МІНІСТРЕСТВО ОСВІТИ Й НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“Харківський Політехнічний Інститут”

Кафедра управління проєктами в інформаційних технологіях

Звіт з лабораторної роботи №1

“Однозв’язні та двозв’язні списки й робота з ними”

з дисципліни

“Алгоритми та структури даних”

Варіант №2

Перевірив: ст. викл. каф. УПІТ Мошко Є.О.

Виконав: ст. гр. КН-1223г Ставицький А.А.

Харків – 2024

**Мета**: Ознайомитися із основними способами організації списків та особливостями їх програмної реалізації. Набути практичних навичок по роботі зі однозв’язними та двозв’язними списками

**Завдання**:

1. Для організації однозвʼязного списку визначити структурний тип, що містить покажчик на свій тип і поля.

Варіант завдання вибирається за номером студента в списку групи.

Варіанти завдань наведені в таблиці 1.

2. Написати програму, яка демонструє основні дії по роботі із однозвʼязним списком і пропонує користувачу вибір дії:

* ﻿﻿сформувати список;
* ﻿﻿додати елемент до списку (в кінець);
* ﻿﻿вилучити вказаний елемент 31 списку;
* ﻿﻿визначити кількість елементів у списку;
* ﻿﻿поміняти два сусідні елементи місцями;
* ﻿﻿очистити список.

3. Перетворити лінійний список в двозвʼязний. Внести необхідні зміни в операції роботи зі списком.

Написати програму, яка демонструє основні дії по роботі із двозвʼязним

списком і пропонує користувачу вибір дії:

* ﻿﻿сформувати список;
* ﻿﻿додати елемент до списку;
* ﻿﻿вилучити вказаний елемент з1 списку;
* ﻿﻿поміняти два сусідні елементи місцями;

﻿﻿очистити список.

**Формування** **списку**:

Цей код є частиною функції, яка рекурсивно створює однозв’язний список із заданою кількістю елементів. Ось докладний опис, що відбувається на кожному кроці:

if (n > 0):

• Ця умова перевіряє, чи більше нуля значення n (кількість елементів, які потрібно додати до списку). Якщо n більше нуля, виконується наступна частина коду. Якщо ж n дорівнює нулю або менше, функція зупиняється, і список більше не буде доповнюватися новими елементами.

Single\_List\* newNode = new Single\_List();:

• Створюється новий вузол (елемент) для списку типу Single\_List. Цей новий вузол (вказівник newNode) динамічно виділяється в пам’яті.

cout << "Введіть значення для елемента: "; **і** cin >> newNode->data;:

• Користувачу пропонується ввести значення для нового елемента списку. Введене значення зберігається у змінній data нового вузла (елемента) newNode.

newNode->next = head;:

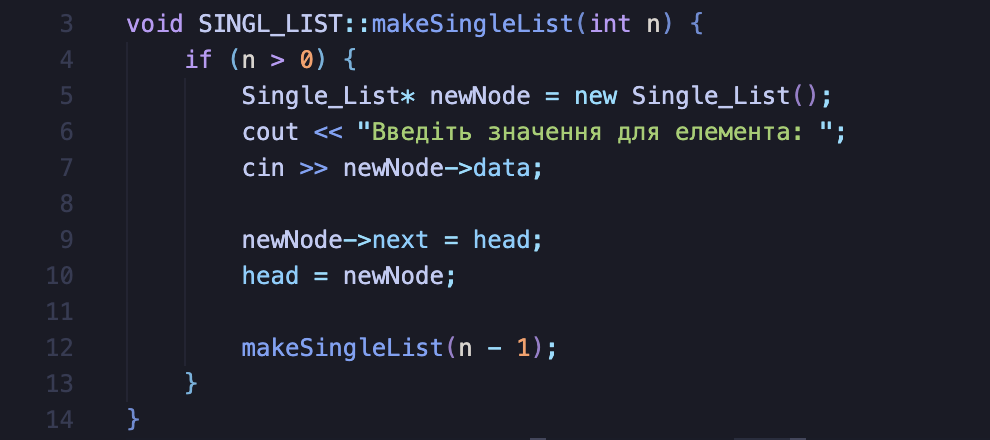
• Вказівник next нового вузла newNode встановлюється на поточний головний елемент списку (head). Це означає, що новий елемент буде вказувати на попередній головний елемент списку.

head = newNode;:

• Головний вказівник списку (head) оновлюється і тепер вказує на новостворений елемент newNode. Таким чином, новий елемент стає головою списку, а попередній головний елемент тепер є другим елементом у списку.

makeSingleList(n - 1);:

• Викликається функція makeSingleList() рекурсивно з аргументом n - 1, щоб продовжити додавати елементи до списку, доки n не стане рівним нулю.



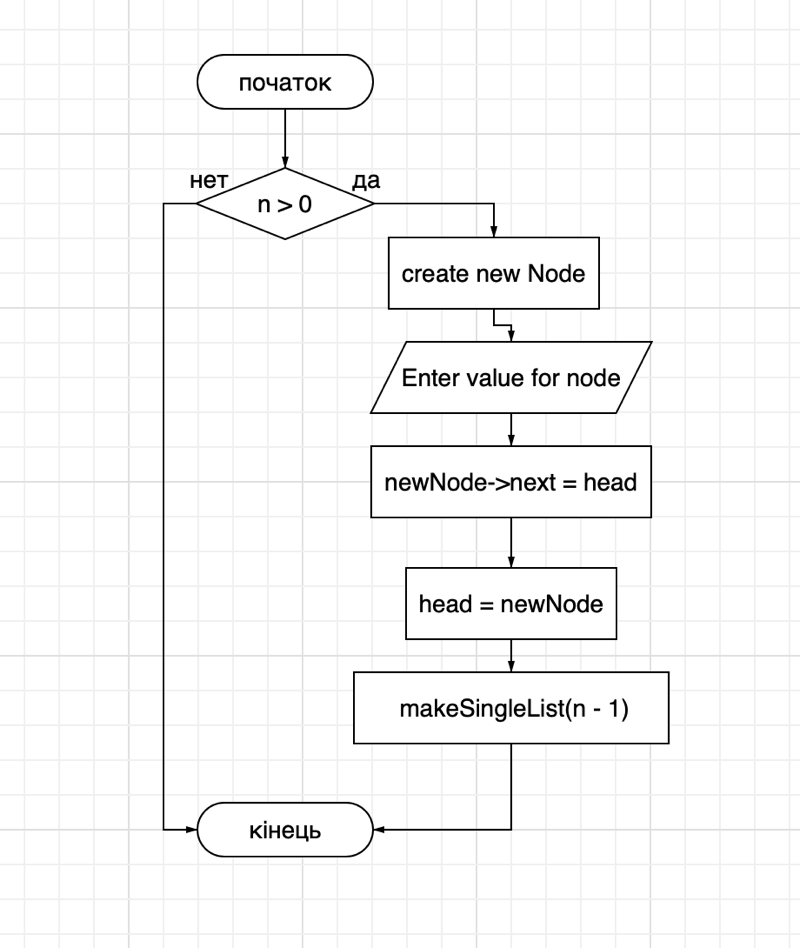


Рис 1. Блок-схема функції makeSingleList()

**Показ списку**:

if (Head != nullptr):

• Ця умова перевіряє, чи поточний елемент списку не є порожнім (не nullptr). Якщо список ще містить елементи, виконується наступна частина коду.

cout << setw(DISTANCE) << Head->data << '\t';:

• Поточний елемент списку (Head->data) виводиться на екран з відступом, який визначається значенням DISTANCE.

