МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет

«Харківський політехнічний інститут»

Кафедра «Стратегічного управління проектами»

**Лабораторна робота №1**

з дисципліни «Алгоритми і структури даних»

Виконав:

студент групи КН-320(Б)

Колій Дмитро

Перевірив:

Мошко Є.О.

Харків 2022

ПЕРЕДМОВА

Дана програма була реалізована на мові програмування С++ у середовищі VS 2019. Сама програма розбита на три файли, а саме: «Header.h» - файл з описом класів, підключенням необхідних бібліотек і простору імен; «function.cpp» - файл з описом методів класів; «main.cpp» - основний файл в якому відбувається всі дії і маніпуляції із списками та методами.

class List // Клас списку

{

private:

bool type; // Якщо true - двозв'язний, false - кільцевий

int size = 0; // Розмір списку

Node\* head = NULL, \* tail = NULL; // Вказівники на перший останній елементи списку

public:

List();

List(bool who);

bool Ret\_type(); //Повернення типу списку

Node\* Ret\_ptr\_head\_node(); //Повернення вказівника на початок списку

void Create\_list(int numb, bool how); //Створення списку

void Add\_node(char new\_simb); //Додавання вузлу в кінець списку

bool Del\_node(int index\_node); //Видалення вузлу в списку

int Count\_nodes(); //Підрахування кількості елементів у списку

bool Swap\_nodes(int index\_first\_node, int index\_second\_node); //Зміна місцями двох вузлів

void Combi\_lists(Node\* ptr\_head\_first\_list, Node\* ptr\_head\_second\_list); //Об'єднання двох списків в один

bool Clear\_list(); //Видалення всіх вузлів у списку

void Print\_list(); //Виведення списку на екран

char Read\_list\_from\_file(ifstream& f\_cin); //Зчитування і додавання вузлів із списку

void Write\_list\_to\_file(ofstream& f\_cout, bool last\_element); //Записування списку в файл

~List();

};

class Node // Клас вузлу

{

public:

char simb; // Змінна символьного типу(інформація)

Node\* next = NULL, \* back = NULL; // Вказівники на наступний, попередній елементи

Node();

~Node();

};

ФУНКЦІЯ СТВОРЕННЯ СПИСКУ

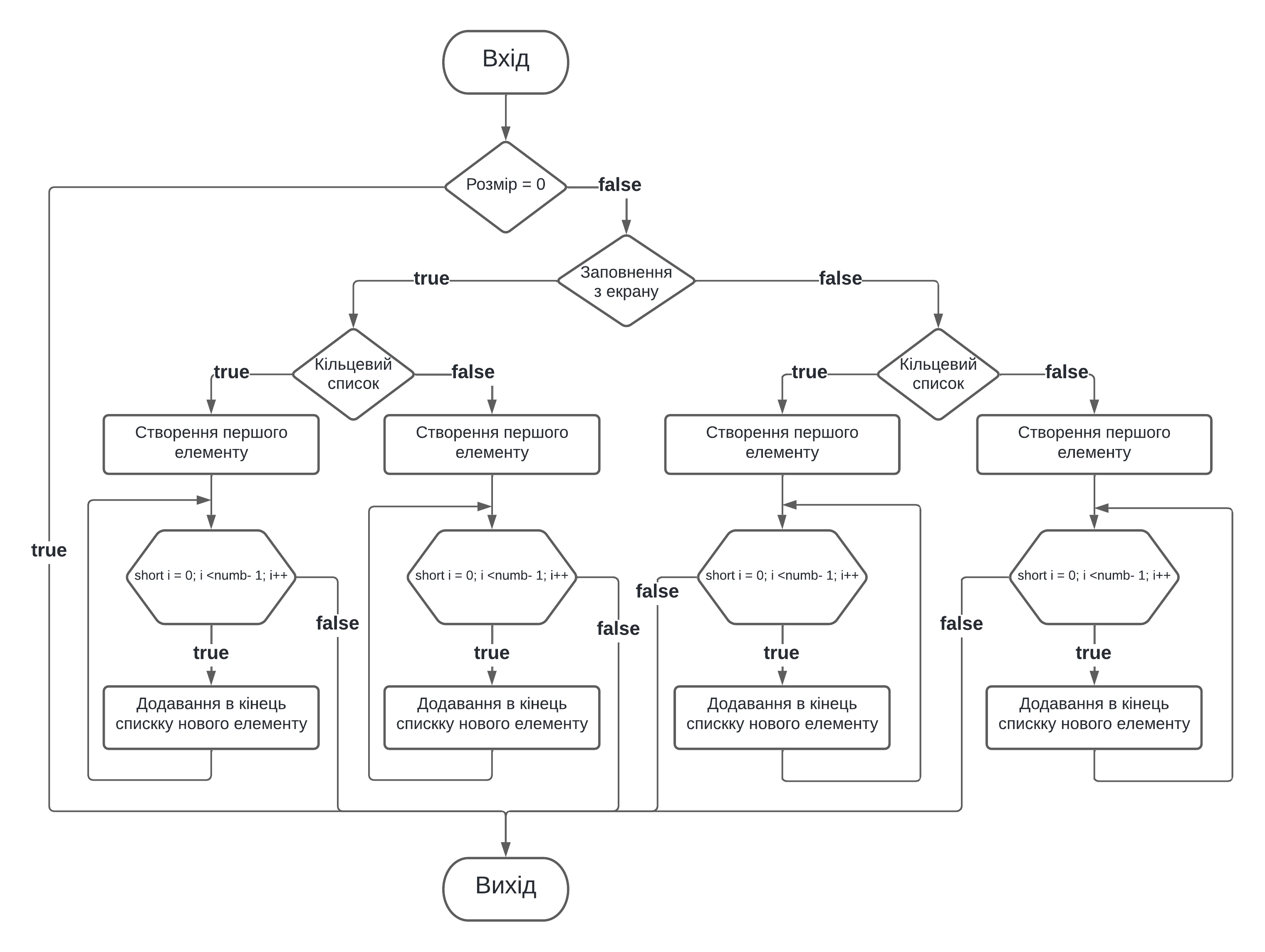


Рисунок 1 - блоксхема функції створення списку

void List::Create\_list(int numb, bool how) {

if (numb == 0) return;

if (how == true) {

cout << "\nВведіть символи через пробіл: ";

if (this->type == false) {

this->head = this->tail = new Node;

this->head->next = this->head->back = this->head;

cin >> this->tail->simb;

for (short i = 0; i < numb - 1; i++) {

this->tail->next = new Node;

this->tail->next->back = this->tail;

this->tail->next->next = this->head;

this->tail = this->tail->next;

this->head->back = this->tail;

cin >> this->tail->simb;

}

}

else {

this->head = this->tail = new Node;

cin >> this->head->simb;

for (short i = 0; i < numb - 1; i++) {

this->tail->next = new Node;

this->tail->next->back = this->tail;

this->tail = tail->next;

cin >> this->tail->simb;

}

}

}

else {

if (this->type == false) {

this->head = this->tail = new Node;

this->head->next = this->head->back = this->head;

this->tail->simb = (char)(rand() % 95 + 32);

for (short i = 0; i < numb - 1; i++) {

this->tail->next = new Node;

this->tail->next->back = this->tail;

this->tail->next->next = this->head;

this->tail = this->tail->next;

this->head->back = this->tail;

this->tail->simb = (char)(rand() % 95 + 32);

}

}

else {

this->head = this->tail = new Node;

this->tail->simb = (char)(rand() % 95 + 32);

for (short i = 0; i < numb - 1; i++) {

this->tail->next = new Node;

this->tail->next->back = this->tail;

this->tail = this->tail->next;

this->tail->simb = (char)(rand() % 95 + 32);

}

}

}

this->size = numb;

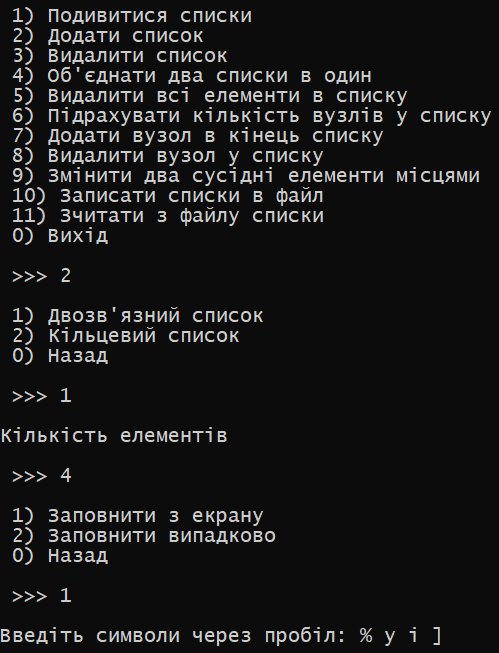
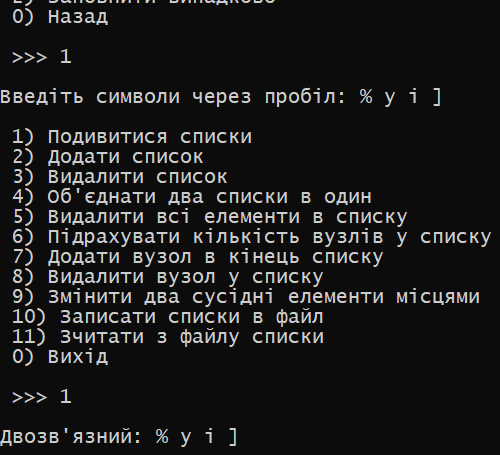
}

