Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Стратегічного управління в сфері інформаційних технологій"

|  |  |
| --- | --- |
|  |  |

**Звітів до лабораторної роботи №1**

з дисципліни "Алгоритми на структури даних"

Виконав: студент групи КН-321А

Бородай Д. А.

Перевірив: старший викладач

Мошко Є.О.

Харків 2022

**Тема лабораторної роботи.** Однозв’язні та двозв’язні списки та робота з ними.

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

**Порядок виконання роботи**:

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

2. Написати програму, яка демонструє основні дії по роботі із однозв’язним списком і пропонує користувачу вибір дії:  
− сформувати список;  
− додати елемент до списку (в кінець);  
− вилучити вказаний елемент зі списку;  
− визначити кількість елементів у списку;  
− поміняти два сусідні елементи місцями;  
− об’єднати двох списків в один, з утворенням нового списку шляхом дописування другого списку в кінець першого списку;  
− очистити список.

3. Перетворити лінійний список в двозв’язний. Внести необхідні зміни в операції роботи зі списком.  
Написати програму, яка демонструє основні дії по роботі із двозв’язним списком і пропонує користувачу вибір дії:  
− сформувати список;  
− додати елемент до списку;  
− вилучити вказаний елемент зі списку;  
− визначити кількість елементів у списку;  
− поміняти два сусідні елементи місцями;  
− очистити список

# Завдання 1

Згідно таблиці варіантів завдань для другого варіанту тип змінної – змінна цілого типу. Отже списки у інформаційному полі матимуть змінну типу int. Оголошення структур списків з типом даної змінної наведено на рисунку 1.1.

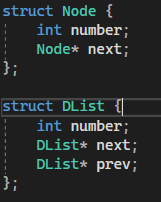


Рисунок 1.1 — Структури списків

Повний текст програми файлу заголовків наведено у додатку В.

# Завдання 2

Для початку роботи з однозв’язним списком було створено структуру Node, що продемонстровано на рисунку 2.1. У структурі інформаційного поля оголошено змінну number типу int та покажчик next на наступний елемент структури.

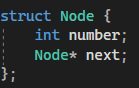
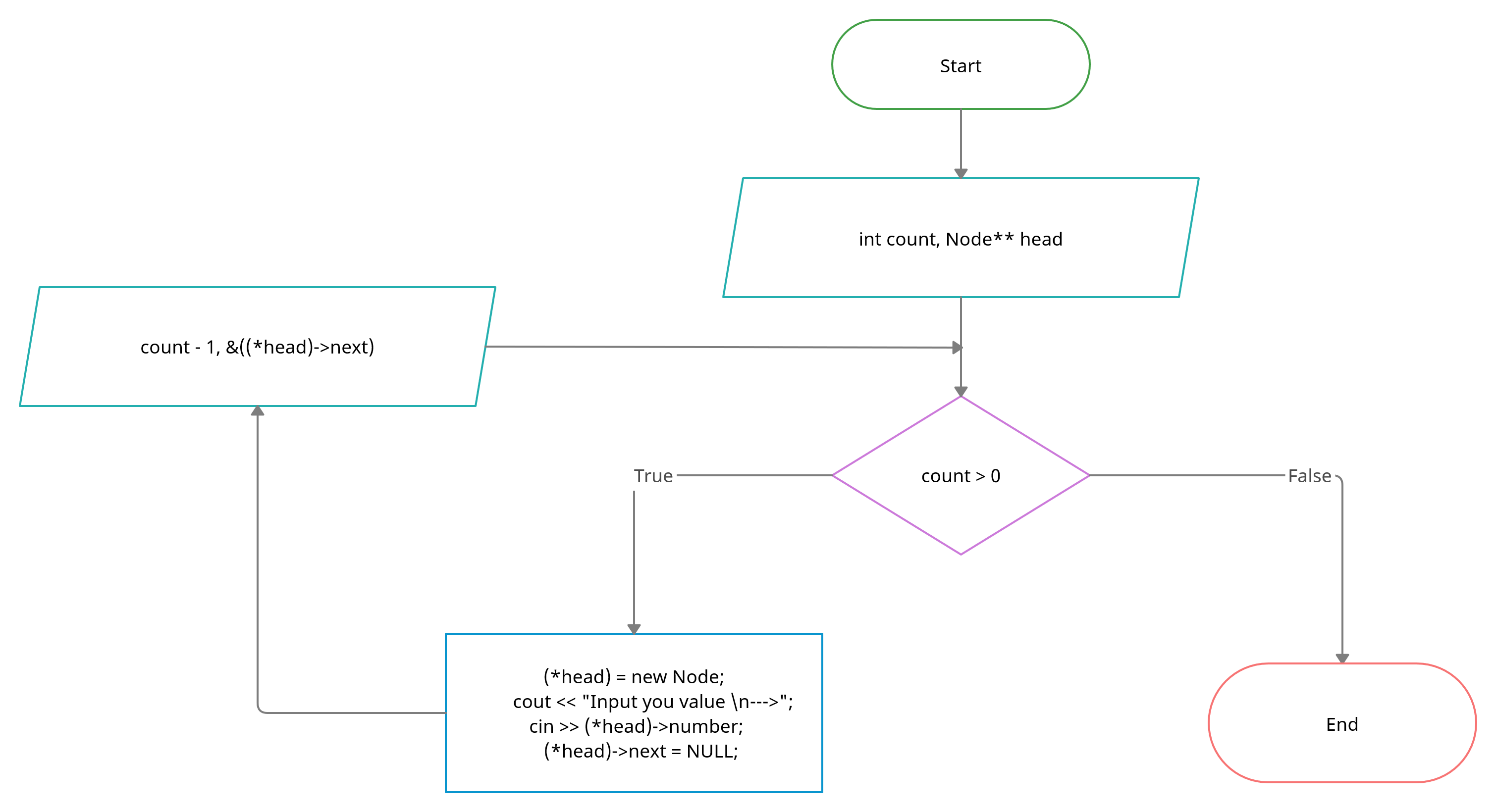


Рисунок 2.1 — Структура списку Node

**Створенная односвязного списку:**



Блок-схема 2.1— Алгоритм створення однозв’язного списку

void CreateNodeConsole(int count, Node\*\* head) {

if (count > 0) {

(\*head) = new Node;

cout << "Input you value \n--->"; cin >> (\*head)->number;

(\*head)->next = NULL;

CreateNodeConsole(count - 1, &((\*head)->next));

}

}

Повний текст програми наведений у додатку А.

Результат роботи алгоритму наведено на рисунках 2.2 та 2.3.

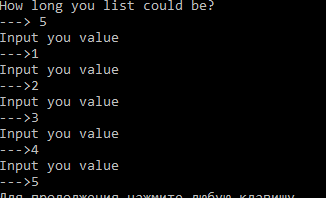
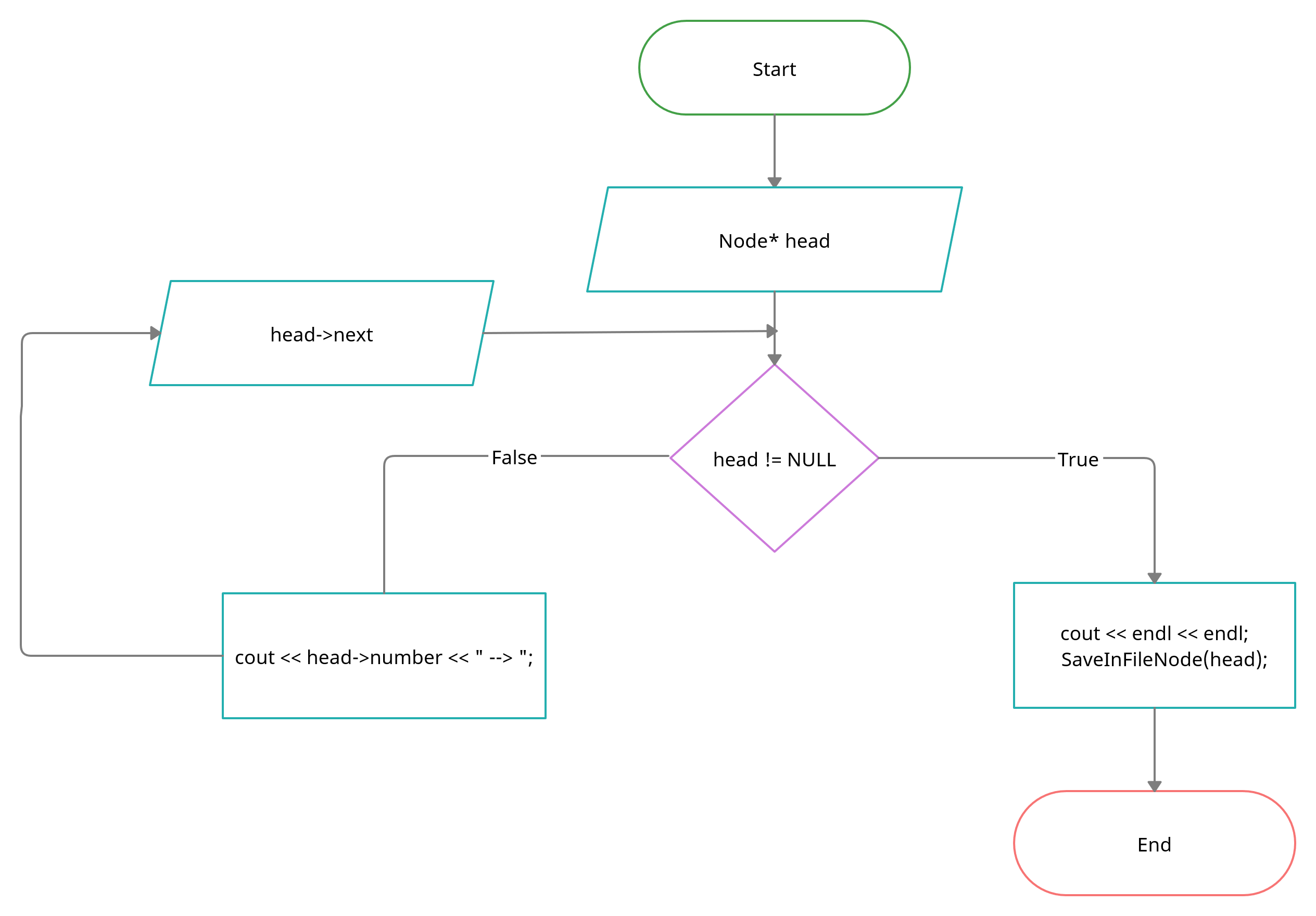


