МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет

«Харківський політехнічний інститут»

Кафедра «Стратегічного управління проектами»

**Лабораторна робота №2**

з дисципліни «Алгоритми і структури даних»

Виконав:

студент групи КН-320(б)

Колій Дмитро

Перевірив:

Є.О.Мошко

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 Single\_List //Клас однозв'язного списку

{

public:

double numb = 0; //Інформація

Single\_List\* next = NULL; //Вказівник на наступний елемент

Single\_List();

Single\_List(double value);

~Single\_List();

};

class Double\_List //Клас двозв'язного списку

{

public:

double numb = 0; //Інформація

Double\_List\* next = NULL, \* back = NULL; //Вказівники на наступний і попередній елемент

Double\_List();

Double\_List(double value);

~Double\_List();

};

class Stack //Клас стеку

{

private:

Single\_List\* top = NULL; //Вказівник на вершину стеку

int size = 0; //Розмір стеку

public:

Stack();

void Create\_stack(int count, bool flag); //Створення декількох елементів

void Print\_stack(); //Роздрукування стеку

void Push\_top(double value); //Додавання елементу на вершину стеку

void Pop\_top(); //Видалення елементу з вершини стеку

void Clear\_stack(); //Очищення стеку

void Swap\_first\_last(); //Зміна першого і останього елемента місцями

void Flip\_stack(); //Перегортання стеку

void Delete\_everyone\_second(); //Видалення кожного другого елементу

void Insert\_star(); //Вставлення зірки у стек

void Search\_max\_and\_insert\_after\_zero(); //Вставлення після максимального елементу - нуль

void Delete\_min(); //Видалити мінімальний

void Clear\_besides\_first(); //Видалити всі елементи крім першого

void Clear\_besides\_last(); //Видалити всі елементи крім останього

void Read\_from\_file(ifstream& f\_in); //Зчитати з файлу

void Write\_to\_file(ofstream& f\_out); //Записату в файл

~Stack();

};

class Queue //Клас черги

{

private:

Double\_List\* first = NULL, \* last = NULL; //Вказівники на початок і кінець черги

public:

Queue();

void Create\_queue(int count, bool flag); //Створення декількох елементів

void Print\_queue(); //Роздрукування черги

void Push\_back(double value); //Додавання елементу в кінець черги

void Pop\_front(); //Видалення елемент з початку черги

void Clear\_queue(); //Очищення черги

void Count(); //Пдірахувати кількість елементів черги

void Average\_arithmetic(); //Череднє арифметичне

void Search\_max\_and\_min(); //Пошук максимального і мінімального елементів

void Search\_item\_after\_min\_item(); //Пошук елементу, що йде перед мінімальним

void Read\_from\_file(ifstream& f\_in); //Зчитування з файлу

void Write\_to\_file(ofstream& f\_out); //Записування у файл

~Queue();

};

class Deque //Клас деки

{

private:

Double\_List\* first = NULL, \* last = NULL; //Вказівники на початок і кінець черги

public:

Deque();

void Create\_deque(int count, bool how); //Створення декількох елементів

void Print\_deque(); //Роздрукування деки

void Push\_front(double value); //Додавання елементу в початок

void Push\_back(double value); //Додавання елементу в кінець

void Pop\_front(); //Видалення елементу з початку

void Pop\_back(); //Видалення елементу з кінця

void Clear\_deque(); //Очищення деки

void Read\_sixth\_item(); //Зчитати шостий елемент

void Read\_last\_item(); //Зчитати останній елемент

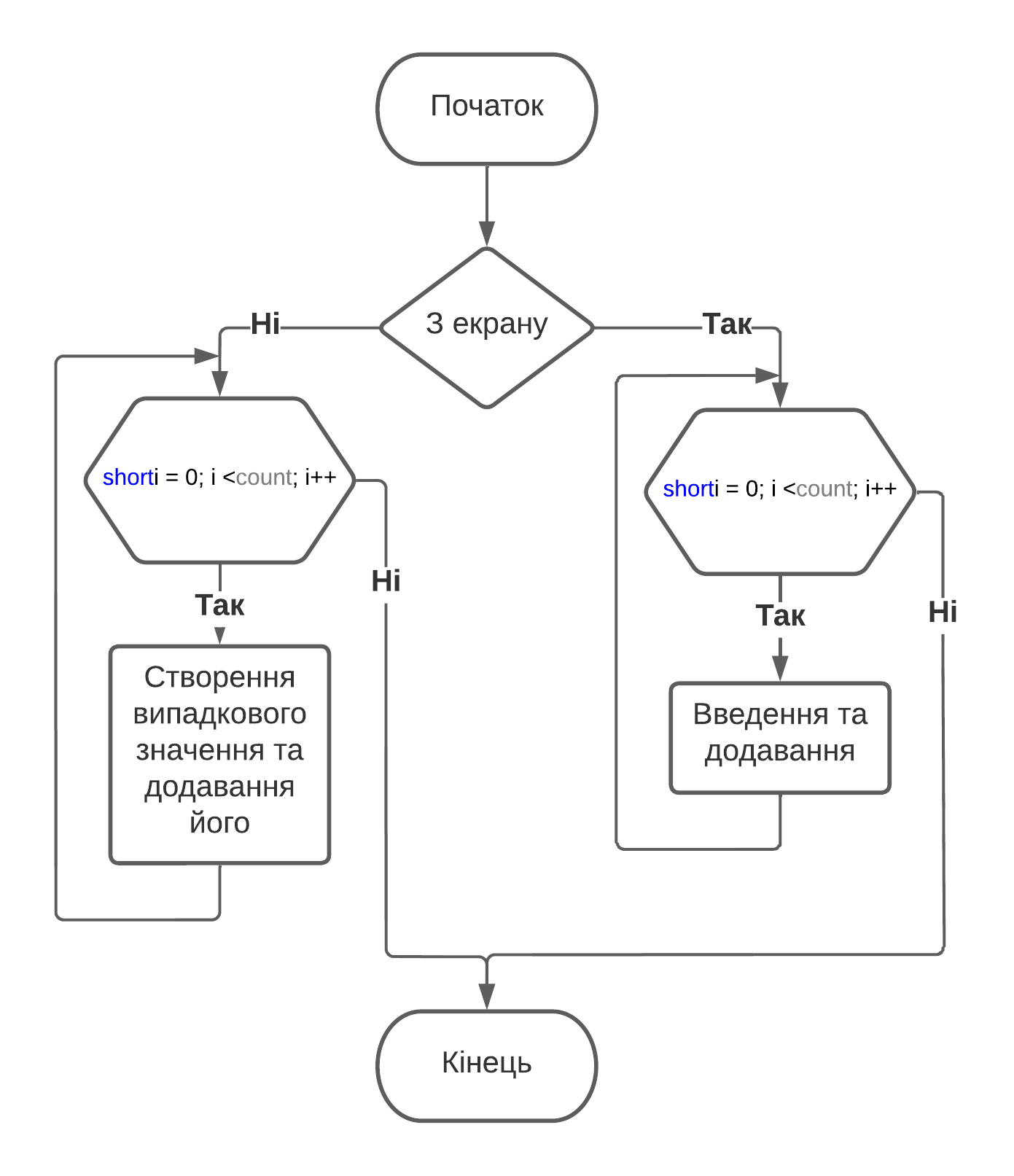
void Read\_from\_file(ifstream& f\_in); //Зчитати з файлу

void Write\_to\_file(ofstream& f\_out); //Записати у файл

~Deque();

};

СТЕК

ФУНКЦІЯ СТВОРЕННЯ СТЕКУ

void Stack::Create\_stack(int count, bool flag) {

double temp;

if (flag == true) {

for (int i = 0; i < count; i++) {

cin >> temp;

this->Push\_top(temp);

}

}

else {

for (int i = 0; i < count; i++) {

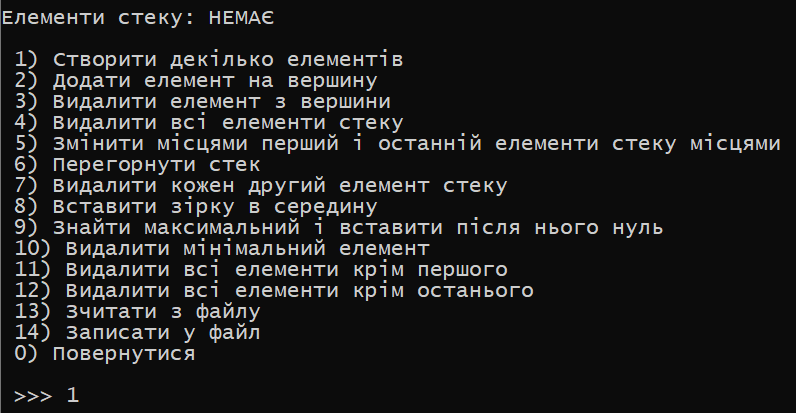
temp = rand() % 101 - 50 + rand() % 101 \* 0.01;

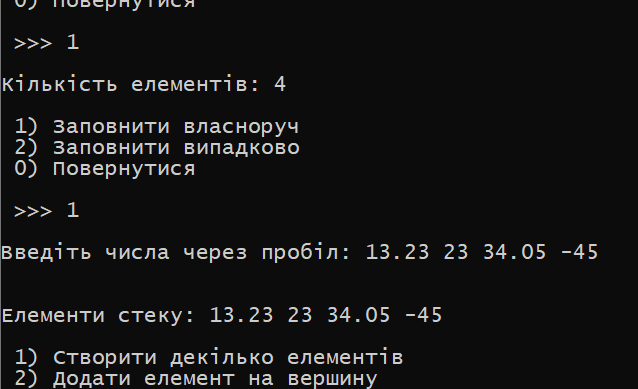
this->Push\_top(temp);

}

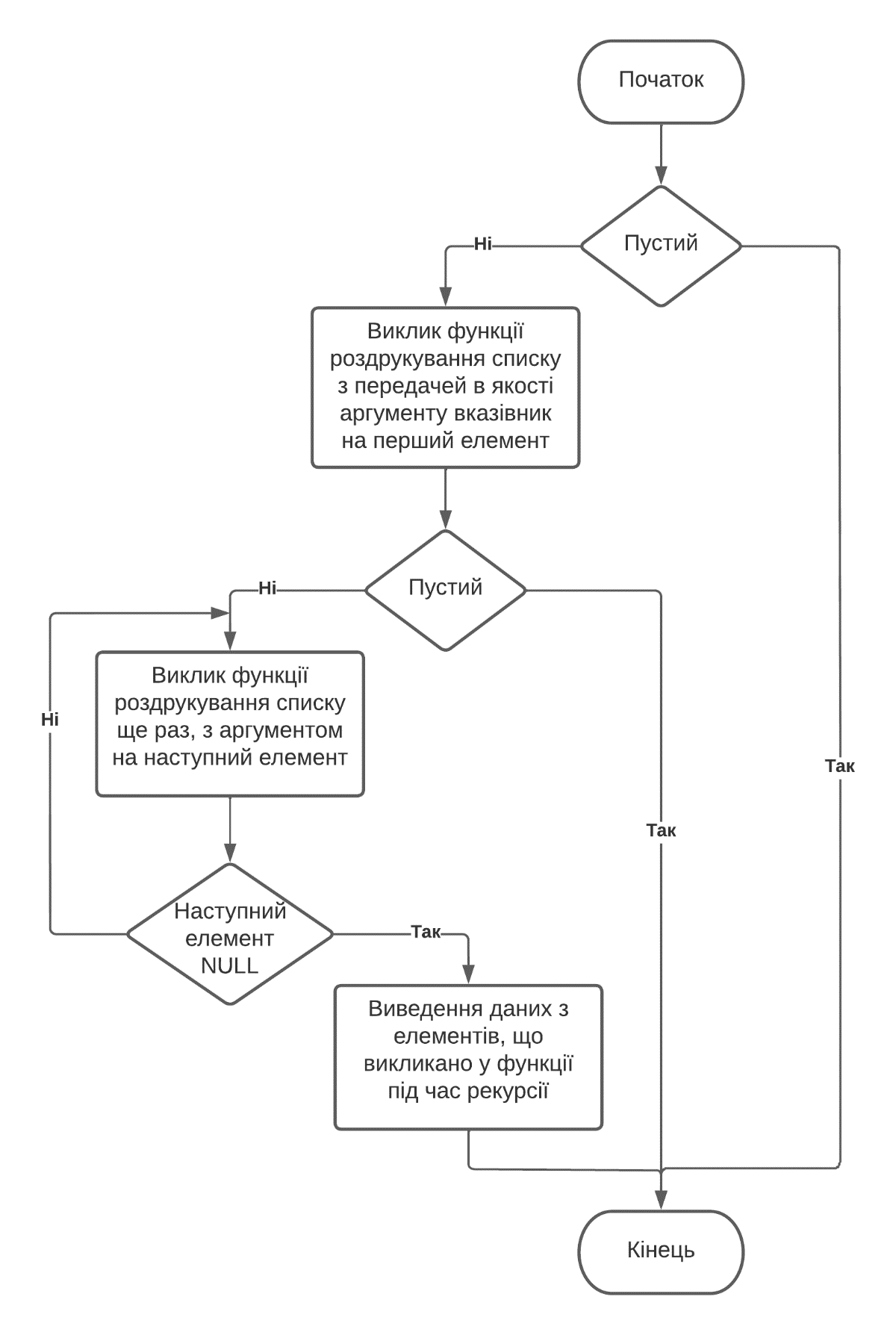
}

}





ФУНКЦІЯ РОЗРОКУВАННЯ СТЕКУ



void Stack::Print\_stack() {

cout << "Елементи стеку: ";

if (this->top == NULL) {

cout << "НЕМАЄ";

return;

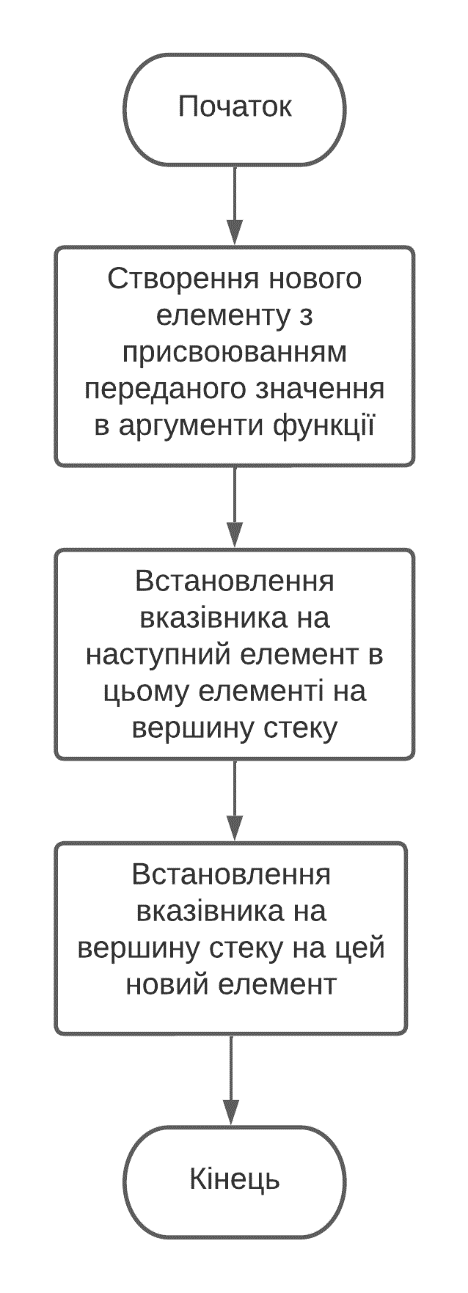
}

Print\_single\_list(this->top);

}



ФУНКЦІЯ ДОДАВАННЯ ЕЛЕМЕНТУ НА ВЕРШИНУ СТЕКУ



void Stack::Push\_top(double value) {

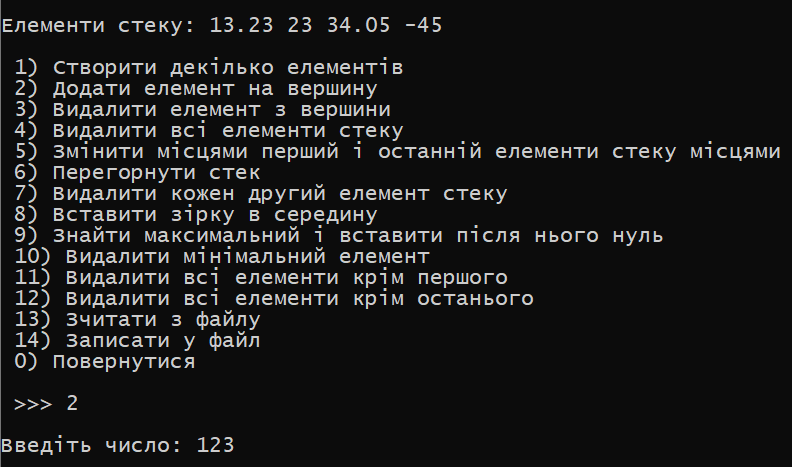
Single\_List\* ptr = new Single\_List(value);

ptr->next = this->top;

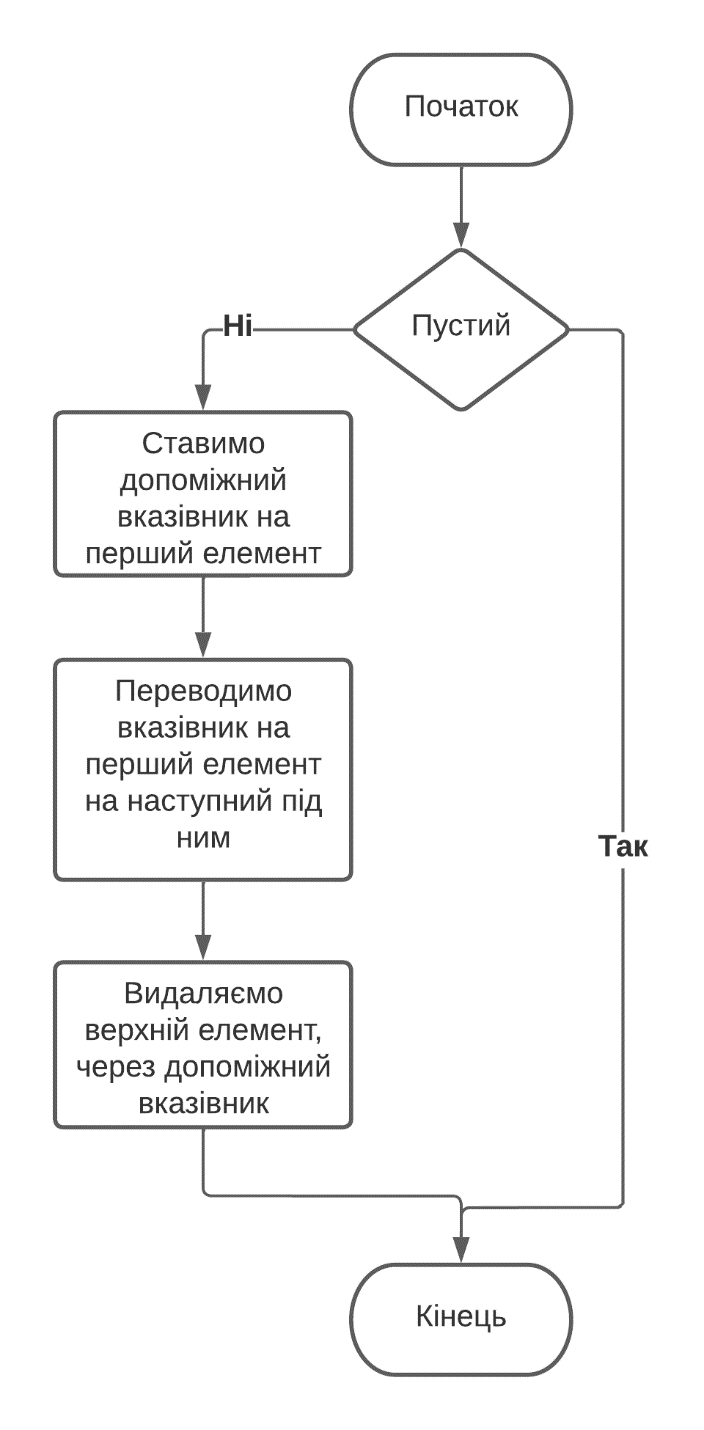
this->top = ptr;

this->size++;

}



ФУНКЦІЯ ВИДАЛЕННЯ ЕЛЕМЕНТУ З ВЕРШИНИ СТЕКУ



void Stack::Pop\_top() {

if (this->top == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

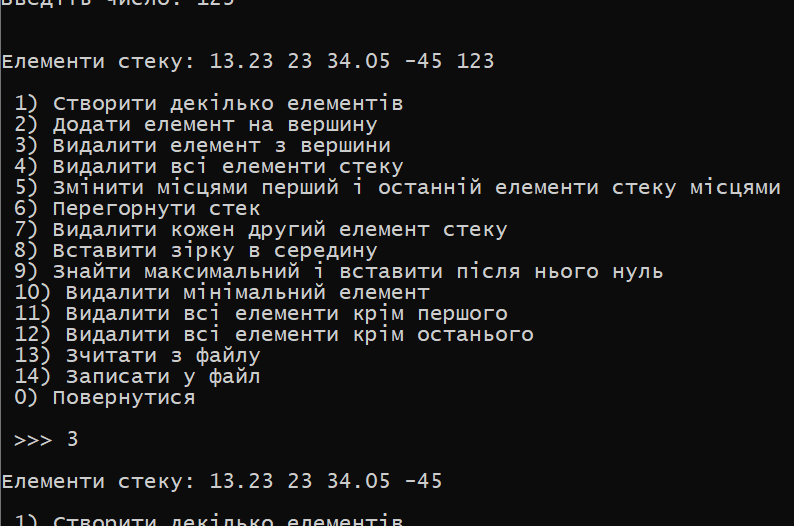
Single\_List\* ptr = this->top;

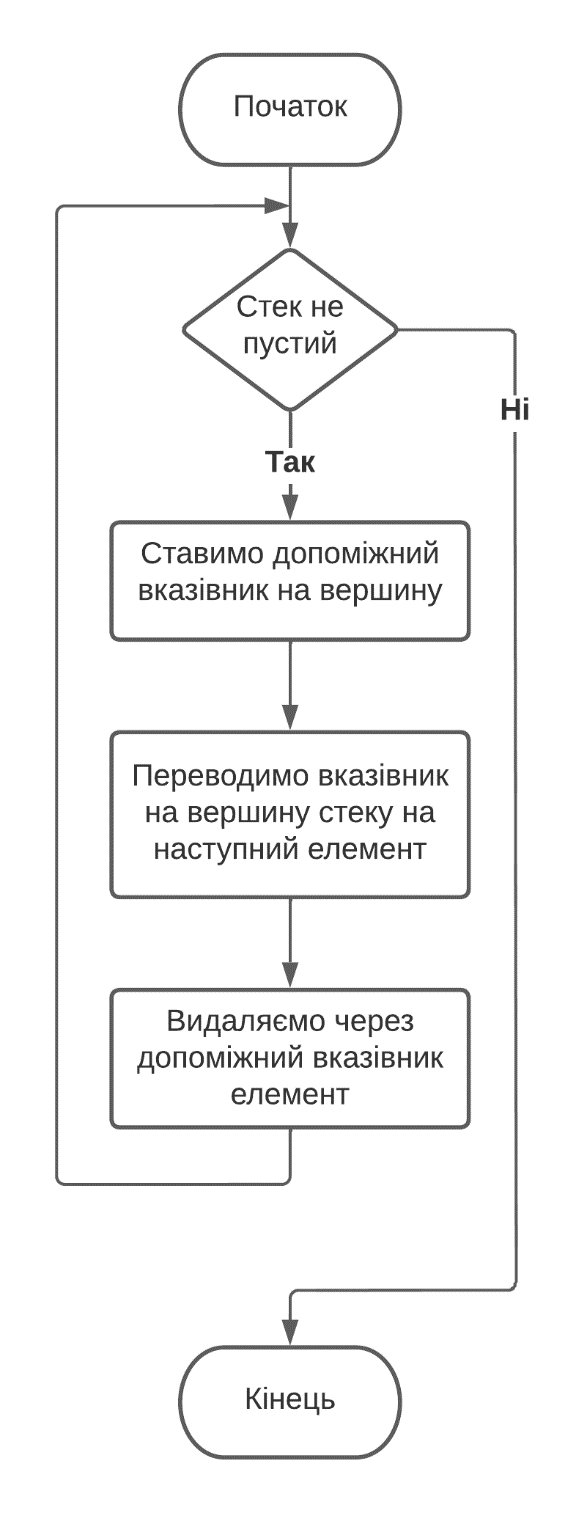
this->top = this->top->next;

delete ptr;

this->size--;

