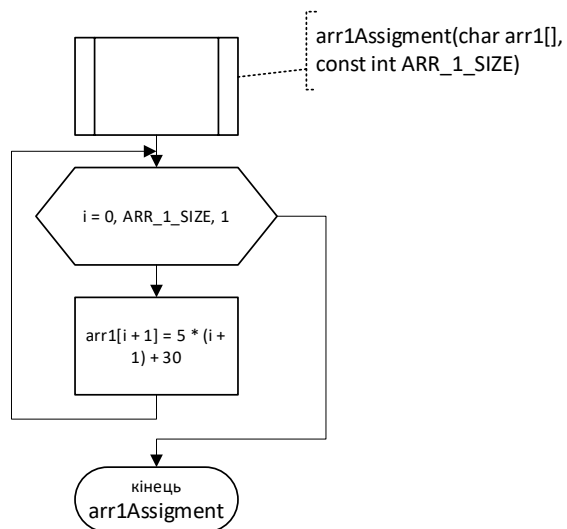


Підпрограми arr1Assignment(char arr1[], const int ARR_1_SIZE):

Крок 1

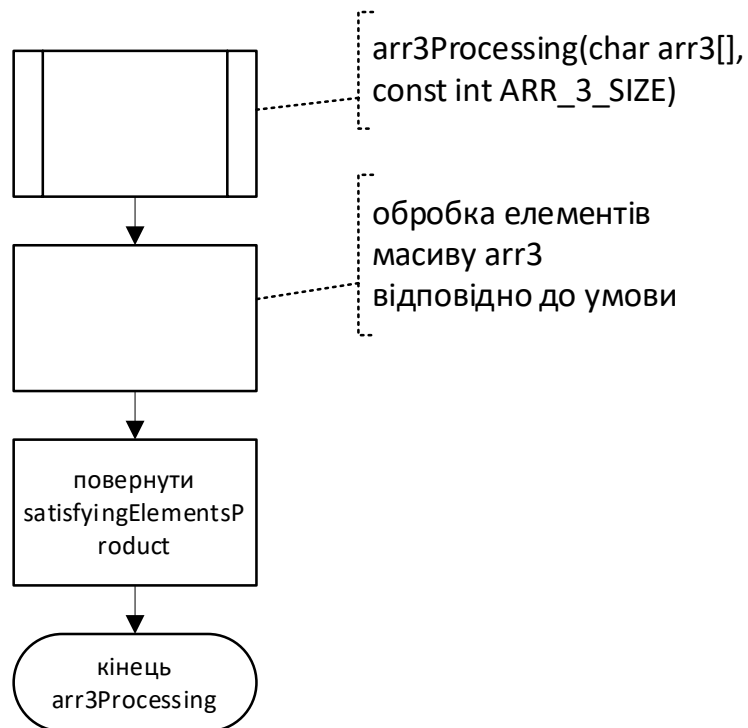