Рисунок 2.2 — заповнення списку цілими числами



Рисунок 2.3 — новостворений однозв’язний список

**Виведення однозв’язного списку у командний рядок**:



Блок-схема 2.2— Алгоритм виведення списку у командний рядок

void PrintNode(Node\* head) {

if (head != NULL) {

cout << head->number << " --> ";

PrintNode(head->next);

}

else cout << endl << endl;

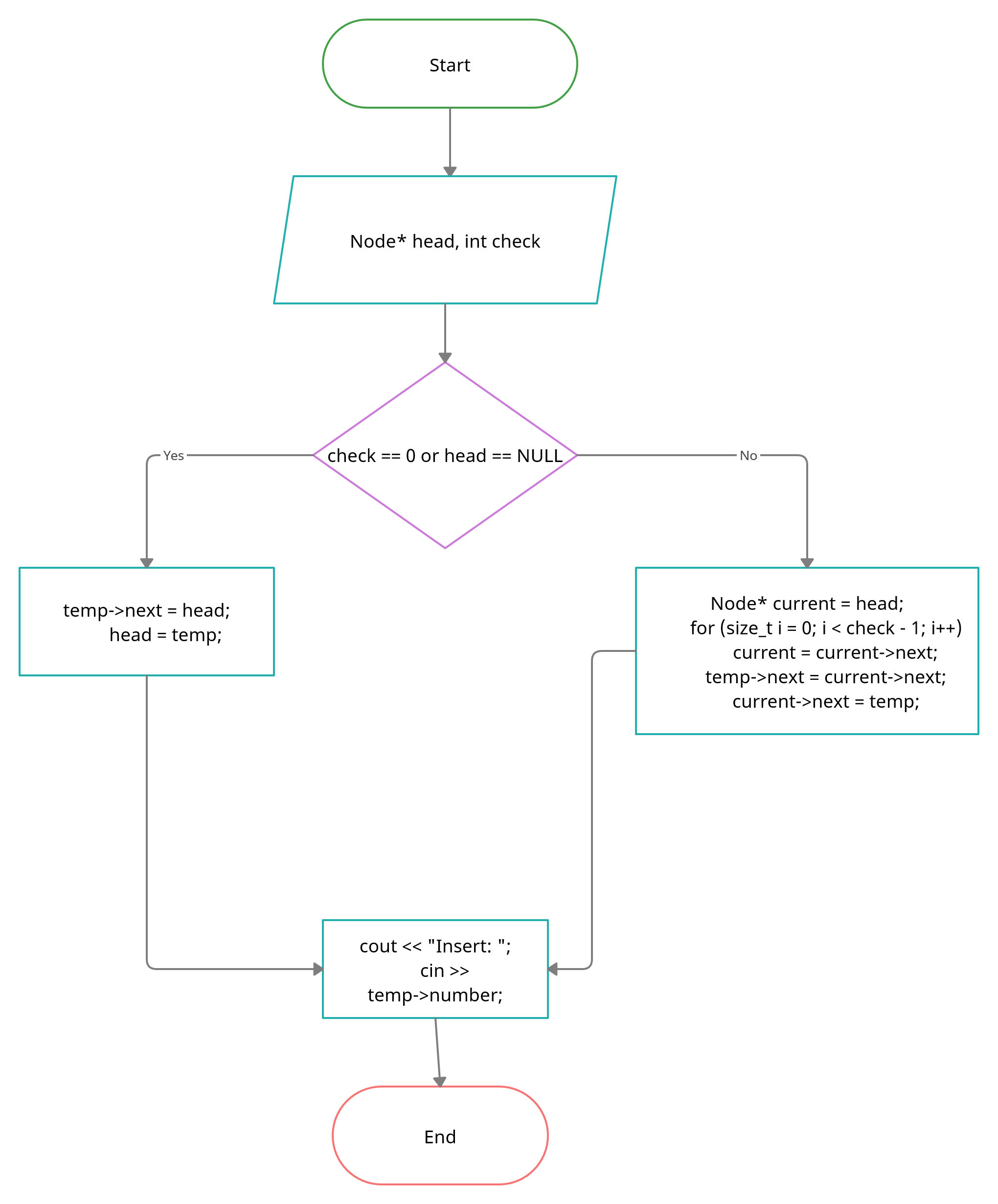
SaveInFileNode(head);

}

Повний текст програми наведений у додатку А.

Результат роботи алгоритму наведено на рисунку 2.3.

**Вставка нового елементу в список:**



Блок-схема 2.3— Алгоритм вставки нового елементу до списку

void Insert(Node\* head, int check) {

Node\* temp = new Node;

if (check == 0 or head == NULL) {

temp->next = head;

head = temp;

}

else {

Node\* current = head;

for (size\_t i = 0; i < check - 1; i++)

current = current->next;

temp->next = current->next;

current->next = temp;

}

cout << "Insert: ";

cin >> temp->number;

}

Повний текст програми наведений у додатку А.

Результат роботи алгоритму наведено на рисунку 2.4.

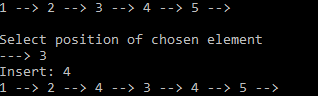


Рисунок 2.4— Результат вставки

**Видалення елементу:**

void Delete(Node\*\* head, int check) {

Node\* current = \*head;

if (check == 1) {

(\*head) = (\*head)->next;

delete current;

}

else {

for (size\_t i = 0; i < check - 1; i++)

current = current->next;

Node\* temp = current->next;

current->next = temp->next;

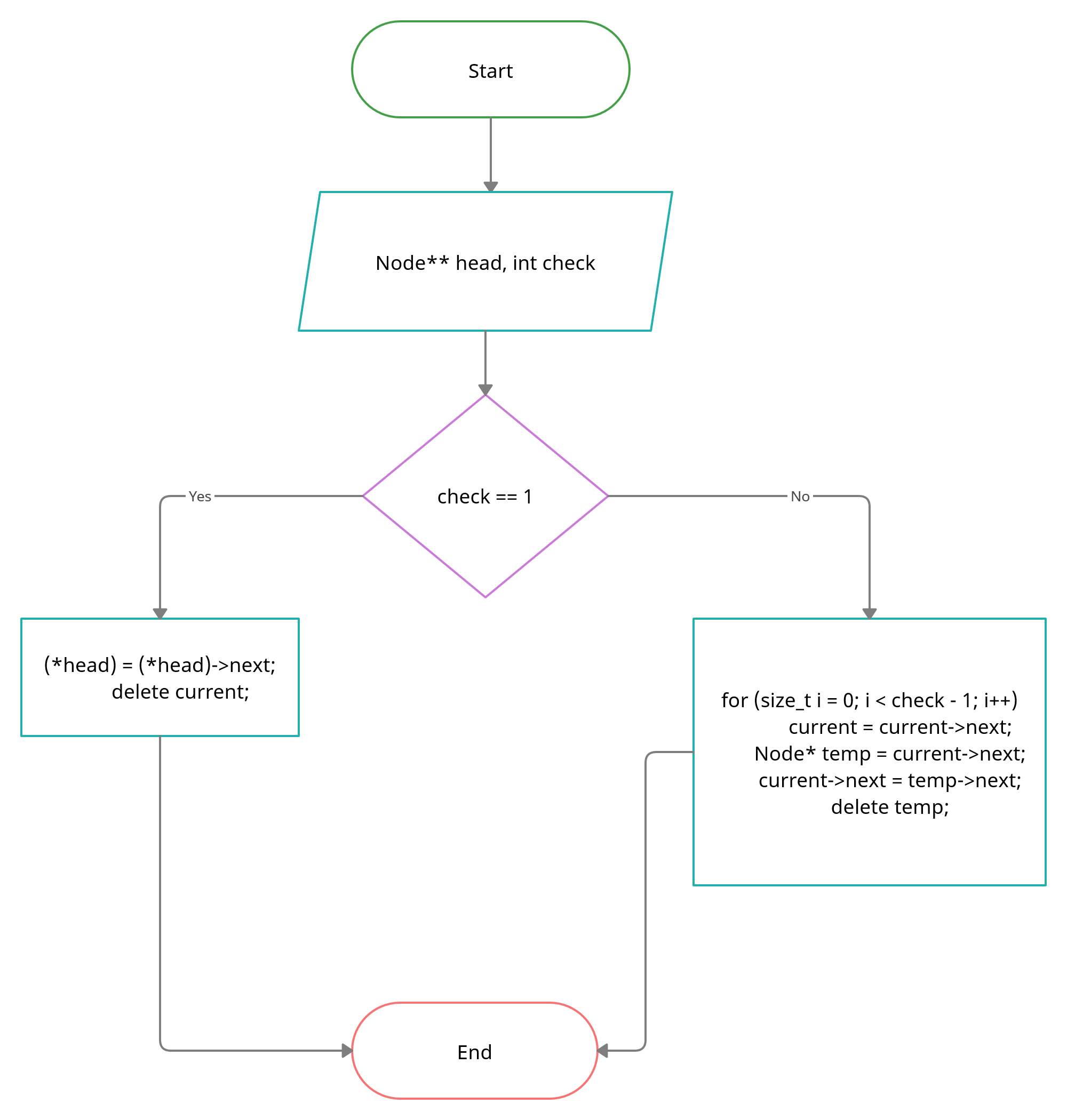
delete temp;