}



ФУНКЦІЯ ОЧИЩЕННЯ СТЕКУ

void Stack::Clear\_stack() {

Single\_List\* ptr\_del;

while (this->top != NULL) {

ptr\_del = this->top;

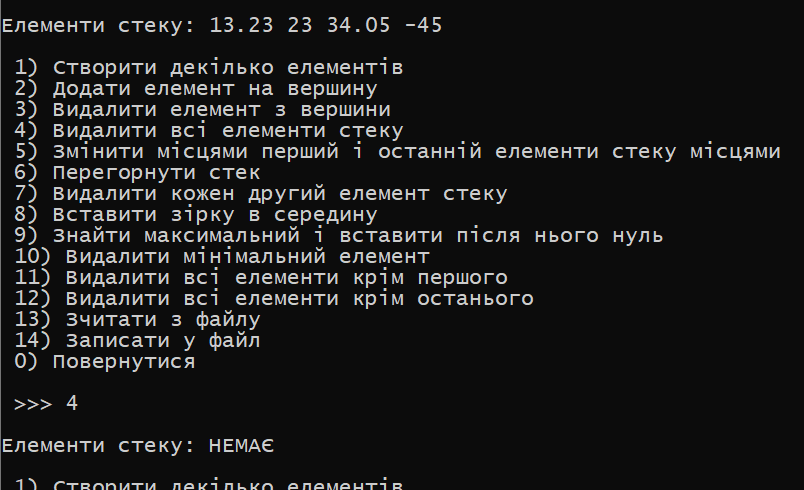
this->top = this->top->next;

delete ptr\_del;

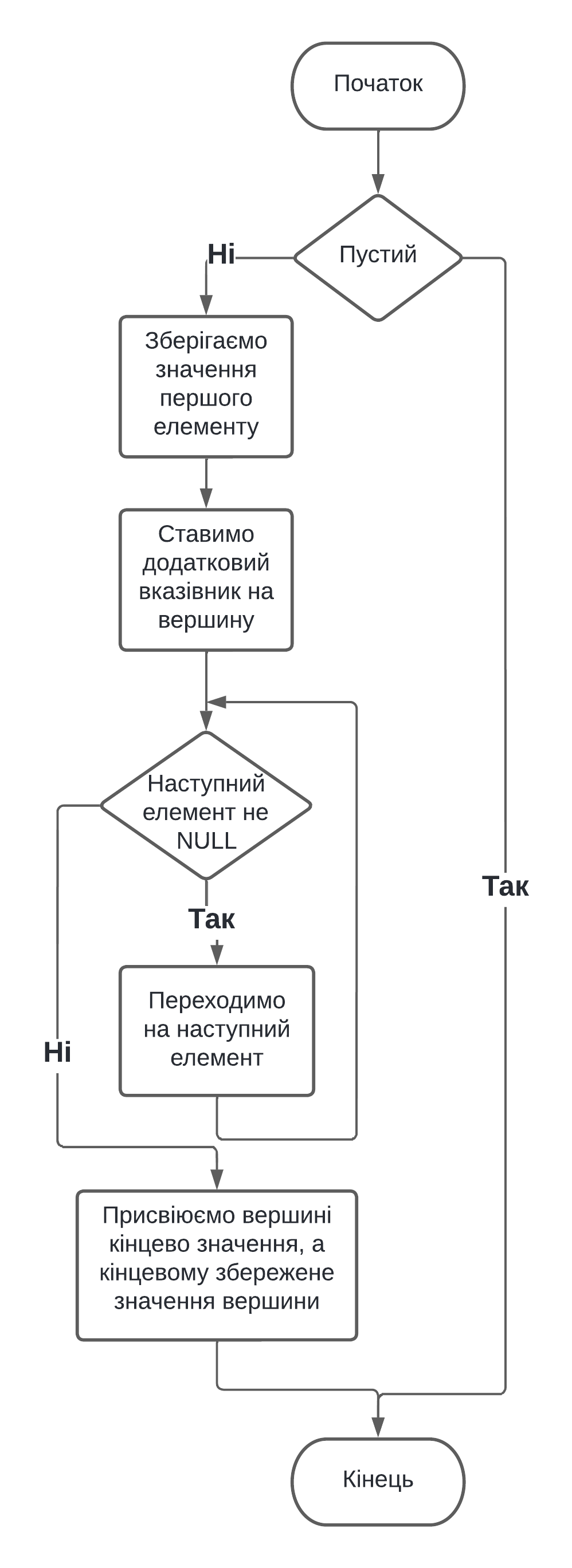
}

this->size = 0;

}



ФУНКЦІЯ ЗМІНИ ПЕРШОГО І ОСТАНЬОГО ЕЛЕМЕНТУ МІСЦЯМИ



void Stack::Swap\_first\_last() {

if (this->top == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

double temp = this->top->numb;

Single\_List\* ptr = this->top;

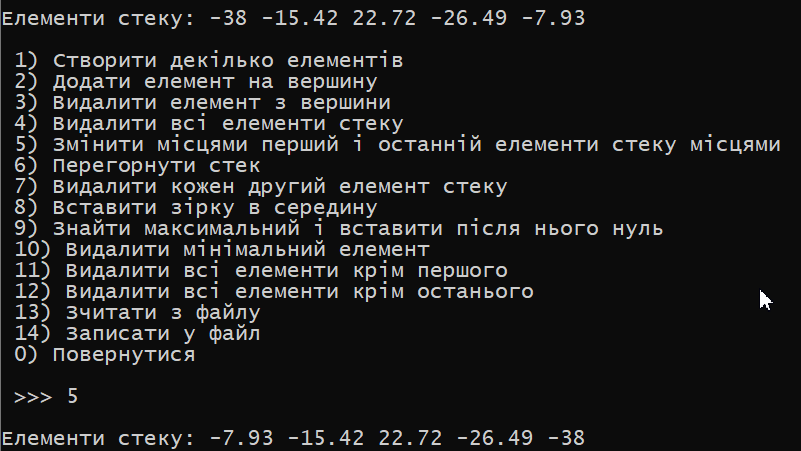
while (ptr->next != NULL)

ptr = ptr->next;

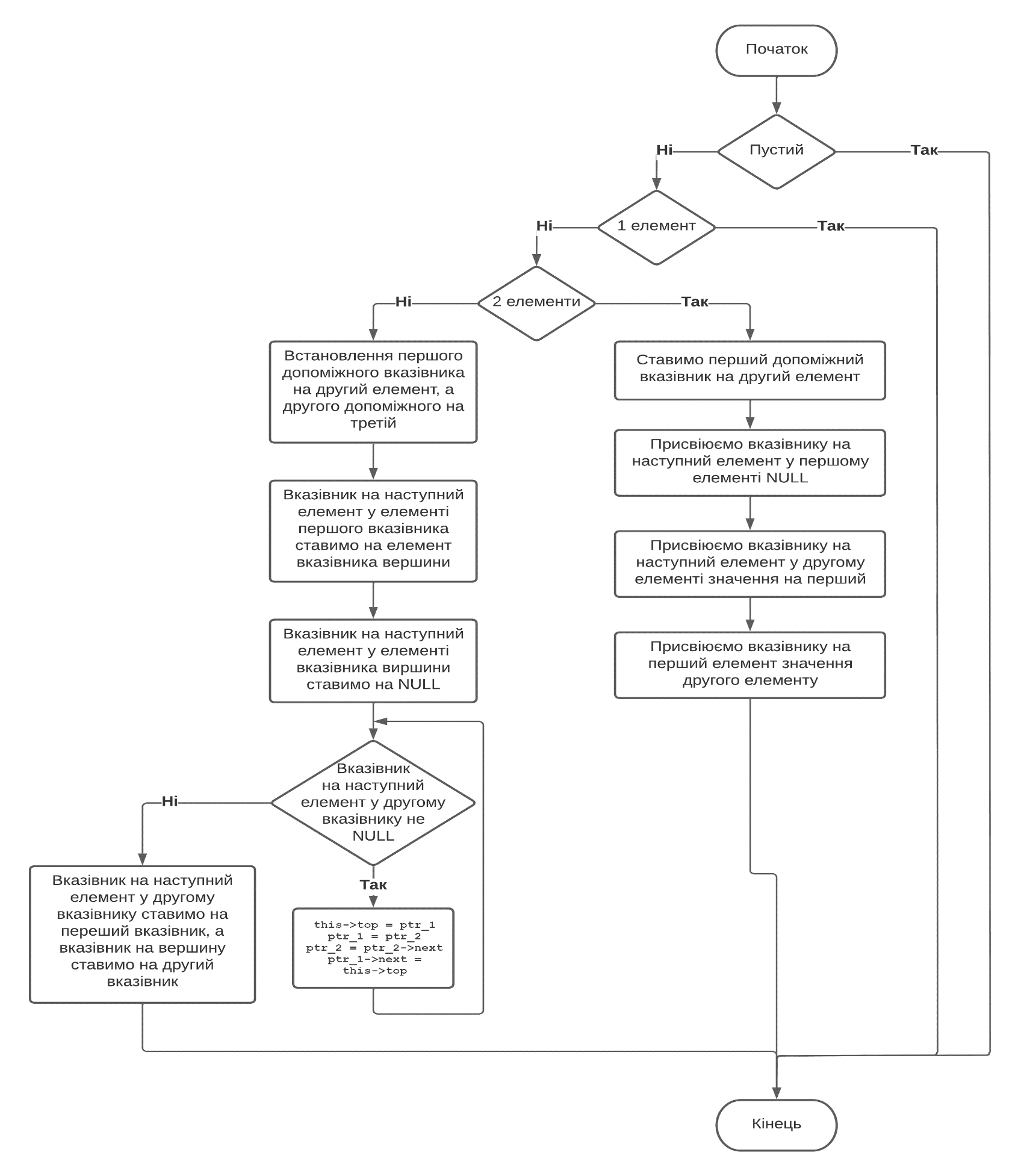
this->top->numb = ptr->numb;

ptr->numb = temp;

}



ФУНКЦІЯ ПЕРЕГОРТАННЯ СТЕКУ



void Stack::Flip\_stack() {

if (this->top == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

Single\_List\* ptr\_1 = NULL, \* ptr\_2 = NULL;

if (this->top->next == NULL)

return;

if (this->top->next->next == NULL) {

ptr\_1 = this->top->next;

this->top->next = NULL;

ptr\_1->next = this->top;

this->top = ptr\_1;

return;

}

ptr\_1 = this->top->next;

ptr\_2 = ptr\_1->next;

ptr\_1->next = this->top;

this->top->next = NULL;

while (ptr\_2->next != NULL) {

this->top = ptr\_1;

ptr\_1 = ptr\_2;

ptr\_2 = ptr\_2->next;

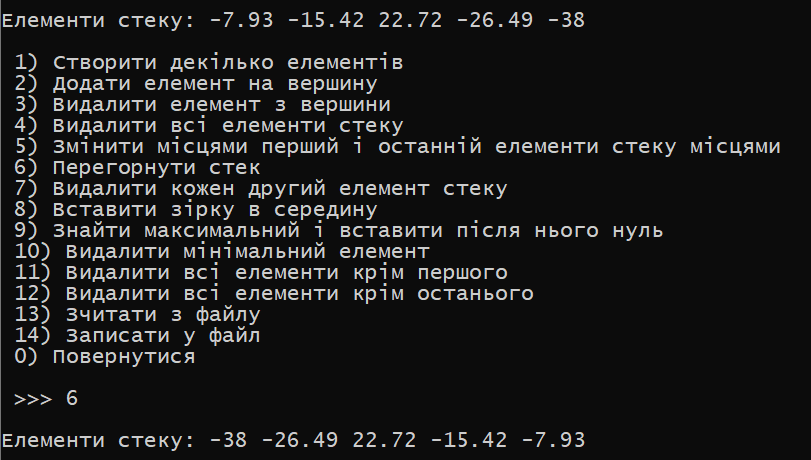
ptr\_1->next = this->top;

}

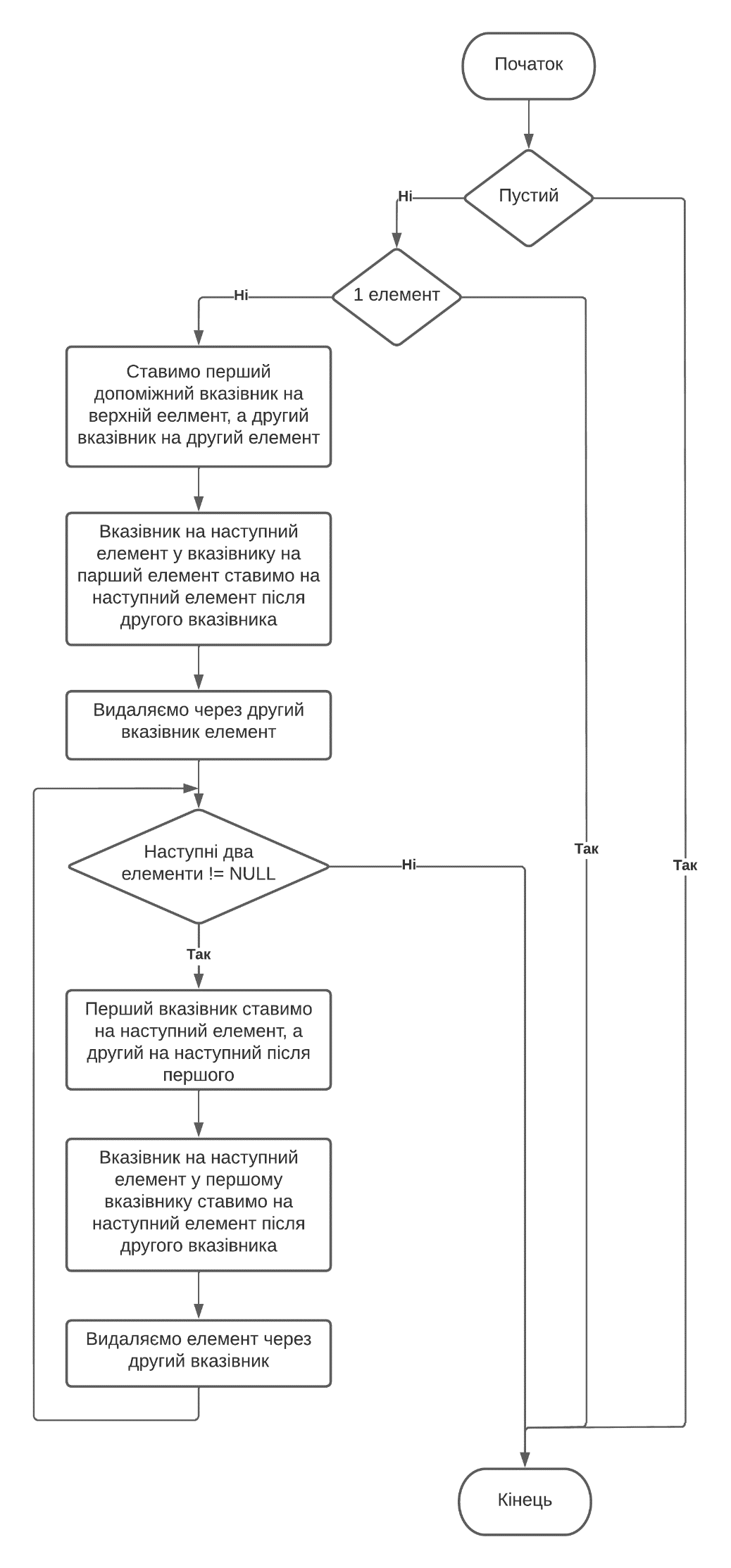
ptr\_2->next = ptr\_1;

this->top = ptr\_2;

}



ФУНКЦІЯ ВИДАЛЕННЯ КОЖНОГО ДРУГОГО ЕЛЕМЕНТУ



void Stack::Delete\_everyone\_second() {

if (this->top == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

Single\_List\* ptr\_back = this->top, \* ptr\_del = this->top->next;

if (this->top->next == NULL)

return;

ptr\_back->next = ptr\_del->next;

delete ptr\_del;

this->size--;

while (ptr\_back->next != NULL && ptr\_back->next->next != NULL) {

ptr\_back = ptr\_back->next;

ptr\_del = ptr\_back->next;

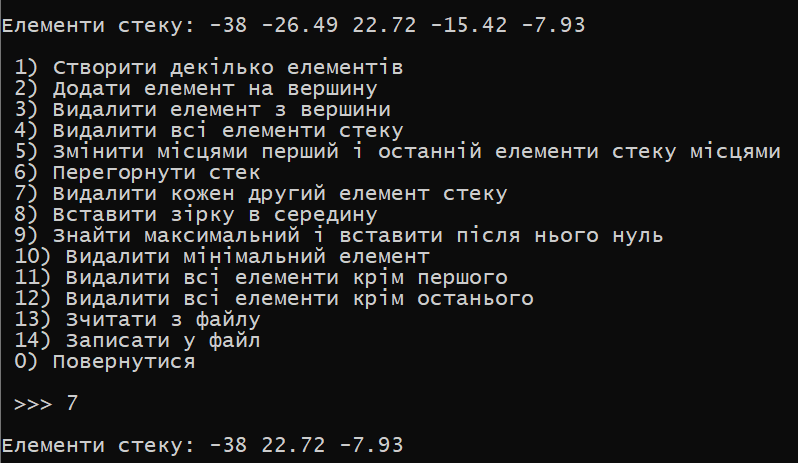
ptr\_back->next = ptr\_del->next;

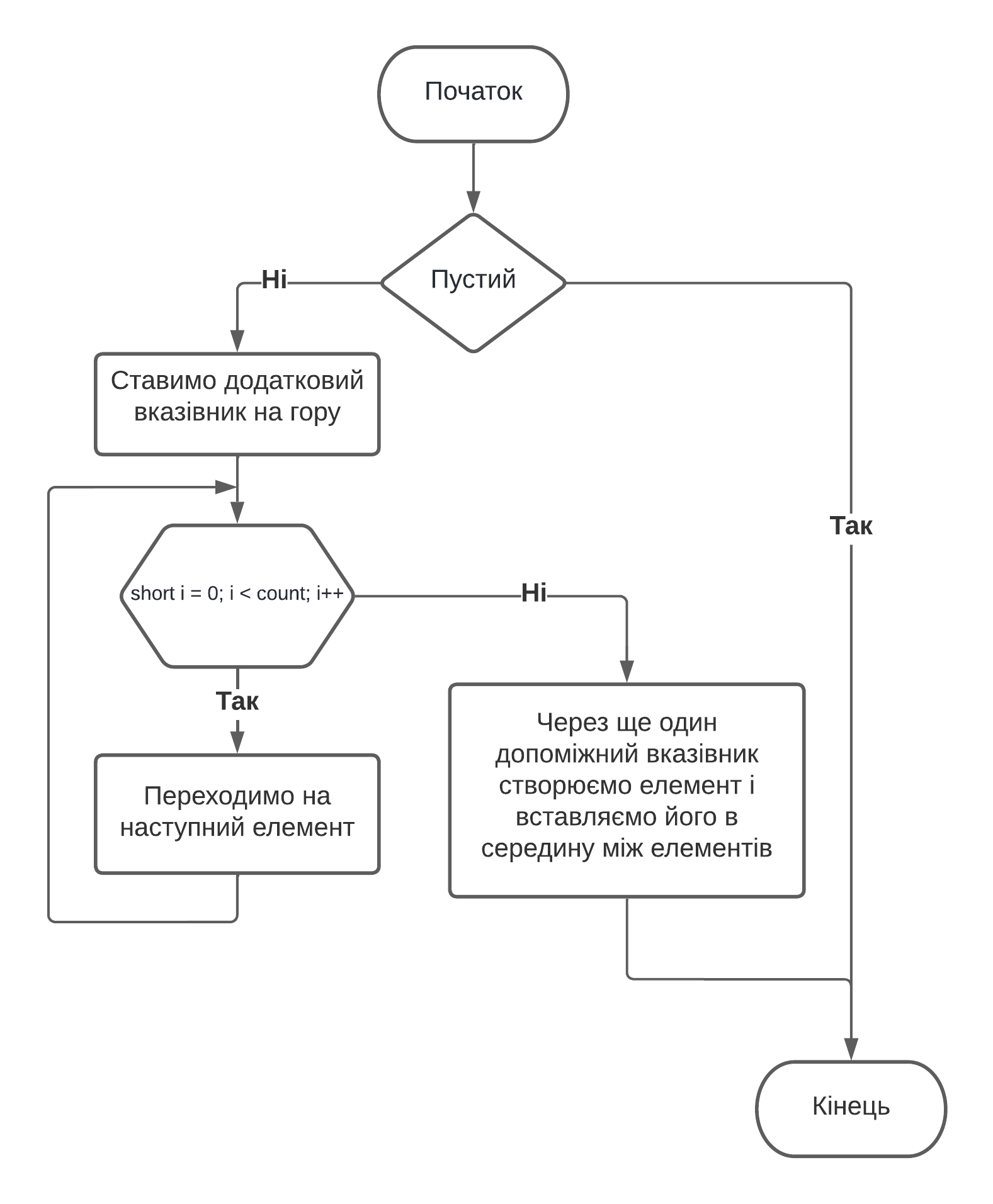
delete ptr\_del;

this->size--;

}

}



ФУНКЦІЯ ВСТАВЛЕННЯ ЗІРКИ В СЕРЕДИНУ СТЕКУ

void Stack::Insert\_star() {

if (this->top == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

Single\_List\* ptr\_search = this->top, \* ptr = NULL;

int count = ceil(this->size / 2.0) - 1;

for (short i = 0; i < count; i++)

ptr\_search = ptr\_search->next;

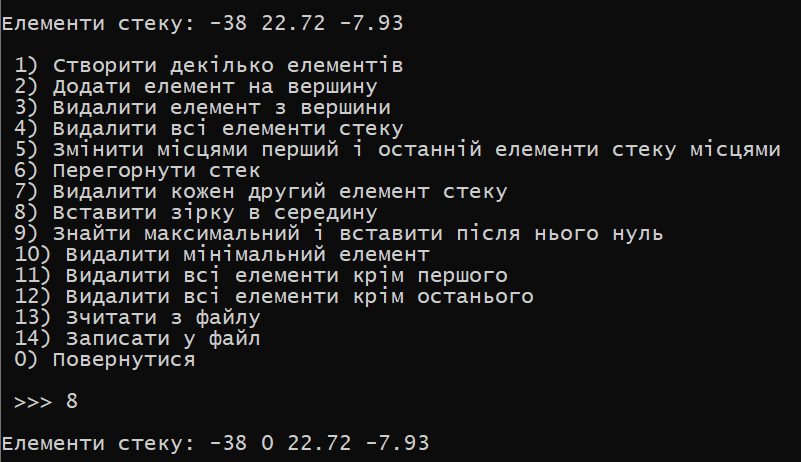
ptr = new Single\_List(0);

ptr->next = ptr\_search->next;