• setw(DISTANCE) використовується для того, щоб вирівняти виведені елементи на певну відстань для кращого форматування виводу.

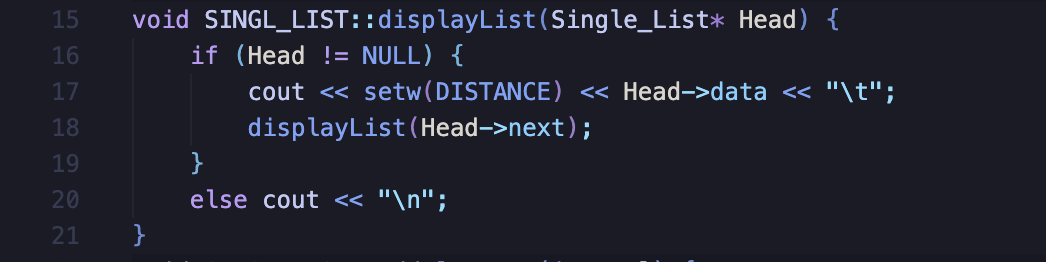
• Після виводу елемента додається символ табуляції ('\t') для створення простору між елементами.

displayList(Head->next);:

• Після виведення поточного елемента функція рекурсивно викликається для наступного елемента списку. Це відбувається за допомогою вказівника next, який вказує на наступний вузол у списку.

else { cout << "\n"; }:

• Якщо досягається кінець списку (вказівник Head дорівнює nullptr), функція не викликається повторно, а просто виводиться новий рядок ("\n"), що вказує на завершення виводу всіх елементів списку.



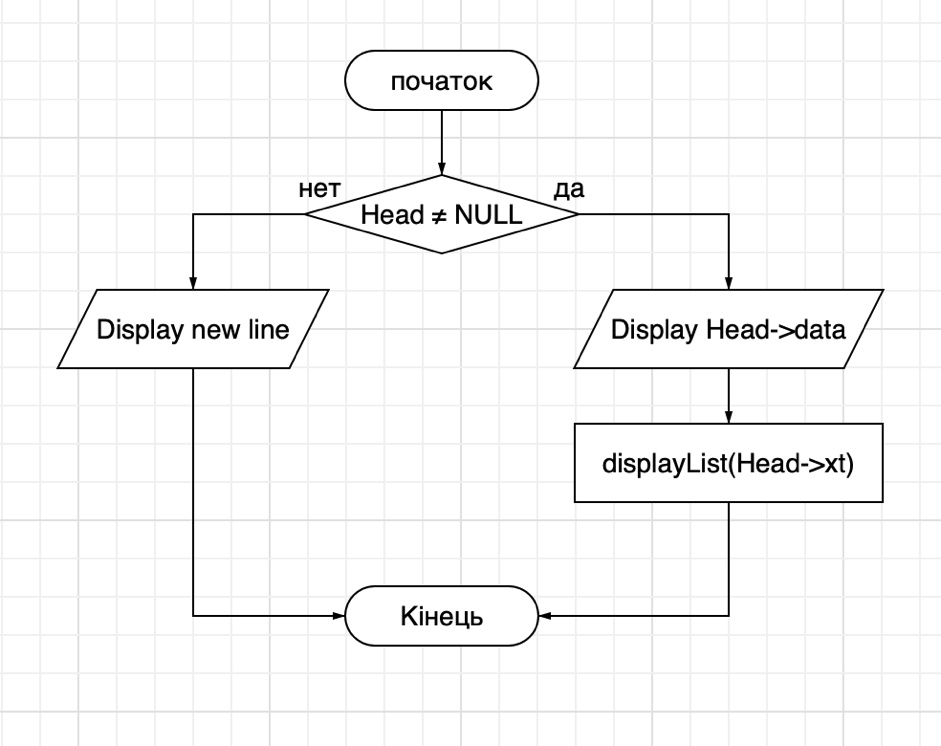


Рис. 2 блок-схема функції displayList()

**Додавання** **елементів**:

Single\_List\* newNode = new Single\_List();:

• Створюється новий вузол (елемент) списку.

newNode->data = val;:

• Присвоюється значення val, яке було передано функції, новому вузлу.

newNode->next = NULL;:

• Оскільки цей вузол додається в кінець списку, вказівник на наступний вузол (next) встановлюється в NULL.

if (head == NULL):

• Перевіряється, чи список порожній. Якщо вказівник на голову списку (перший елемент) є NULL, це означає, що список порожній.

head = newNode;:

• Якщо список порожній, новий елемент стає головою списку, тобто першим елементом.

else:

• Якщо список не порожній, виконується пошук останнього елемента.

Single\_List\* temp = head;:

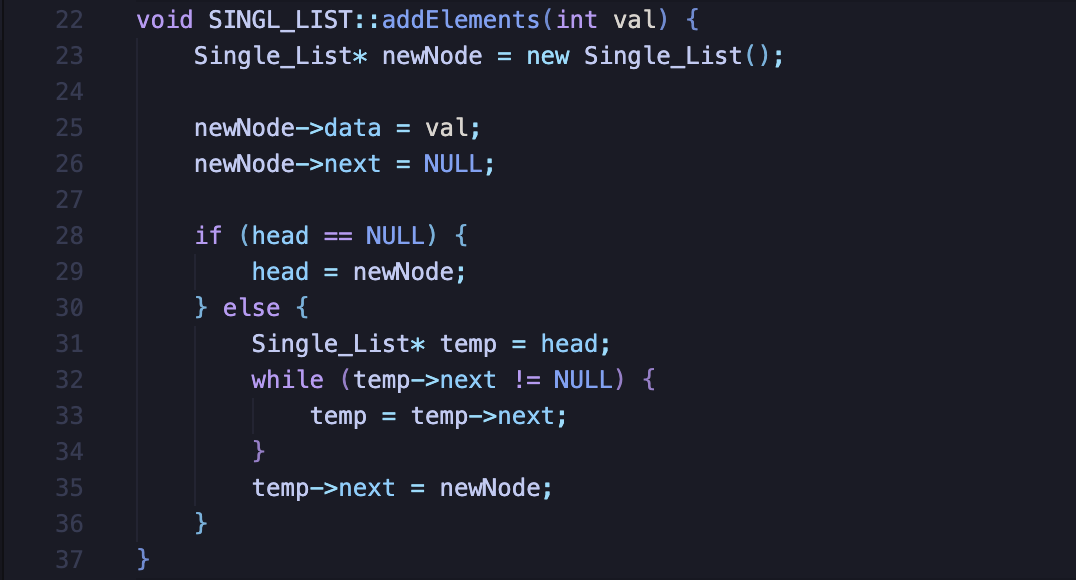
• Створюється тимчасовий вказівник temp, який спочатку вказує на перший елемент списку.