}

}

Повний текст програми наведений у додатку А.

Блок-схема алгоритму видалення елементу наведено нижче.



Блок-схема 2.4— Алгоритм видалення елементу за номером списку

Результат роботи алгоритму наведено на рисунку 2.5.

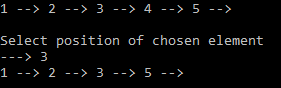


Рисунок 2.5— Результат видалення

**Перестановка місцями двох елементів:**

void SwapNode(const int value1, const int value2, Node\* head) {

Node\* inex1 = head;

for (int i = 0; i < value1-1; i++)

inex1 = inex1->next;

Node\* inex2 = head;

for (int i= 0; i < value2-1; i++)

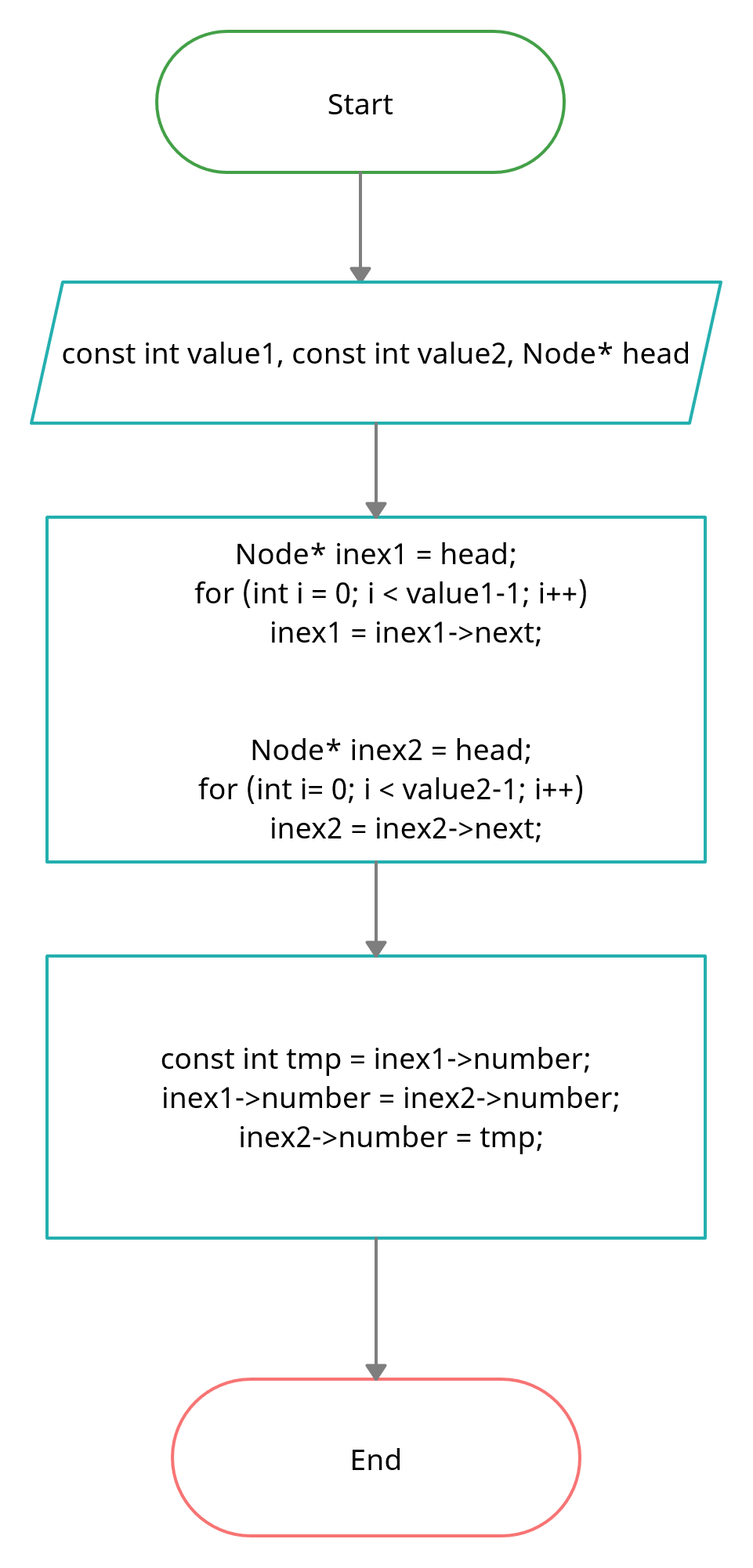
inex2 = inex2->next;

const int tmp = inex1->number;

inex1->number = inex2->number;

inex2->number = tmp;}

Повний текст програми наведений у додатку А.



Блок-схема 2.5— Алгоритм перестановки двох елементів

Результат роботи алгоритму наведено на рисунку 2.6.

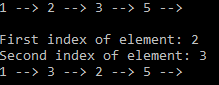
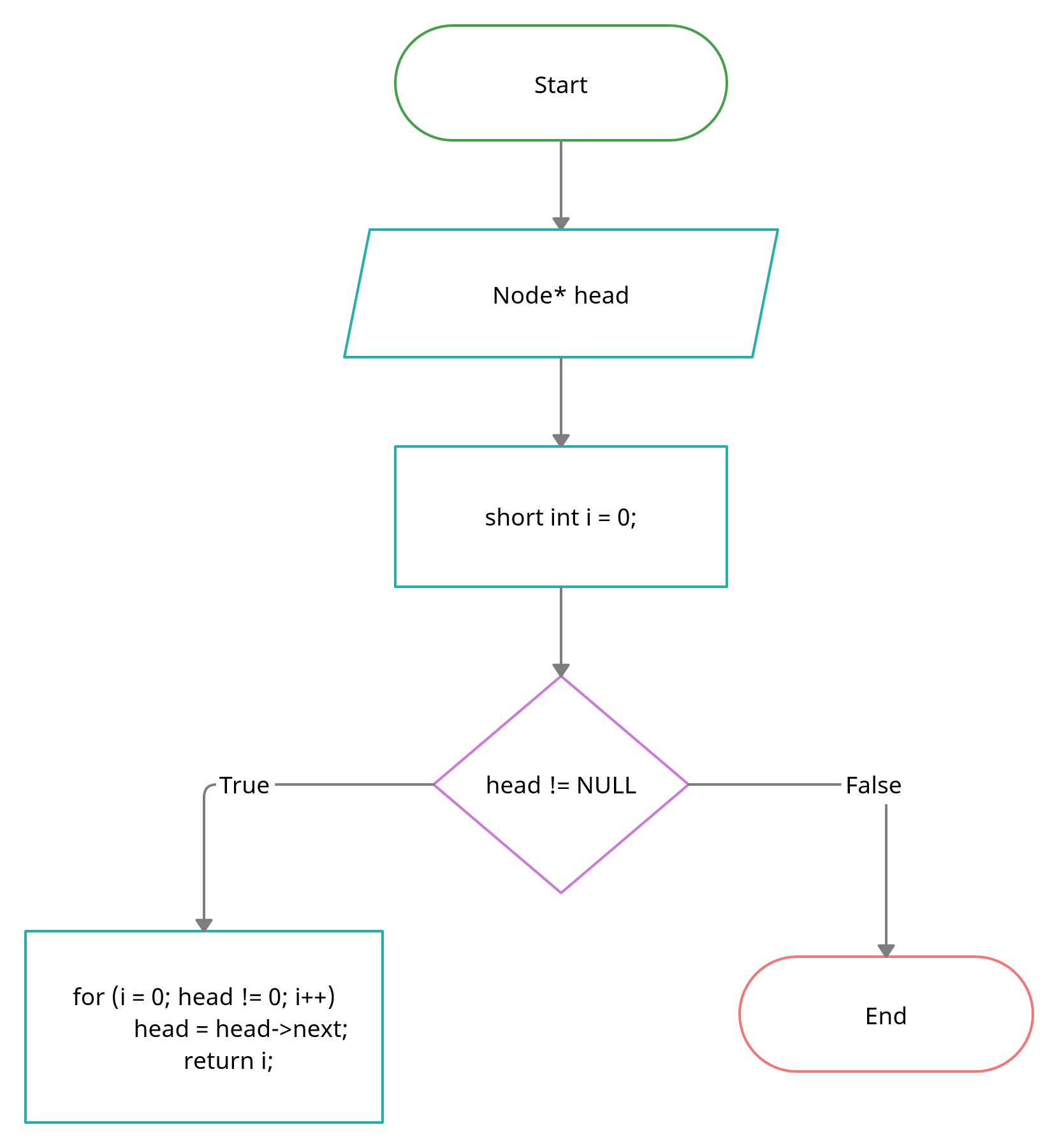