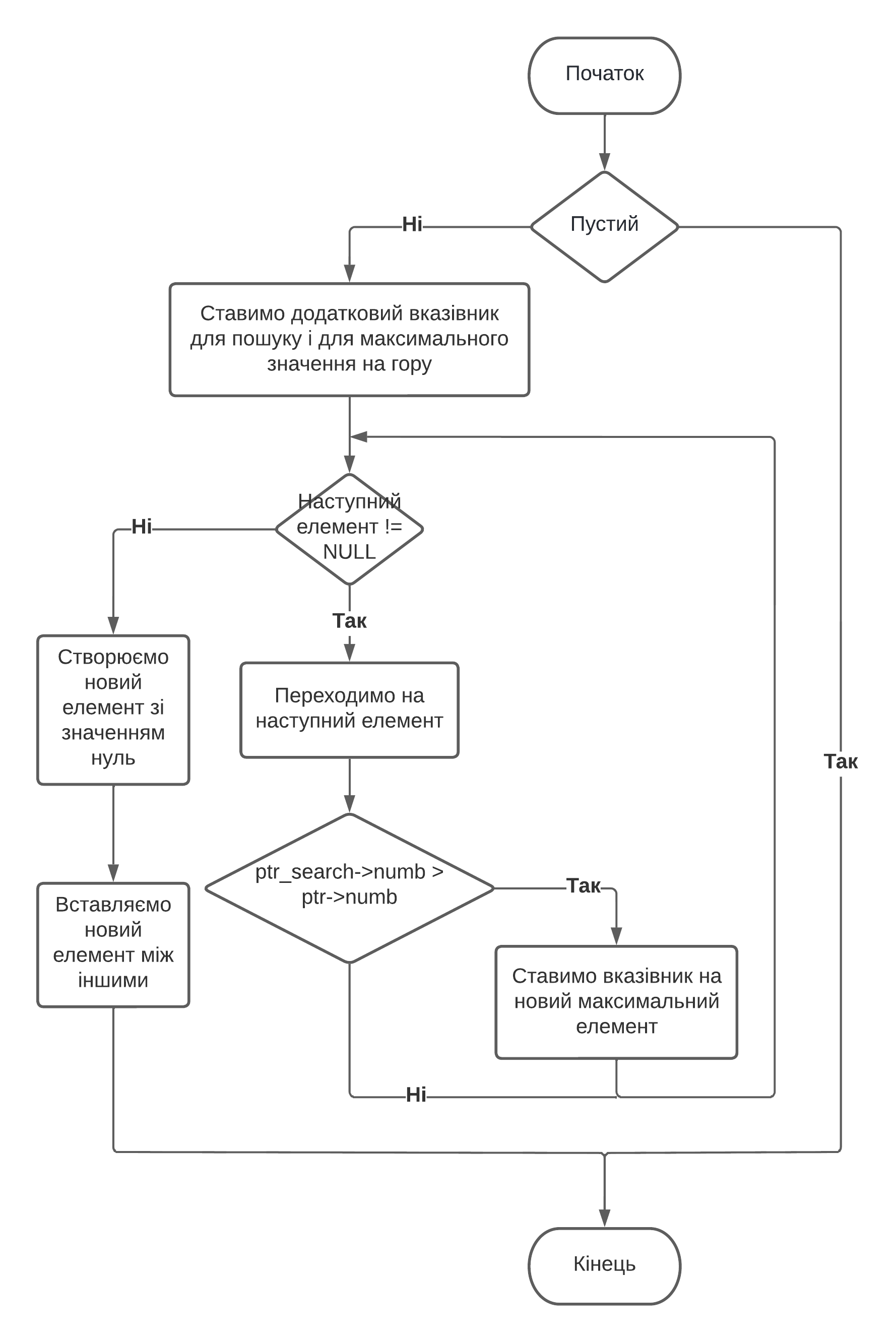
ptr\_search->next = ptr;

this->size++;

}



ФУНКЦІЯ ВСТАВЛЕННЯ НУЛЮ ПІСЛЯ МАКСИМАЛЬНОГО ЕЛЕМЕНТУ



void Stack::Search\_max\_and\_insert\_after\_zero() {

if (this->top == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

Single\_List\* ptr = this->top, \* ptr\_search = this->top;

while (ptr\_search->next != NULL) {

ptr\_search = ptr\_search->next;

if (ptr\_search->numb > ptr->numb)

ptr = ptr\_search;

}

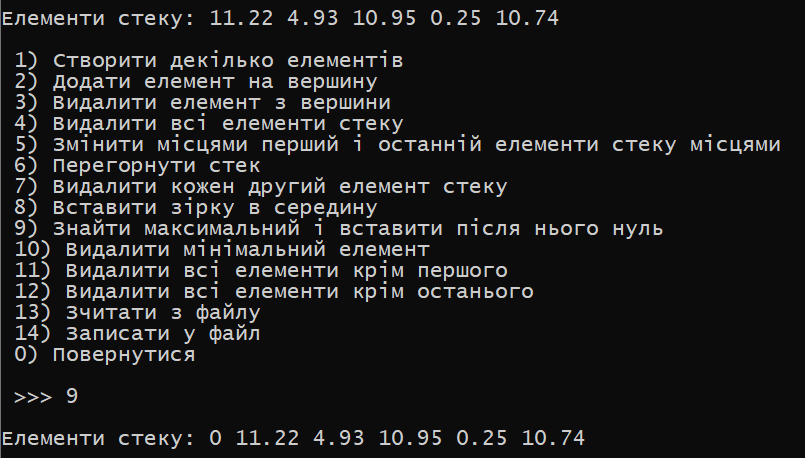
ptr\_search = new Single\_List(0);

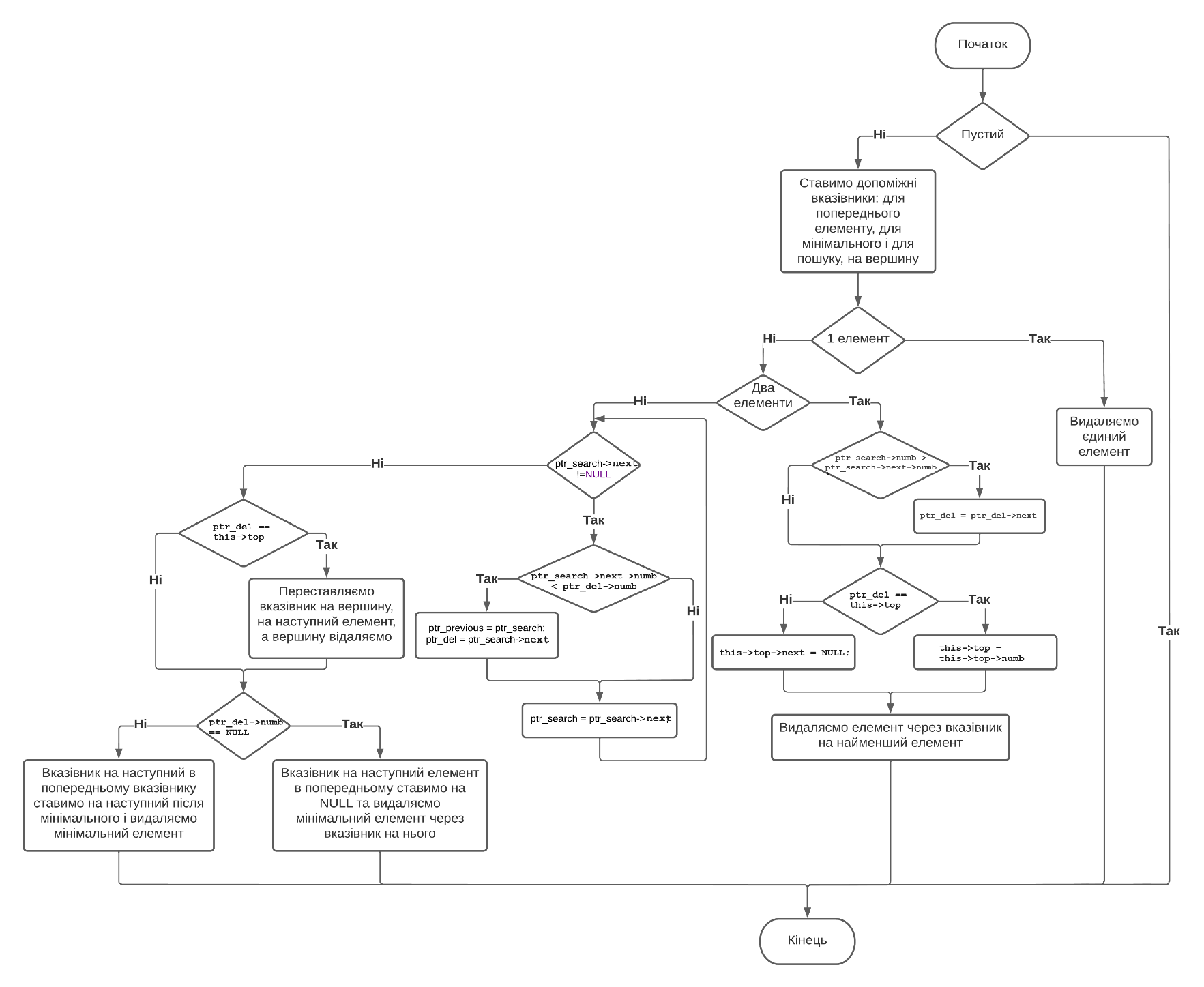
ptr\_search->next = ptr->next;

ptr->next = ptr\_search;

this->size++;

}



ФУНКЦІЯ ВИДАЛЕННЯ МІНІМАЛЬНОГО ЕЛЕМЕНТУ

void Stack::Delete\_min() {

if (this->top == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

Single\_List\* ptr\_back = this->top,

\* ptr\_del = this->top,

\* ptr\_search = this->top;

if (this->top->next == NULL) {

delete this->top;

this->size--;

this->top = NULL;

return;

}

if (this->top->next->next == NULL) {

if (ptr\_search->numb > ptr\_search->next->numb)

ptr\_del = ptr\_del->next;

if (ptr\_del == this->top)

this->top = this->top->next;

else

this->top->next = NULL;

delete ptr\_del;

this->size--;

return;

}

while (ptr\_search->next != NULL) {

if (ptr\_search->next->numb < ptr\_del->numb) {

ptr\_back = ptr\_search;

ptr\_del = ptr\_search->next;

}

ptr\_search = ptr\_search->next;

}

if (ptr\_del == this->top) {

this->top = this->top->next;

delete ptr\_del;

this->size--;

return;

}

if (ptr\_del->next == NULL) {

ptr\_back->next = NULL;

delete ptr\_del;

this->size--;

return;

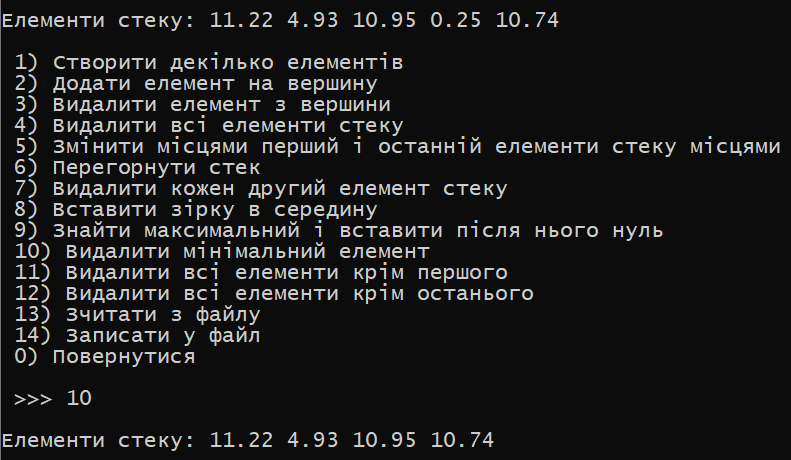
}

ptr\_back->next = ptr\_del->next;

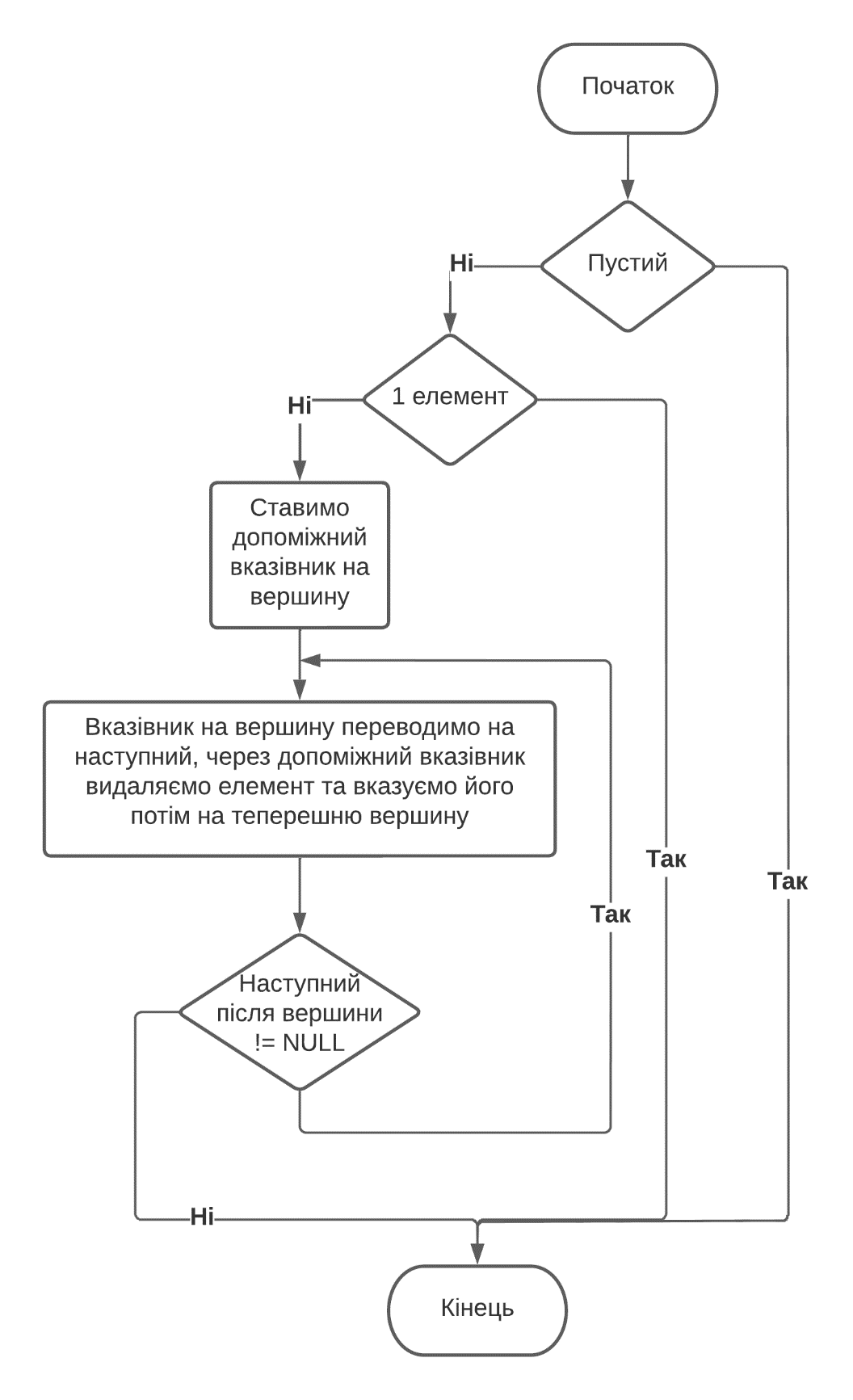
delete ptr\_del;

this->size--;

}



ФУНКЦІЯ ВИДАЛЕННЯ ВСІХ ЕЛЕМЕНТІВ КРІМ ПЕРШОГО



void Stack::Clear\_besides\_first() {

if (this->top == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

Single\_List\* ptr\_del = this->top;

if (this->top->next == NULL)

return;

do {

this->top = this->top->next;

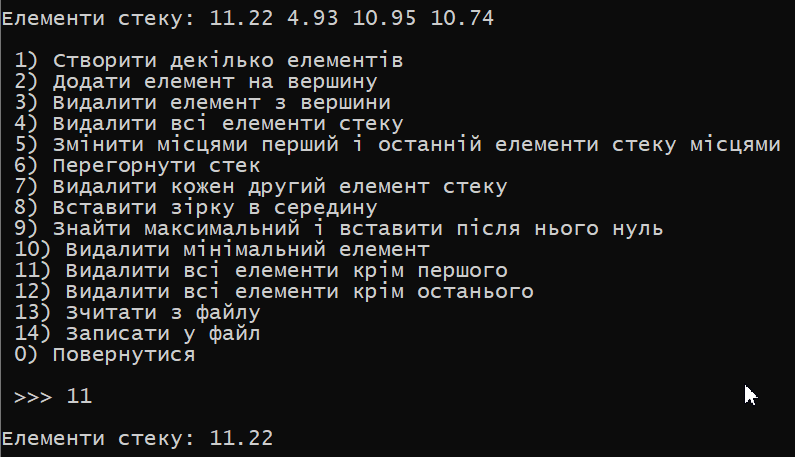
delete ptr\_del;

ptr\_del = this->top;

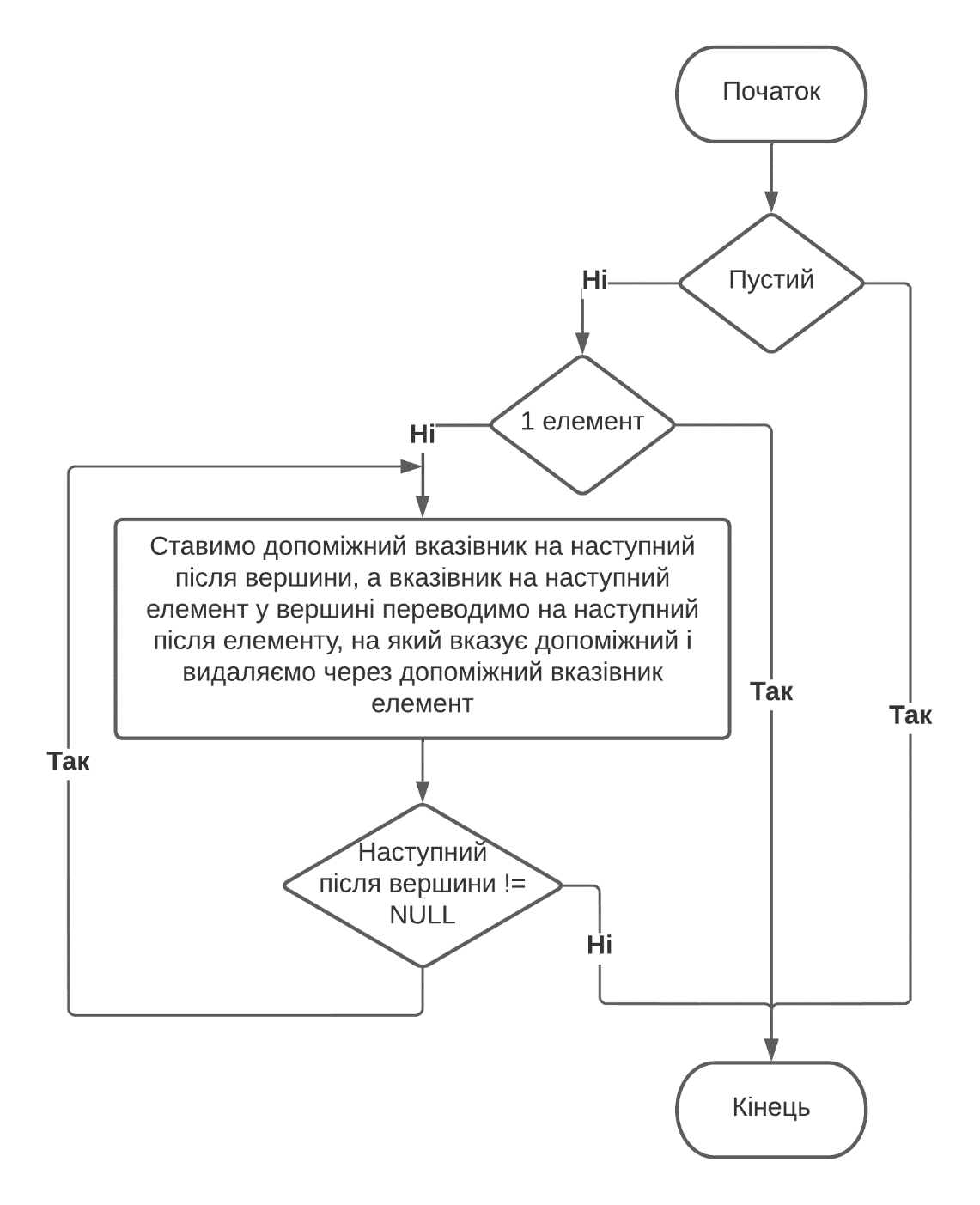
} while (this->top->next != NULL);

this->size = 1;

}



ФУНКЦІЯ ВИДАЛЕННЯ ВСІХ ЕЛЕМЕНТІВ КРІМ ОСТАНЬОГО



void Stack::Clear\_besides\_last() {

if (this->top == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

Single\_List\* ptr\_del = NULL;

if (this->top->next == NULL)

return;

do {

ptr\_del = this->top->next;

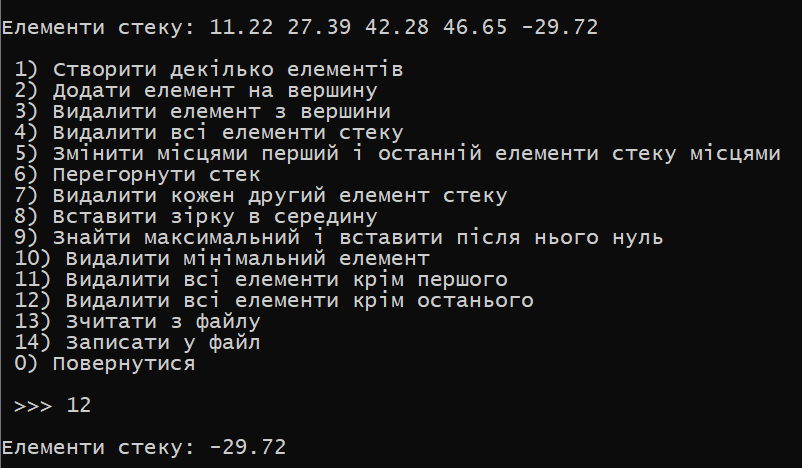
this->top->next = ptr\_del->next;