while (temp->next != NULL):

• Цикл проходить по списку, шукаючи останній елемент (той, у якого next дорівнює NULL).

temp->next = newNode;:

• Як тільки знайдено останній елемент, новий вузол додається в його кінець, оновлюючи вказівник next.



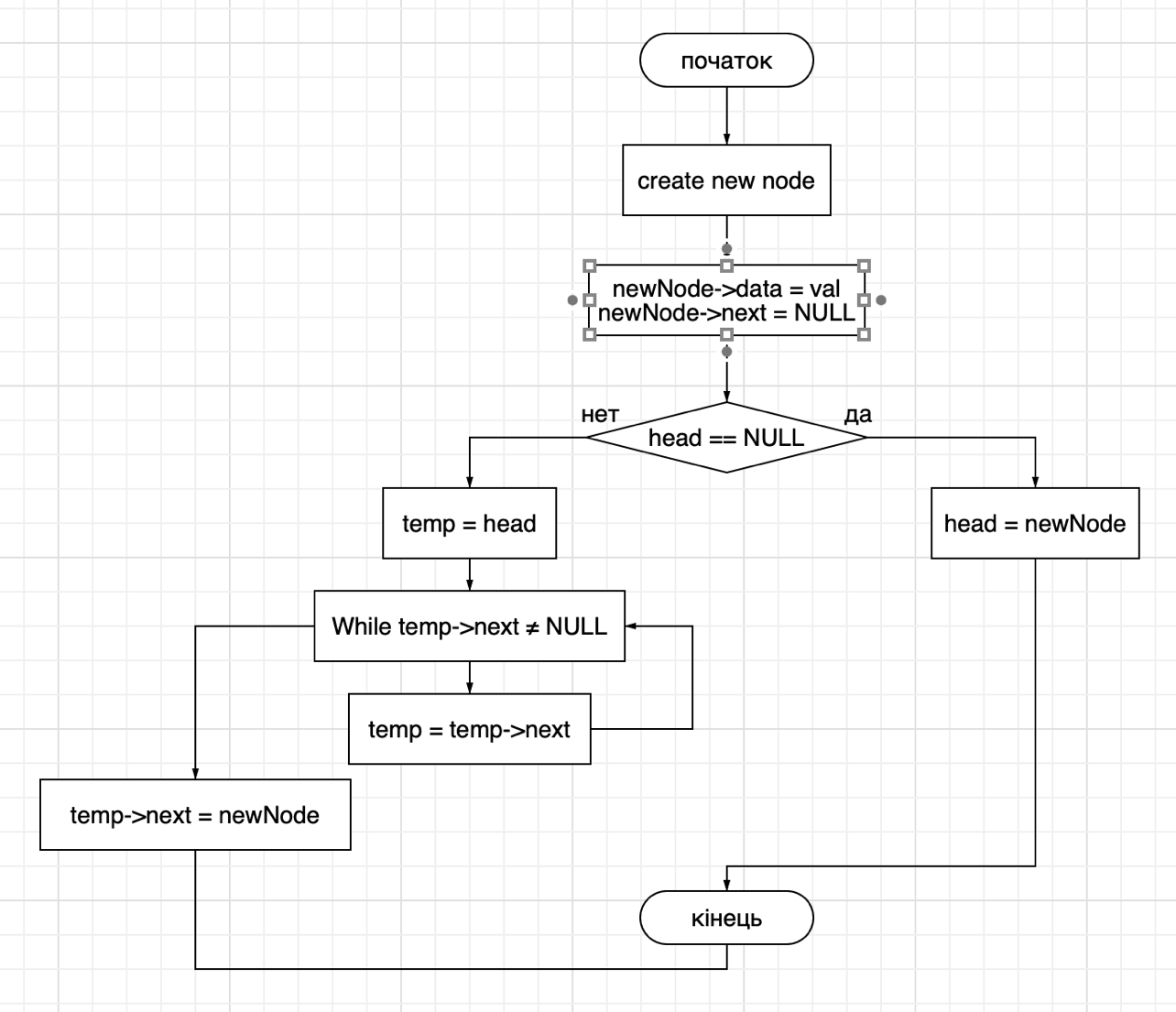


Рис. 3 блок-схема функції addElements()

**Очищення списку:**

Single\_List\* current = head;:

• Створюється вказівник current, який спочатку вказує на головний елемент списку.

Single\_List\* next;:

• Створюється додатковий вказівник next, який буде використовуватися для збереження наступного елемента перед тим, як видалити поточний.

while(current != NULL):

• Цикл проходить по всьому списку, поки поточний елемент не стане NULL (тобто поки не досягнуто кінця списку).

next = current->next;:

• Перед видаленням поточного елемента зберігається вказівник на наступний елемент, щоб не втратити доступ до нього після видалення.

delete current;:

• Видаляється поточний елемент, звільняючи пам’ять, яку він займав.

current = next;:

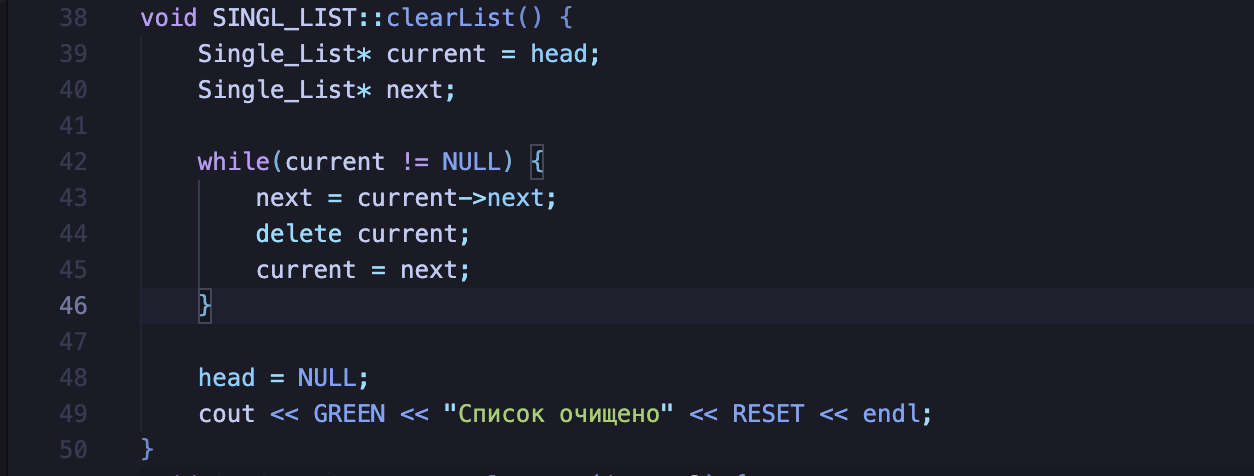
• Після видалення поточного елемента вказівник current оновлюється і починає вказувати на наступний елемент.