Рисунок 3 - Результат виконання по створенню двозв’язного списку

Рисунок 2 - Виклик і введення значень для створення двозв’язного списку

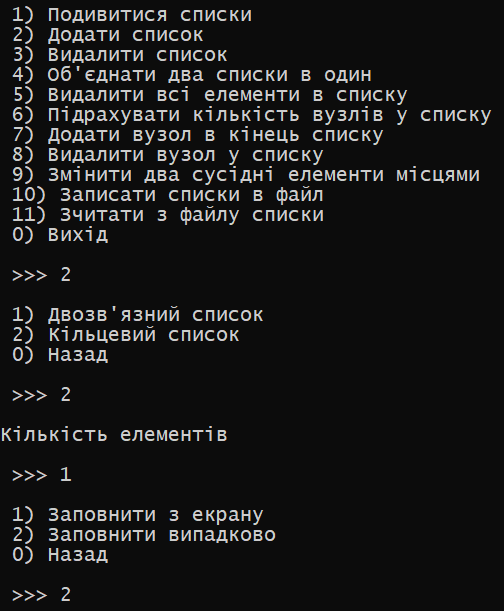
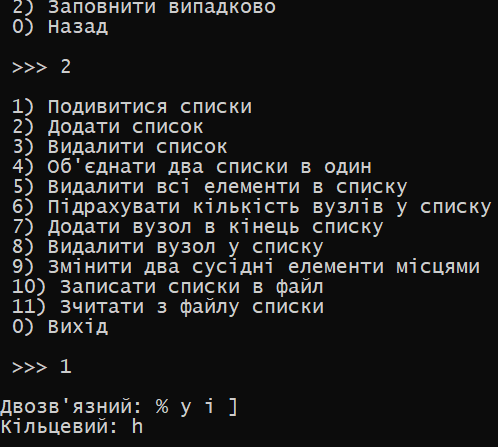


Рисунок 5 - Результат виконання по створенню кільцевого списку

Рисунок 4 - Виклик функції для створення кільцевого списку

Даний метод приймає два параметри, перший кількість елементів, а другий яким способом записувати дані, з екрану вводити, чи випадково.

ФУНКЦІЯ ДОДАВАННЯ ВУЗЛУ В КІНЕЦЬ СПИСКУ

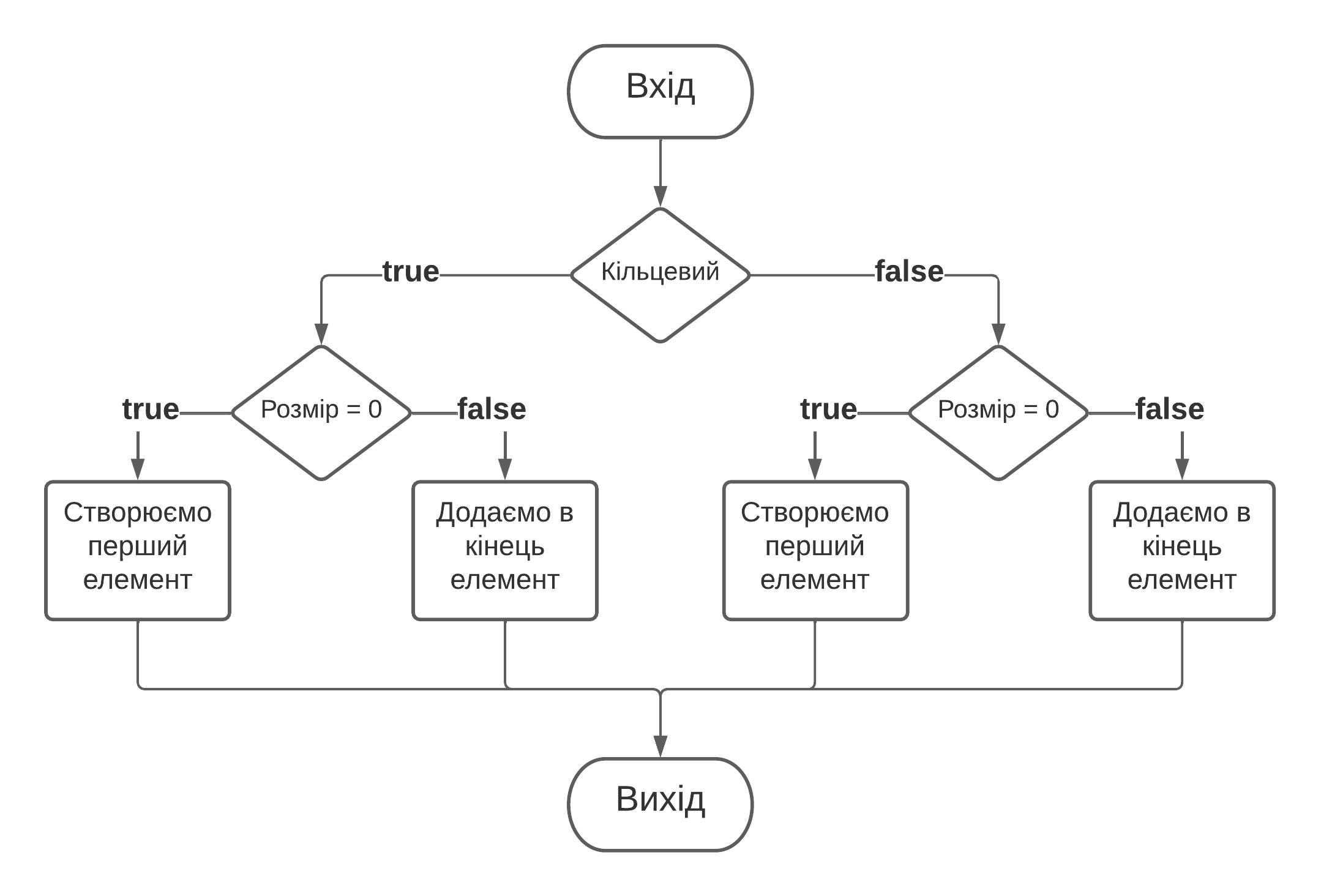


Рисунок 6 - блоксхема функції додавання елементу в кінець списку

void List::Add\_node(char new\_simb) {

if (this->type == false)

{

if (this->size == 0)

{

this->head = this->tail = new Node;

this->head->next = this->head->back = this->head;

this->tail->simb = new\_simb;

}

else

{

tail->next = new Node;

tail->next->back = tail;

tail->next->next = head;

tail = tail->next;

head->back = tail;

this->tail->simb = new\_simb;

}

}

else

{

if (this->size == 0)

{

this->head = this->tail = new Node;

this->tail->simb = new\_simb;

}

else

{

tail->next = new Node;

tail->next->back = tail;

tail = tail->next;

this->tail->simb = new\_simb;

}

}

this->size++;

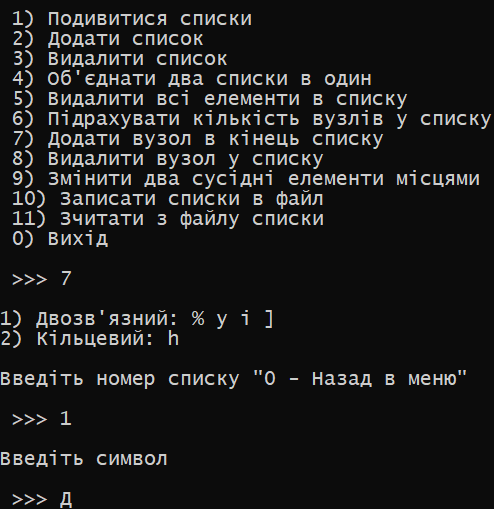
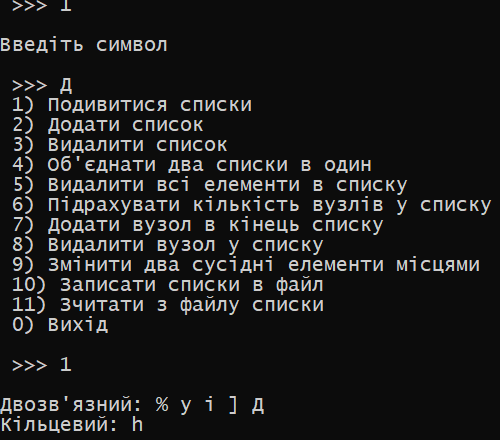
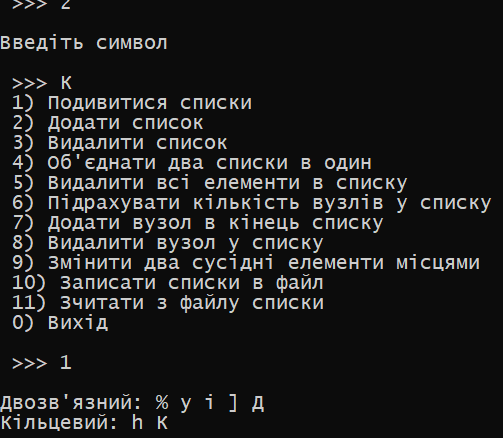
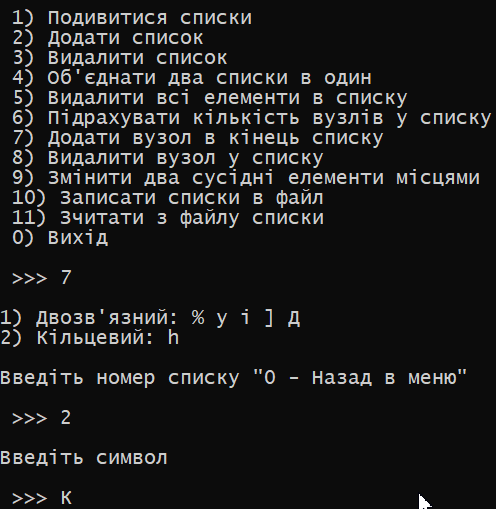
}

Рисунок 9 - Виклик функції додавання елементу в кінець списку для кільцевого списку

Рисунок 10 - Результат виконання для кільцевого списку

Рисунок 8 - Результат виконання для двозв'язного списку

Рисунок 7 - Виклик функції для додавання елементу в кінець двозв'язного списку

Дана функція отримує значення, яка буде додано в кінець.