Крок 2

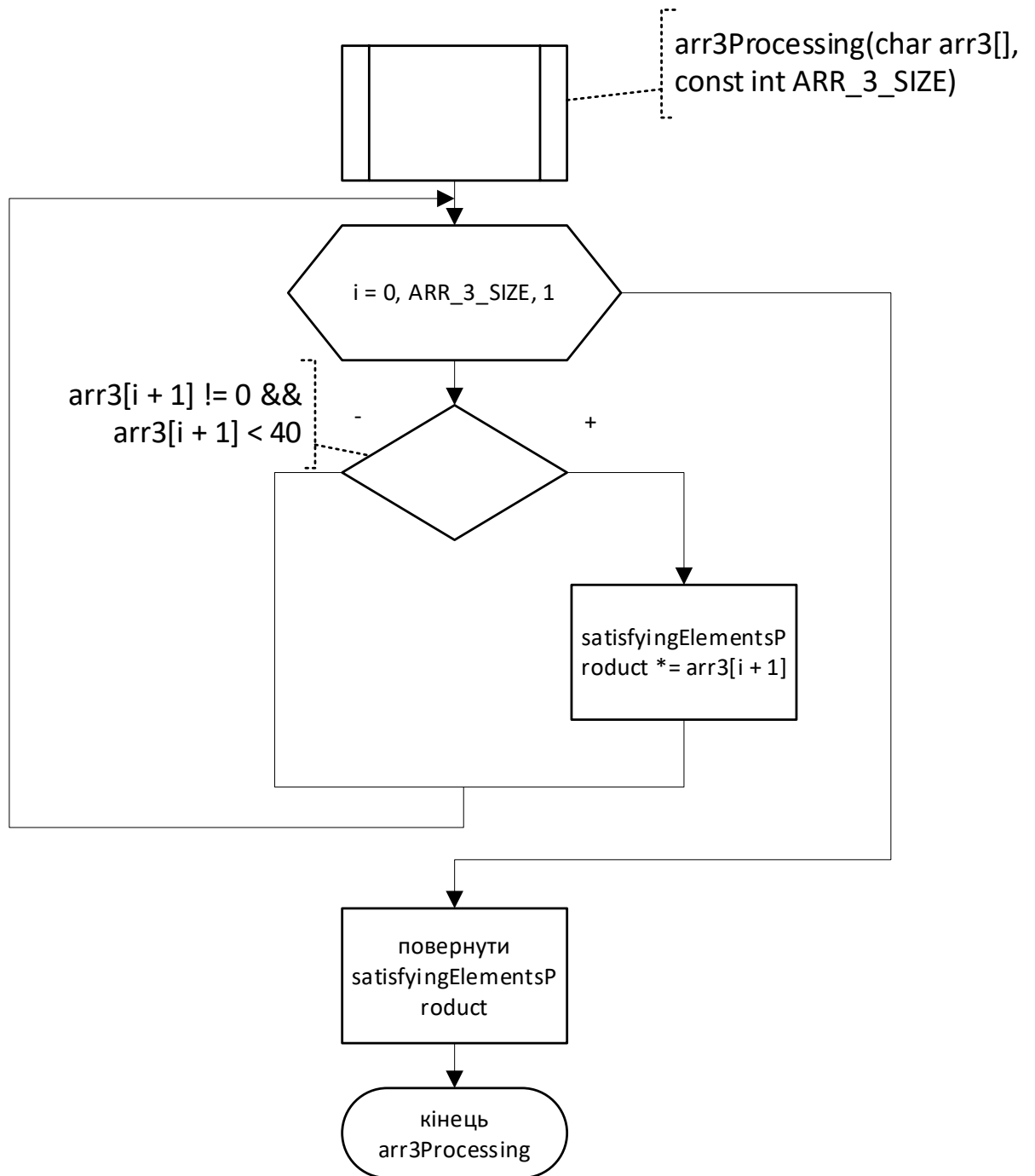


Підпрограми arr3Processing(char arr3[], const int ARR_3_SIZE):