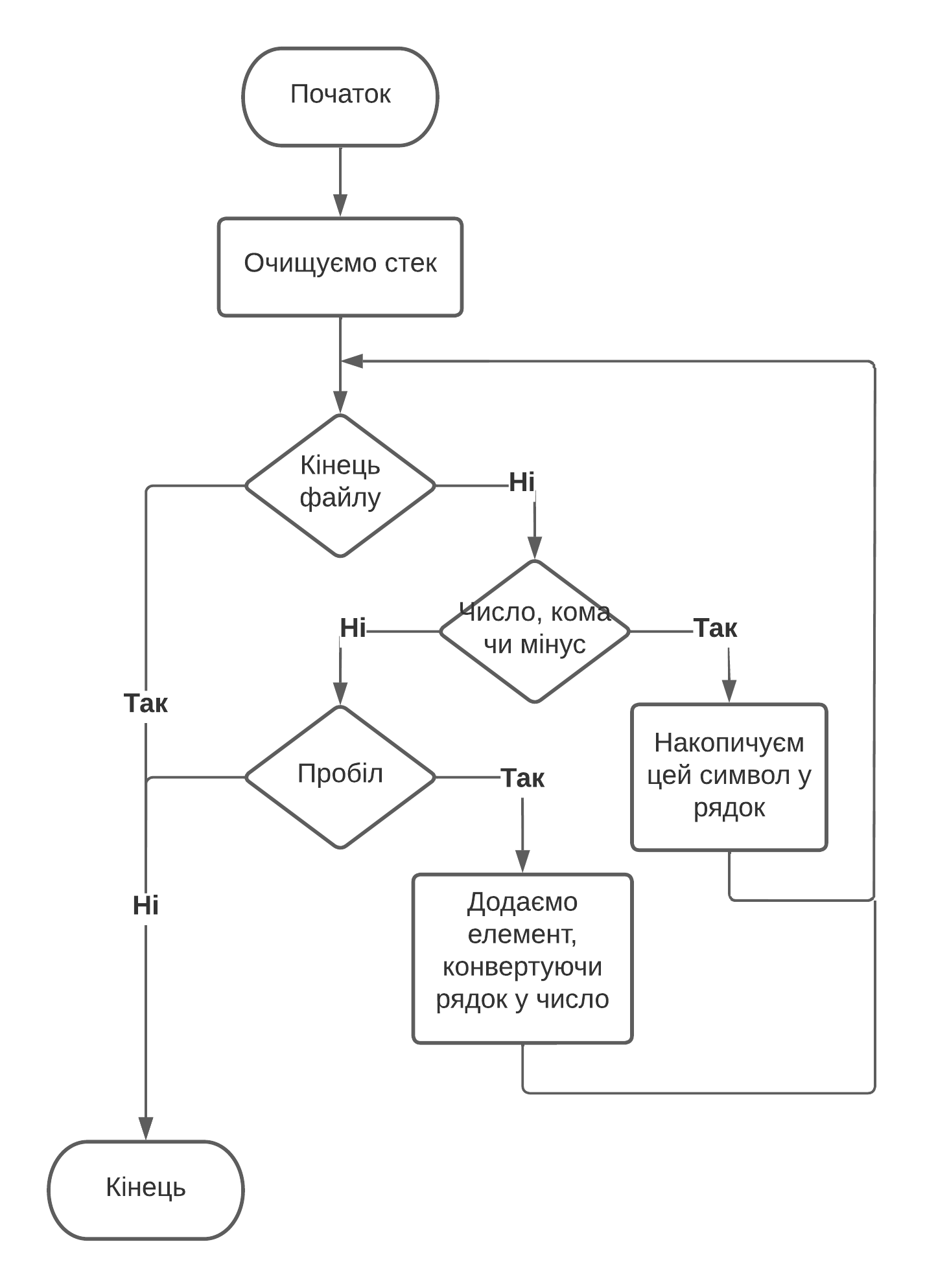
delete ptr\_del;

} while (this->top->next != NULL);

this->size = 1;

}



ФУНКЦІЯ ЗЧИТУВАННЯ З ФАЙЛУ

void Stack::Read\_from\_file(ifstream& f\_in) {

this->Clear\_stack();

Single\_List\* ptr = NULL;

string str = "";

char ch;

while ((ch = f\_in.get()) != EOF) {

if ((ch > 47 && ch < 58) || ch == 46 || ch == 45)

str += (char)ch;

else if (ch == 32) {

if (this->top == NULL)

this->top = ptr = new Single\_List(atof(str.c\_str()));

else {

ptr->next = new Single\_List(atof(str.c\_str()));

ptr = ptr->next;

}

this->size++;

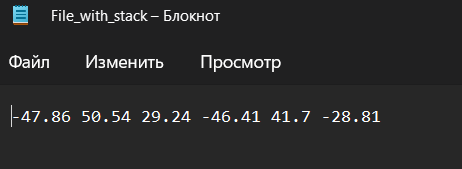
str = "";

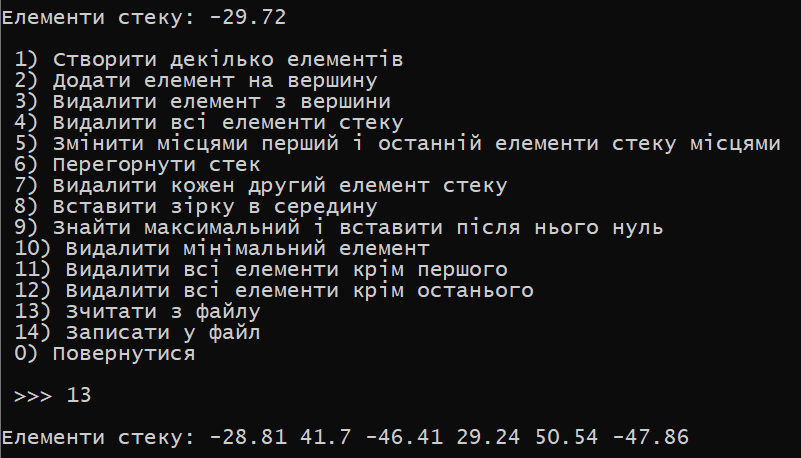
}

else break;

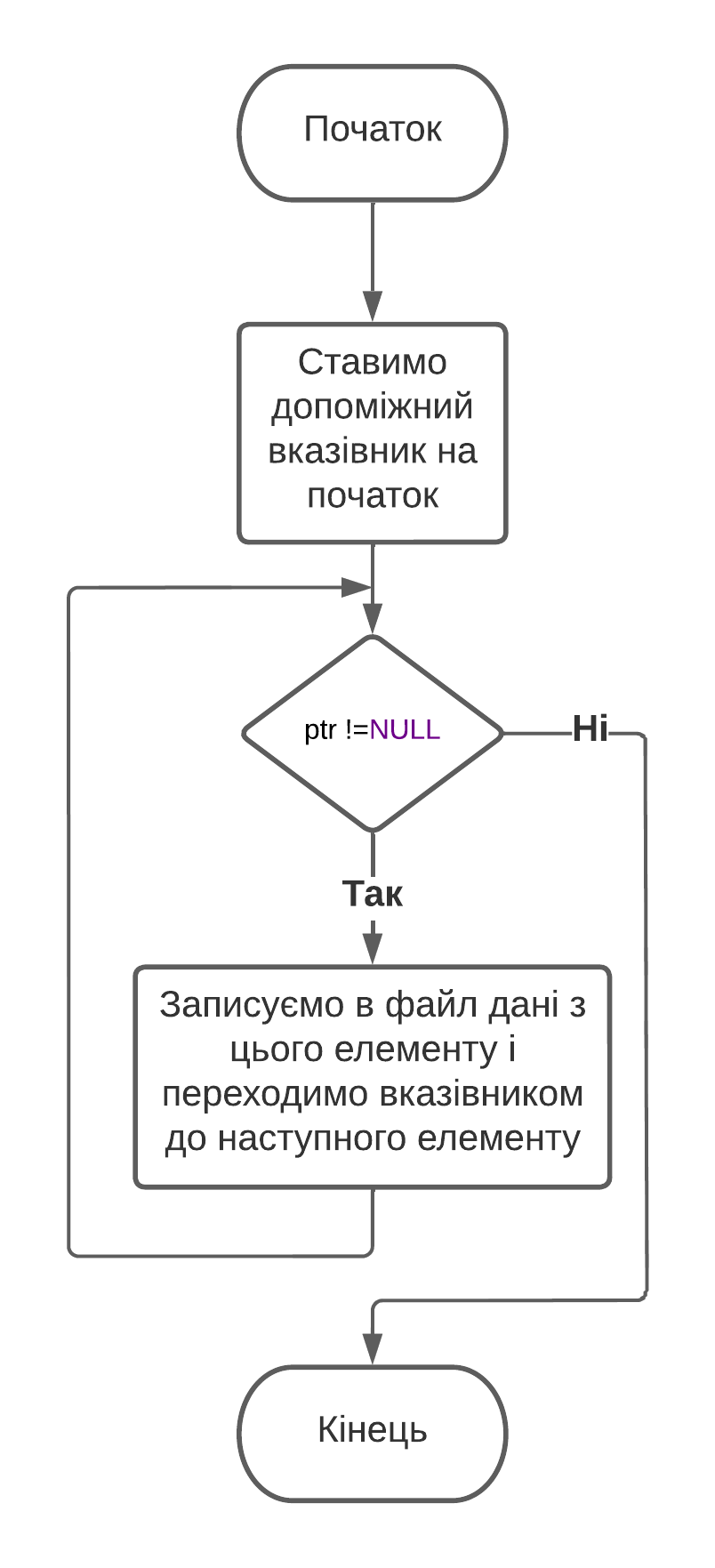
}

}





ФУНКЦІЯ ЗАПИСУВАННЯ У ФАЙЛ



void Stack::Write\_to\_file(ofstream& f\_out) {

Single\_List\* ptr = this->top;

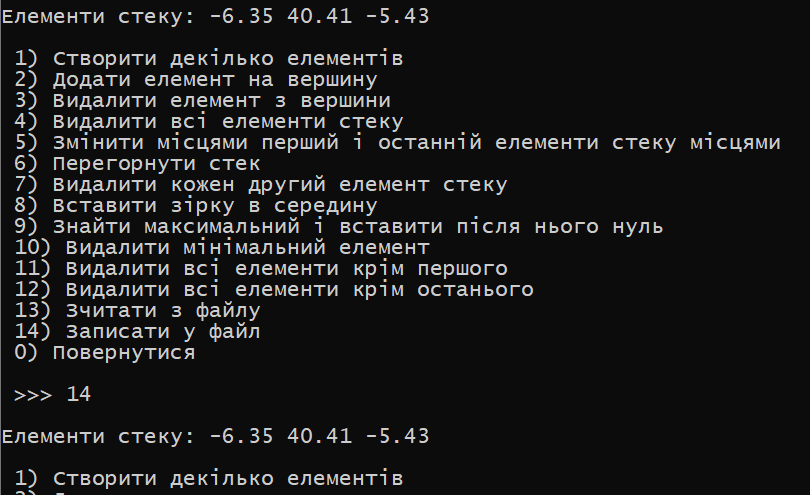
while (ptr != NULL) {

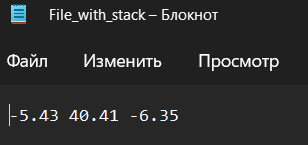
f\_out << ptr->numb << " ";

ptr = ptr->next;

}

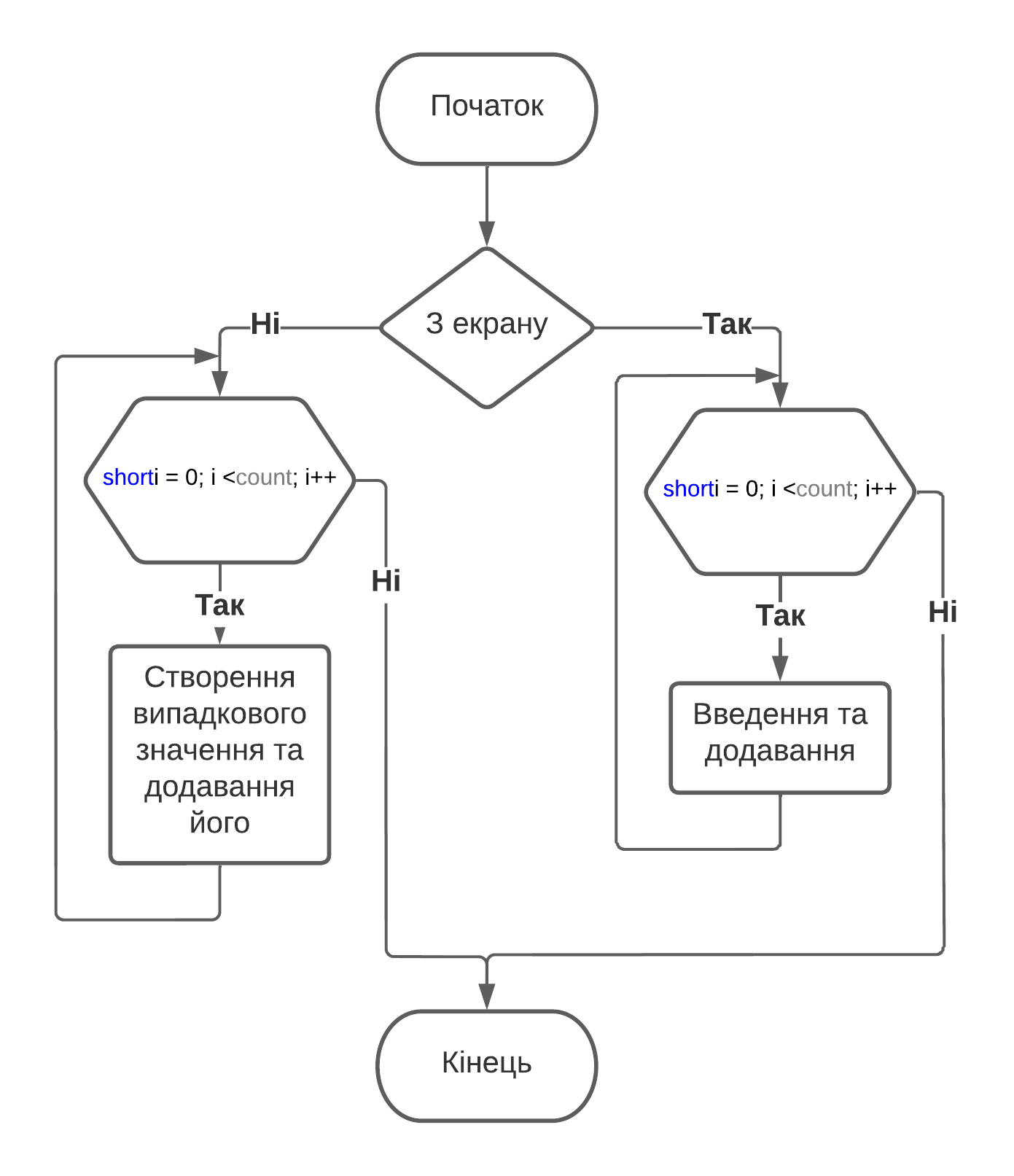
}





ЧЕРГА

ФУНКЦІЯ СТВОРЕННЯ ЧЕРГИ



void Queue::Create\_queue(int count, bool flag) {

double temp;

if (flag == true) {

for (short i = 0; i < count; i++) {

cin >> temp;

this->Push\_back(temp);

}

}

else {

for (short i = 0; i < count; i++) {

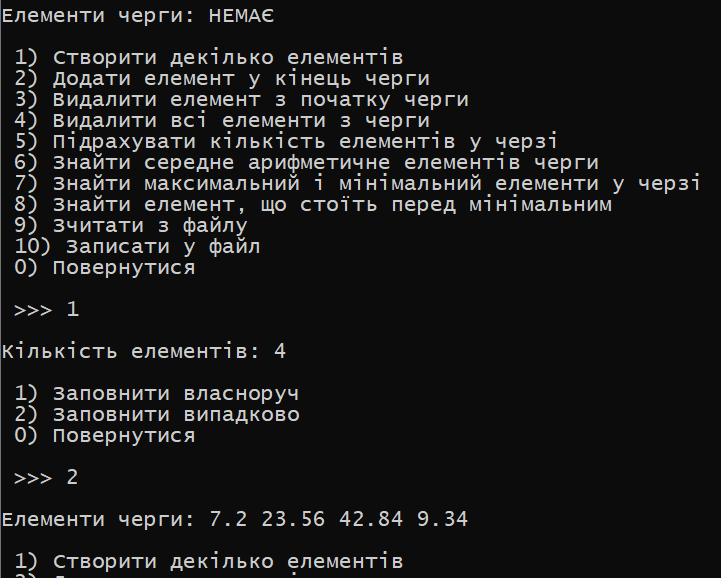
temp = rand() % 101 - 50 + rand() % 101 \* 0.01;

this->Push\_back(temp);

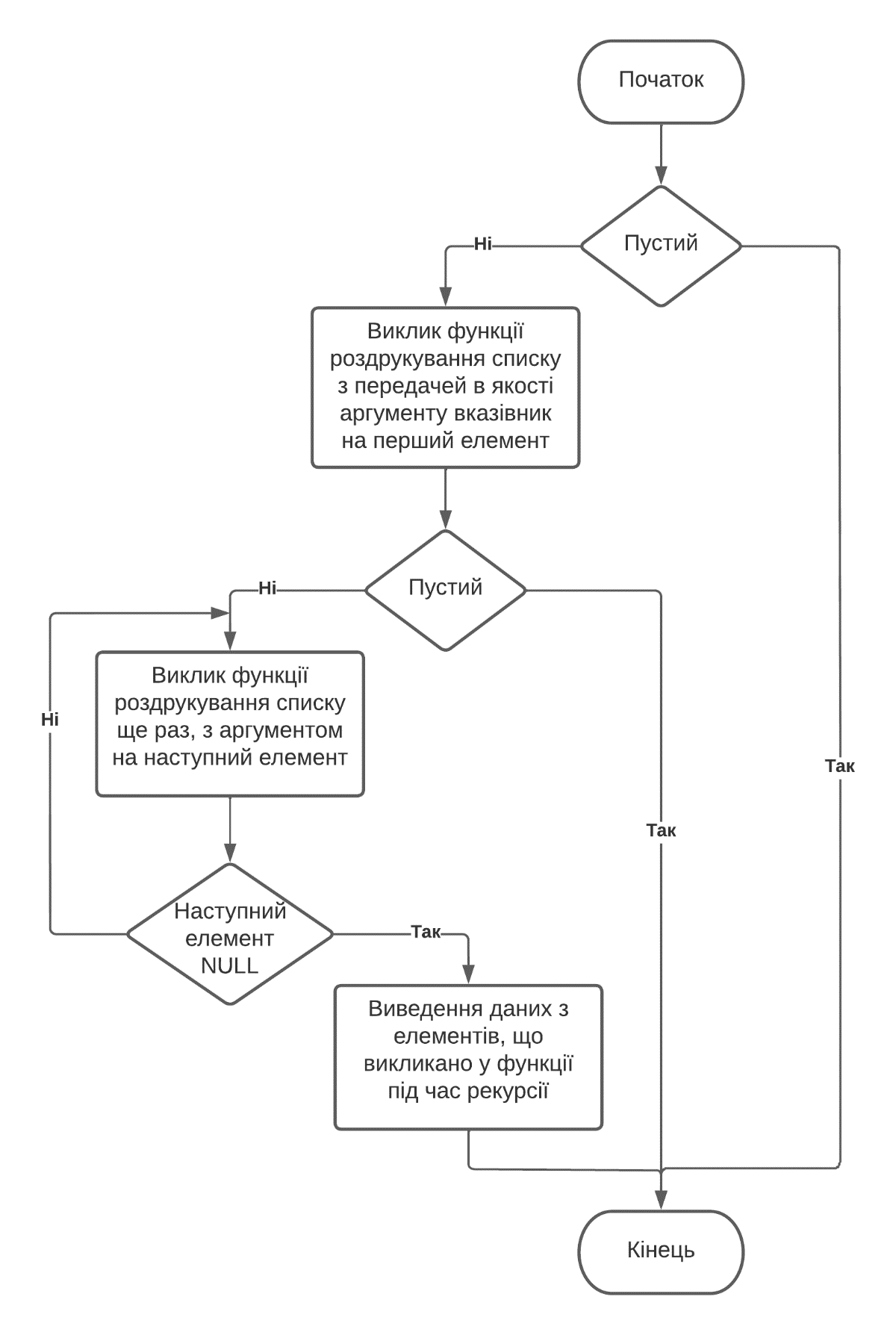
}

}

}



ФУНКЦІЯ ВИВЕДЕННЯ ЧЕРГИ



void Queue::Print\_queue() {

cout << "Елементи черги: ";

if (this->first == NULL) {

cout << "НЕМАЄ";

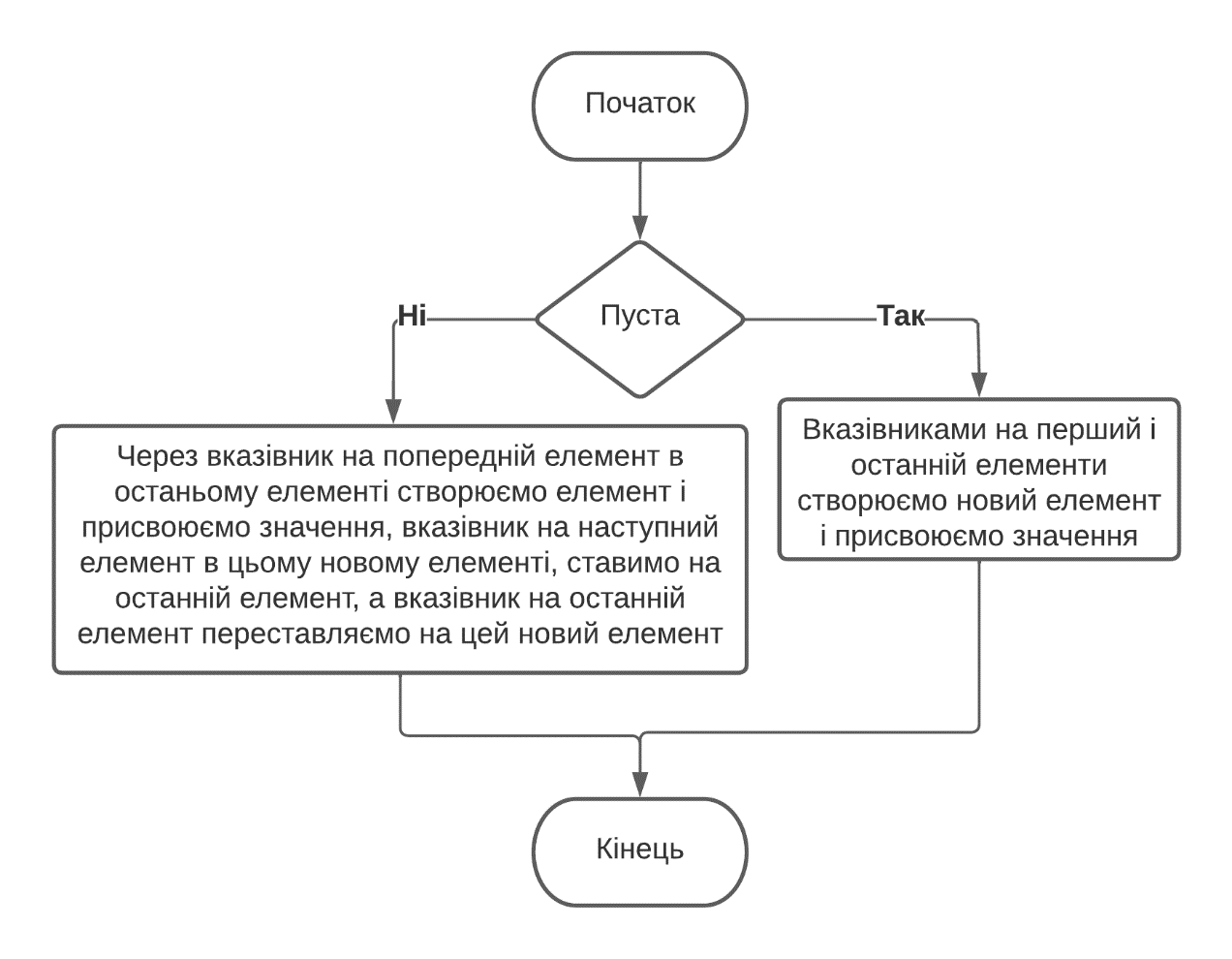
return;