head = NULL;:

• Після завершення циклу всі елементи списку видалено, тому голова списку (head) встановлюється в NULL.

cout << GREEN << "Список очищено" << RESET << endl;:

• Виводиться повідомлення про успішне очищення списку.



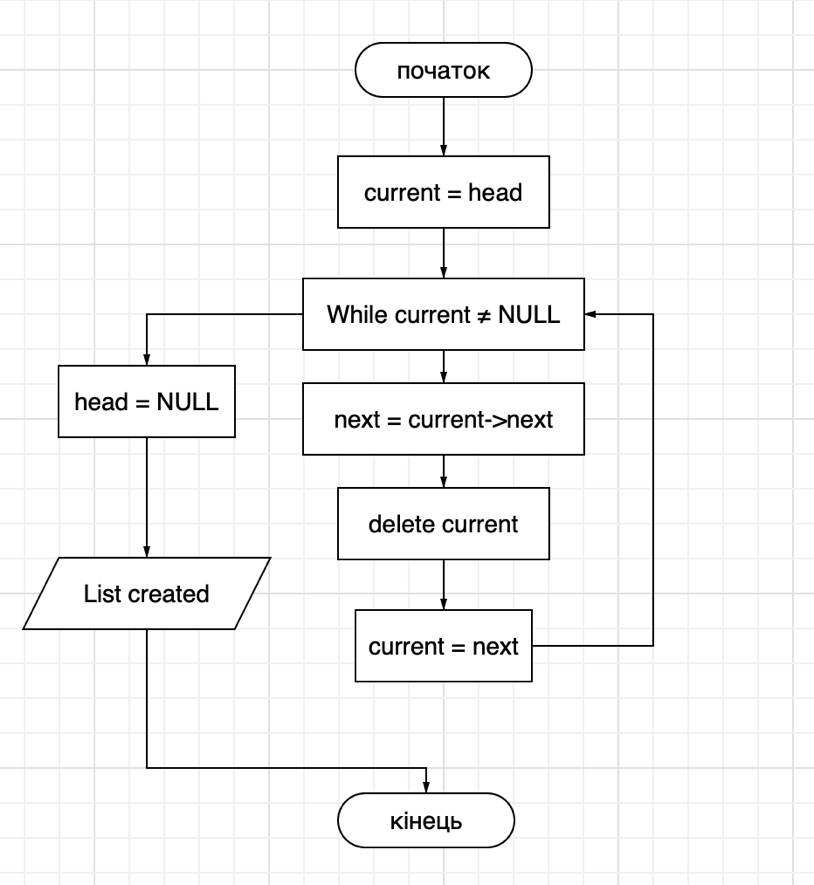


Рис 4. Блок-схема функції clearList()

**Видалення елементу зі списку:**

**Перевірка на порожній список**:

• Якщо head == nullptr, тобто список порожній, виводиться повідомлення про те, що неможливо видалити елемент, і функція завершується.

**Видалення елемента на початку списку**:

• Якщо перший елемент списку (голова) містить значення, яке потрібно видалити, то:

• Голова списку (head) оновлюється, щоб вказувати на наступний елемент.

• Поточний елемент видаляється за допомогою delete.

**Пошук елемента в списку**:

• За допомогою циклу while функція проходить через всі елементи списку, поки не знайде потрібне значення.

• prew зберігає попередній елемент перед поточним, а current використовується для проходження списку.

**Якщо елемент не знайдений**:

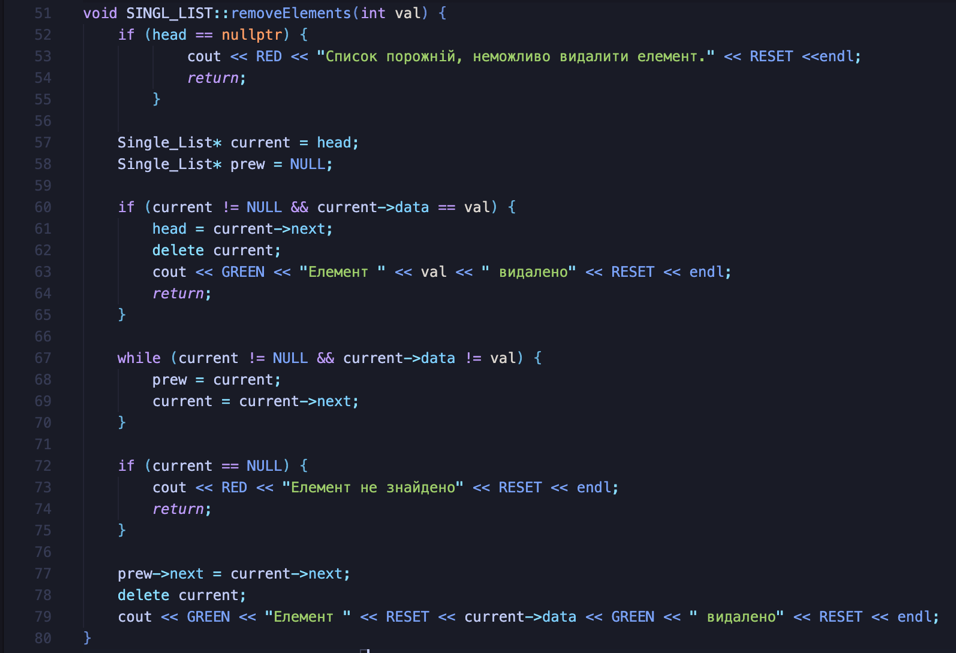
• Якщо після проходження списку потрібний елемент не знайдений, виводиться повідомлення про відсутність елемента.

**Видалення елемента в середині чи кінці списку**:

• Коли елемент знайдено, функція:

• Зв’язує попередній елемент з наступним після поточного.

• Видаляє поточний елемент за допомогою delete.