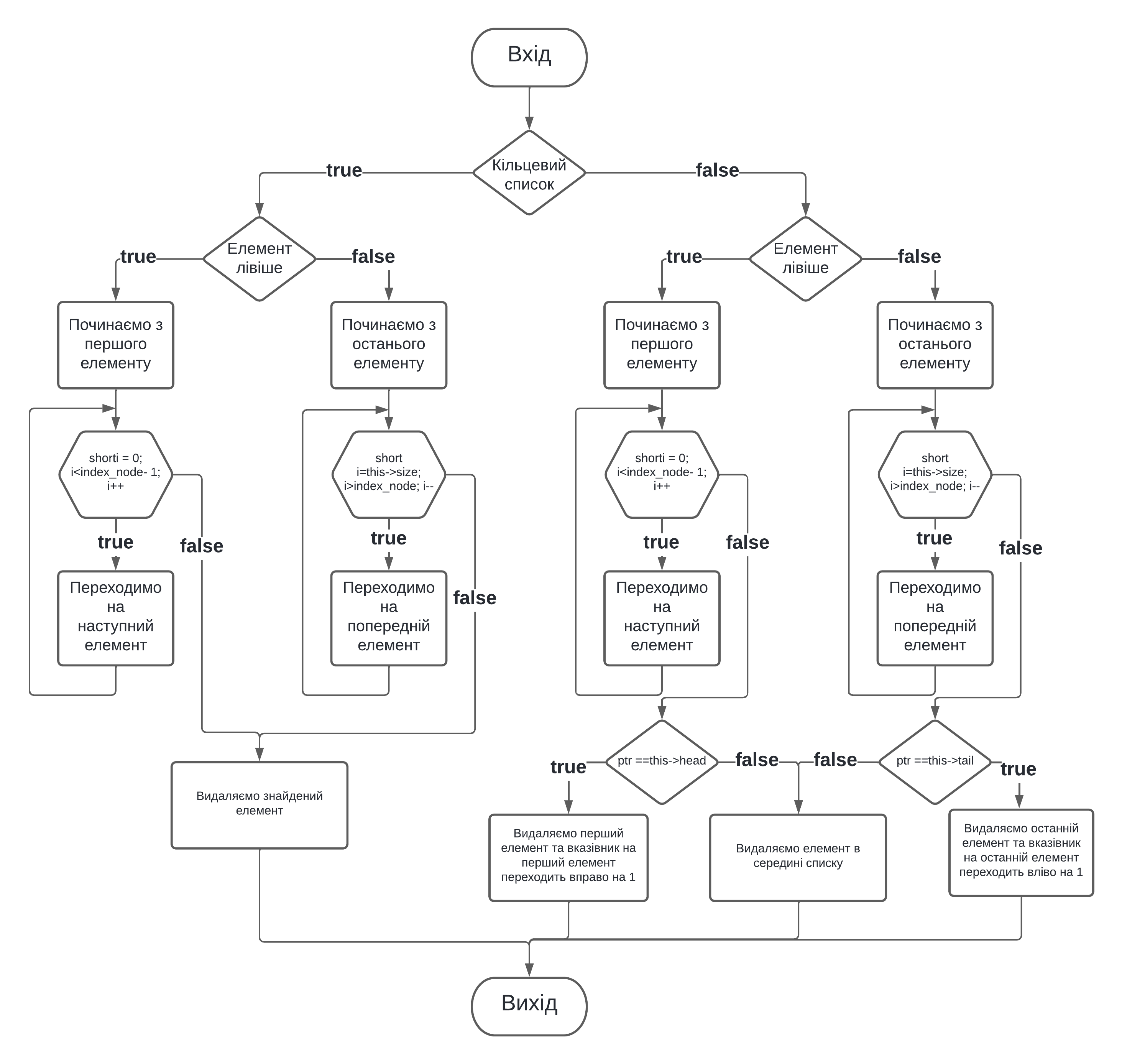
ФУНКЦІЯ ВИДАЛЕННЯ ВУЗЛУ ІЗ СПИСКУ

Рисунок 11 - блоксхема функції видалення елементу

bool List::Del\_node(int index\_node) {

Node\* ptr;

if (this->type == false) {

if (this->size / 2 >= index\_node) {

ptr = this->head;

for (short i = 0; i < index\_node - 1; i++)

ptr = ptr->next;

if (ptr == this->head)

this->head = this->head->next;

}

else {

ptr = this->tail;

for (short i = this->size; i > index\_node; i--)

ptr = ptr->back;

if (ptr == this->tail)

this->tail = this->tail->back;

}

ptr->back->next = ptr->next;

ptr->next->back = ptr->back;

delete ptr;

}

else

{

if (this->size / 2 >= index\_node)

{

ptr = this->head;

for (short i = 0; i < index\_node - 1; i++)

ptr = ptr->next;

if (ptr == this->head)

{

this->head = this->head->next;

this->head->back = NULL;

delete ptr;

this->size--;

return true;

}

}

else

{

ptr = this->tail;

for (short i = this->size; i > index\_node; i--)

ptr = ptr->back;

if (ptr == this->tail)

{

this->tail = this->tail->back;

this->tail->next = NULL;

delete ptr;

this->size--;

return true;

}

}

ptr->back->next = ptr->next;

ptr->next->back = ptr->back;

delete ptr;

}

this->size--;

return true;

}

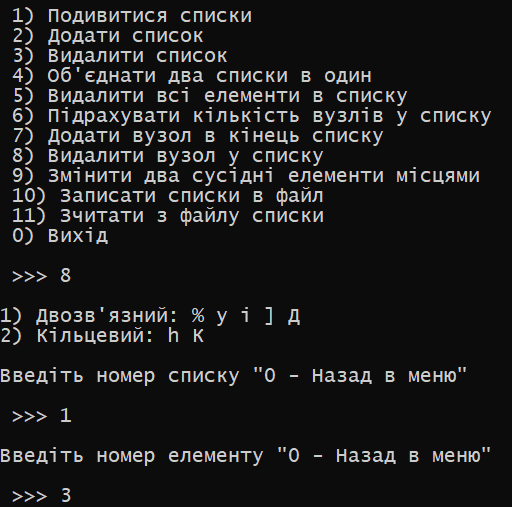
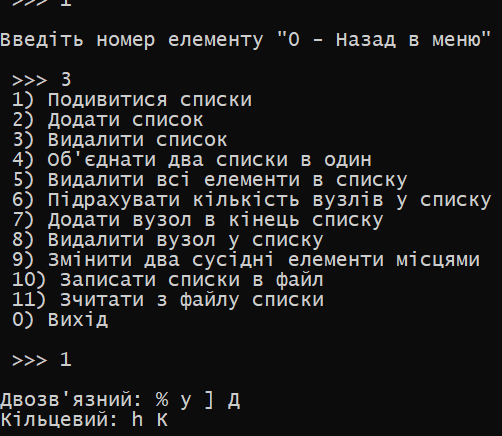
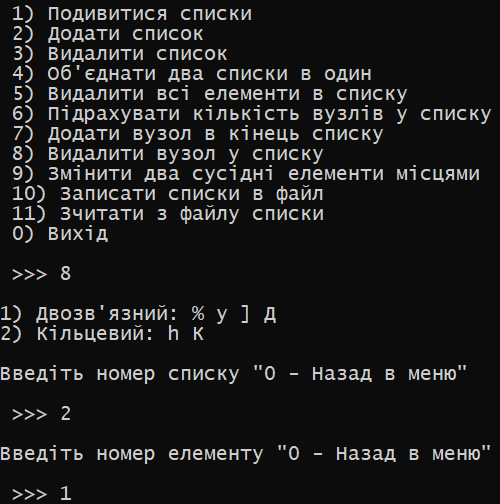
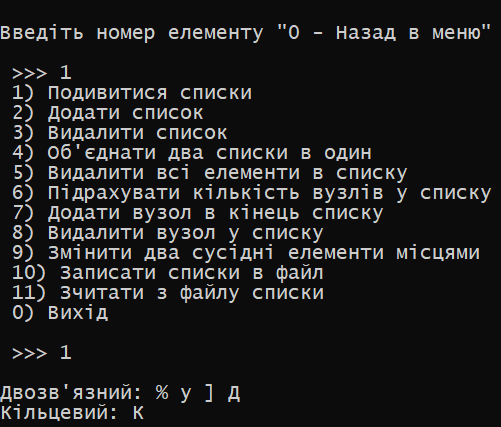


Рисунок 15 - Результат виконання функції для кільцевого списку

Рисунок 14 - Виклик функції для видалення елементу в кільцевому списку

Рисунок 13 - Результат виконання функції для двозв'язного списку

Рисунок 12 - Виклик функції для видалення елементу в двозв'язному списку

Функція, приймає індекс елемента, якого потрібно видали, вона повертає істину, якщо елемент було видалено та неправду, якщо такого елементу не було у списку, тобто елемент не видалено.