}

Print\_double\_list(this->last);

}



ФУНКЦІЯ ДОДАВАННЯ ЕЛЕМЕНТУ В КІНЕЦЬ ЧЕРГИ

void Queue::Push\_back(double value) {

if (this->first == NULL) {

this->first = this->last = new Double\_List(value);

return;

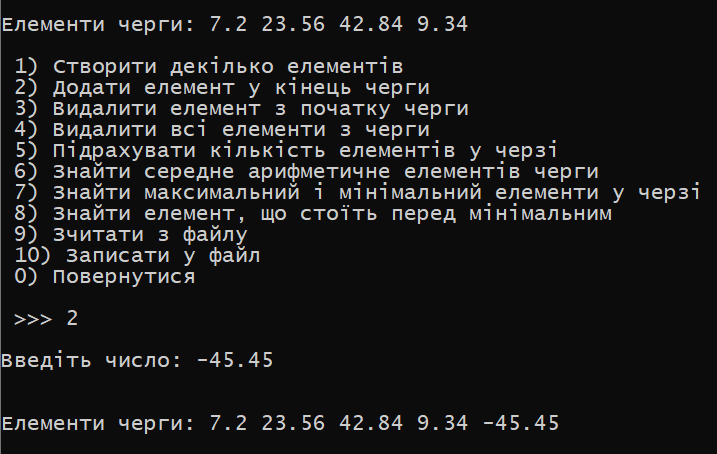
}

this->last->back = new Double\_List(value);

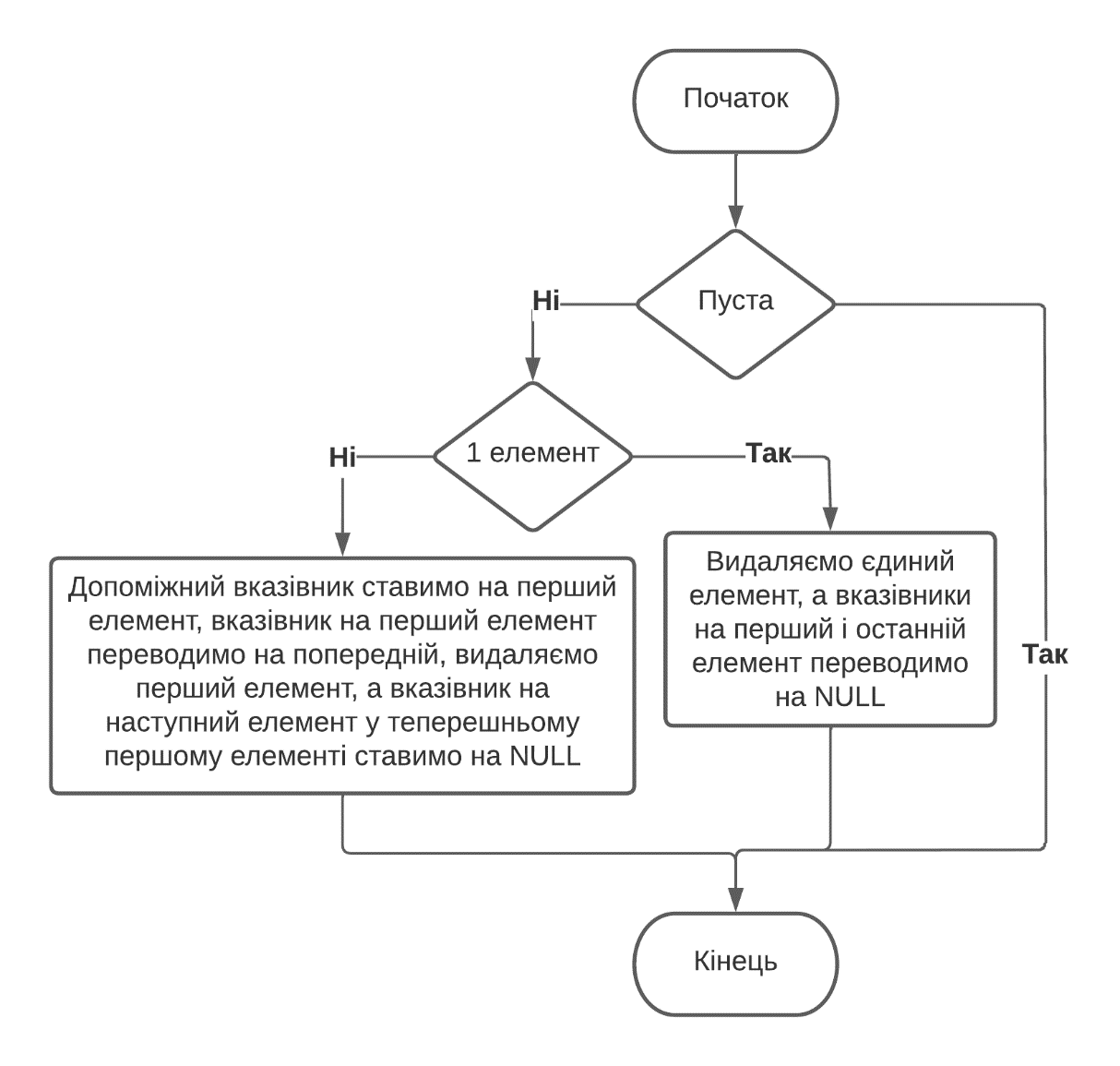
this->last->back->next = this->last;

this->last = this->last->back;

}



ФУНКЦІЯ ВИДАЛЕННЯ ЕЛЕМЕНТУ З ПОЧАТКУ ЧЕРГИ



void Queue::Pop\_front() {

if (this->first == NULL)

return;

Double\_List\* ptr\_del = this->first;

if (this->first->back == NULL) {

delete this->first;

this->first = this->last = NULL;

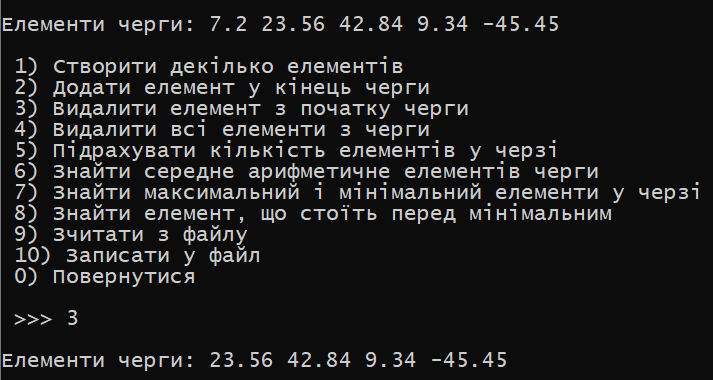
return;

}

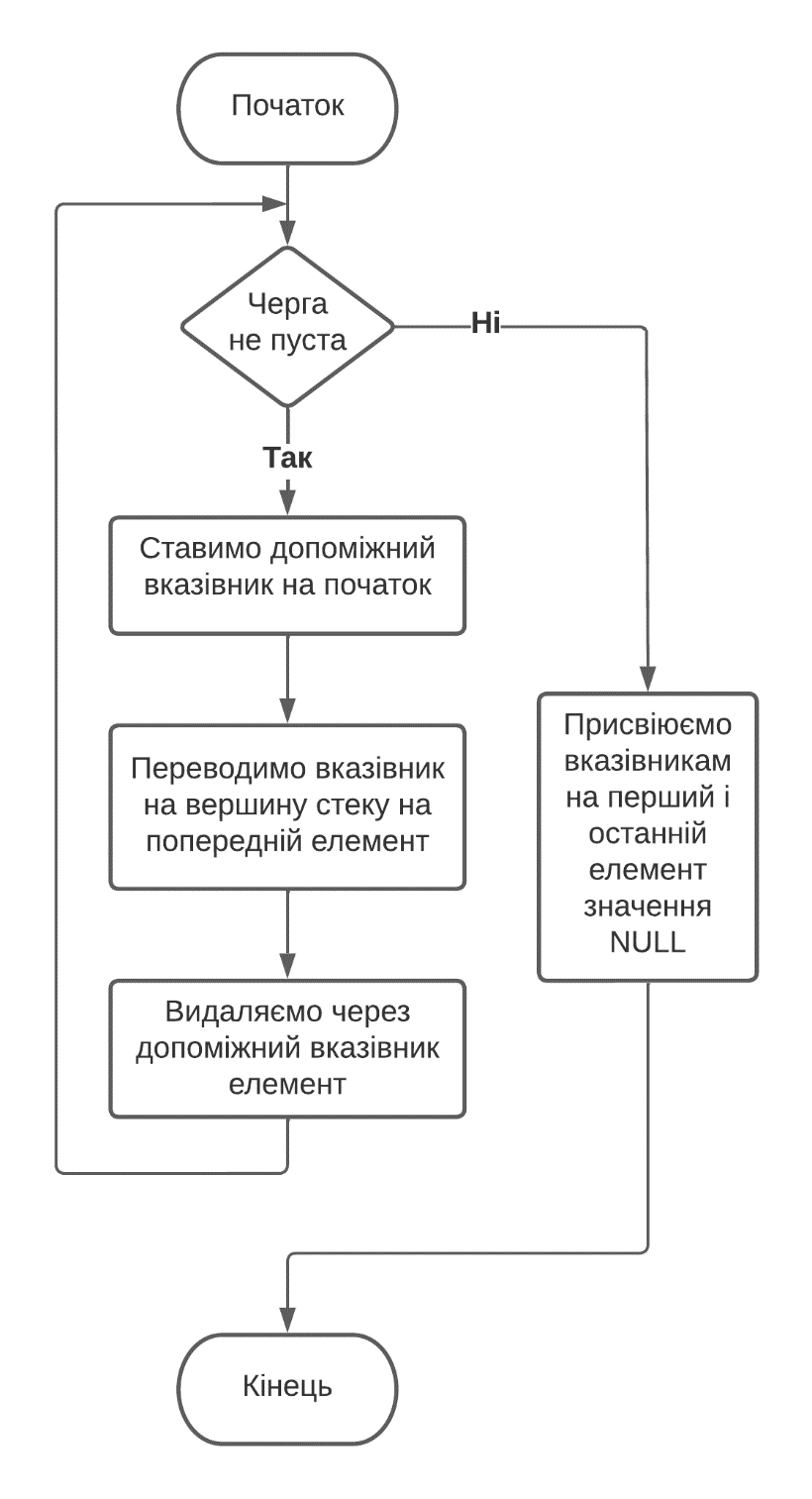
this->first = this->first->back;

this->first->next = NULL;

delete ptr\_del;

}

ФУНКЦІЯ ОЧИЩЕННЯ ЧЕРГИ



void Queue::Clear\_queue() {

Double\_List\* ptr\_del = NULL;

while (this->first != NULL) {

ptr\_del = this->first;

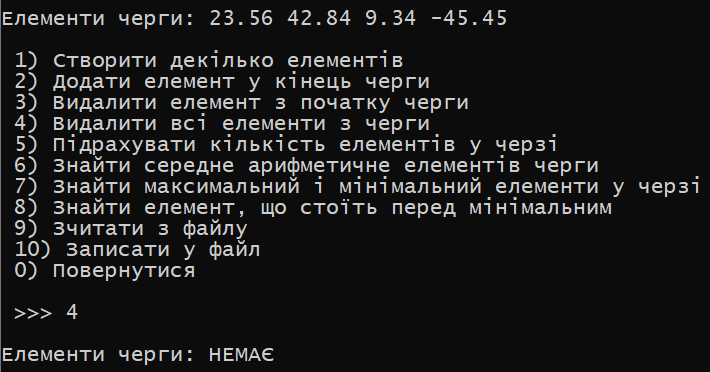
this->first = this->first->back;

delete ptr\_del;

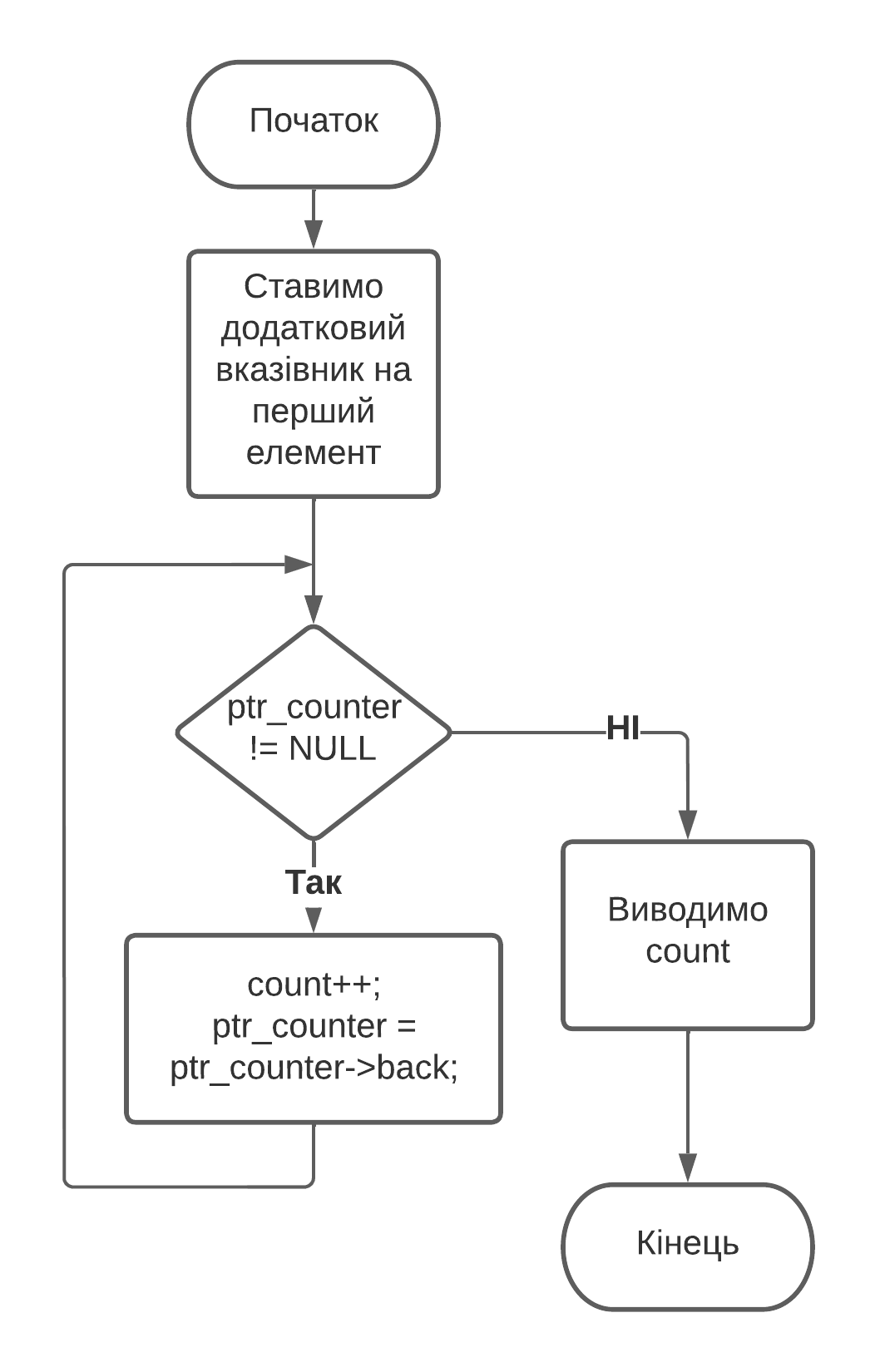
}

this->first = this->last = NULL;

}



ФУНКЦІЯ ПІДРАХУВАННЯ КІЛЬКОСТІ ЕЛЕМЕНТІВ У ЧЕРЗІ



void Queue::Count() {

int count = 0;

Double\_List\* ptr\_counter = this->first;

while (ptr\_counter != NULL) {

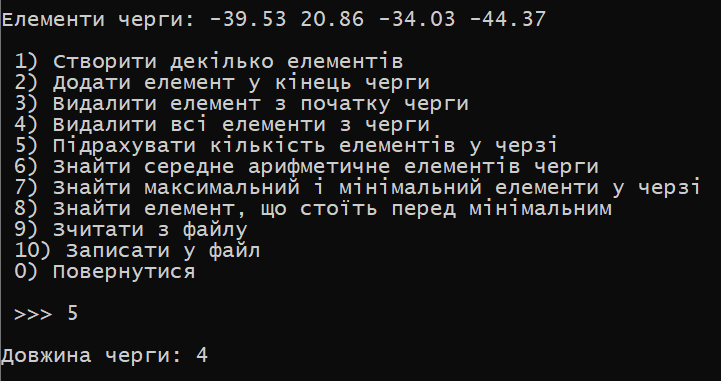
count++;

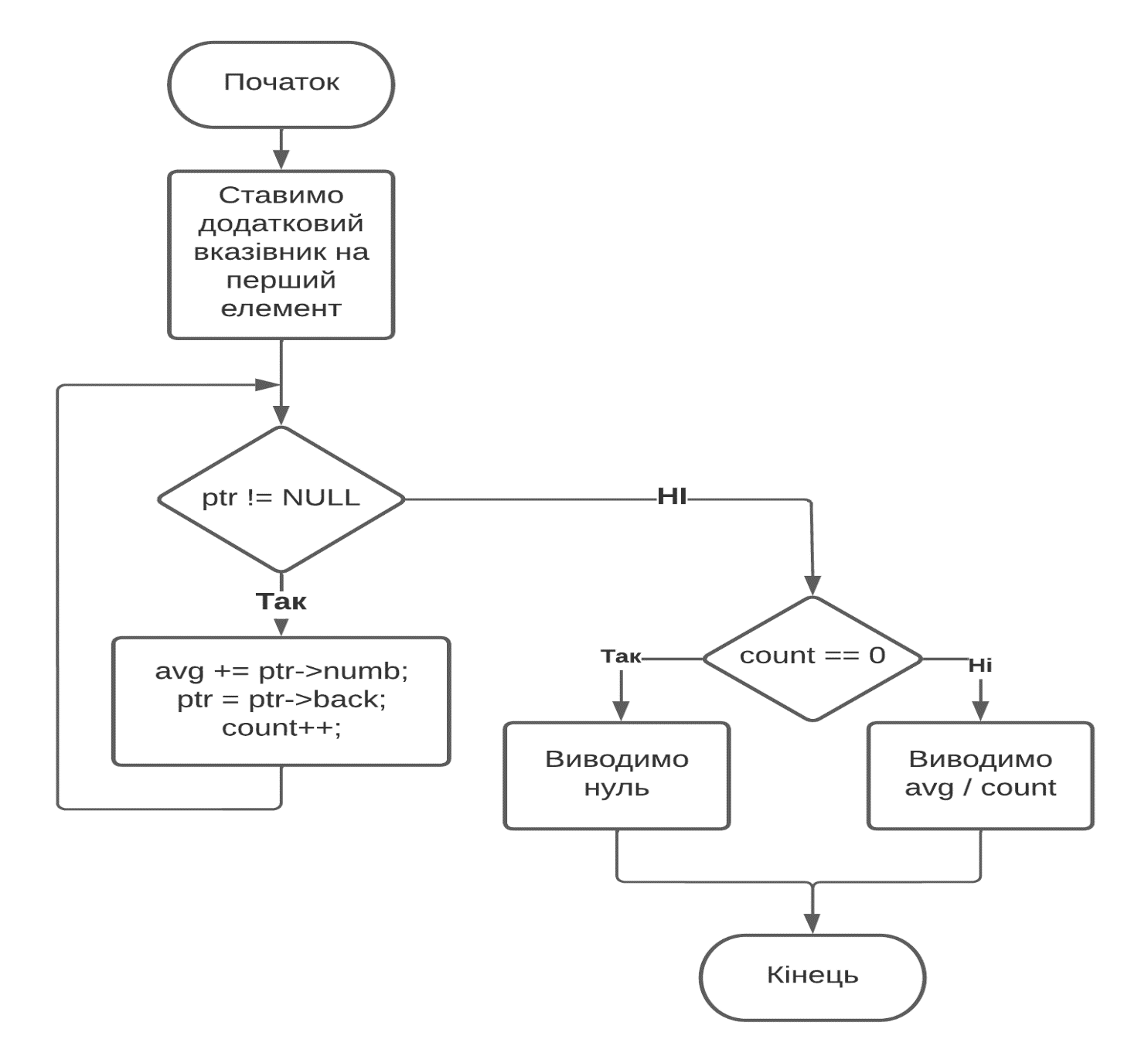
ptr\_counter = ptr\_counter->back;

}

cout << "\nДовжина черги: " << count << endl;

}



ФУНКЦІЯ ПІДРАХУВАННЯ СЕРЕДНЬОГО АРИФМЕТИЧНОГО ЗНАЧЕНЬ ЧЕРГИ

void Queue::Average\_arithmetic() {

double avg = 0, count = 0;

Double\_List\* ptr = this->first;

while (ptr != NULL) {

avg += ptr->numb;

count++;

ptr = ptr->back;

}

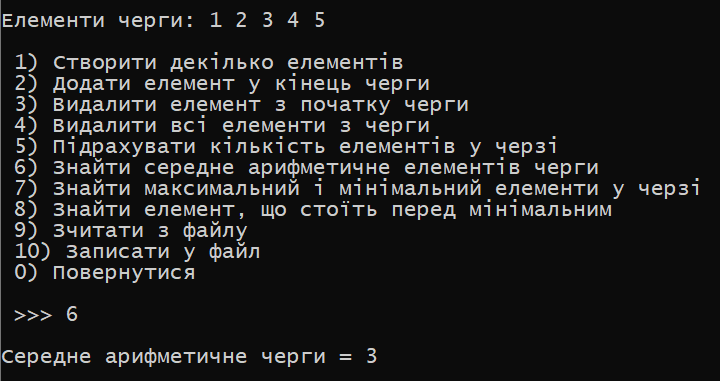
if (count == 0)