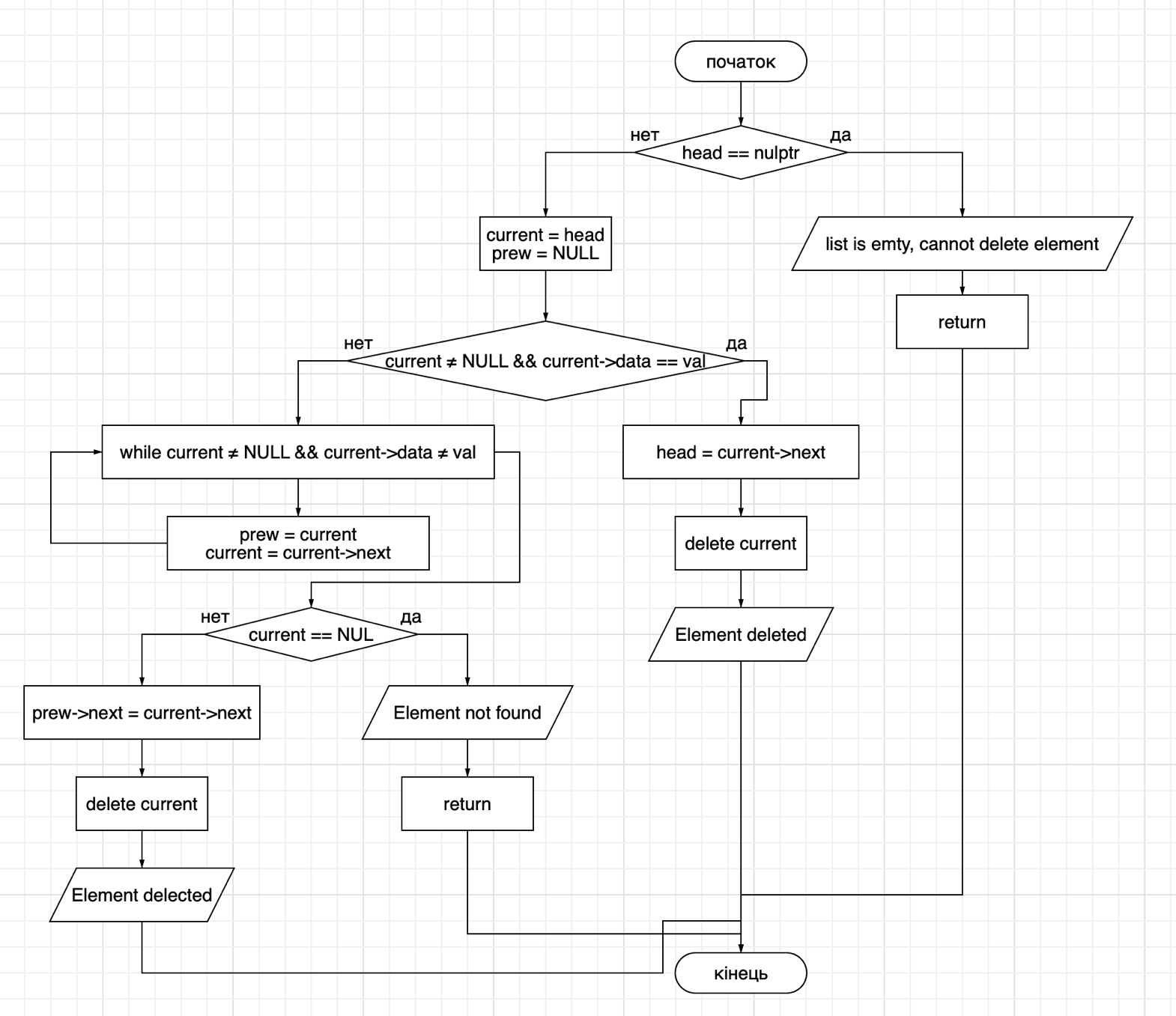


Рис. 5 блок-схема removeElements()

**Підрахунок елеметів**:

**Ініціалізація**:

• Змінна current ініціалізується як покажчик на голову списку (head).

• Змінна count ініціалізується як 0 і використовується для підрахунку елементів списку.

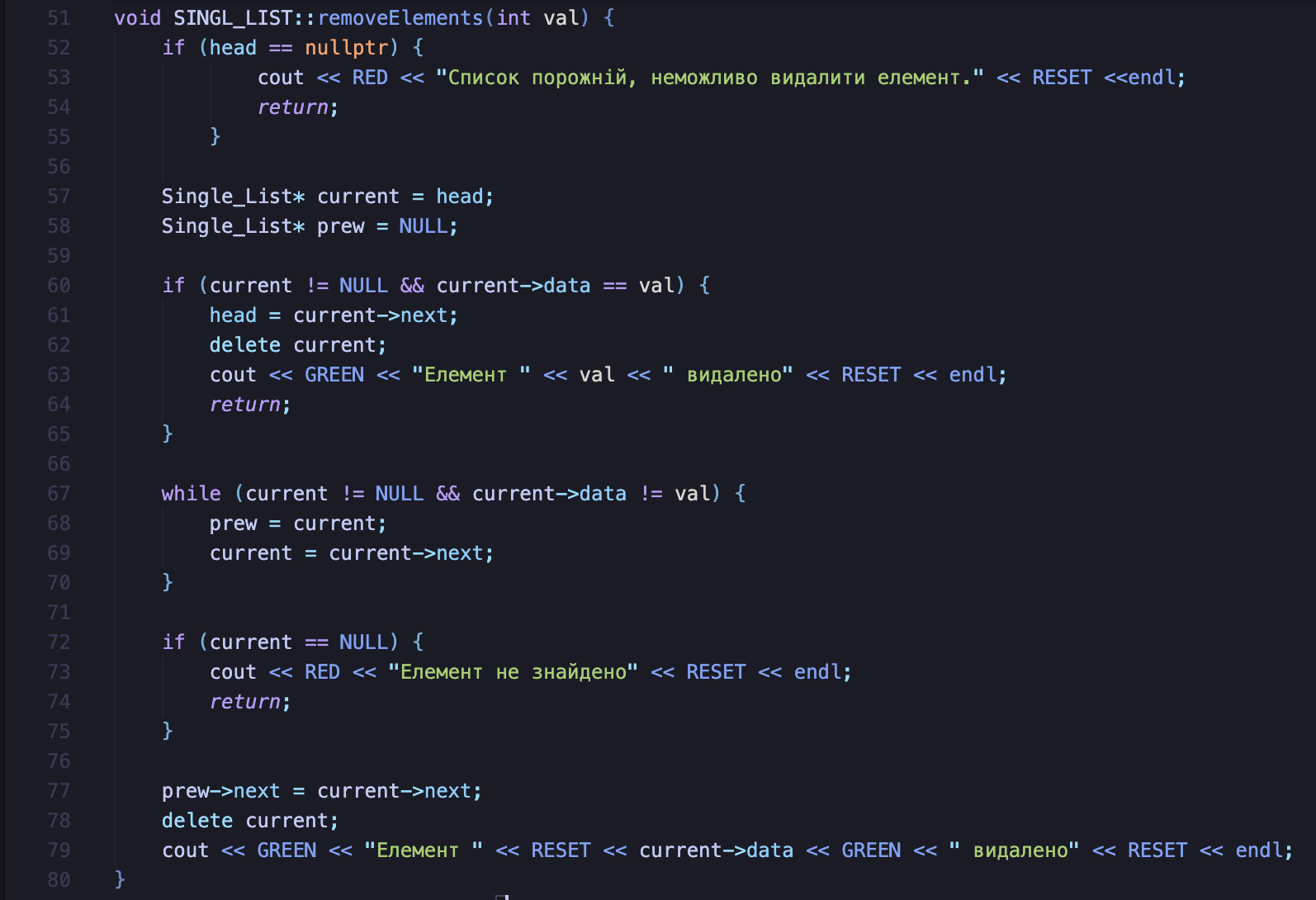
**Прохід по списку**:

• Функція використовує цикл while, який проходить через кожен елемент списку, поки не дійде до кінця, тобто поки current != NULL.

• На кожній ітерації лічильник count збільшується на 1.

**Повернення результату**:

• Після завершення проходу по всіх елементах списку, функція повертає значення змінної count, яка містить кількість елементів у списку.



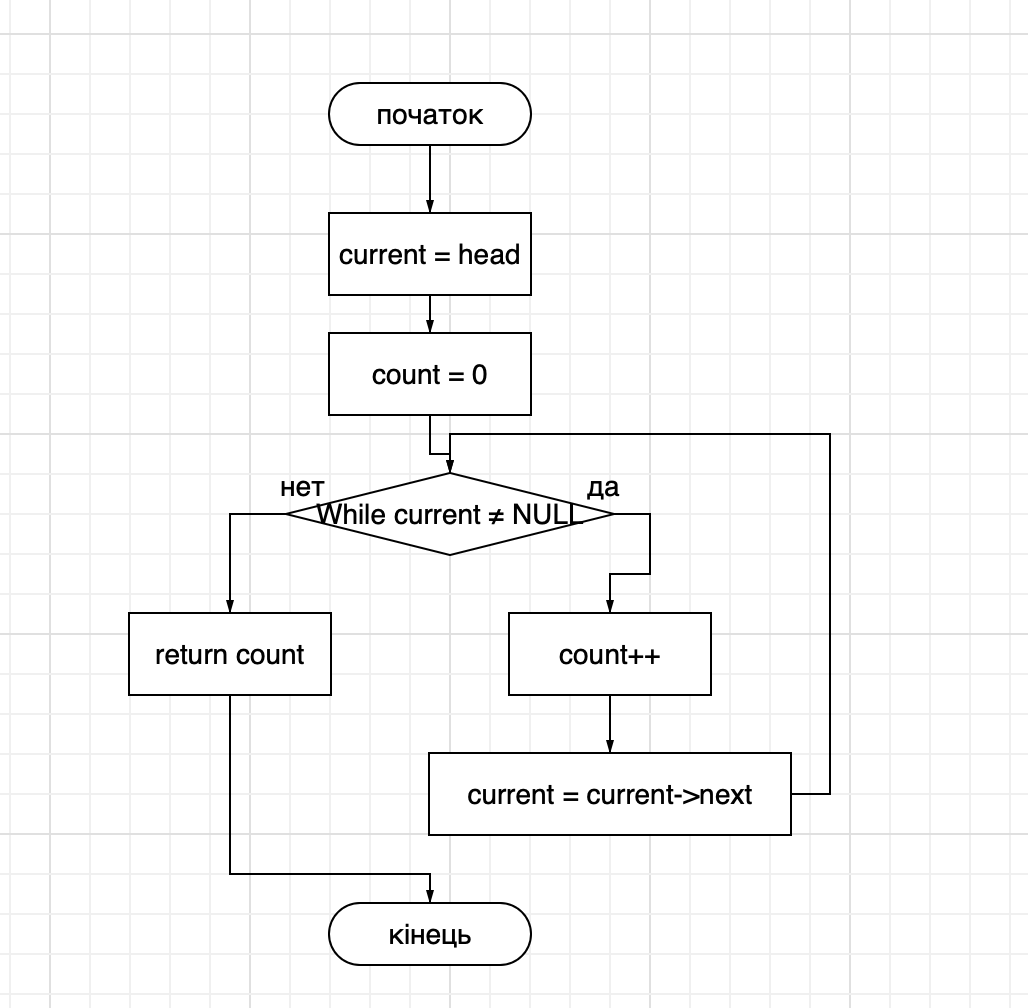


Рис 6. Блок-схема сountElements()

**Обмін елементів:**

**Перевірки на коректність**:

• Перевіряється, чи є позиція валідною: якщо позиція pos менша або рівна 0, або якщо список порожній (head == NULL), або в ньому тільки один елемент (head->next == NULL), операція не може бути виконана, і виводиться повідомлення про помилку.

**Обмін елементів на першій позиції**:

• Якщо потрібно обміняти перший і другий елементи (pos == 1), вони просто міняються місцями:

• Перший елемент (current) стає другим, а другий (next) стає першим.

• Оновлюється вказівник на голову списку.

**Пошук елементів, які треба поміняти**:

• Цикл for проходить по списку і знаходить потрібні елементи, які знаходяться на позиціях pos і pos + 1.

• На кожній ітерації цикл переміщає покажчики prew, current, і next, щоб знайти відповідні елементи.

**Перевірка на наявність елементів для обміну**:

• Якщо список закінчився до того, як було знайдено два сусідніх елементи (current == NULL або next == NULL), функція припиняє роботу і виводить повідомлення про помилку.