ФУНКЦІЯ ПІДРАХУВАНН КІЛЬКОСТІ ЕЛЕМЕНТІВ У СПИСКУ

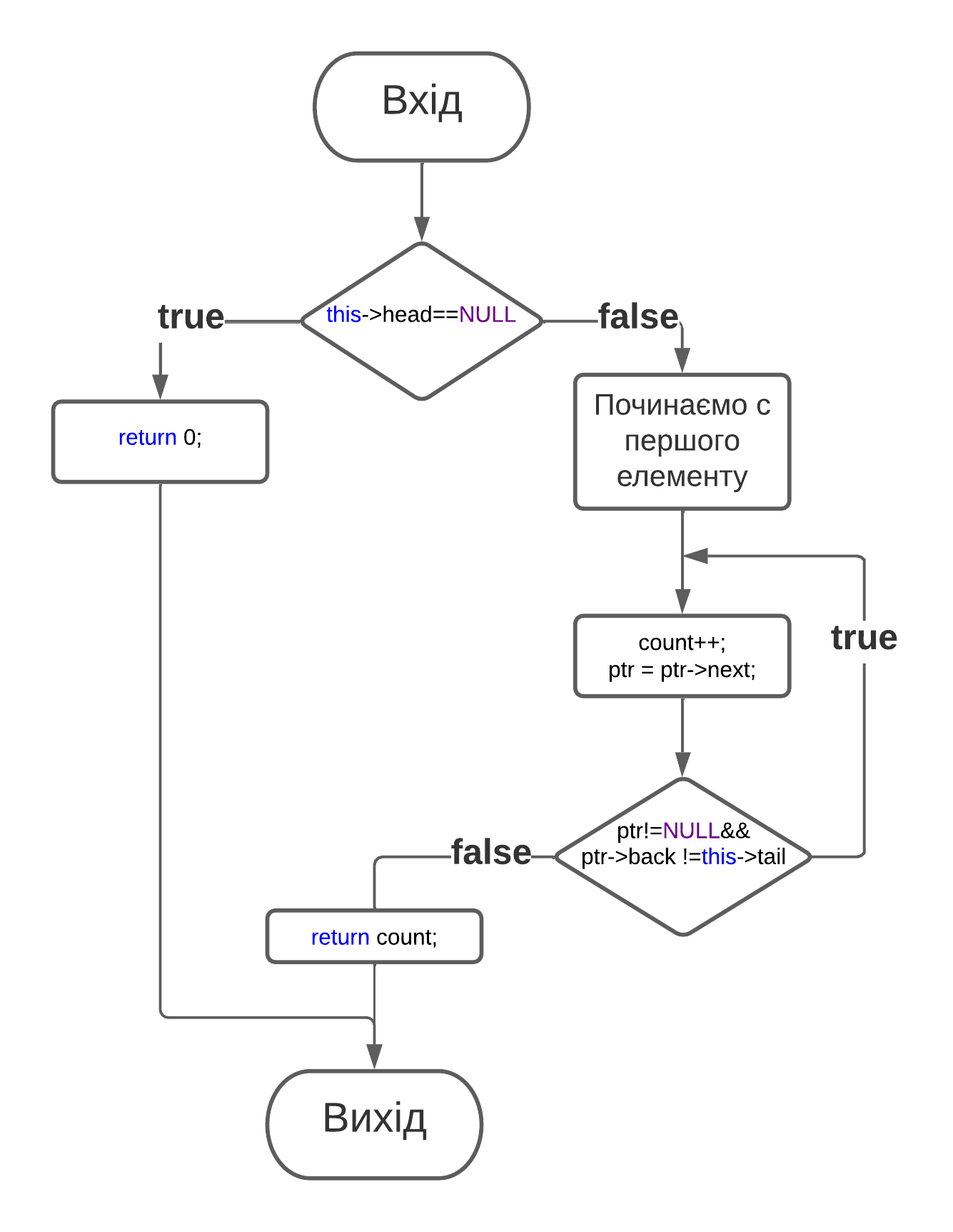


Рисунок 16 - блоксхема функції підрахування кількості елементів у списку

int List::Count\_nodes() {

if (this->head == NULL) return 0;

int count = 0;

Node\* ptr = this->head;

do {

count++;

ptr = ptr->next;

} while (ptr != NULL && ptr->back != this->tail);

return count;

}

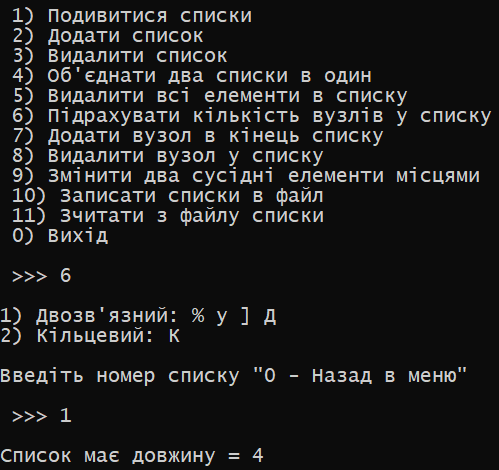
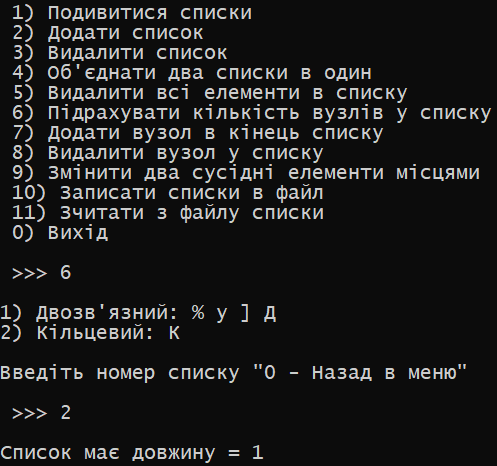


Рисунок 18 - Результат виклику функції підрахування кількості елементів у кільцевому списку

Рисунок 17 - Результат виклику функції підрахування кількості елементів у двозв’язному списку

Функція повертає значення кількості елементів у списку.

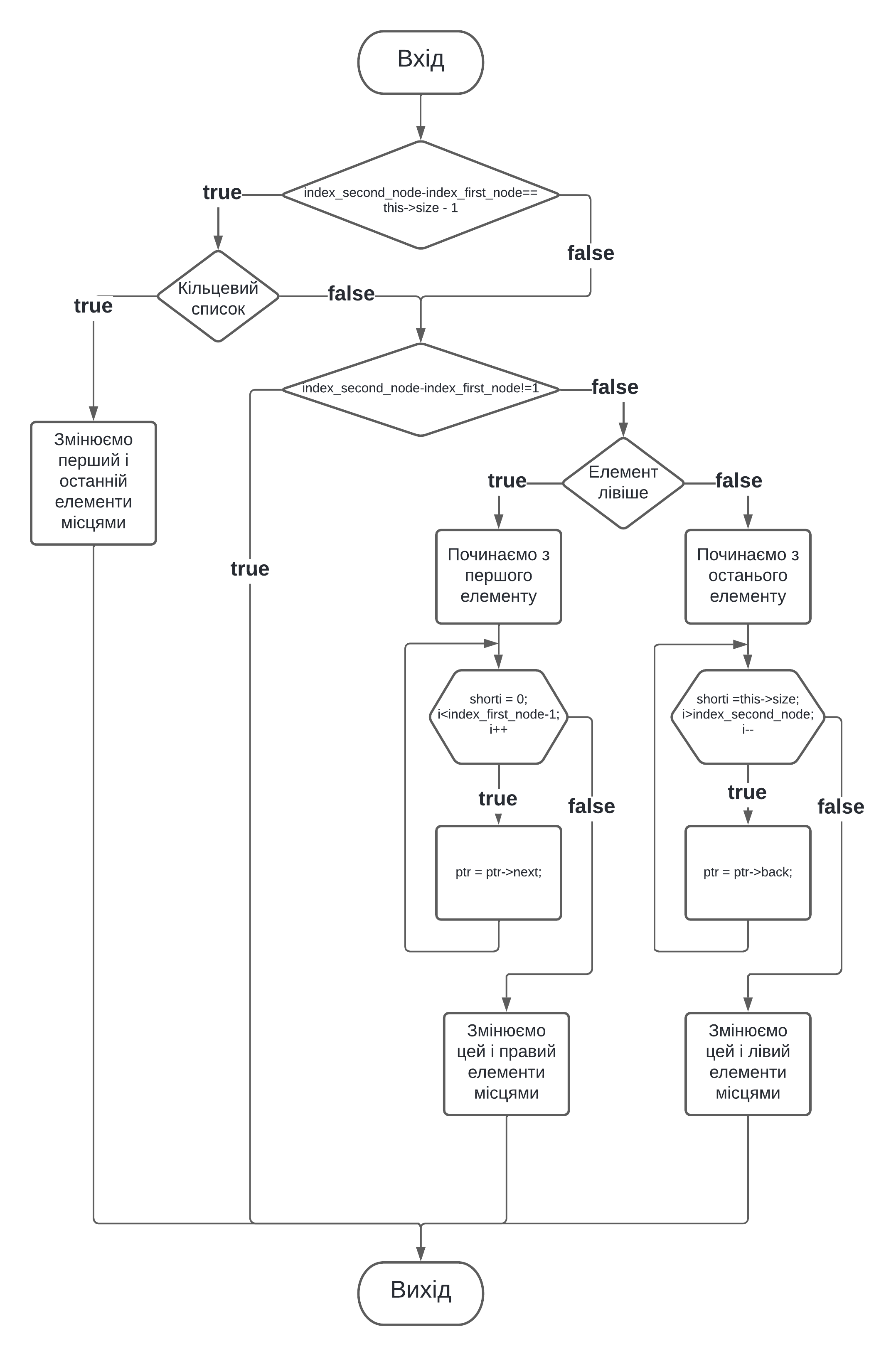
ФУНКЦІЯ ЗМІНИ ДВОХ СУСІДНІХ ЕЛЕМЕНТІВ МІСЦЯМИ