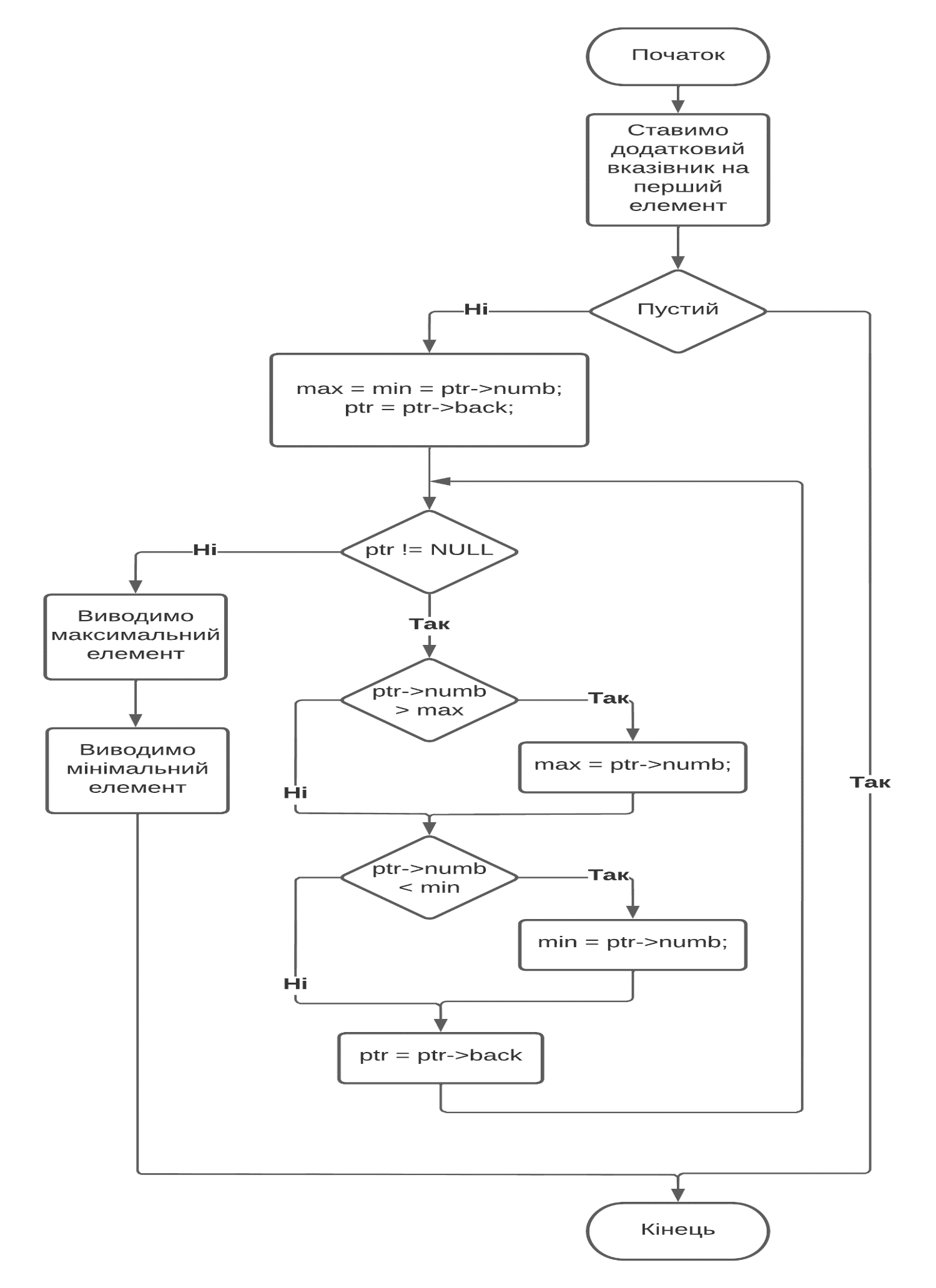
cout << "\nСередне арифметичне черги = 0\n\n";

else

cout << "\nСередне арифметичне черги = " << avg / count << "\n\n";

}



ФУНКЦІЯ ПОШУКУ МАКСИМАЛЬНОГО І МІНІМАЛЬНОГО ЕЛЕМЕНТІВ

void Queue::Search\_max\_and\_min() {

if (this->first == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

double max = 0, min = 0;

Double\_List\* ptr = this->first;

max = min = ptr->numb;

ptr = ptr->back;

while (ptr != NULL) {

if (ptr->numb > max)

max = ptr->numb;

if (ptr->numb < min)

min = ptr->numb;

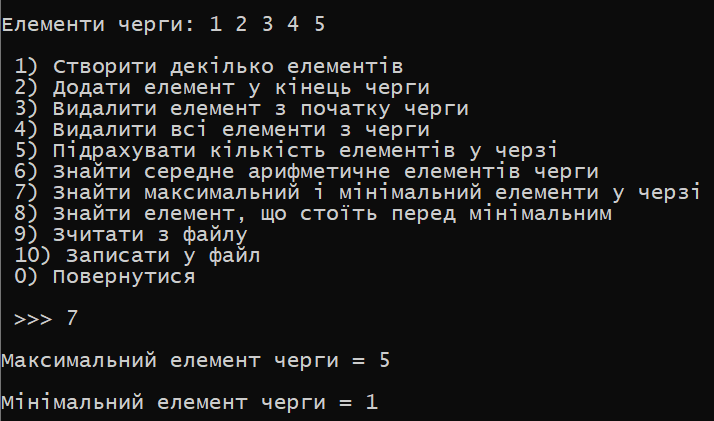
ptr = ptr->back;

}

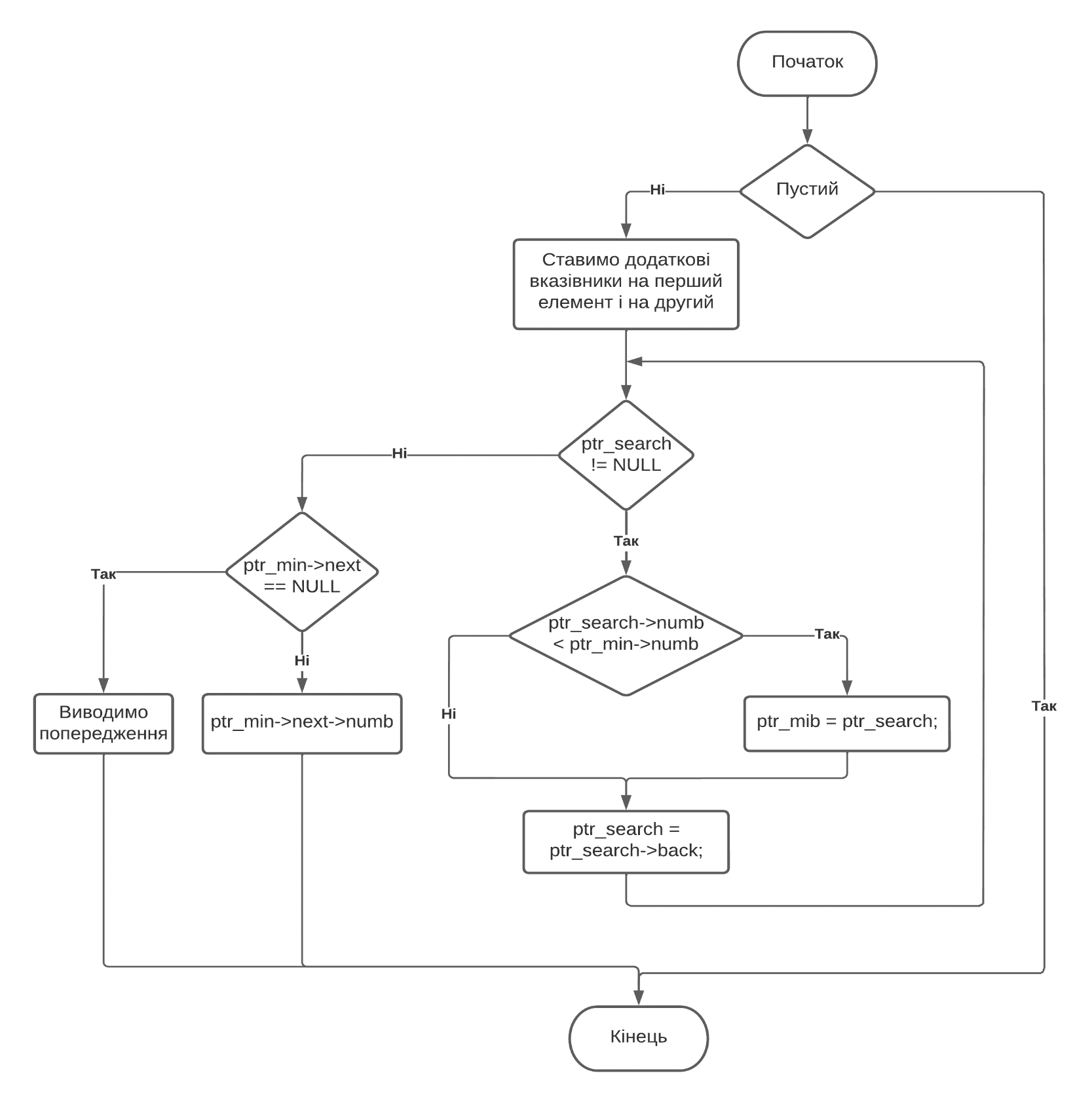
cout << "\nМаксимальний елемент черги = " << max;

cout << "\n\nМінімальний елемент черги = " << min << "\n\n";

}



ФУНКЦІЯ ПОШУКУ ЕЛЕМЕНТА ПЕРЕДУЮЧОГО МІНІМАЛЬНИЙ



void Queue::Search\_item\_after\_min\_item() {

if (this->first == NULL) {

cout << "\nНемає елементів!!!\n\n";

return;

}

Double\_List\* ptr\_search = NULL, \* ptr\_min = NULL;

ptr\_min = this->first;

ptr\_search = this->first->back;

while (ptr\_search != NULL) {

if (ptr\_search->numb < ptr\_min->numb)

ptr\_min = ptr\_search;

ptr\_search = ptr\_search->back;

}

if (ptr\_min->next == NULL) {

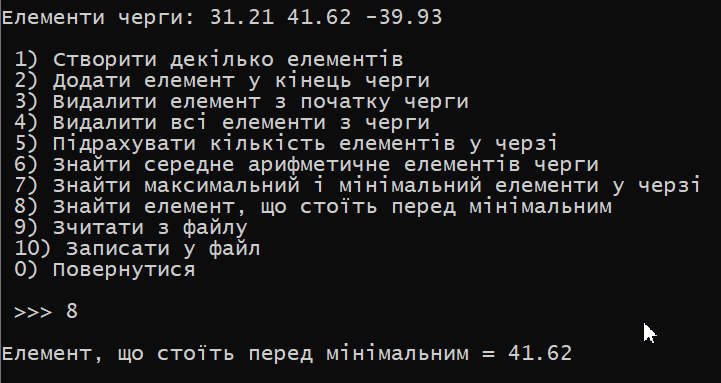
cout << "\nТакого елементу немає у черзі!!!\n\n";

return;

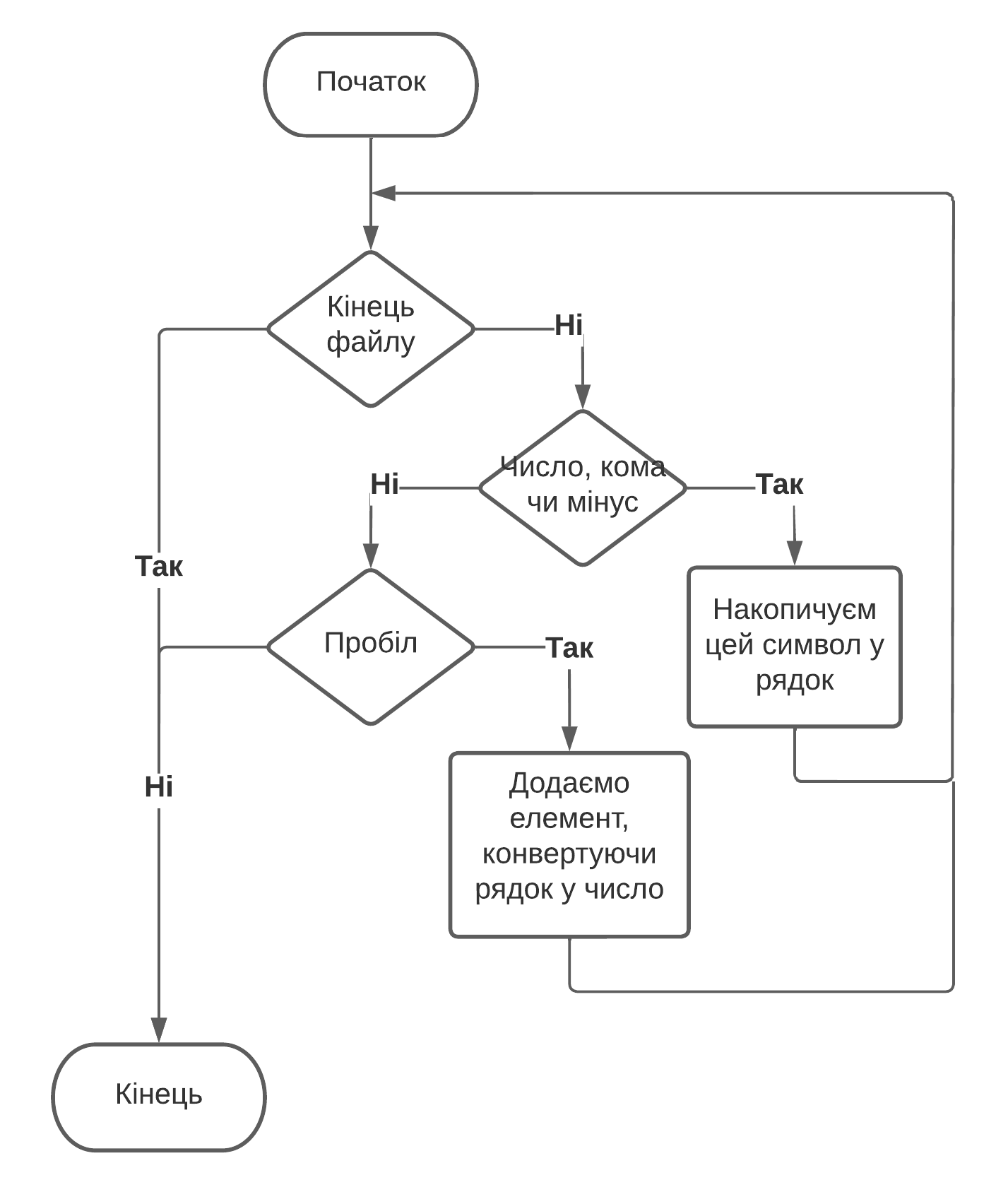
}

cout << "\nЕлемент, що стоїть перед мінімальним = " << ptr\_min->next->numb << "\n\n";

}



ФУНКЦІЯ ЗЧИТУВАННЯ З ФАЙЛУ



void Queue::Read\_from\_file(ifstream& f\_in) {

string str = "";

char ch;

while ((ch = f\_in.get()) != EOF) {

if ((ch > 47 && ch < 58) || ch == 46 || ch == 45)

str += (char)ch;

else if (ch == 32) {

this->Push\_back(atof(str.c\_str()));

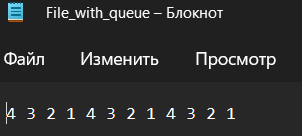
str = "";

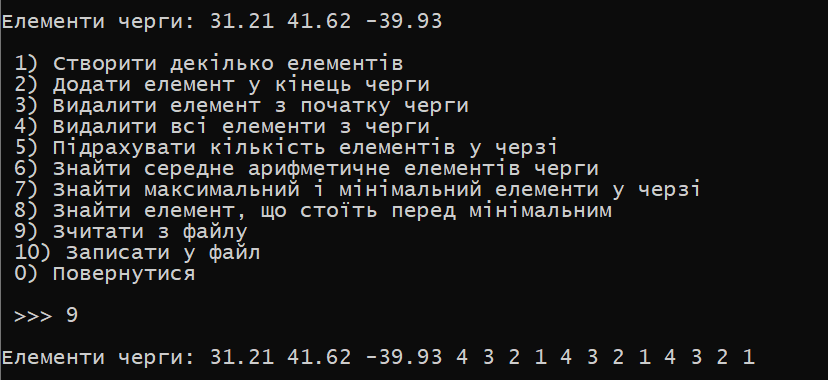
}

else break;

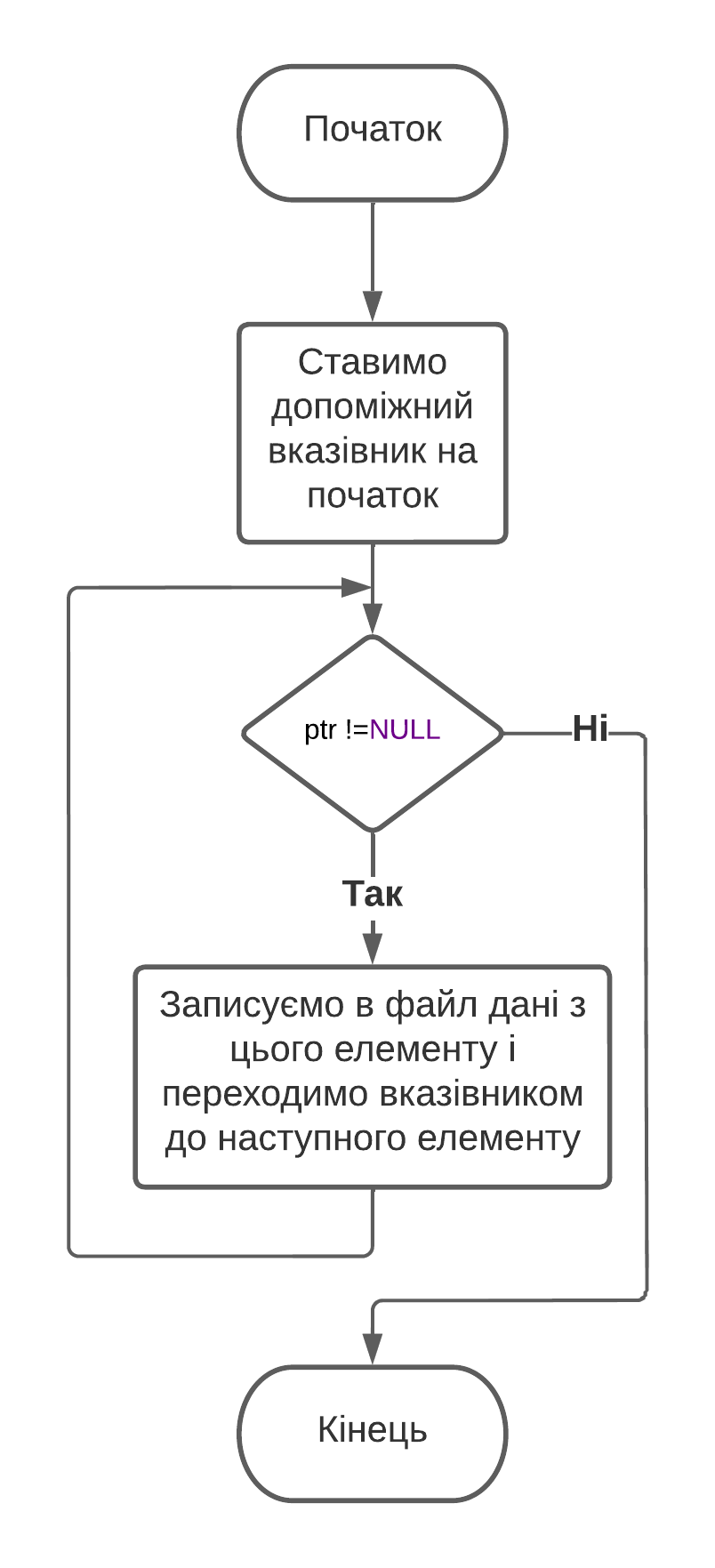
}

}





ФУНКЦІЯ ЗАПИСУВАННЯ У ФАЙЛ



void Queue::Write\_to\_file(ofstream& f\_out) {

Double\_List\* ptr = this->first;

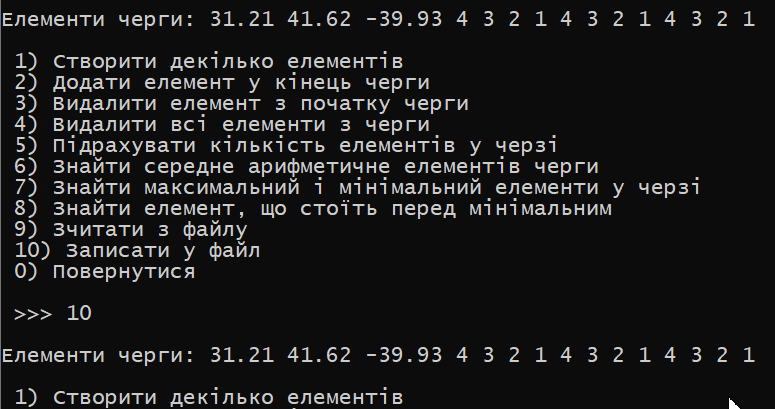
while (ptr != NULL) {

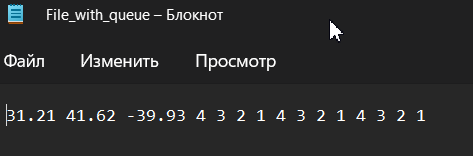
f\_out << ptr->numb << " ";

ptr = ptr->back;

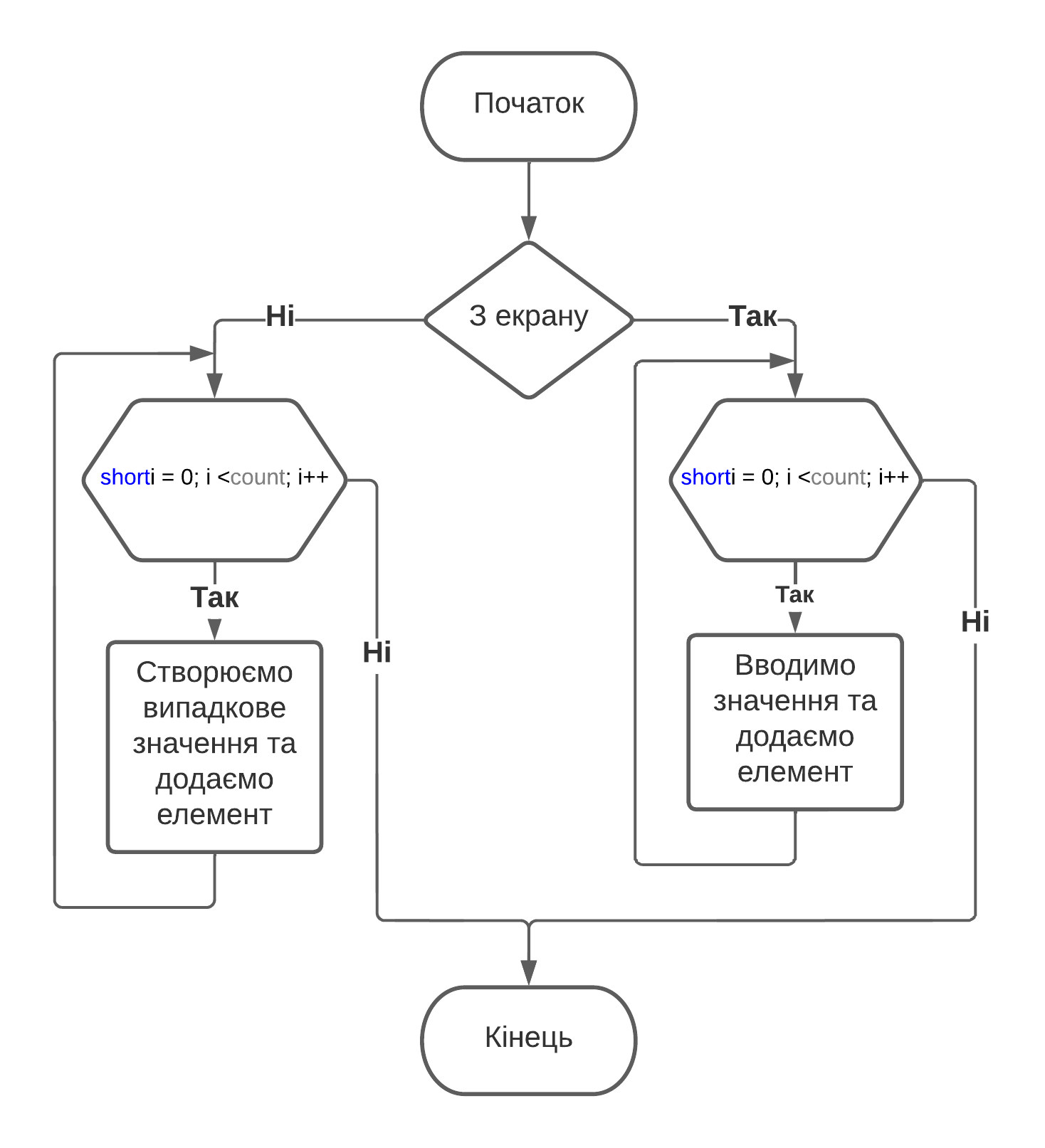
}

}





ДЕКА

ФУНКЦІЯ СТВОРЕННЯ ДЕКИ

void Deque::Create\_deque(int count, bool flag) {

double temp;

if (flag == true) {

for (short i = 0; i < count; i++) {

cin >> temp;

this->Push\_back(temp);