Рисунок 2.6— Результат перестановки

**Підрахунок елементів однозв’язного списку:**



Блок-схема 2.6— Алгоритм підрахунку елементів списку

short int CountOfNode(Node\* head) {

short int i = 0;

if (checkNULL(head) != 0) {

for (i = 0; head != 0; i++)

head = head->next;

return i;

}

}

Повний текст програми наведений у додатку А.

Результат роботи алгоритму наведено на рисунках 2.7 та 2.8.

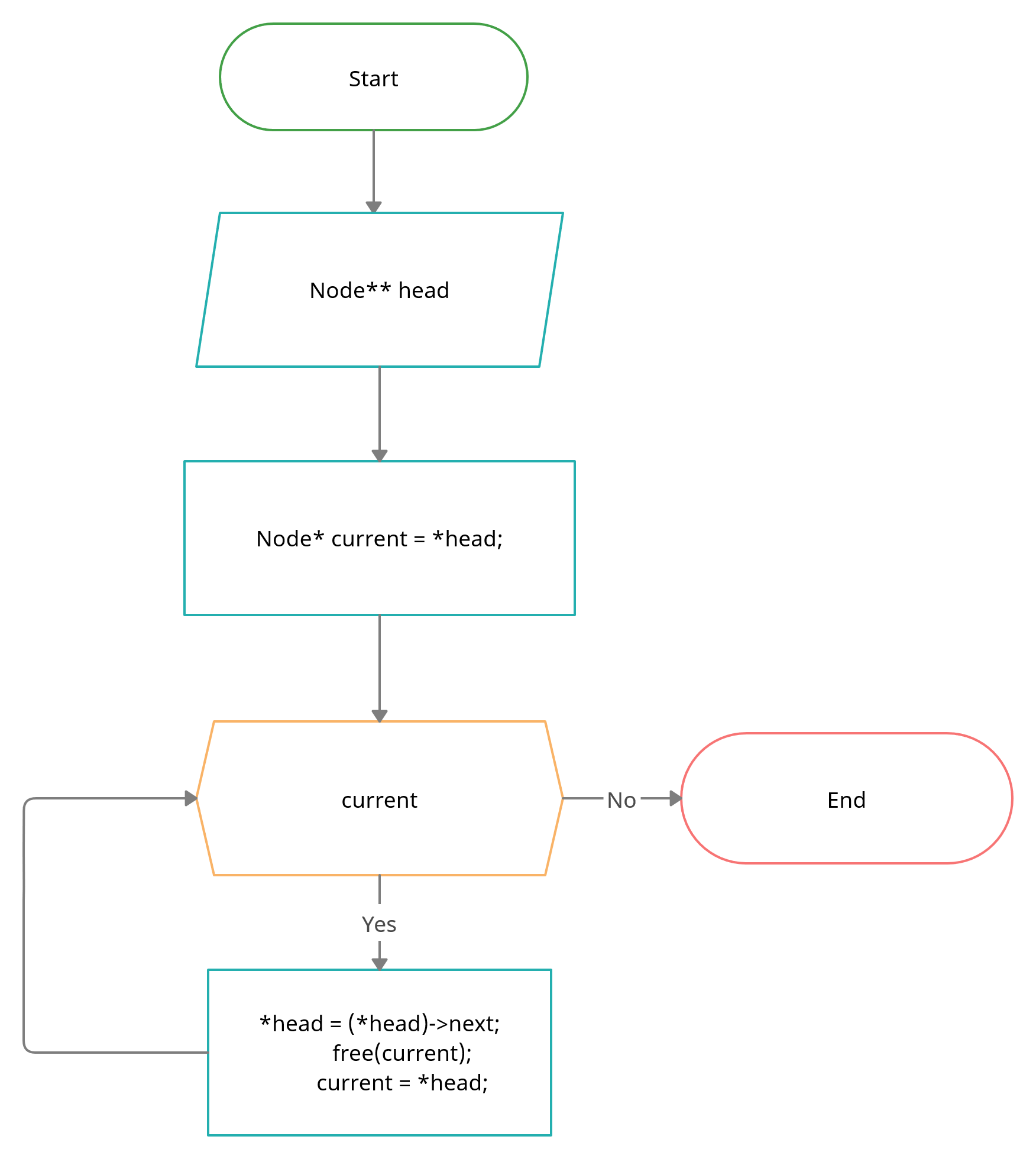


Рисунок 2.7— Список до підрахунку



Рисунок 2.8— Результат підрахунку

**Очищення однозв’язного списку:**



Блок-схема 2.7— Алгоритм очищення списку з пам’яті

void FreeNode(Node\*\* head) {

Node\* current = \*head;

while (current) {

\*head = (\*head)->next;

free(current);

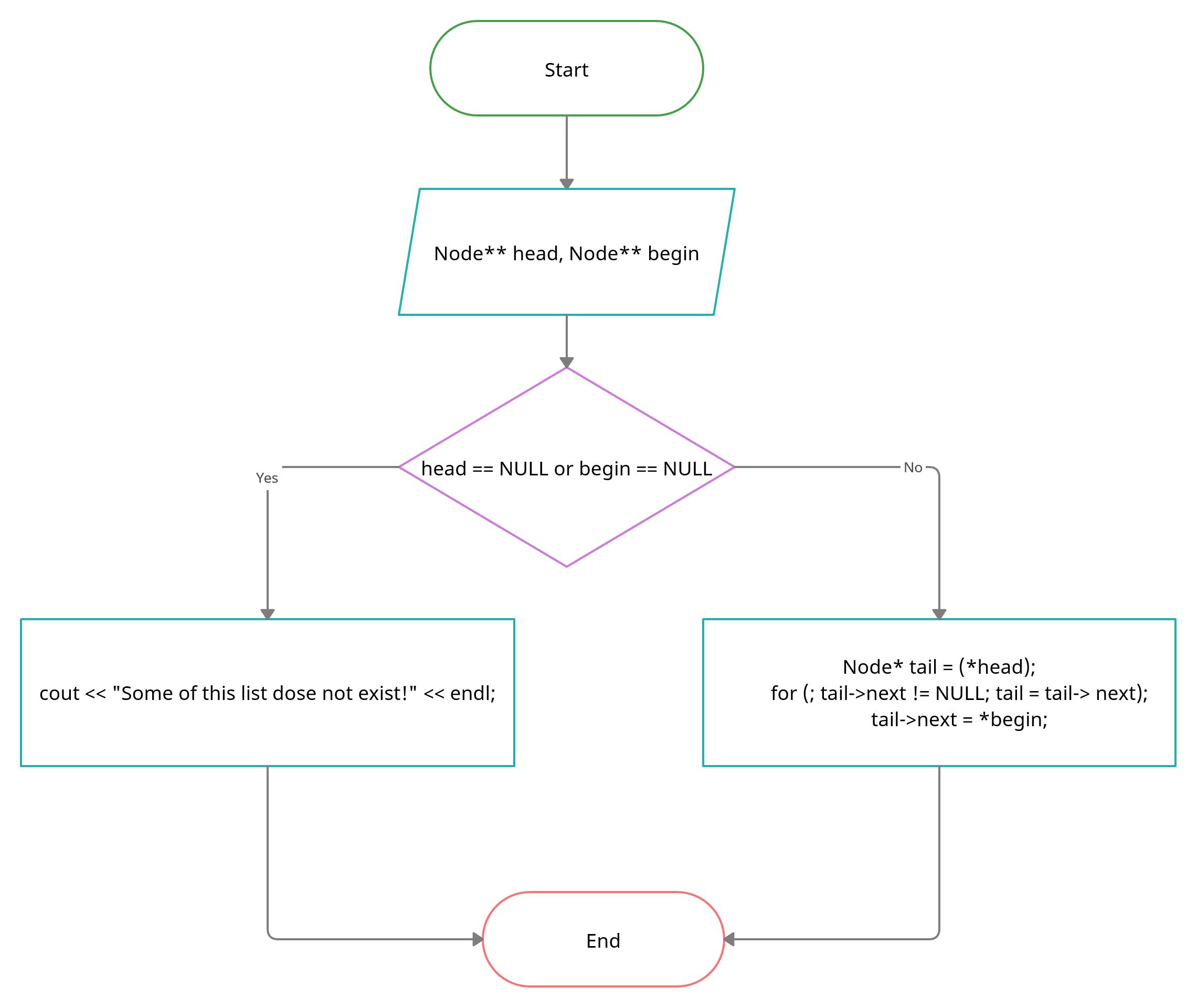
current = \*head;

}

}

Повний текст програми наведений у додатку А.