Рисунок 19 - блоксхема функції зміни двох сусідніх елементів списку

bool List::Swap\_nodes(int index\_first\_node, int index\_second\_node) {

char temp;

Node\* ptr = NULL;

if (index\_second\_node - index\_first\_node == this->size - 1) {

if (this->type == false) {

temp = this->head->simb;

this->head->simb = this->tail->simb;

this->tail->simb = temp;

return true;

}

}

if (index\_second\_node - index\_first\_node != 1)

return false;

if (this->size / 2 >= index\_first\_node) {

ptr = this->head;

for (short i = 0; i < index\_first\_node - 1; i++)

ptr = ptr->next;

temp = ptr->simb;

ptr->simb = ptr->next->simb;

ptr->next->simb = temp;

}

else {

ptr = this->tail;

for (short i = this->size; i > index\_second\_node; i--)

ptr = ptr->back;

temp = ptr->simb;

ptr->simb = ptr->back->simb;

ptr->back->simb = temp;

}

return true;

}

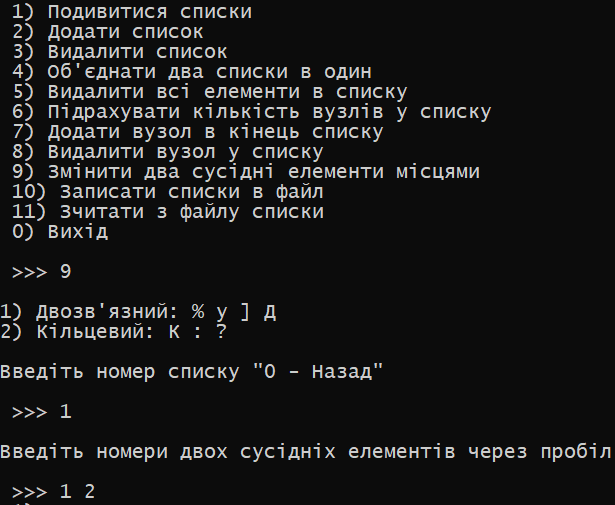
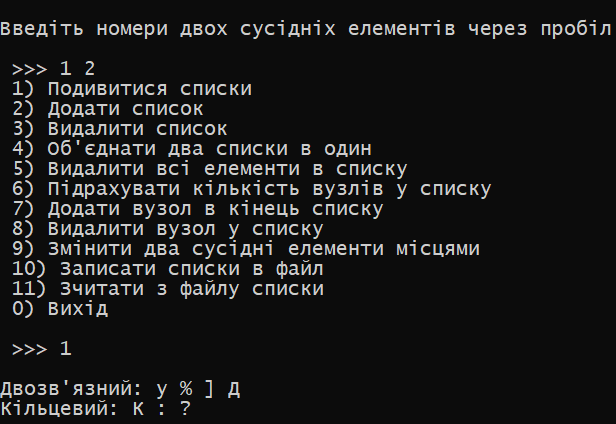


Рисунок 21 - Результат виконання функції для двозв'язного списку

Рисунок 20 - Виклик функції для двозв'язного списку

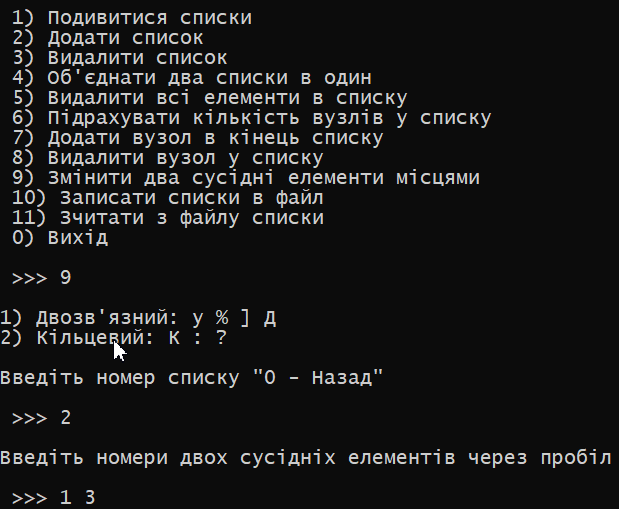
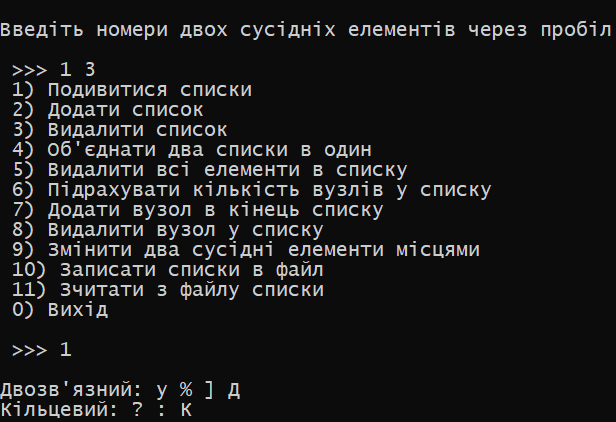
Даний метод приймає індекси лівого та правого елементів та повертає значення істини, якщо елементи було змінено та неправду, якщо елементи не є сусідніми

Рисунок 23 - Результат виконання функції для кільцевого списку

Рисунок 22 - Виклик функції для кільцевого списку

ФУНКЦІЯ ОБ’ЄДНАННЯ ДВОХ СПИСКІВ У ОДИН

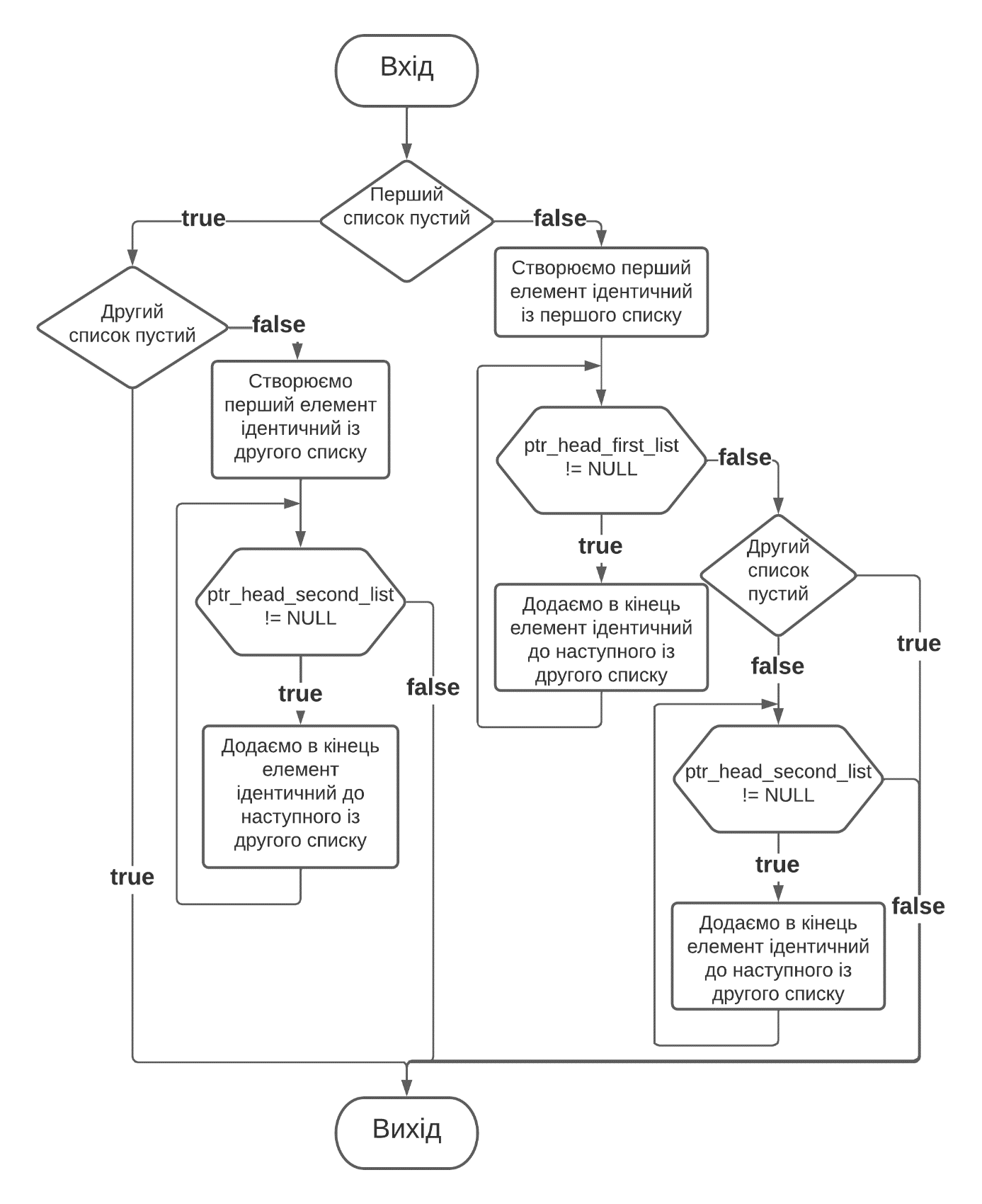


Рисунок 24 - блоксхема функції об'єднання двох списків в один

void List::Combi\_lists(Node\* ptr\_head\_first\_list, Node\* ptr\_head\_second\_list) {

if (ptr\_head\_first\_list == NULL) {

if (ptr\_head\_second\_list == NULL)

return;

this->head = this->tail = new Node;

this->tail->simb = ptr\_head\_second\_list->simb;

ptr\_head\_second\_list = ptr\_head\_second\_list->next;

this->size++;

while (ptr\_head\_second\_list != NULL) {

this->tail->next = new Node;

this->tail->next->back = this->tail;

this->tail = this->tail->next;

this->tail->simb = ptr\_head\_second\_list->simb;

ptr\_head\_second\_list = ptr\_head\_second\_list->next;

this->size++;