Крок 1

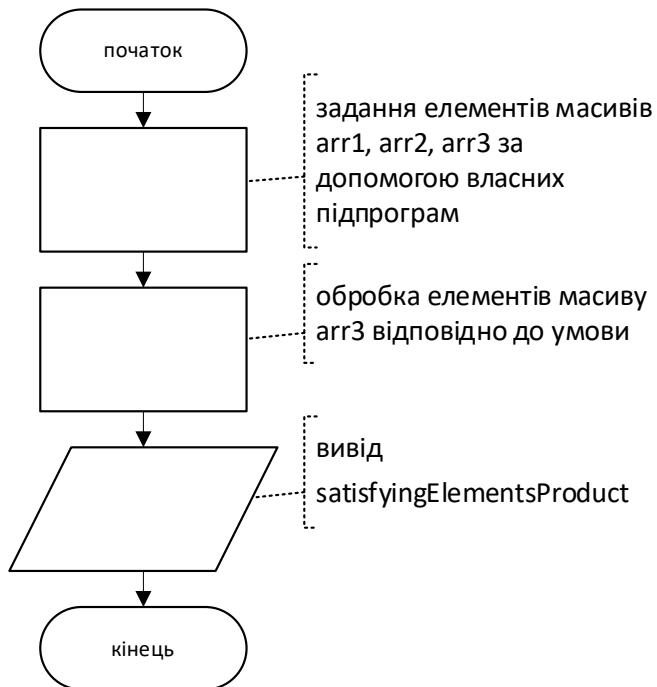


Крок 2

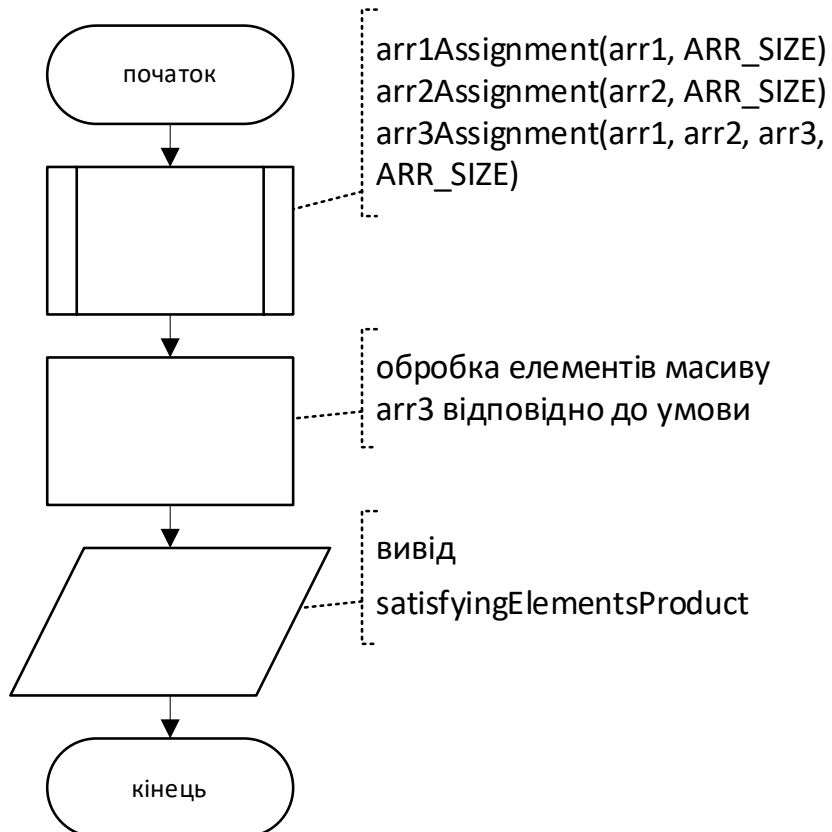


Основна програма:

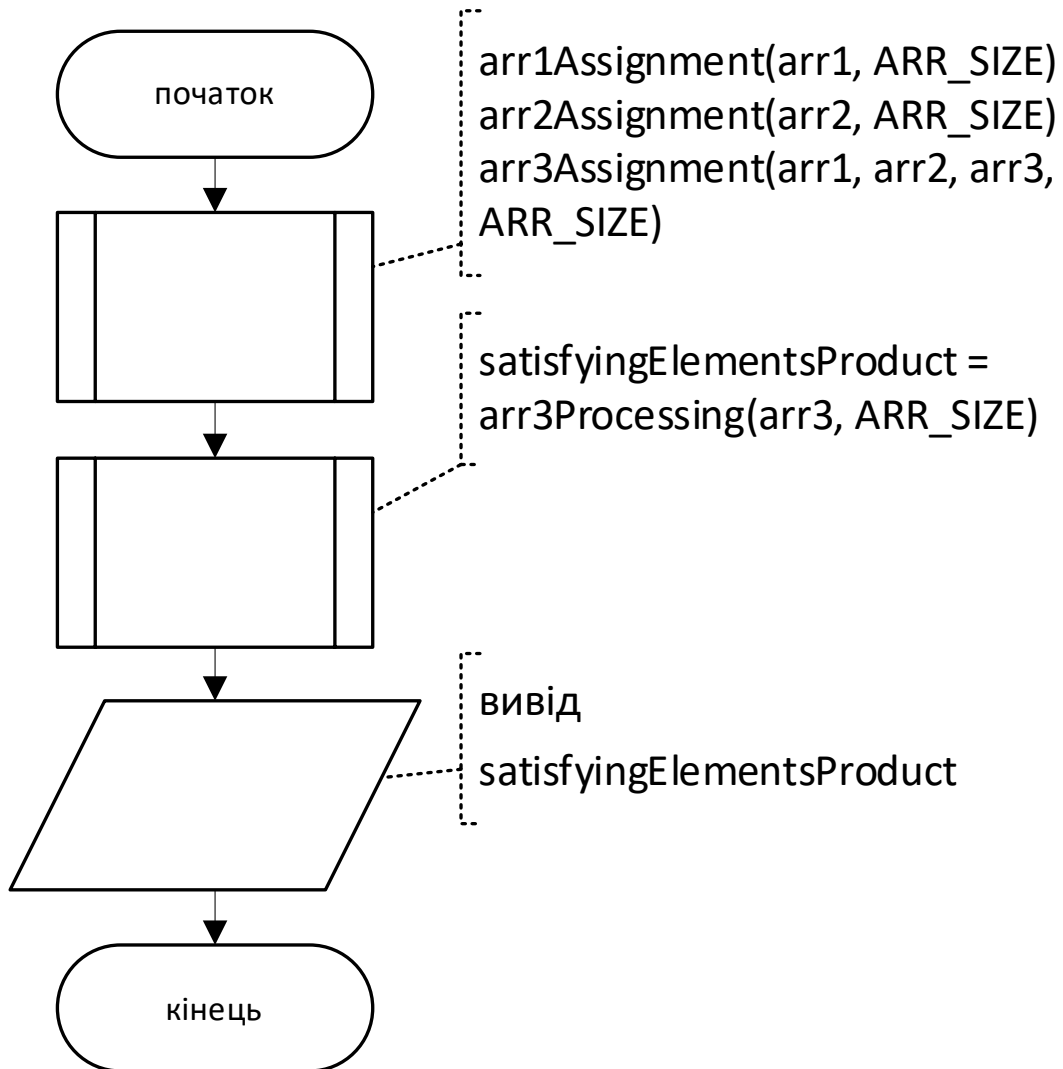
Крок 1



Крок 2



Крок 3



Реалізація алгоритму на мові C++:

```
#include <iostream>

void arr1Assignment(char arr1[], const int ARR_1_SIZE);

void arr2Assignment(char arr2[], const int ARR_2_SIZE);

void arr3Assignment(char arr1[], char arr2[], char arr3[], const int ARR_SIZE);

void displayCharArray(char arr[], const int ARR_SIZE);

char arr3Processing(char arr3[], const int ARR_3_SIZE);

int main()
{
    const int ARR_SIZE{ 10 };

    char arr1[ARR_SIZE]{};
    arr1Assignment(arr1, ARR_SIZE);
    std::cout << "Array 1:\n";
    displayCharArray(arr1, ARR_SIZE);

    char arr2[ARR_SIZE]{};
    arr2Assignment(arr2, ARR_SIZE);
    std::cout << "Array 2:\n";
    displayCharArray(arr2, ARR_SIZE);

    char arr3[ARR_SIZE]{};
    arr3Assignment(arr1, arr2, arr3, ARR_SIZE);
    std::cout << "Array 3:\n";
    displayCharArray(arr3, ARR_SIZE);

    char satisfyingElementsProduct{ 0 };
    satisfyingElementsProduct = arr3Processing(arr3, ARR_SIZE);
    std::cout << "Result: " << satisfyingElementsProduct << '\n';
    return 0;
}

void arr1Assignment(char arr1[], const int ARR_1_SIZE)
{
    for (int i{ 0 }; i < ARR_1_SIZE; ++i)
    {
        arr1[i] = 5 * (i + 1) + 30;
    }
}

void arr2Assignment(char arr2[], const int ARR_2_SIZE)
{
    for (int i{ 0 }; i < ARR_2_SIZE; ++i)
    {
        arr2[i] = 60 - 5 * (i + 1);
    }
}

void arr3Assignment(char arr1[], char arr2[], char arr3[], const int ARR_SIZE)
{
    bool elementIsAlreadyInArray3{ false };
    int arr3Index{ 0 };
}
```

```

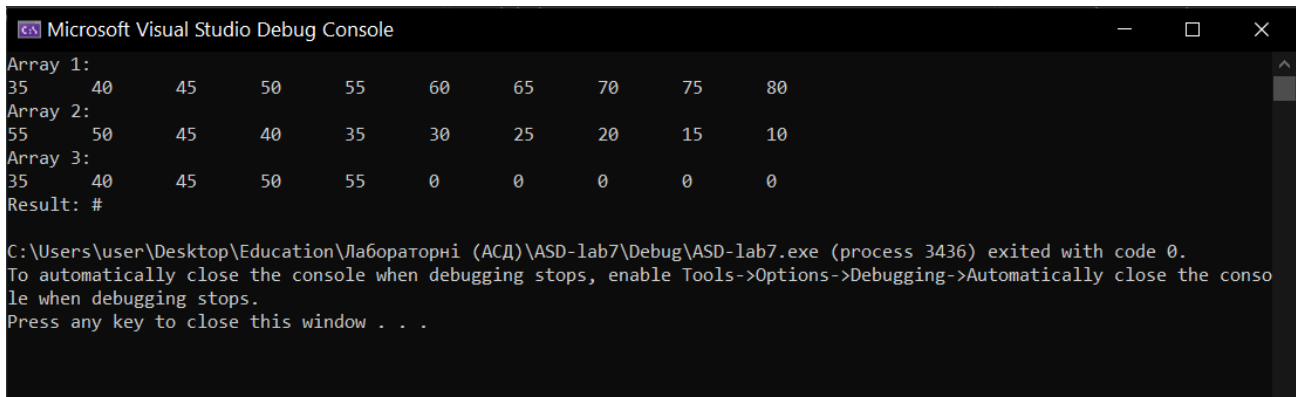
for (int i{ 0 }; i < ARR_SIZE; ++i)
{
    for (int j{ 0 }; j < ARR_SIZE; ++j)
    {
        if (arr1[i] == arr2[j])
        {
            elementIsAlreadyInArray3 = false;