**Об’єднання двох списків:**



Блок-схема 2.8— Алгоритм ооб’єднання двох однозв’язних списків

void Split(Node\*\* head, Node\*\* begin) {

if (head == NULL or begin == NULL)

cout << "Some of this list dose not exist!" << endl;

else {

Node\* tail = (\*head);

for (; tail->next != NULL; tail = tail-> next);

tail->next = \*begin;

}

}

Повний текст програми наведений у додатку А.

Результат роботи алгоритму наведено на рисунках 2.9 та 2.10.



Рисунок 2.9— Перший список

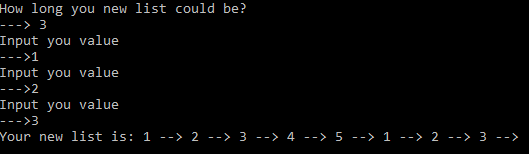


Рисунок 2.10— Створення другого за умови, якщо не існує, та об’єднання з першим

# Завдання 3

Оскільки, двозв’язний список передбачає в адресному полі посилання на попередній елемент, то у такому адресному полі списку Dlist було оголошено покажчик prev, що означає попередній. Таке оголошення продемонстровано на рисунку 3.1.

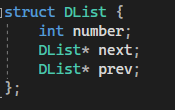


Рисунок 3.1 — оголошення структури DList

Різницю між оголошеннями списків у файлу заголовків можна порівняти за допомогою рисунка 1.1. А також у функції main при оголошенні покажчиків на початок та кінець списків, що продемонстровано на рисунку 3.2.

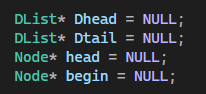


Рисунок 3.2 — оголошення та ініціалізація початків та кінця списків

Повний текст програми наведений у додатку Б.

**Створення двозв’язного списку:**

void CreateDListConsole(int count, DList\*\* Dhead, DList\* Dtail) { // user`s input

if (count > 0) {

(\*Dhead) = new DList;

cout << "Input you value \n--->"; cin >> (\*Dhead)->number;

(\*Dhead)->prev = Dtail;

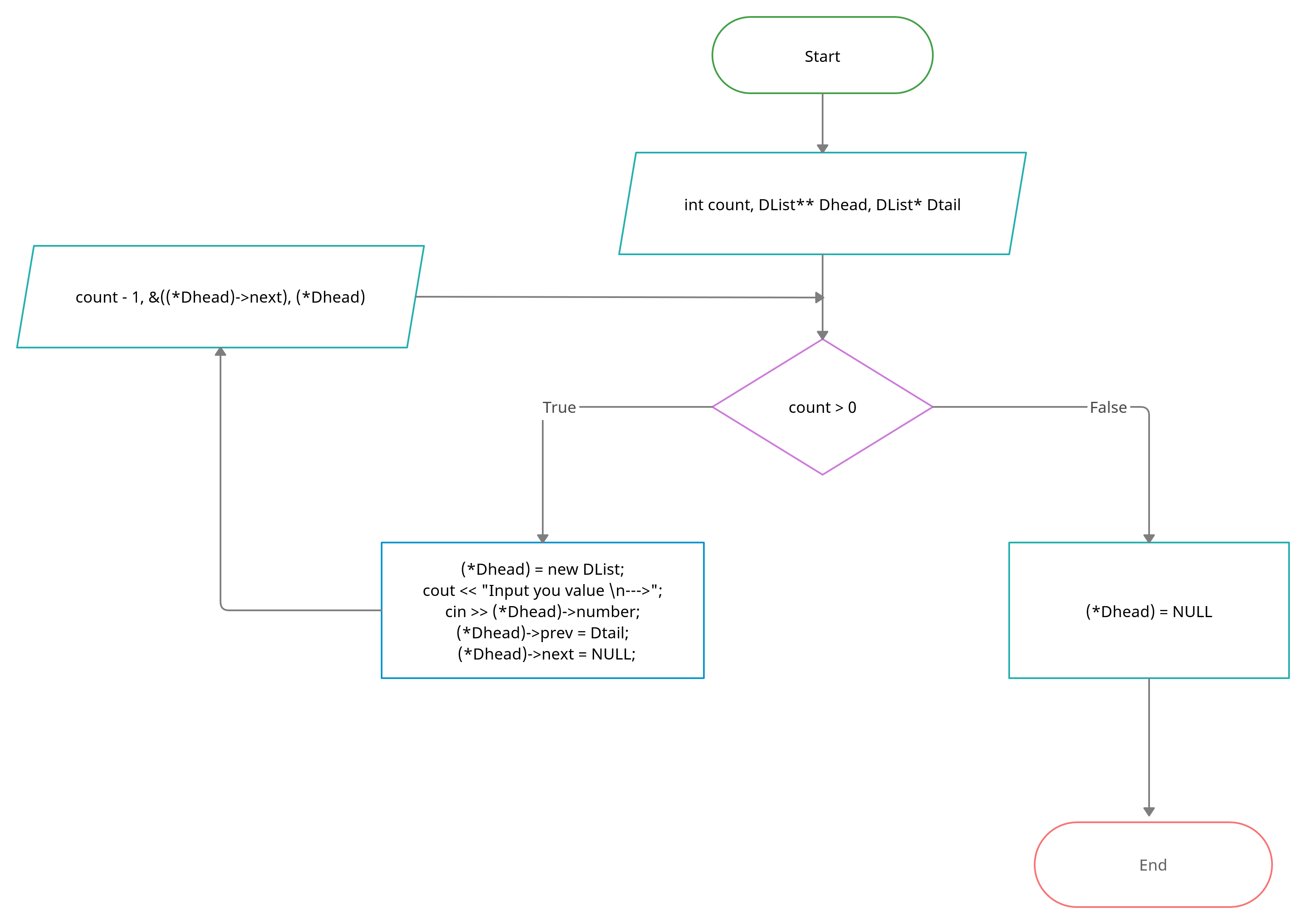
(\*Dhead)->next = NULL;

CreateDListConsole(count - 1, &((\*Dhead)->next), (\*Dhead));

}

else (\*Dhead) = NULL; }

Повний текст програми наведений у додатку А



Блок-схема 3.1— Алгоритм створення двозв’язного списку

Результат роботи алгоритму продемонстровано на рисунках 3.3 та 3.4.

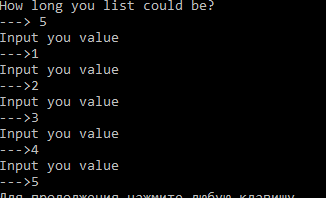


Рисунок 3.3 — заповнення списку цілими числами



Рисунок 3.4 — новостворений однозв’язний список

**Виведення двозв’зного списку у командний рядок:**

Алгоритм виведення аналогічне як у однозв’язного, за винятком коду, який наведено нижче, див. на блок-схему 2.2.

void PrintDlist(DList\* Dhead) {

if (Dhead != NULL) {

cout << Dhead->number << " --> ";

PrintDlist(Dhead->next);

}

else cout << endl << endl;

SaveInFileDlist(Dhead);

}

Повний текс програми наведений у додатку А.