}

}

else {

for (short i = 0; i < count; i++) {

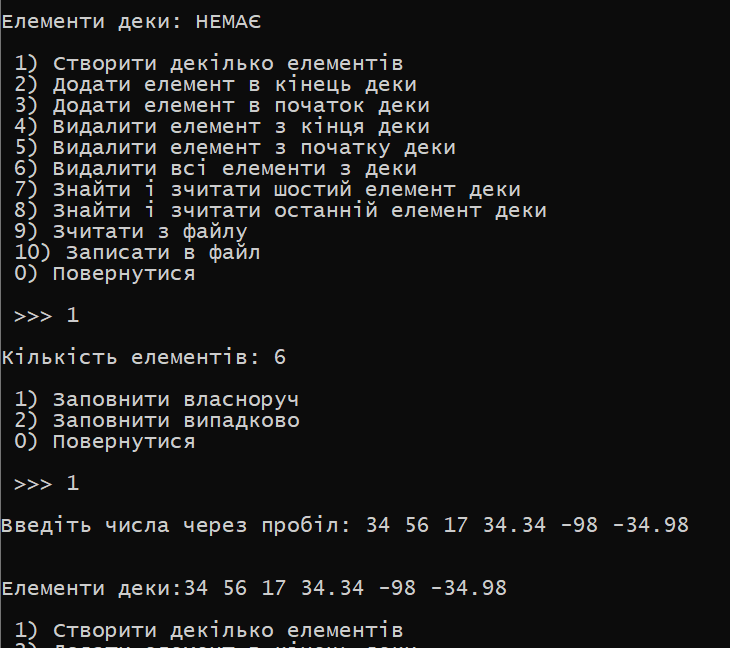
temp = rand() % 101 - 50 + rand() % 101 \* 0.01;

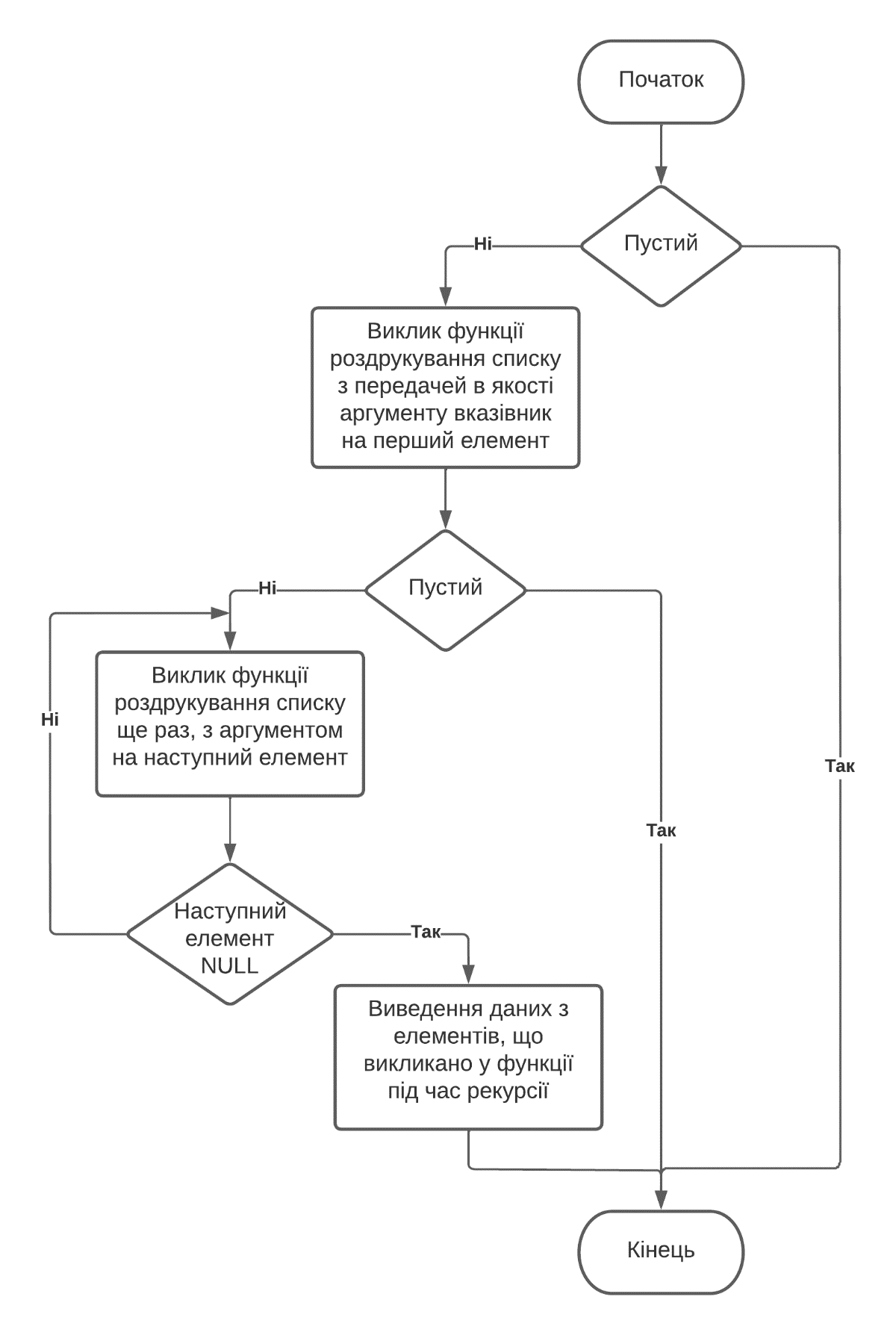
this->Push\_back(temp);

}

}

}



ФУНКЦІЯ ВИВЕДЕННЯ ДЕКИ

void Deque::Print\_deque() {

cout << "Елементи деки:";

if (this->first == NULL) {

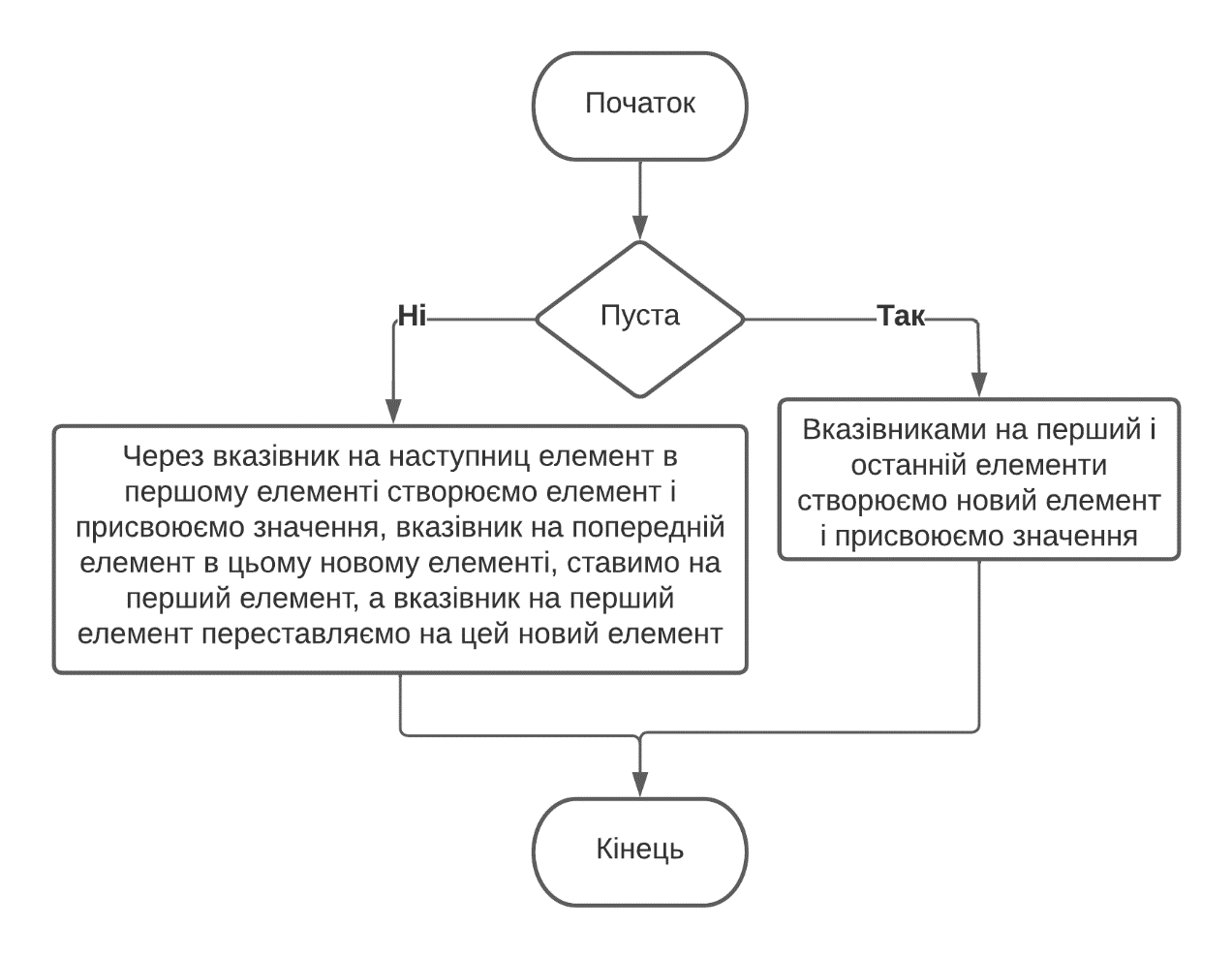
cout << " НЕМАЄ";

return;

}

Print\_double\_list(this->last);

}

ФУНКЦІЯ ДОДАВАННЯ ЕЛЕМЕНТУ В ПОЧАТОК ДЕКИ

void Deque::Push\_front(double value) {

if (this->first == NULL) {

this->first = this->last = new Double\_List(value);

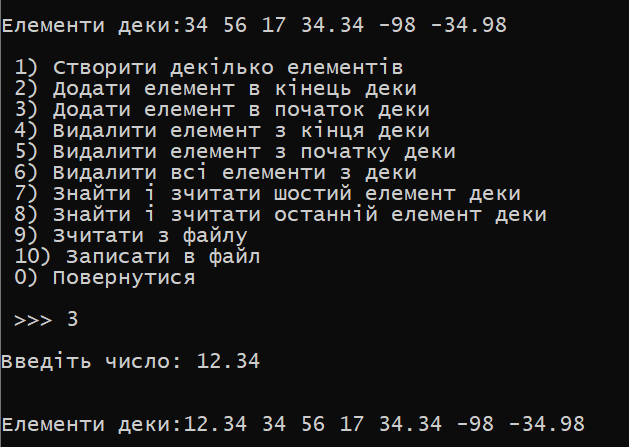
return;

}

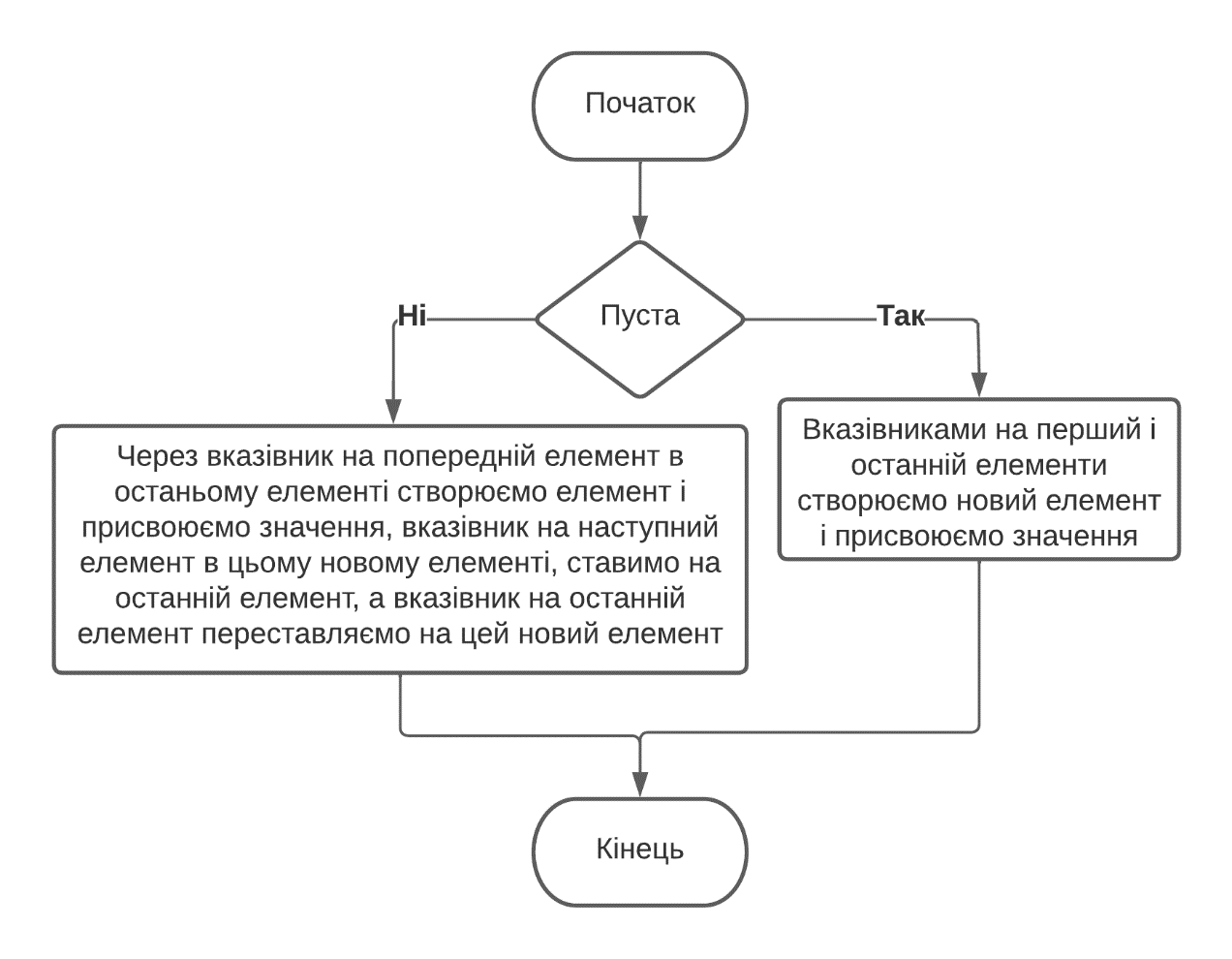
this->first->next = new Double\_List(value);

this->first->next->back = this->first;

this->first = this->first->next;

}

ФУНКЦІЯ ДОДАВАННЯ ЕЛЕМЕНТУ В КІНЕЦЬ ДЕКИ



void Deque::Push\_back(double value) {

if (this->first == NULL) {

this->first = this->last = new Double\_List(value);

return;

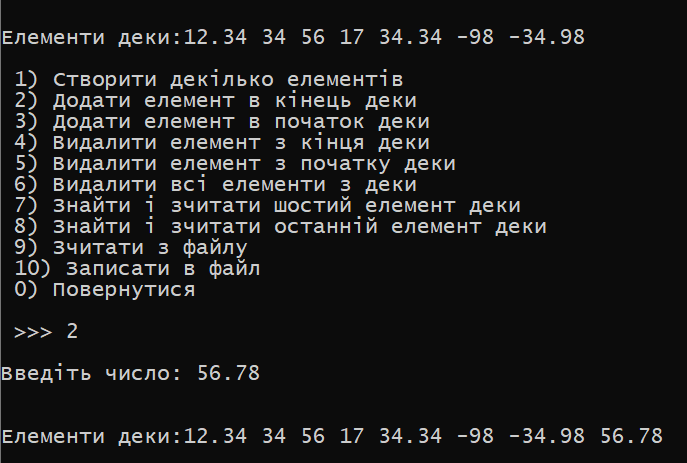
}

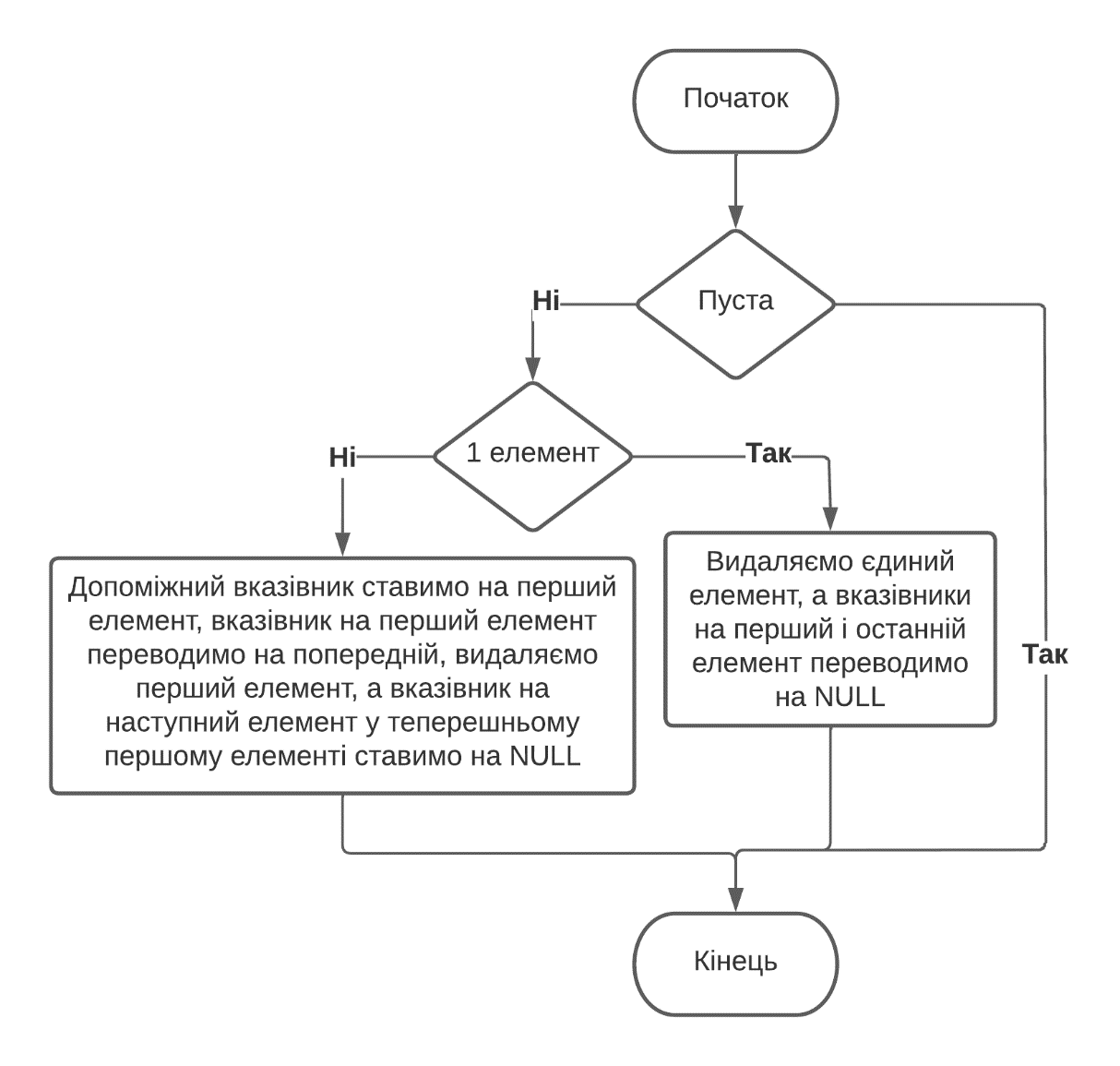
this->last->back = new Double\_List(value);

this->last->back->next = this->last;

this->last = this->last->back;

}



ФУНКЦІЯ ВИДАЛЕННЯ ЕЛЕМЕНТУ З ПОЧАТКУ ДЕКИ

void Deque::Pop\_front() {

if (this->first == NULL)

return;

Double\_List\* ptr\_del = this->first;

if (this->first->back == NULL) {

delete this->first;

this->first = this->last = NULL;

return;

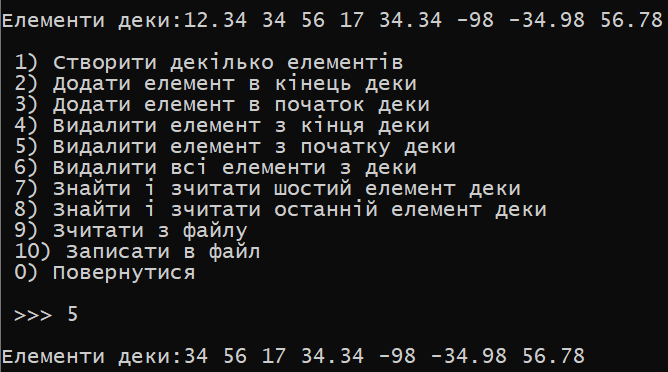
}

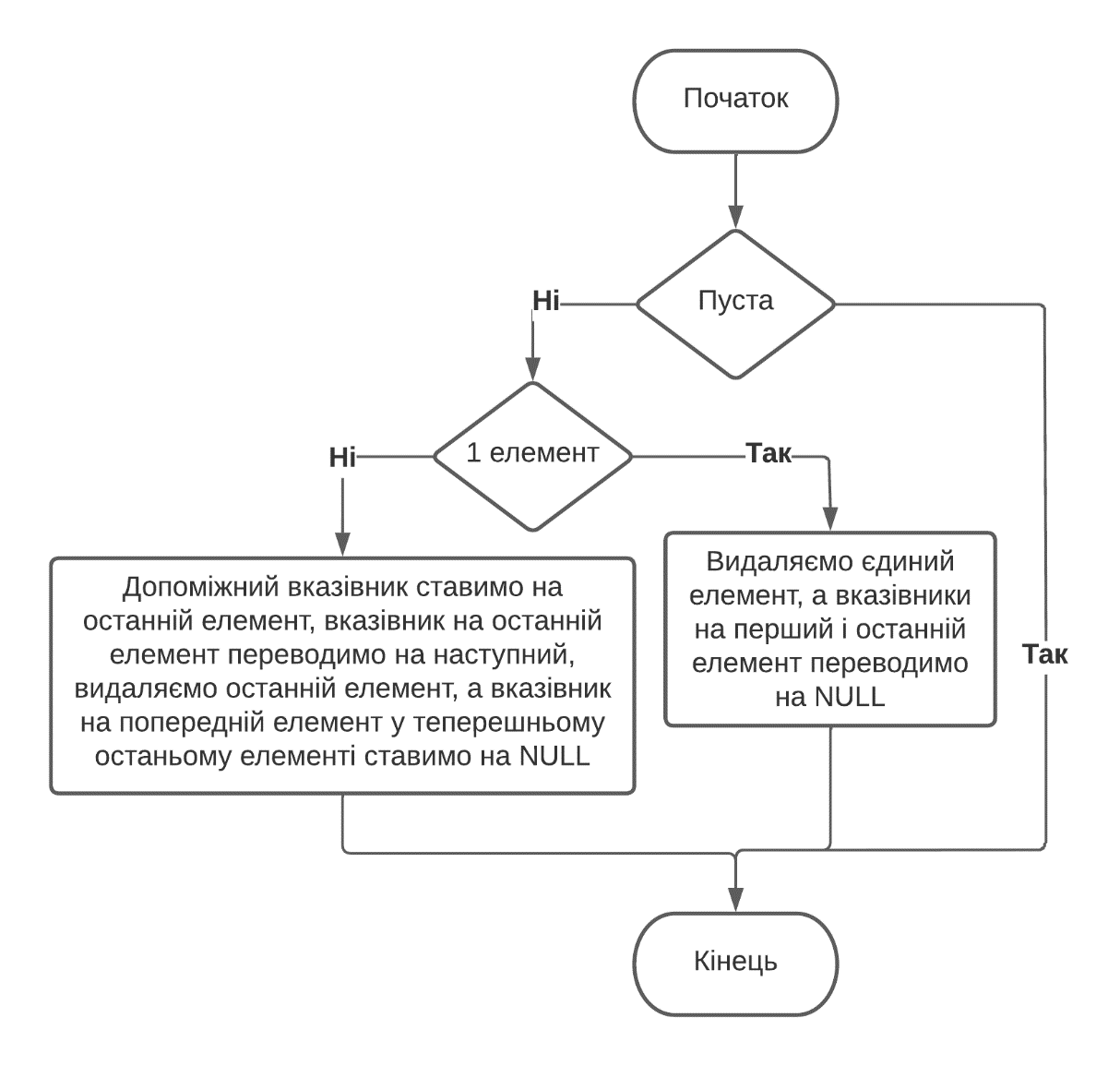
this->first = this->first->back;

this->first->next = NULL;

delete ptr\_del;