}

}

else {

this->head = this->tail = new Node;

this->tail->simb = ptr\_head\_first\_list->simb;

ptr\_head\_first\_list = ptr\_head\_first\_list->next;

this->size++;

while (ptr\_head\_first\_list != NULL) {

this->tail->next = new Node;

this->tail->next->back = this->tail;

this->tail = this->tail->next;

this->tail->simb = ptr\_head\_first\_list->simb;

ptr\_head\_first\_list = ptr\_head\_first\_list->next;

this->size++;

}

if (ptr\_head\_second\_list == NULL)

return;

while (ptr\_head\_second\_list != NULL)

{

this->tail->next = new Node;

this->tail->next->back = this->tail;

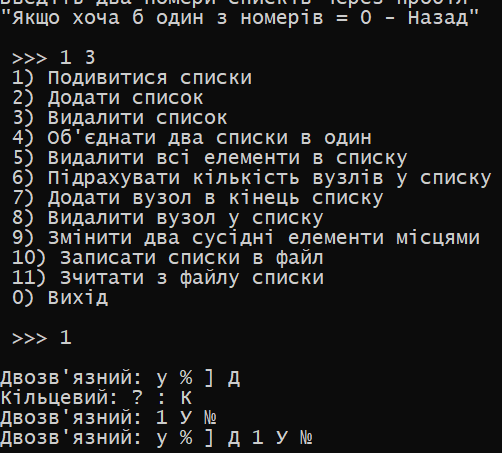
this->tail = this->tail->next;

this->tail->simb = ptr\_head\_second\_list->simb;

ptr\_head\_second\_list = ptr\_head\_second\_list->next;

this->size++;

}

 }

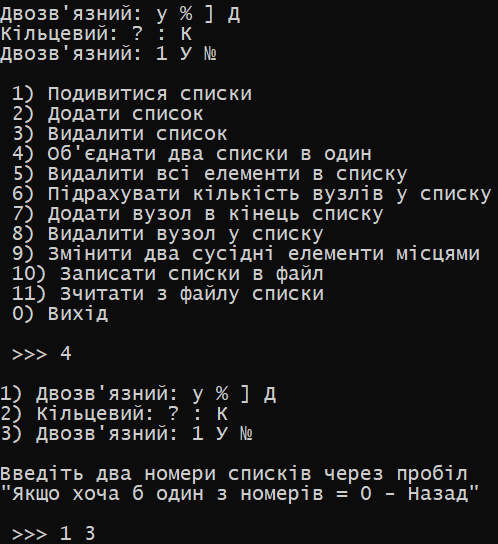
}

Рисунок 25 - Виклик функції об'єднання двох двозв'язних списків в один

Рисунок 26 - Результат виконання функції об'єднання двох двозв'язних списків в один

Приймає в якості аргументів, вказівники на перший елемент першого та другого списків.

ФУНКЦІЯ ВИДАЛЕННЯ ВСІХ ЕЛЕМЕНТІВ У СПИСКУ

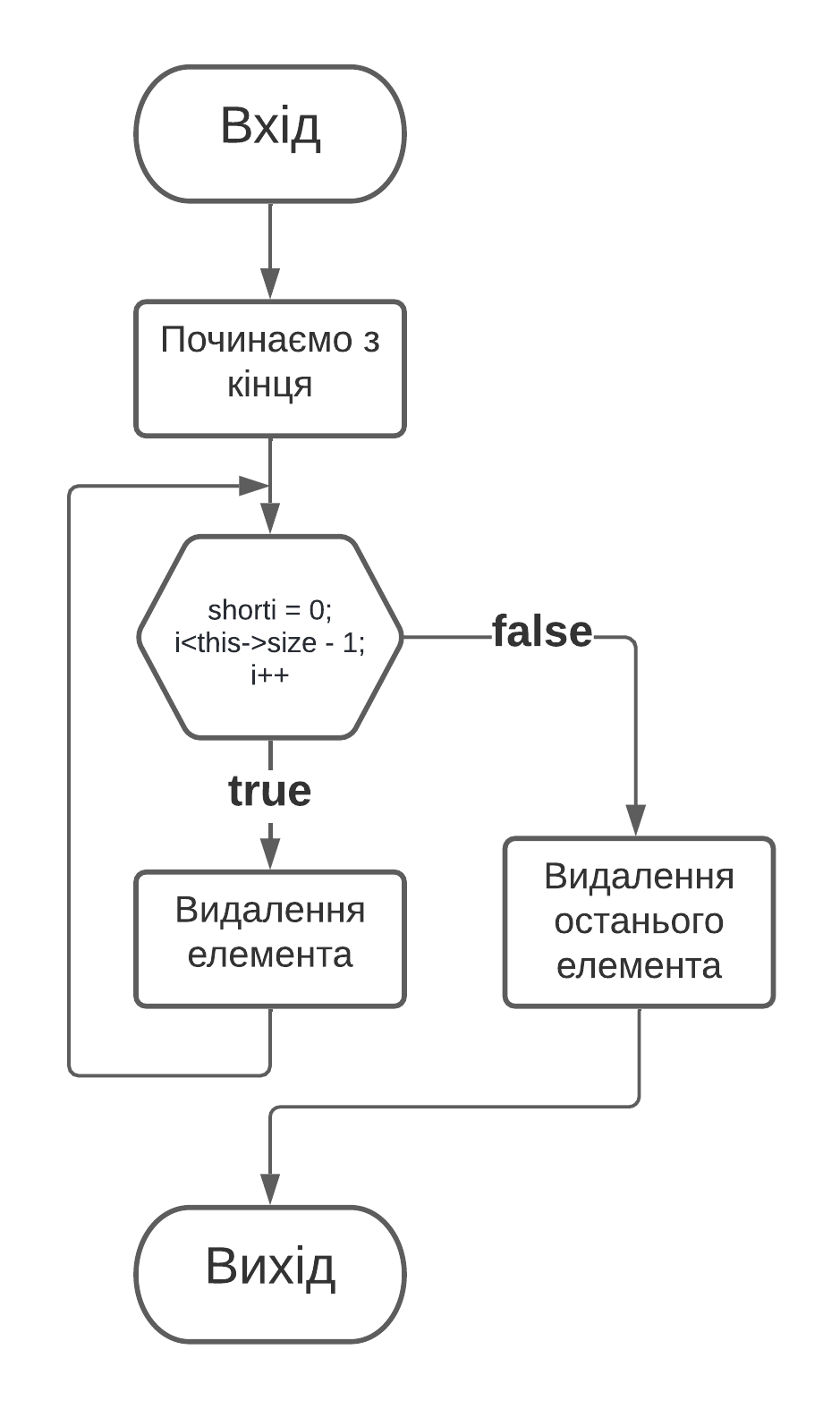


Рисунок 27 - блоксхема функції видалення всіх елементів списку

bool List::Clear\_list() {

Node\* ptr = this->tail;

for (short i = 0; i < this->size - 1; i++){

this->tail = this->tail->back;

delete ptr;

ptr = this->tail;

}

delete this->head;

this->head = this->tail = NULL;

this->size = 0;

return true;

}

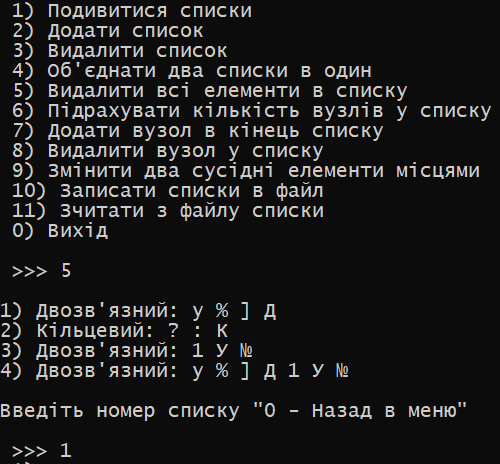
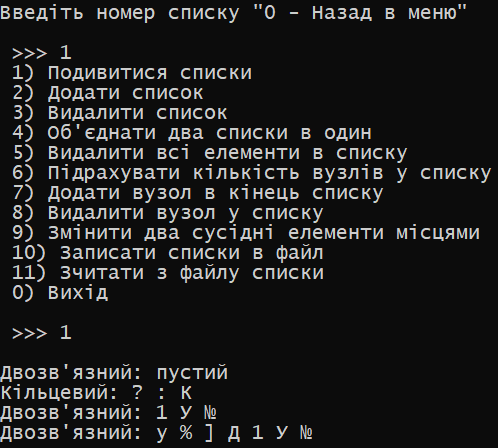
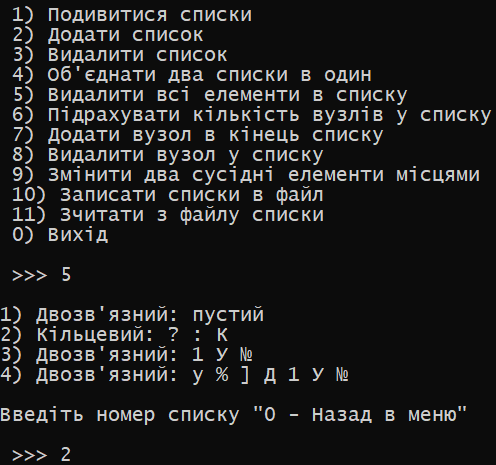
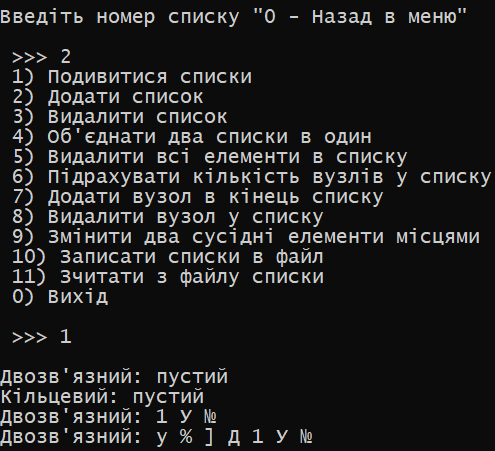