            for (int k{ 0 }; k < arr3Index && !elementIsAlreadyInArray3; ++k)
            {
                if (arr1[i] == arr3[k])
                {
                    elementIsAlreadyInArray3 = true;
                }
            }
            if (!elementIsAlreadyInArray3)
            {
                arr3[arr3Index] = arr1[i];
                arr3Index++;
            }
        }
    }
}

void displayCharArray(char arr[], const int ARR_SIZE)
{
    for (int i{ 0 }; i < ARR_SIZE; ++i)
    {
        std::cout << (int)arr[i] << '\t';
    }
    std::cout << '\n';
}

char arr3Processing(char arr3[], const int ARR_3_SIZE)
{
    char satisfyingElementsProduct{ 1 };
    for (int i{ 0 }; i < ARR_3_SIZE; ++i)
    {
        if (arr3[i] && (int)arr3[i] < 40)
        {
            satisfyingElementsProduct *= arr3[i];
        }
    }
    return satisfyingElementsProduct;
}

```

Результат виконання даної програми:



```
Microsoft Visual Studio Debug Console
Array 1:
35    40    45    50    55    60    65    70    75    80
Array 2:
55    50    45    40    35    30    25    20    15    10
Array 3:
35    40    45    50    55    0    0    0    0    0
Result: #

C:\Users\user\Desktop\Education\Лабораторні (АСД)\ASD-lab7\Debug\ASD-lab7.exe (process 3436) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Висновки. Таким чином, в результаті виконання лабораторної роботи було досліджено методи послідовного пошуку у впорядкованих і неупорядкованих послідовностях (на прикладі пошуку однакових елементів в двох масивах та запису їх у третій) та набути практичних навичок їх використання під час складання програмних специфікацій. Особливістю цього варіанту лабораторної роботи є здійснення непрямого зведення типів даних під час обчислень (char та int) та створення власних підпрограм для заповнення та обробки масивів.