Вставка нового елементу у двозв’язний список:

void DInsert(DList\*\* Dhead, DList\*\* Dtail, int pos) {

DList\* temp = new DList;

if (pos == 0) {

temp->next = \*Dhead;

(\*Dhead)->prev = temp;

\*Dhead = temp;

temp->prev = NULL;

}

else {

DList\* current = \*Dhead;

for (size\_t i = 0; i < pos - 1; i++)

current = current->next;

if (current->next == NULL) {

current->next = temp;

temp->prev = \*Dtail;

temp->next = NULL;

\*Dtail = temp;

}

else{

temp->next = current->next;

current->next = temp;

temp->next->prev = temp;

temp->prev = current;

}

}

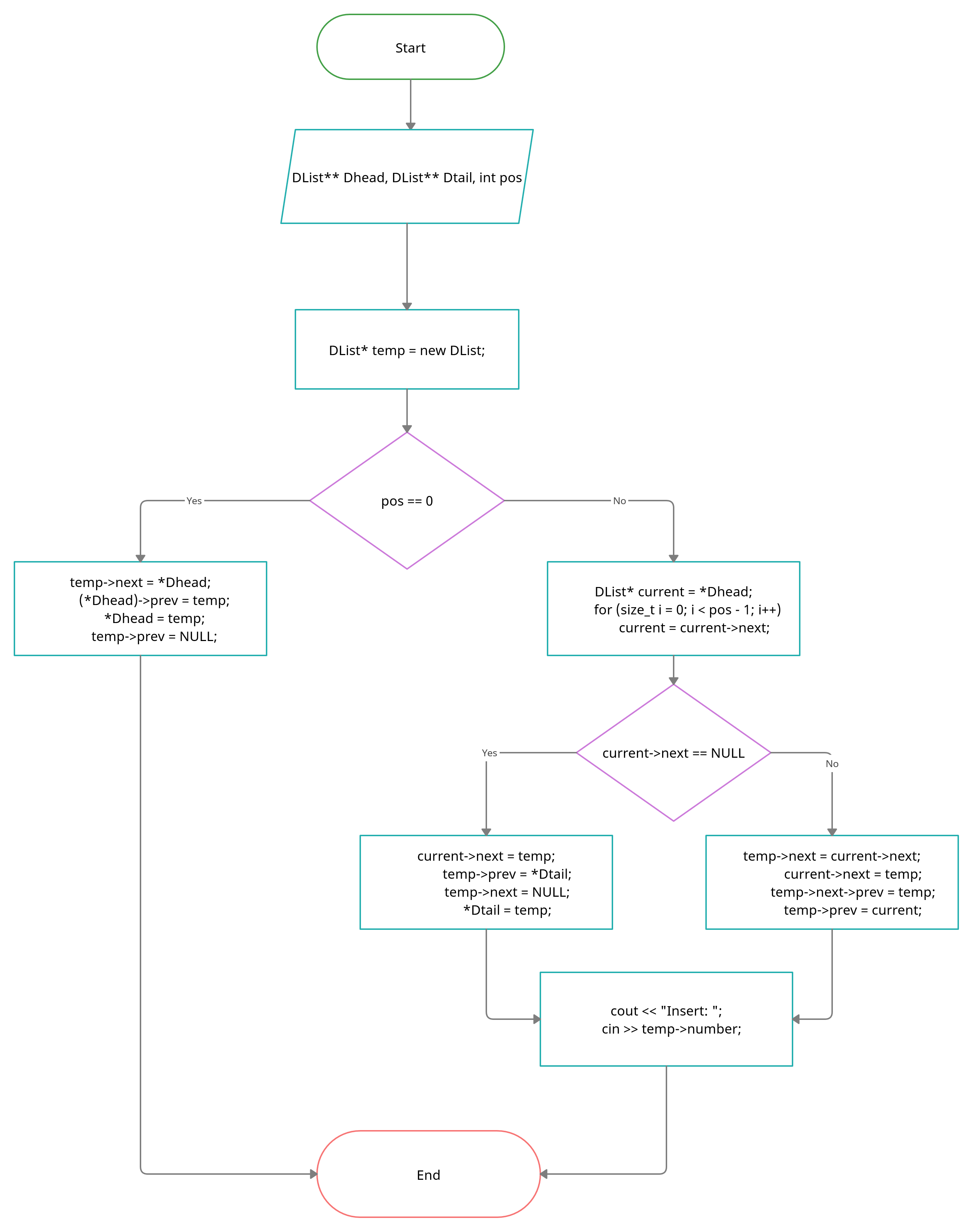
cout << "Insert: ";

cin >> temp->number;

}

Повний текс програми наведений у додатку А.

Блок-схему алгоритму наведено нижче.



Блок-схема 3.2— Алгоритм вставки нового елементу у двоз’язний список

Результат роботи алгоритму наведено на рисунку 3.5.

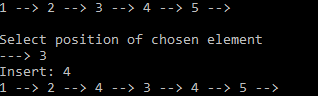
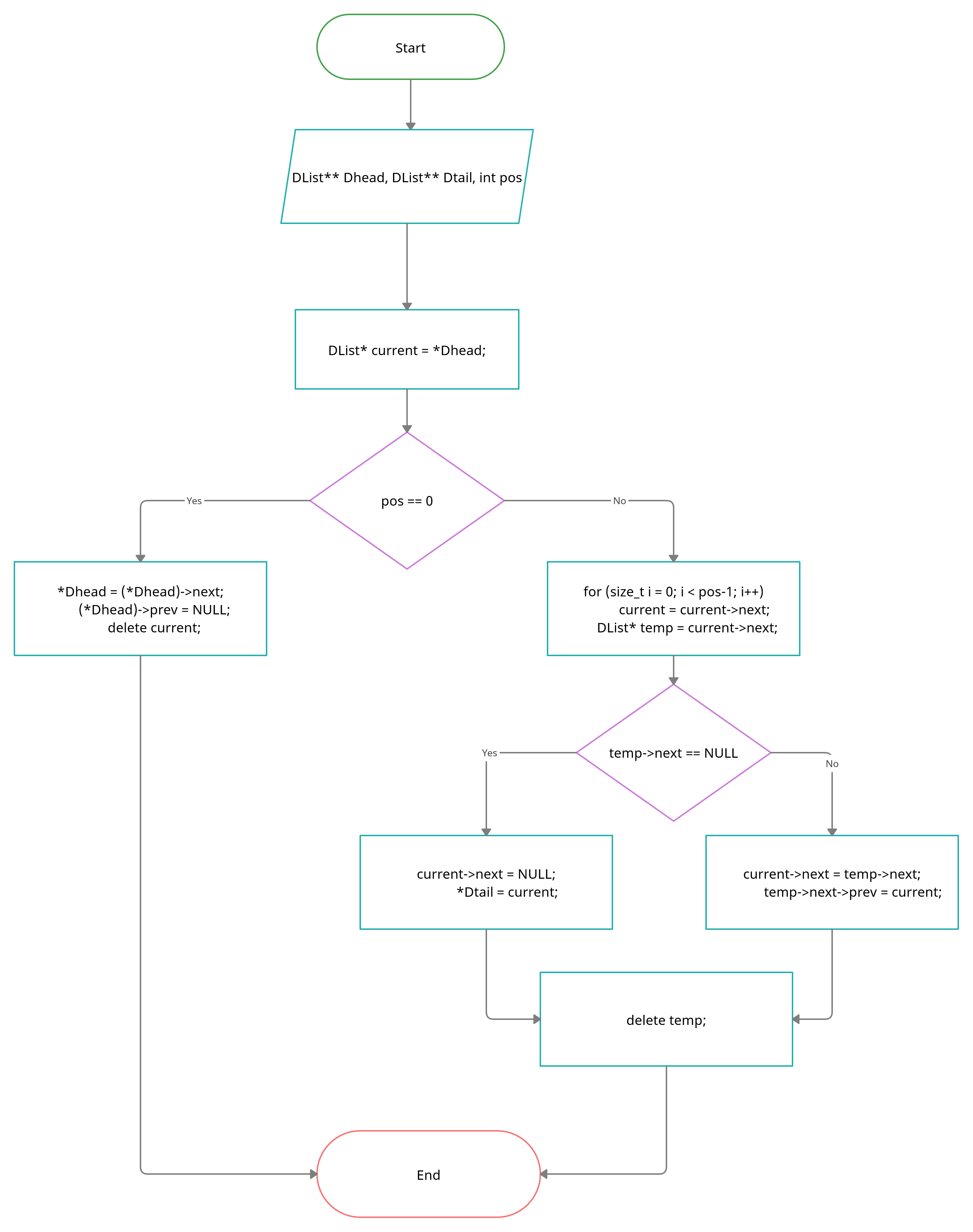


Рисунок 3.5— Результат вставки

**Видалення елементу зі списку:**



Блок-схема 3.1— Алгоритм вставки нового елементу у двоз’язний список

void DeleteDlist(DList\*\* Dhead, DList\*\* Dtail, int pos) {

DList\* current = \*Dhead;

if (pos == 0) {

\*Dhead = (\*Dhead)->next;

(\*Dhead)->prev = NULL;

delete current;

}

else {

for (size\_t i = 0; i < pos-1; i++)

current = current->next;

DList\* temp = current->next;

if (temp->next == NULL) {

current->next = NULL;

\*Dtail = current;

}

else {

current->next = temp->next;

temp->next->prev = current;

}

delete temp;

}

}

Повний текс програми наведений у додатку А.