Рисунок 31 - Результат виконання функції по очищенню кільцевого списку

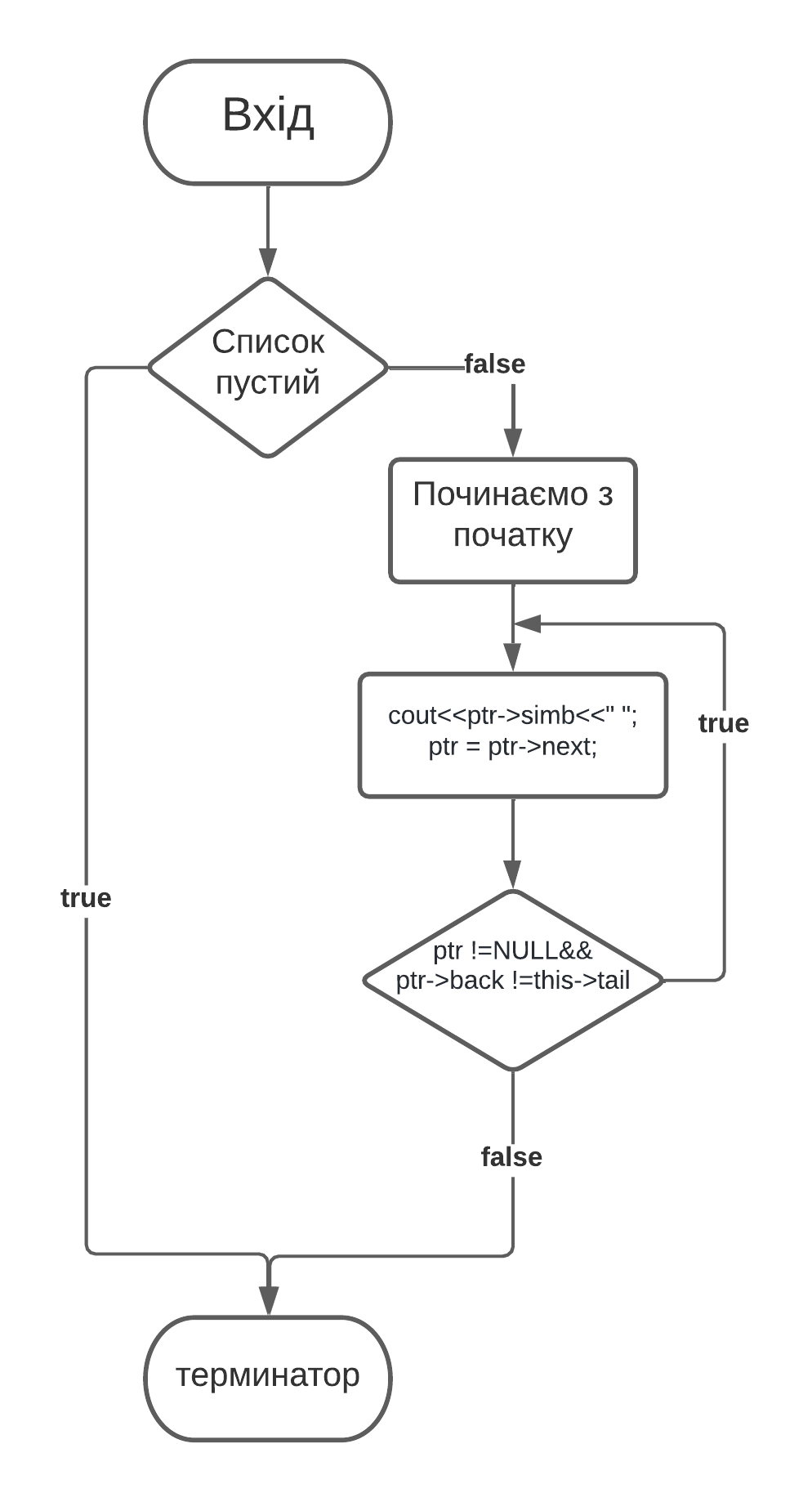
Рисунок 30 - Виклик функції по видаленню всіх елементів у кільцевому списку

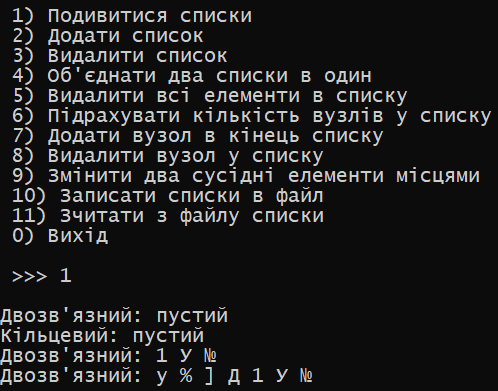
Рисунок 29 - Результат виконання функції по очищенню двозв’язного списку

Рисунок 28 - Виклик функції по видаленню всіх елементів для двозв'язного списку

Метод повертає істину, якщо елементи були видалено та неправду, якщо список пустий і видаляти нема чого.

ФУНКЦІЯ ВИВЕДЕННЯ СПИСКУ НА ЕКРАН



Рисунок 32 - блоксхема функції виведення списку на екран

void List::Print\_list() {

if (this->type == false)

cout << "Кільцевий: ";

else

cout << "Двозв'язний: ";

if (this->head == NULL) {

cout << "пустий\n";

return;

}

Node\* ptr = this->head;

do {

cout << ptr->simb << " ";

ptr = ptr->next;

Рисунок 33 - Результат виконання

} while (ptr != NULL && ptr->back != this->tail);

cout << endl;

}

ФУНКЦІЯ ЗЧИТУВАННЯ СПИСКУ ІЗ ФАЙЛУ

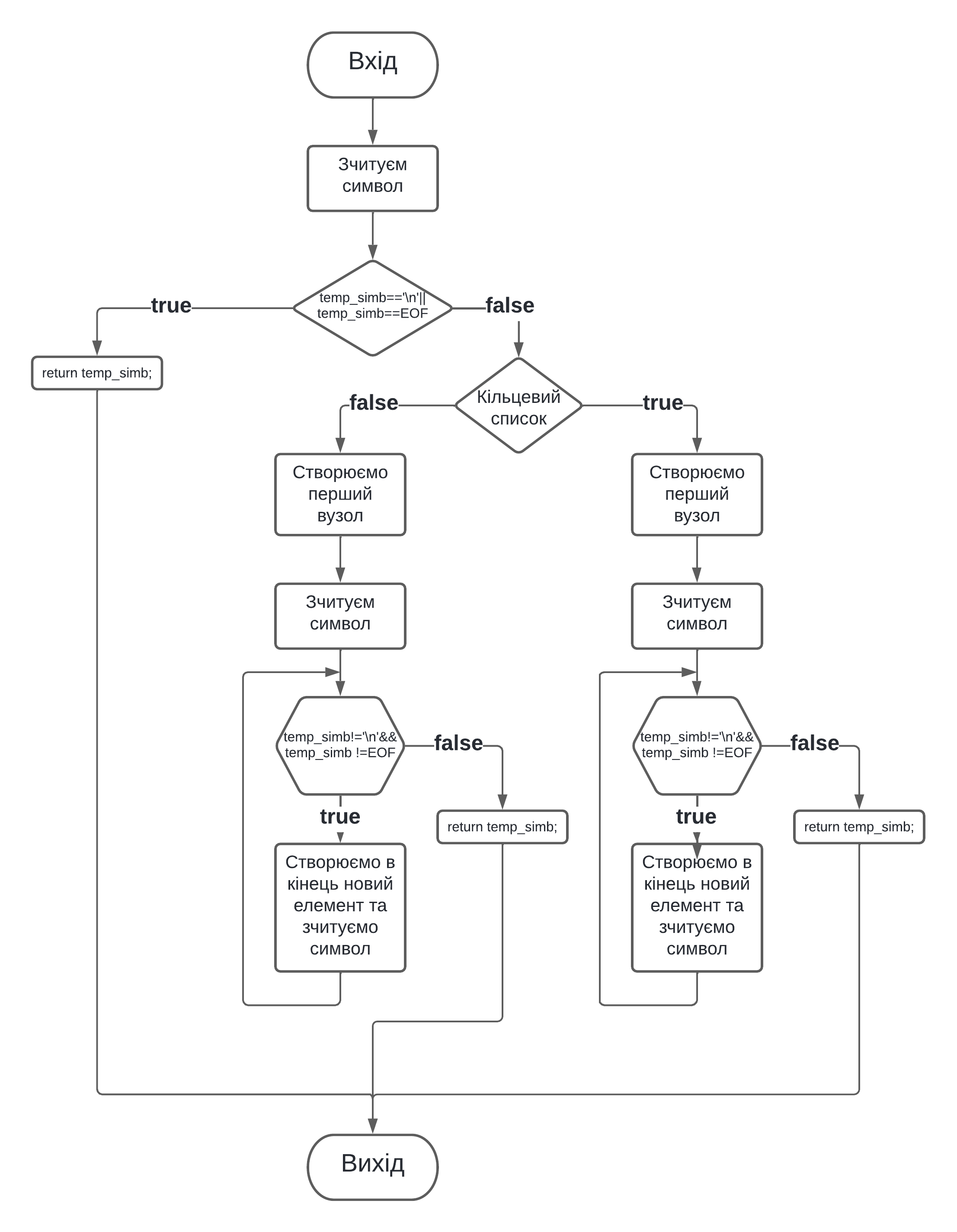


Рисунок 34 - блоксхема функції зчитування списку з файлу

char List::Read\_list\_from\_file(ifstream& f\_cin) {

char temp\_simb;