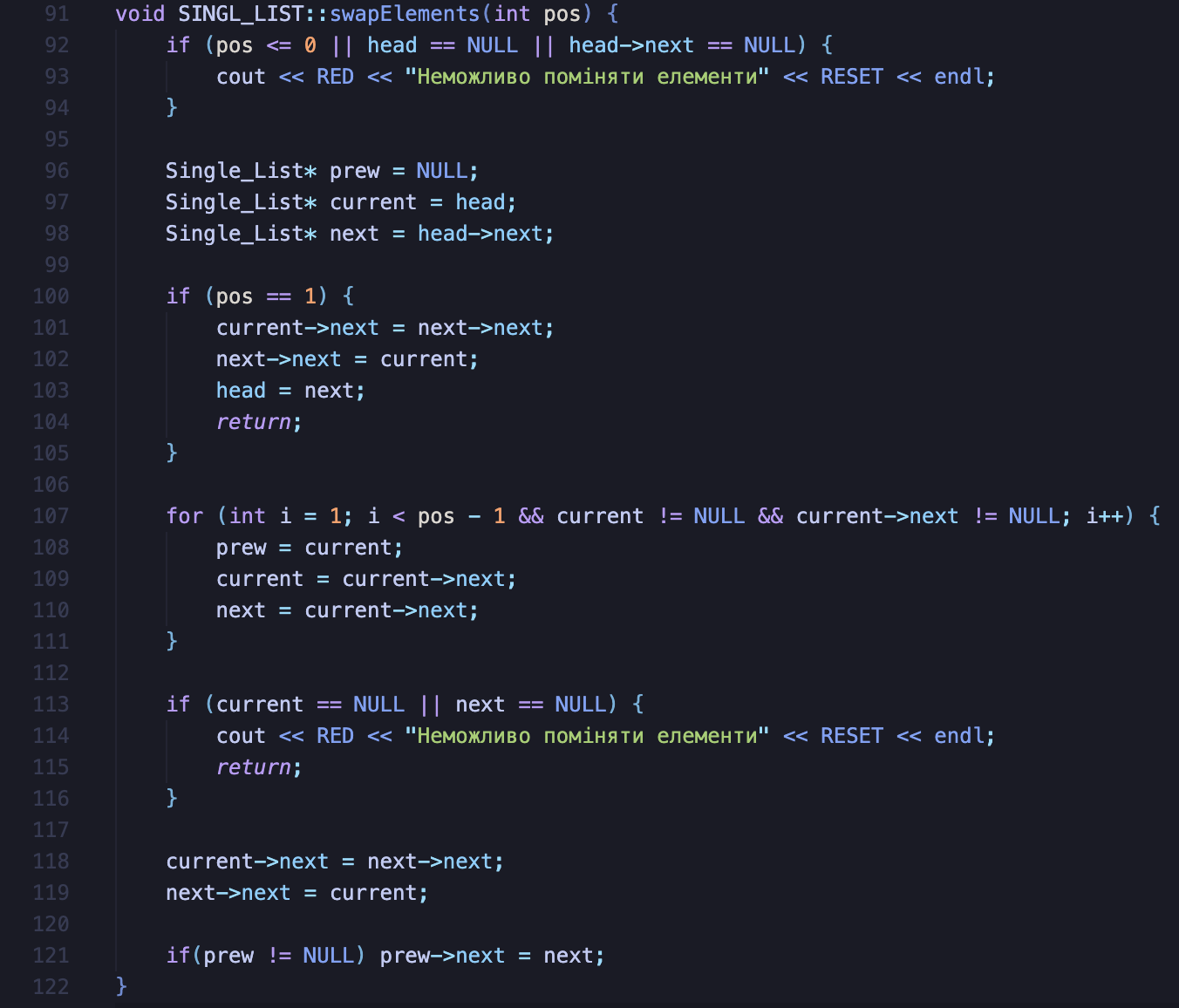
**Обмін сусідніми елементами**:

• Якщо елементи знайдені, вони міняються місцями:

• current змінює своє посилання на наступний елемент (next->next).

• next змінює своє посилання на current, фактично міняючи їх місцями.

• Якщо існує попередній елемент (prew), він тепер повинен вказувати на новий перший елемент (теперішній next).



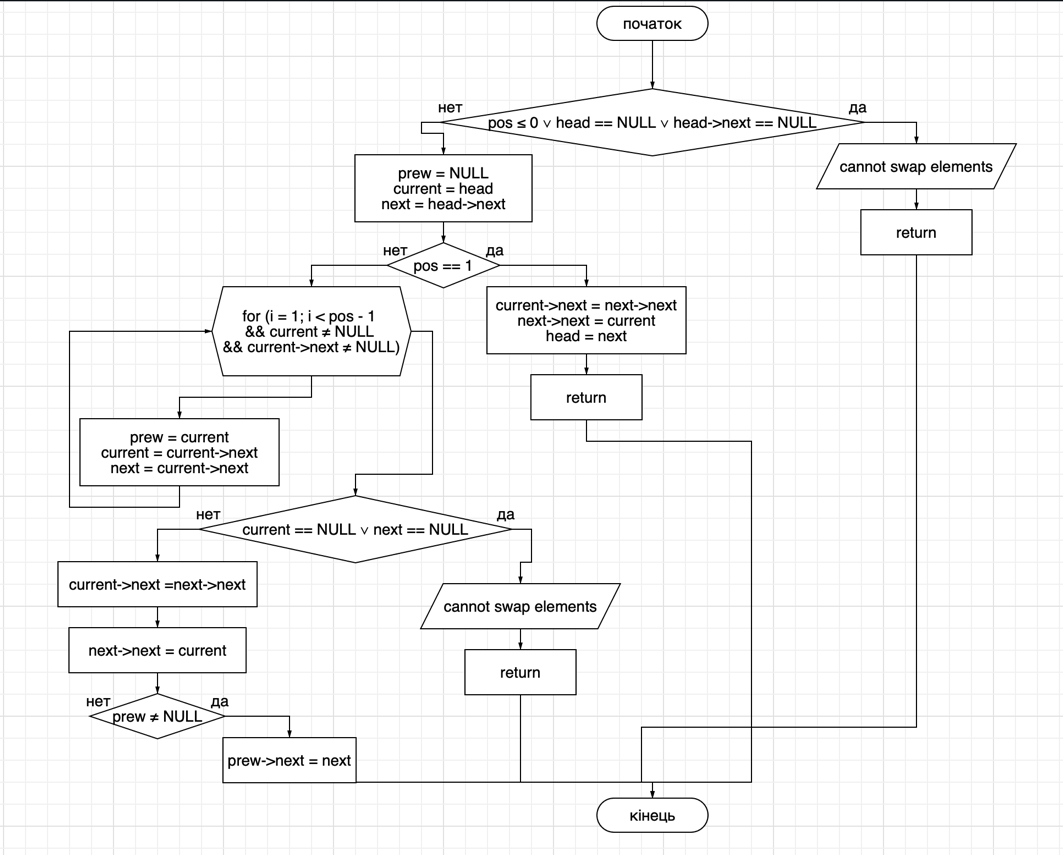


Рис. 7 блок-схема swapElements()

**Записування в файл:**

**Відкриття файлу**:

• Функція приймає ім’я файлу filename як аргумент і намагається відкрити його для запису.

• Використовується клас ofstream для запису даних у файл.

• Якщо файл не вдалося відкрити, виводиться повідомлення про помилку, і функція завершується.

**Прохід по списку**:

• Використовується вказівник current, який початково вказує на голову списку (head).

• В циклі while функція проходить по кожному елементу списку, доки не дійде до кінця (current != NULL).

**Запис даних у файл**:

• Кожен елемент списку (current->data) записується у файл.

• Між елементами додається пробіл, щоб розділити значення, але після останнього елемента пробіл не додається.