**Зміна місцями двох елементів:**

Алгоритм аналогічний алгоритму однозв’язного списку, тому див. на блок-схему 2.5.

void SwapDlsit(const int value1, const int value2, DList\* Dhead ) {

DList\* inex1 = Dhead;

for (int i = 0; i < value1 - 1; i++)

inex1 = inex1->next;

DList\* inex2 = Dhead;

for (int i = 0; i < value2 - 1; i++)

inex2 = inex2->next;

const int tmp = inex1->number;

inex1->number = inex2->number;

inex2->number = tmp;

}

Повний текс програми наведений у додатку А.

**Підрахунок елементів списку:**

Алгоритм аналогічний алгоритму однозв’язного списку, тому див. на блок-схему 2.6.

short int CountOfDlist(DList\* Dhead) {

short int i = 0;

if (DcheckNULL(Dhead) != 0) {

for (i = 0; Dhead != 0; i++)

Dhead = Dhead->next;

return i;

}

}

Повний текст програми наведений у додатку А.

Результат роботи алгоритму наведено на рисунках 2.7 та 2.8.



Рисунок 3.6— Список до підрахунку



Рисунок 3.7— Результат підрахунку

Очищення двозв’зного писка:

Алгоритм аналогічний алгоритму однозв’язного списку, тому див. на блок-схему 2.7.

void FreeDlist(DList\*\* Dhead) { // clean list

DList\* current = \*Dhead;

while (current) {

\*Dhead = (\*Dhead)->next;

free(current);

current = \*Dhead;

}

}

Повний текст програми наведений у додатку А.

# Висновок

Під час цієї лабораторної роботи було закріплено знання з тем однозв’язного та двозв’язного списку.

Списки це один з видів динамічних структур, яка представляє послідовність елементів, в кожному якому зберігається покажчик на наступний та/або на попередній елемент списку. Він обов’язково має містити інформаційне поле та адресне поле. На відміну від масивів, вставка та вилучення елементів у Зв'язний список потребують сталого часу. Також значною перевагою зв'язних списків є можливість легкого розширення: щоб збільшити розмір списку, треба лише додати ще один елемент. Недоліком зв'язних списків є необхідність проходити весь список, щоб знайти елемент (тобто час доступу до елемента списку.

Існують операції над списками такі як: створення списку, виведення на екран, вставка елементу в список, видалення елементу зі списку, пошук елементу в списку, перевірка на відсутність елементу, очистка списку.

Відмінність між двозв’язним та однозвязним полягає у тому, що кожен елемент двозв’язного списку має покажчик на попередній елемент. Тому, перший елемент також як и останній посилаються на NULL вказівник, це свідчить на те, що не обов’язково потрібно проініціалізовувати перший елемент списку. Також організація двозв’зного списку має переваги над однозв’язним, оскільки пошук елементу швидший через те, що пошук може початися одночасно з початку та кінця списку.

Робота з пам’яттю такої структури даних це робота з динамічною пам’яттю, тому важливо дотримуватися контролю виділення пам’яті. Також потрібно після закінчення роботи очистити список з пам’яті, оскільки можливий витік пам’яті і може викликати деякі програмні помилки навіть у інших користувачів.

# Додаток А

Код Source.cpp

#include "Header.h"

void inputCheker() {

if (cin.fail()) {

system("cls");

cin.clear();

cin.ignore(32767, '\n');

cout << " \t\t\tIncorrect input! ERROR! " << endl;

}

}

///////////////////////////////////////////////////////////////////////

///////////////////// Doubly Node ///////////////////////////////////

/////////////////////////////////////////////////////////////////////

void CreateDListConsole(int count, DList\*\* Dhead, DList\* Dtail) { // user`s input

if (count > 0) {

(\*Dhead) = new DList;

cout << "Input you value \n--->"; cin >> (\*Dhead)->number;

(\*Dhead)->prev = Dtail;

(\*Dhead)->next = NULL;

CreateDListConsole(count - 1, &((\*Dhead)->next), (\*Dhead));

}

else (\*Dhead) = NULL;

}

void PrintDlist(DList\* Dhead) {

if (Dhead != NULL) {

cout << Dhead->number << " --> ";

PrintDlist(Dhead->next);

}

else cout << endl << endl;

SaveInFileDlist(Dhead);

}

void DInsert(DList\*\* Dhead, DList\*\* Dtail, int pos) {

DList\* temp = new DList;

if (pos == 0) {

temp->next = \*Dhead;

(\*Dhead)->prev = temp;

\*Dhead = temp;

temp->prev = NULL;

}

else {

DList\* current = \*Dhead;

for (size\_t i = 0; i < pos - 1; i++)

current = current->next;

if (current->next == NULL) {

current->next = temp;

temp->prev = \*Dtail;

temp->next = NULL;

\*Dtail = temp;

}

else{

temp->next = current->next;

current->next = temp;

temp->next->prev = temp;

temp->prev = current;

}

}

cout << "Insert: ";

cin >> temp->number;

}

void DeleteDlist(DList\*\* Dhead, DList\*\* Dtail, int pos) {

DList\* current = \*Dhead;

if (pos == 0) {

\*Dhead = (\*Dhead)->next;

(\*Dhead)->prev = NULL;

delete current;

}

else {

for (size\_t i = 0; i < pos-1; i++)

current = current->next;

DList\* temp = current->next;

if (temp->next == NULL) {

current->next = NULL;

\*Dtail = current;

}

else {

current->next = temp->next;

temp->next->prev = current;

}

delete temp;

}

}

void SwapDlsit(const int value1, const int value2, DList\* Dhead ) {

DList\* inex1 = Dhead;

for (int i = 0; i < value1 - 1; i++)

inex1 = inex1->next;

DList\* inex2 = Dhead;

for (int i = 0; i < value2 - 1; i++)

inex2 = inex2->next;

const int tmp = inex1->number;

inex1->number = inex2->number;

inex2->number = tmp;

}

short int CountOfDlist(DList\* Dhead) {

short int i = 0;

if (DcheckNULL(Dhead) != 0) {

for (i = 0; Dhead != 0; i++)

Dhead = Dhead->next;

return i;

}

}

bool DcheckNULL(DList\* Dhead) { // cheking on empty

if (Dhead == NULL) {

cout << "List is empty." << endl;

return 0;

}

return 1;

}

void FreeDlist(DList\*\* Dhead) { // clean list

DList\* current = \*Dhead;

while (current) {

\*Dhead = (\*Dhead)->next;

free(current);

current = \*Dhead;

}

}

void SaveInFileDlist(DList\* Dhead) { // writing in data.txt

DList\* current = Dhead;

FILE\* data;

fopen\_s(&data, "dataDlist.txt", "w");

if (!data) {

cout << "ERROR. Missing data.txt" << endl;

}

while (current) {

fprintf(data, "\t%d", current->number);

current = current->next;

}

fclose(data);

}

void LoadFromFileDlist(DList\*\* Dhead, DList\*\* Dtail) { // reading from data.txt

DList\* current, \* prev;

prev = current = (\*Dhead) = new DList;

(\*Dhead)->prev = NULL;

FILE\* data;

fopen\_s(&data, "dataDlist.txt", "r");

if (!data) {

cout << "ERROR. Missing data.txt";

delete current;

(\*Dhead) = NULL;

return;

}

if (fgetc(data) == EOF) {

cout << "File is empty.";

delete current;

(\*Dhead) = nullptr;

return;

}

fscanf(data, "%d", &current->number);

while (!feof(data)) {

current = new DList;

fscanf(data, "%d", &current->number);

prev->next = current;

current->prev = prev;

prev = current;

}

current->next = NULL;

\*Dtail = current;

cout << "File has been successfully saved." << endl;

fclose(data);

}

///////////////////////////////////////////////////////////////////////

///////////////////// Singly Node ///////////////////////////////////

/////////////////////////////////////////////////////////////////////

void CreateNodeConsole(int count, Node\*\* head) { // user`s input

if (count > 0) {

(\*head) = new Node;

cout << "Input you value \n--->"; cin >> (\*head)->number;

(\*head)->next = NULL;

CreateNodeConsole(count - 1, &((\*head)->next));

}

}

void Insert(Node\* head, int check) {

Node\* temp = new Node;

if (check == 0 or head == NULL) {

temp->next = head;

head = temp;

}

else {

Node\* current = head;

for (size\_t i = 0; i < check - 1; i++)

current = current->next;

temp->next = current->next;

current->next = temp;

}

cout << "Insert: ";

cin >> temp->number;

}

void Delete(Node\*\* head, int check) {

Node\* current = \*head;

if (check == 1) {

(\*head) = (\*head)->next;

delete current;

}

else {

for (size\_t i = 0; i < check - 1; i++)

current = current->next;

Node\* temp = current->next;

current->next = temp->next;

delete temp;

}

}

void Split(Node\*\* head, Node\*\* begin) {

if (head == NULL or begin == NULL)

cout << "Some of this list dose not exist!" << endl;

else {

Node\* tail = (\*head);

for (; tail->next != NULL; tail = tail-> next);

tail->next = \*begin;

}

}

void SwapNode(const int value1, const int value2, Node\* head) {

Node\* inex1 = head;

for (int i = 0; i < value1-1; i++)

inex1 = inex1->next;