f\_cin.get();

temp\_simb = f\_cin.get();

if (temp\_simb == '\n' || temp\_simb == EOF)

return temp\_simb;

if (this->type == false) {

this->head = this->tail = new Node;

this->tail->next = this->tail->back = this->tail;

this->tail->simb = temp\_simb;

this->size++;

temp\_simb = f\_cin.get();

while (temp\_simb != '\n' && temp\_simb != EOF) {

this->tail->next = new Node;

this->tail->next->back = this->tail;

this->tail = this->tail->next;

this->tail->next = this->head;

this->head->back = this->tail;

this->tail->simb = temp\_simb;

this->size++;

temp\_simb = f\_cin.get();

}

}

else {

this->head = this->tail = new Node;

this->tail->simb = temp\_simb;

this->size++;

temp\_simb = f\_cin.get();

while (temp\_simb != '\n' && temp\_simb != EOF) {

this->tail->next = new Node;

this->tail->next->back = this->tail;

this->tail = this->tail->next;

this->tail->simb = temp\_simb;

this->size++;

temp\_simb = f\_cin.get();

}

}

return temp\_simb;

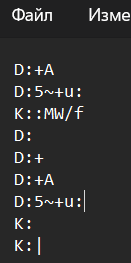
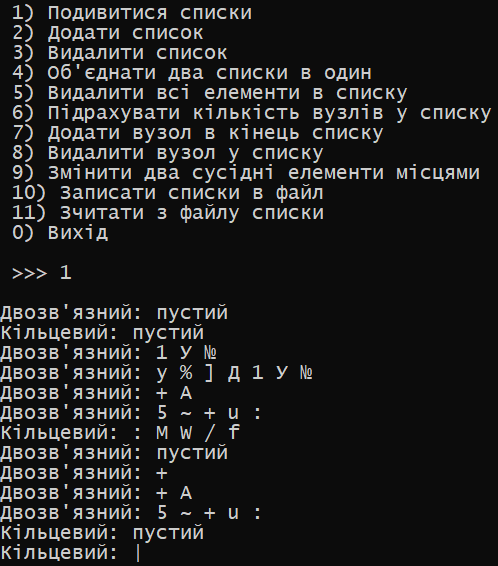
}

Рисунок 36 - Результат виконання функції по зчитуванню списків із файлу

Рисунок 35 - Записи в файлі

Функція отримує в якості аргументу об’єкт типу зчитування з файлу та повертає символ, який буде значити, кінець чи ні для зчитування і записування в список.

ФУНКЦІЯ ЗАПИСУВАННЯ СПИСКУ В ФАЙЛ

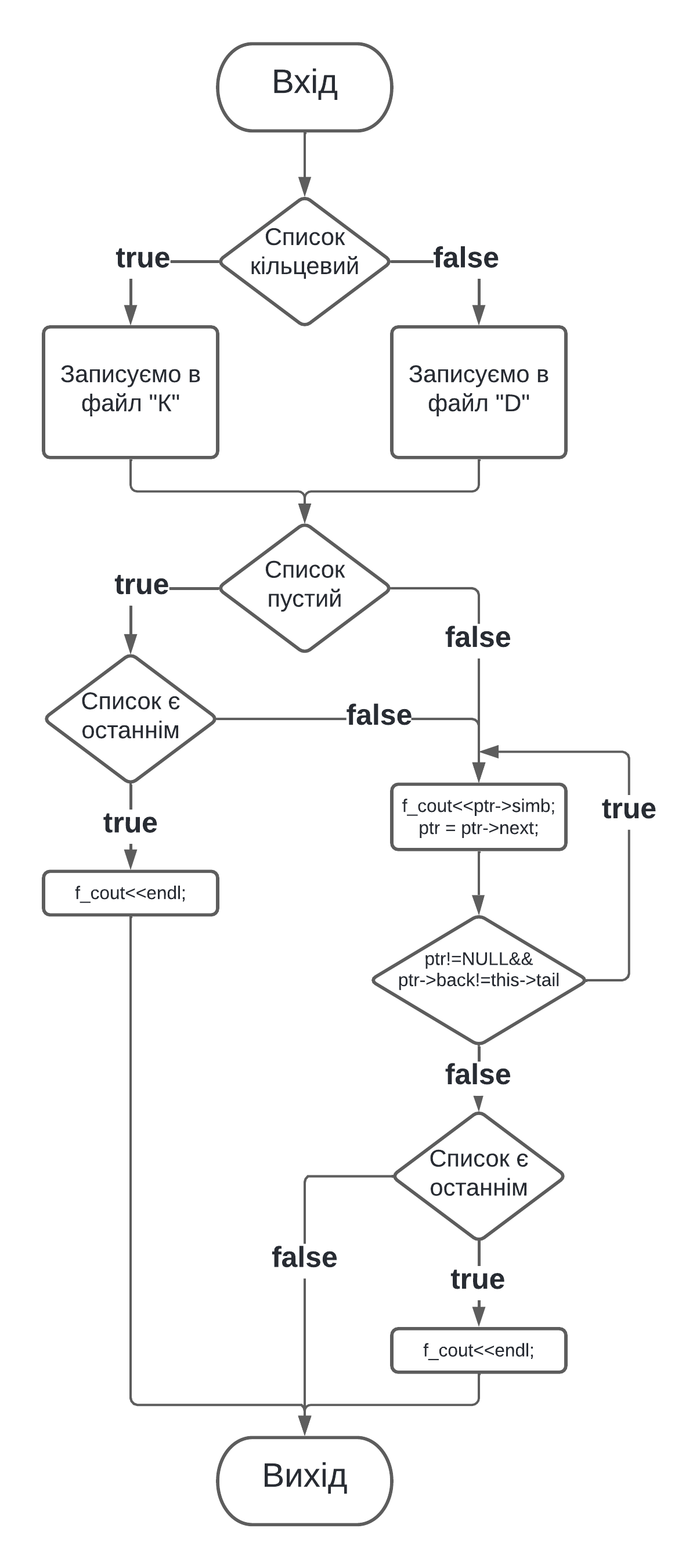


Рисунок 35 - блоксхема функції для запису списка в файл

void List::Write\_list\_to\_file(ofstream& f\_cout, bool last\_element) {

if (this->type == false) f\_cout << "K:";

else f\_cout << "D:";

if (this->head == NULL) {

if (last\_element == false)

f\_cout << endl;

return;

}

Node\* ptr = this->head;

do {

f\_cout << ptr->simb;

ptr = ptr->next;

} while (ptr != NULL && ptr->back != this->tail);

if (last\_element == false)

f\_cout << endl;

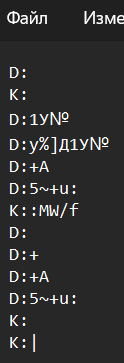
}

Рисунок 38 - Результат виконання запису списків у файл

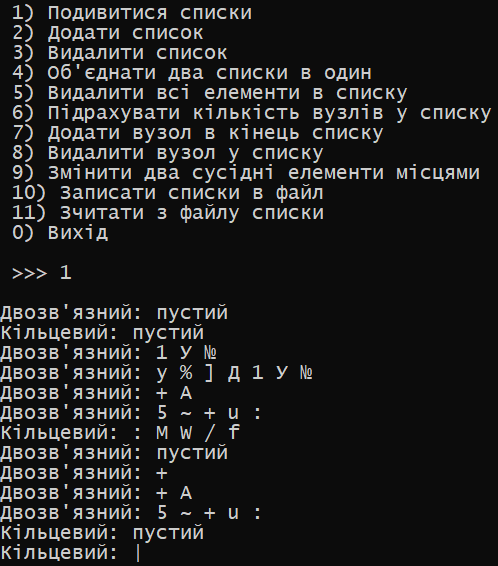


Рисунок 37 - Поточні списки в програмі

Метод приймає об’єкт типу для запису в файл та булеву зміну, яка вказує ставити в кінець перехід на наступний рядок, чи ні.

ВИСНОВОК

Отже, плюси і мінуси двозв’язних списків полягають у тому, що виділення відбувається дінамічно, тобто, можна контролювати скільки і де саме буде відбуватися створення елементу і можна без всяких додаткових виділень і створень копій, для коректування даних. Проте, у цих структур даних, є й недоліки, а саме, це досягання необхідного елементу, їх видалення та перев’язка цих вказівників, але, так як список є двозв’язним, то пошук виконується майже в 2 рази швидше, так як, знаючи з якої сторони ліпше дійти до елементу.