• Це реалізується перевіркою if (current->next != NULL) — пробіл додається тільки якщо це не останній елемент.

**Закриття файлу**:

• Після завершення запису у файл додається новий рядок (\n), і файл закривається методом outFile.close().

**Підтвердження успішного запису**:

• Виводиться повідомлення про те, що список успішно записаний у файл.



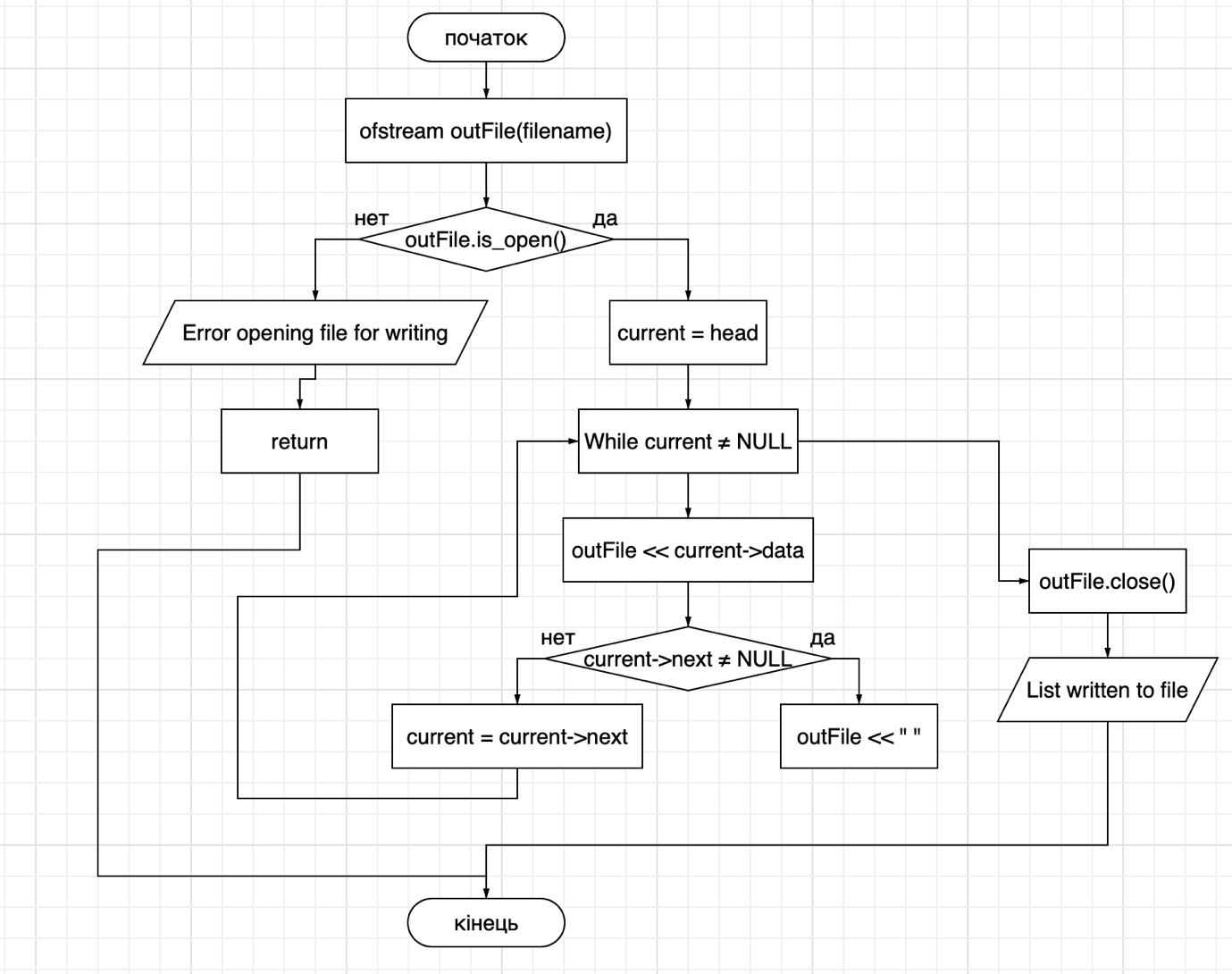


Рис 8. Блок-схема writeToFile()

**Висновки:** Під час виконання лабораторної роботи я здобув важливий практичний досвід у реалізації однозв’язного та двозв’язного списків на мові C++. На початкових етапах я навчився працювати з динамічною пам’яттю, зокрема створювати нові елементи списку, коректно виділяти і звільняти пам’ять. Це стало особливо важливим для запобігання витокам пам’яті під час очищення списку та видалення елементів.

Однією з ключових задач було забезпечення додавання елементів до списку, що вимагало від мене навичок у роботі з покажчиками. Для однозв’язного списку я реалізував функцію додавання елемента в кінець списку, використовуючи ітерацію через наявні елементи, щоб знайти останній вузол. Важливим етапом було забезпечення коректного оновлення покажчиків, щоб новий елемент став останнім у списку.

Також я працював над функціями видалення елементів зі списку. Тут я зіткнувся з необхідністю обробки різних сценаріїв, наприклад, видалення головного елемента або елемента в середині списку. Це вимагало обережної маніпуляції покажчиками на наступні та попередні елементи для уникнення помилок.

Функція підрахунку елементів у списку дозволила мені закріпити навички ітерації через вузли списку, що також було корисно під час реалізації функції виведення елементів на екран. Використання рекурсії для цієї функції допомогло спростити код та покращити його читабельність.

Особливо цікавою частиною роботи було завдання зі зміною двох сусідніх елементів місцями. Ця задача вимагала глибокого розуміння того, як працюють покажчики в контексті динамічних структур, щоб коректно оновити зв’язки між елементами, зберігаючи цілісність списку.

Крім цього, я реалізував функції для роботи з файлами, що включало зчитування даних зі списку та запис його до файлу. Це дозволило поглибити мої знання у сфері файлового вводу/виводу, зокрема роботу з потоками даних, забезпечення коректного формату запису елементів та обробку помилок під час відкриття файлів.

Робота з двозв’язним списком виявилася більш складною, оскільки потрібно було враховувати як попередні, так і наступні елементи кожного вузла. Це додало більше складності в управлінні покажчиками, але допомогло зрозуміти, як такі структури даних можуть полегшувати певні операції, наприклад, видалення або перестановку елементів.

Загалом, під час виконання цієї лабораторної роботи я суттєво покращив свої навички у роботі з динамічними структурами даних і з роботою з файлами. Я навчився ефективно працювати з пам’яттю, працювати з різними типами списків, а також використовувати рекурсію та ітерацію для реалізації різноманітних функцій. Ця робота підготувала мене до подальших складніших завдань та закріпила теоретичні знання на практиці.