Node\* inex2 = head;

for (int i= 0; i < value2-1; i++)

inex2 = inex2->next;

const int tmp = inex1->number;

inex1->number = inex2->number;

inex2->number = tmp;

}

void PrintNode(Node\* head) {

if (head != NULL) {

cout << head->number << " --> ";

PrintNode(head->next);

}

else cout << endl << endl;

SaveInFileNode(head);

}

short int CountOfNode(Node\* head) {

short int i = 0;

if (checkNULL(head) != 0) {

for (i = 0; head != 0; i++)

head = head->next;

return i;

}

}

void FreeNode(Node\*\* head) { // clean list

Node\* current = \*head;

while (current) {

\*head = (\*head)->next;

free(current);

current = \*head;

}

}

bool checkNULL(Node\* head) { // cheking on empty

if (head == NULL) {

cout << "List is empty." << endl;

return 0;

}

return 1;

}

void SaveInFileNode(Node\* head) { // writing in data.txt

Node\* current = head;

FILE\* data;

fopen\_s(&data, "dataNode.txt", "w");

if (!data) {

cout << "ERROR. Missing data.txt" << endl;

}

while (current) {

fprintf(data, "\t%d", current->number);

current = current->next;

}

fclose(data);

}

void LoadFromFile(Node\*\* head) { // reading from dataNode.txt

Node\* current, \* previous;

previous = current = (\*head) = new Node;

FILE\* data;

fopen\_s(&data, "dataNode.txt", "r");

if (!data) {

cout << "ERROR. Missing data.txt";

}

if (fgetc(data) == EOF) {

cout << "File is empty.";

}

current = new Node;

fscanf(data, "%d", &current->number);

previous->next = current;

previous = current;

while (!feof(data)) {

current = new Node;

fscanf(data, "%d", &current->number);

previous->next = current;

previous = current;

}

current->next = NULL;

Delete(head, 1);

cout << "File has been successfully saved." << endl;

fclose(data);

}

# Додаток Б

Код main.cpp

#include "Header.h"

int main()

{

srand(time(NULL));

DList\* Dhead = NULL;

DList\* Dtail = NULL;

Node\* head = NULL;

Node\* begin = NULL;

short int count, nodeSelect, index1, index2, pos;

while (true) {

cout << "Select the list type: \nSingly linked list - 1 \nDoubly linked list - 2 \nLoad singly linked list - 3 \nLoad doubly linked - 4 \nExit - 0 \n---> "; cin >> nodeSelect;

inputCheker();

switch (nodeSelect)

{

///////////////////////////////////////////////////////////////////////

///////////////////// Singly Node ///////////////////////////////////

/////////////////////////////////////////////////////////////////////

case 1:

while (nodeSelect != 0) {

system("cls");

if (head == NULL) {

cout << "How long you list could be? \n---> "; cin >> count;

//inputCheker();

CreateNodeConsole(count, &head);

system("pause");

system("cls");

}

PrintNode(head);

cout << "Select the operation: \nAdd new element - 1 \nCount the number of elements - 2 \nSwap two elements - 3 \nDelete one element - 4 \nSplit two list in one - 5 \nClear list - 6 \nPrint list - 7 \nBack - 0 \n---> "; cin >> nodeSelect;

//inputCheker();

switch (nodeSelect)

{

case 1:

system("cls");

PrintNode(head);

cout << "Select position of chosen element \n---> "; cin >> pos;

if (pos > CountOfNode(head) + 1) {

cout << "You`ve get to far!" << endl;

system("pause");

system("cls");

break;

}

Insert(head, pos - 1);

PrintNode(head);

system("pause");

system("cls");

break;

case 2:

system("cls");

cout << "Amount of elements: " << CountOfNode(head) << "\n";

system("pause");

system("cls");

break;

case 3:

system("cls");

PrintNode(head);

cout << "First index of element: "; cin >> index1;

cout << "Second index of element: "; cin >> index2;

SwapNode(index1, index2, head);

PrintNode(head);

system("pause");

system("cls");

break;

case 4:

system("cls");

PrintNode(head);

cout << "Select position of chosen element \n---> "; cin >> pos;

Delete(&head, pos);

PrintNode(head);

system("pause");

system("cls");

break;

case 5:

system("cls");

cout << "How long you new list could be? \n---> "; cin >> count;

CreateNodeConsole(count, &begin);

cout << "Your new list is: ";

Split(&head, &begin);

PrintNode(head);

system("pause");

system("cls");

break;

case 6:

system("cls");

FreeNode(&head);

FreeNode(&begin);

cout << "List have been cleared\n";

system("pause");

system("cls");

nodeSelect = 0;

break;

case 7:

system("cls");

cout << "Your list is: ";

PrintNode(head);

break;

case 0:

system("cls");

break;

default:

system("cls");

cout << " \t\t\tIncorrect input! ERROR! " << endl;

break;

}

}

break;

///////////////////////////////////////////////////////////////////////

///////////////////// Doubly Node ///////////////////////////////////

/////////////////////////////////////////////////////////////////////

case 2:

system("cls");

while (nodeSelect != 0) {

system("cls");

if (Dhead == NULL) {

cout << "How long you list could be? \n---> "; cin >> count;

//inputCheker();

CreateDListConsole(count, &Dhead, Dtail);

system("pause");

system("cls");

}

PrintDlist(Dhead);

cout << "Select the operation: \nAdd new element - 1 \nCount the number of elements - 2 \nSwap two elements - 3 \nDelete one element - 4 \nClear list - 5 \nPrint list - 6 \nBack - 0 \n---> "; cin >> nodeSelect;

//inputCheker();

switch (nodeSelect)

{

case 1:

system("cls");

PrintDlist(Dhead);

cout << "Select position of chosen element \n---> "; cin >> pos;

if (pos > CountOfDlist(Dhead) + 1) {

cout << "You`ve get to far!" << endl;

system("pause");

system("cls");

break;

}

DInsert(&Dhead, &Dtail, pos - 1);

PrintDlist(Dhead);

system("pause");

system("cls");

break;

case 2:

system("cls");

cout << "Amount of elements: " << CountOfDlist(Dhead) << "\n";

system("pause");

system("cls");

break;

case 3:

system("cls");

PrintDlist(Dhead);

cout << "First index of element: "; cin >> index1;

cout << "Second index of element: "; cin >> index2;

SwapDlsit(index1, index2, Dhead);

PrintDlist(Dhead);

system("pause");

system("cls");

break;

case 4:

system("cls");

PrintDlist(Dhead);

cout << "Select position of chosen element \n---> "; cin >> pos;

DeleteDlist(&Dhead, &Dtail, pos-1);

PrintDlist(Dhead);

system("pause");

system("cls");

break;

case 5:

system("cls");

FreeDlist(&Dhead);

cout << "List have been cleared\n";

system("pause");

system("cls");

nodeSelect = 0;

break;

case 6:

system("cls");

cout << "Your list is: ";

PrintDlist(Dhead);

break;

case 0:

system("cls");

break;

default:

system("cls");

cout << " \t\t\tIncorrect input! ERROR! " << endl;

break;

}

}

break;

case 3:

system("cls");

LoadFromFile(&head);

system("pause");

break;

case 4:

system("cls");

LoadFromFileDlist(&Dhead, &Dtail);

system("pause");

break;

case 0:

system("cls");

FreeDlist(&Dhead);

FreeNode(&head);

FreeNode(&begin);

exit(0);

break;

default:

system("cls");

cout << " \t\t\tIncorrect input! ERROR! " << endl;

break;

}

}

return 0;}

# Додаток В

Код Header.h

#include <string>

#include <ctime>

using namespace std;

struct Node {

int number;

Node\* next;

};

struct DList {

int number;

DList\* next;

DList\* prev;

};

void inputCheker();

// doubly node

void CreateDListConsole(int count, DList\*\* Dhead, DList\* Dtail);

void PrintDlist(DList\* Dhead);

void DInsert(DList\*\* Dhead, DList\*\* Dtail, int pos);

void DeleteDlist(DList\*\* Dhead, DList\*\* Dtail, int pos);

bool DcheckNULL(DList\* Dhead);

short int CountOfDlist(DList\* Dhead);

void FreeDlist(DList\*\* Dhead);

void SwapDlsit(const int value1, const int value2, DList\* Dhead);

void SaveInFileDlist(DList\* Dhead);

void LoadFromFileDlist(DList\*\* Dhead, DList\*\* Dtail);

// singly node

void CreateNodeConsole(int count, Node\*\* head);

void PrintNode(Node\* head);

void Insert(Node\* head, int check);

void Delete(Node\*\* head, int check);

void FreeNode(Node\*\* head);

bool checkNULL(Node\* head); // cheking on empty

short int CountOfNode(Node\* head);

void Split(Node\*\* head, Node\*\* begin);

void SwapNode(const int value1, const int value2, Node\* head);

void SaveInFileNode(Node\* head);

void LoadFromFile(Node\*\* head);