}



ФУНКЦІЯ ВИДАЛЕННЯ ЕЛЕМЕНТУ З КІНЦЯ ДЕКИ

void Deque::Pop\_back() {

if (this->first == NULL)

return;

Double\_List\* ptr\_del = this->last;

if (this->last->next == NULL) {

delete this->last;

this->first = this->last = NULL;

return;

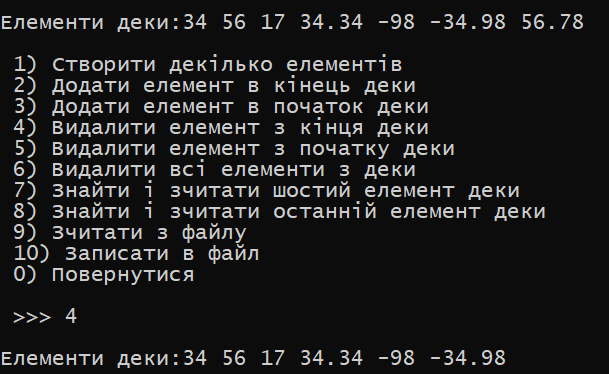
}

this->last = this->last->next;

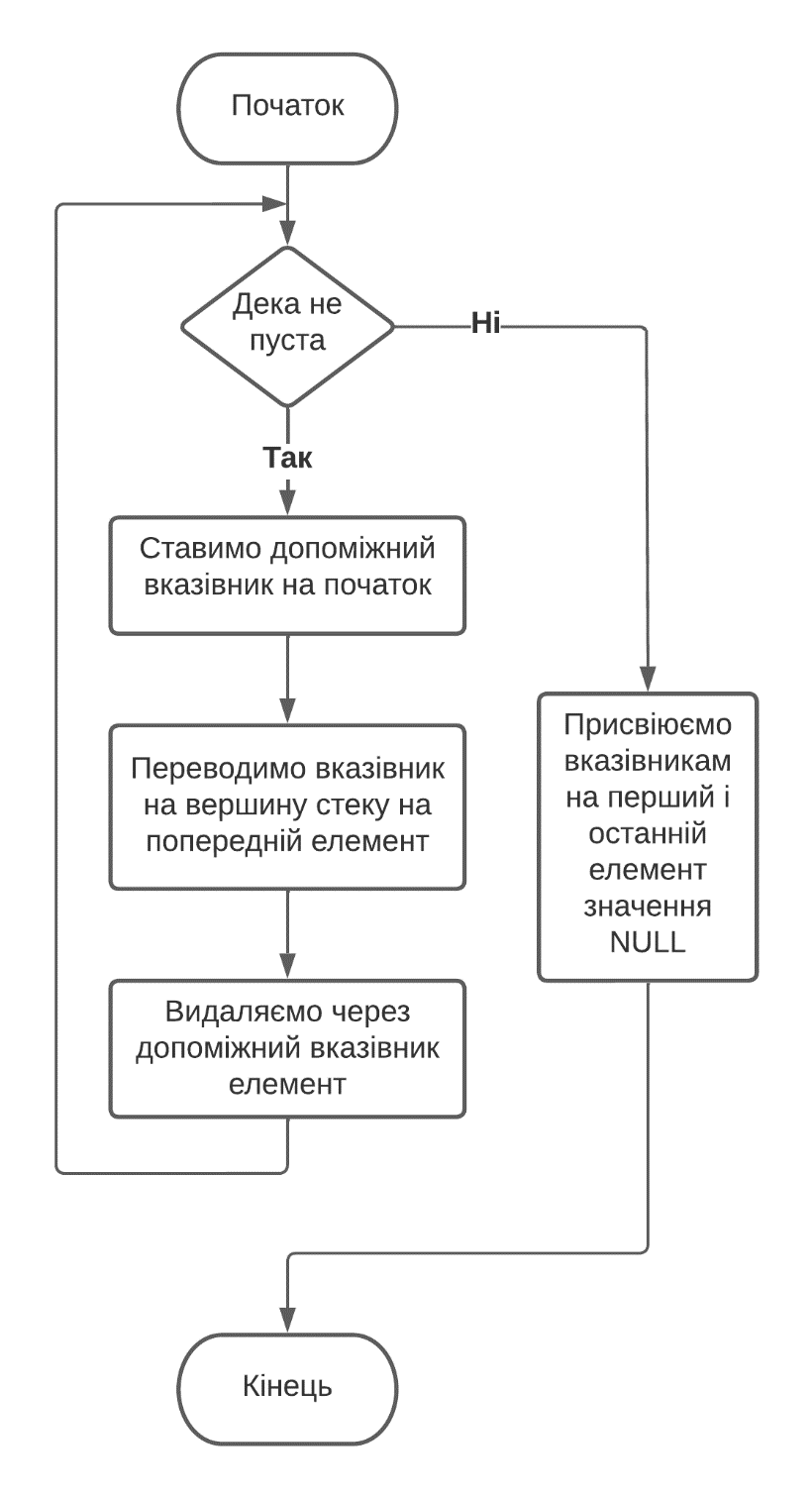
this->last->back = NULL;

delete ptr\_del;

}



ФУНКЦІЯ ОЧИЩЕННЯ ДЕКИ



void Deque::Clear\_deque() {

Double\_List\* ptr\_del = NULL;

while (this->first != NULL) {

ptr\_del = this->first;

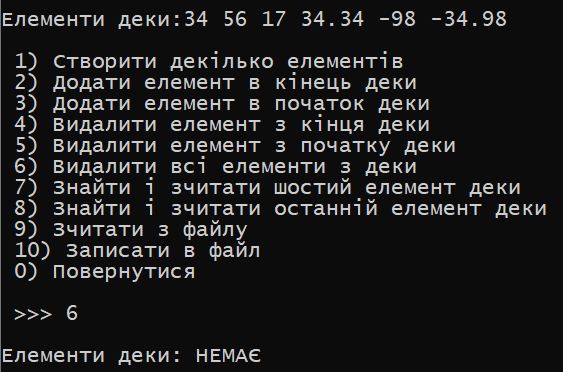
this->first = this->first->back;

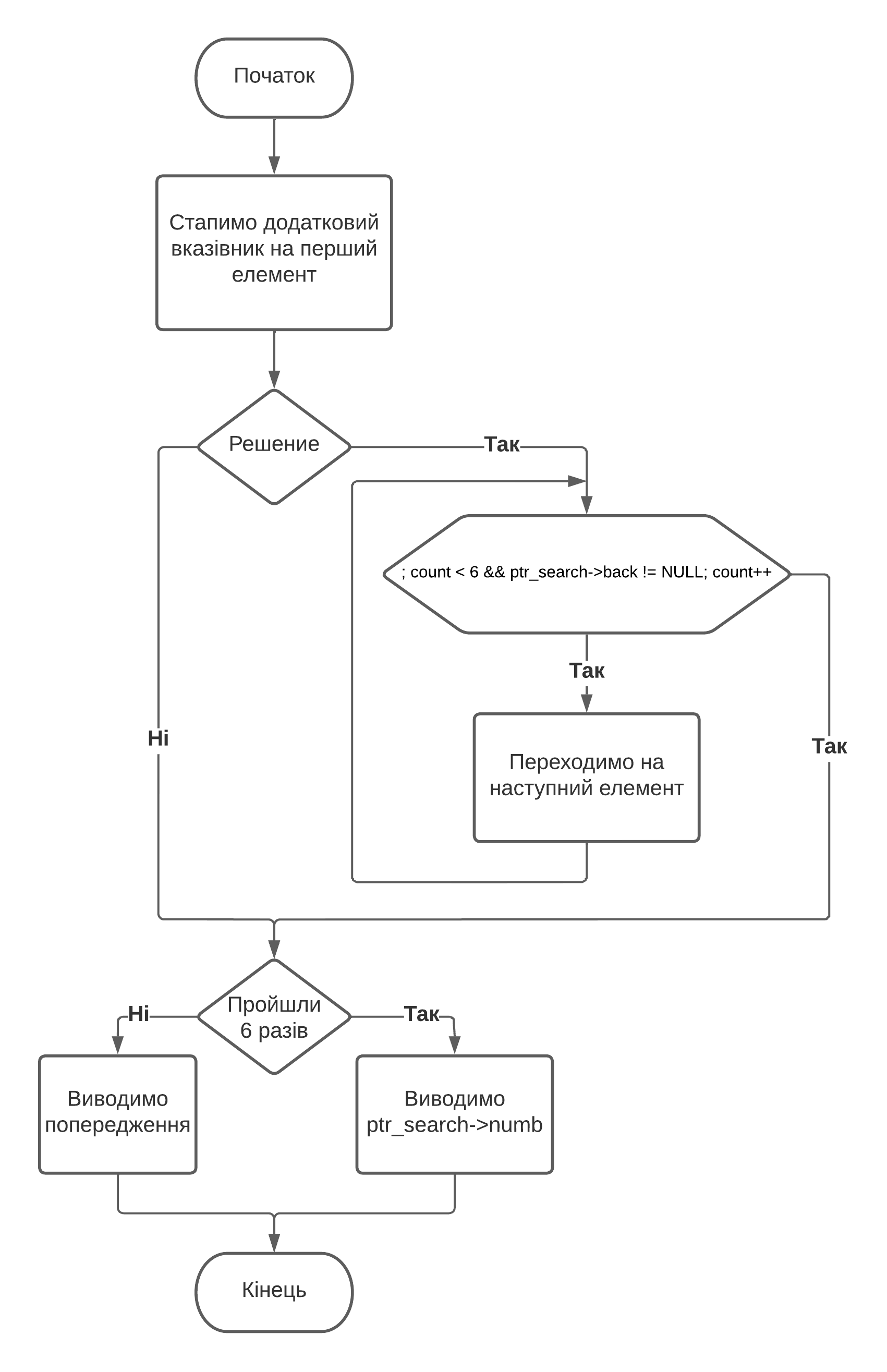
delete ptr\_del;

}

this->first = this->last = NULL;

}



ФУНКЦІЯ ЗЧИТУВАННЯ ШОСТОГО ЕЛЕМЕНТУ

void Deque::Read\_sixth\_item() {

Double\_List\* ptr\_search = this->first;

short count = 1;

if (this->first != NULL)

for (; count < 6 && ptr\_search->back != NULL; count++)

ptr\_search = ptr\_search->back;

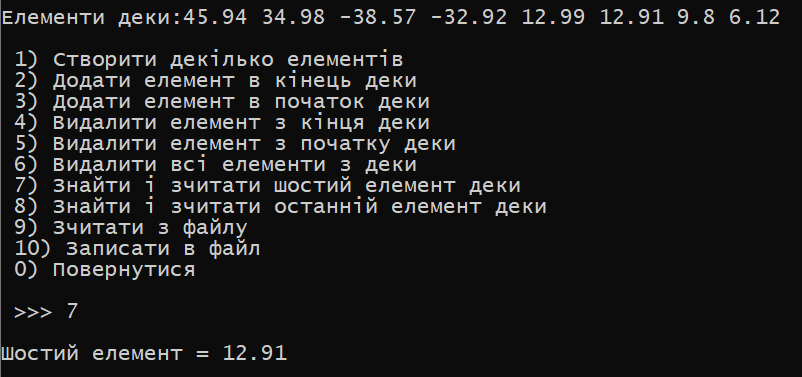
if (count == 6)

cout << "\nШостий елемент = " << ptr\_search->numb << "\n\n";

else

cout << "\nШостого елементу немає у деці!!!\n\n";

}



ФУНКЦІЯ ЗЧИТУВАННЯ ОСТАНЬОГО ЕЛЕМЕНТУ

void Deque::Read\_last\_item() {

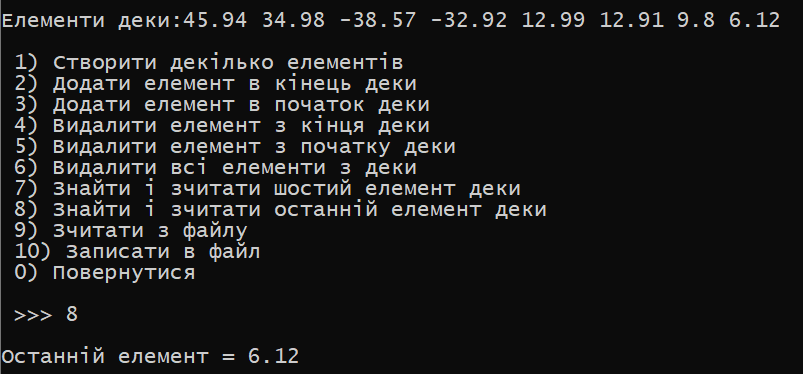
if (this->last == NULL)

cout << "\nНемає елементів!!!\n\n";

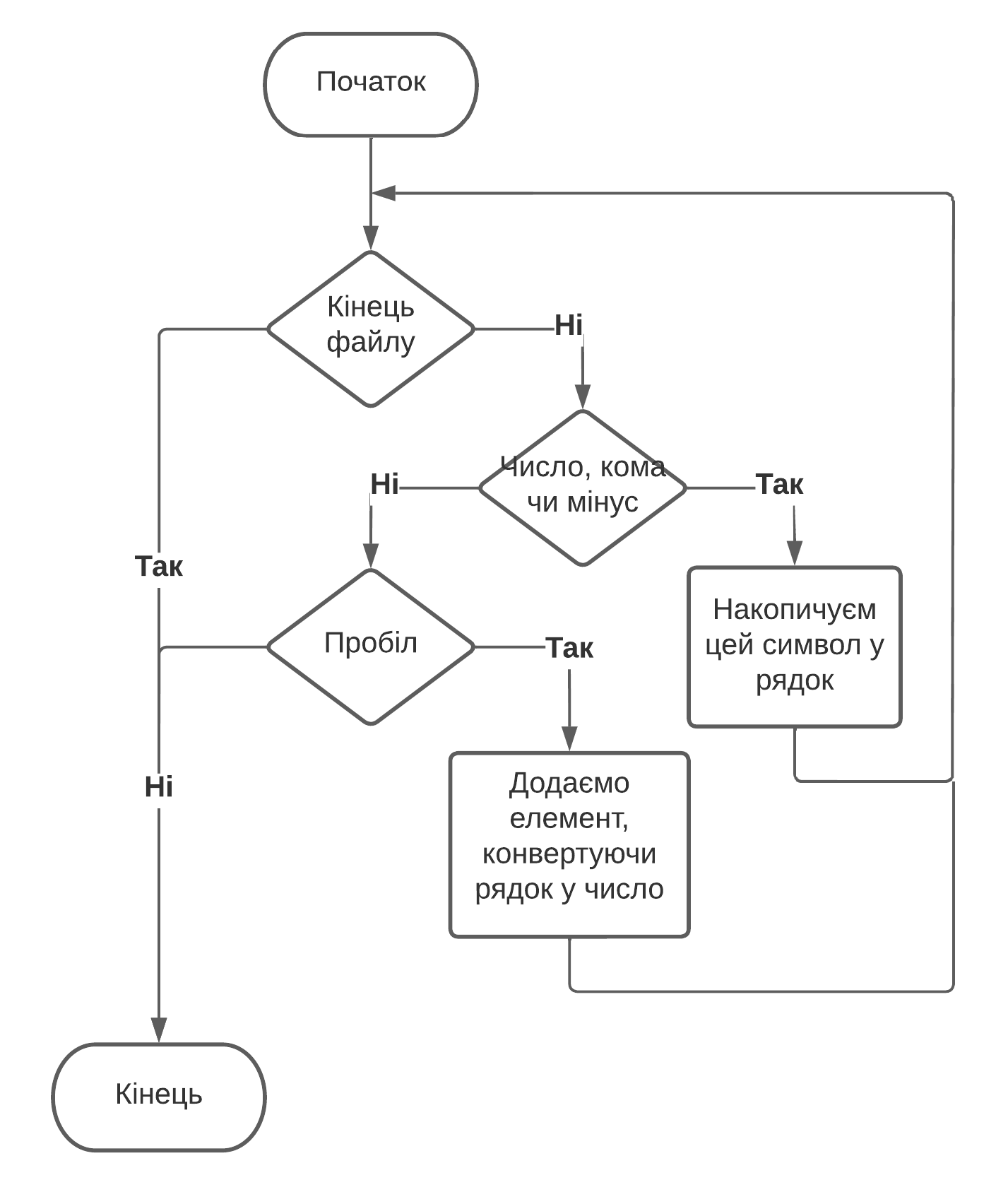
else

cout << "\nОстанній елемент = " << this->last->numb << "\n\n";

}



ФУНКЦІЯ ЗЧИТУВАННЯ З ФАЙЛУ



void Deque::Read\_from\_file(ifstream& f\_in) {

string str = "";

char ch;

while ((ch = f\_in.get()) != EOF) {

if ((ch > 47 && ch < 58) || ch == 46 || ch == 45)

str += (char)ch;

else if (ch == 32) {

this->Push\_back(atof(str.c\_str()));

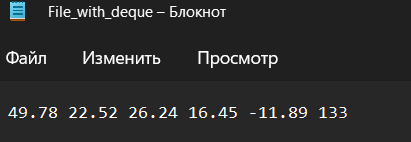
str = "";

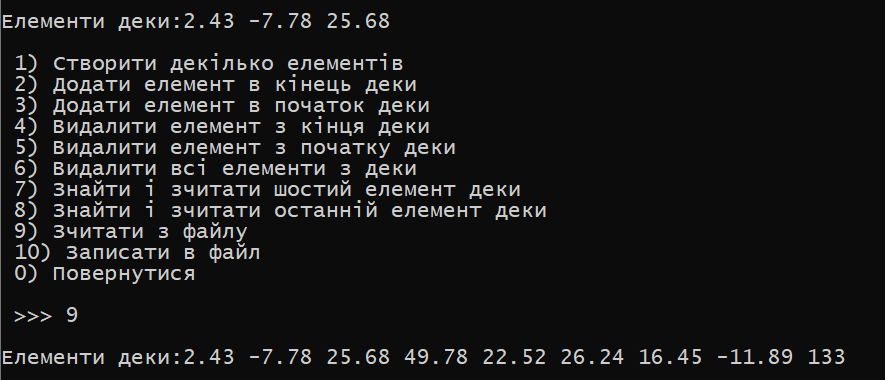
}

else break;

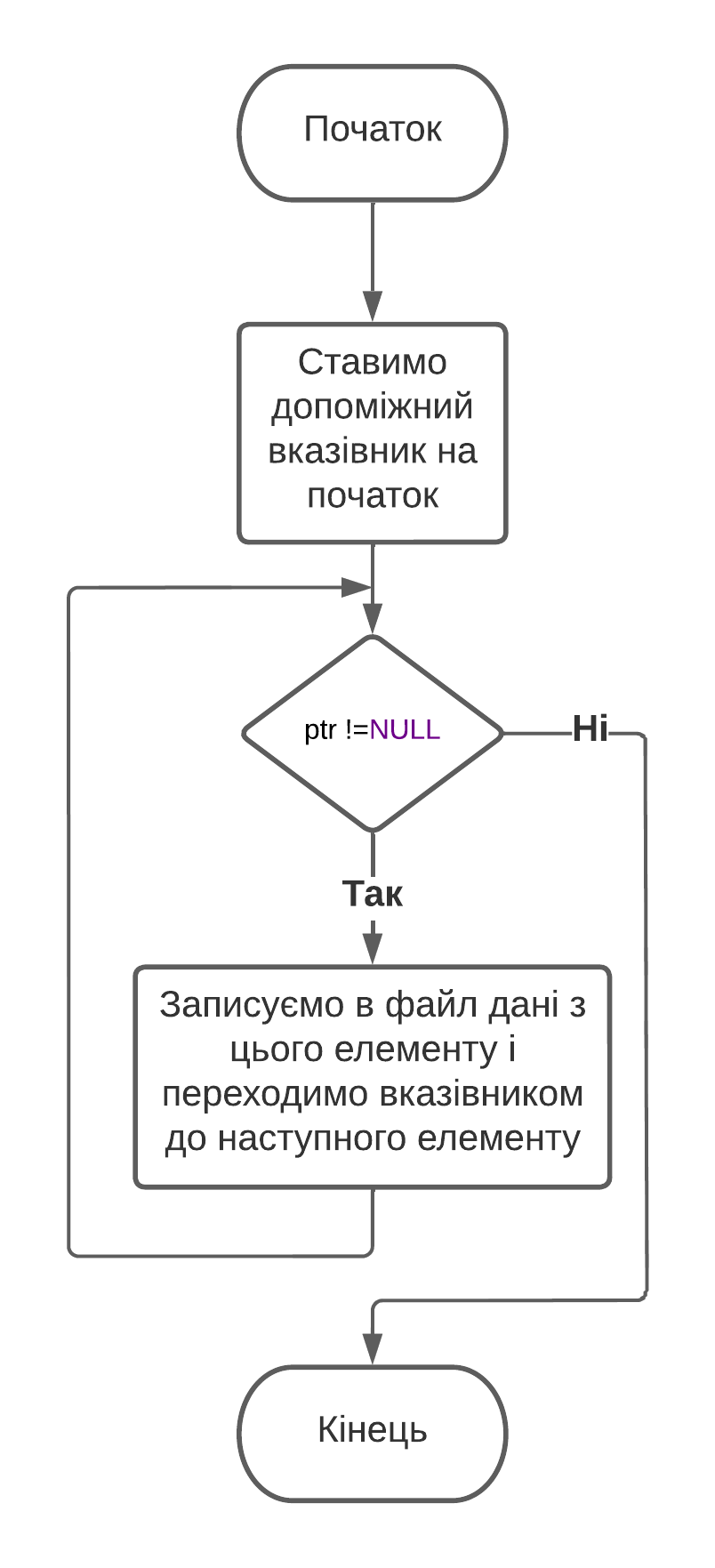
}

}





ФУНКЦІЯ ЗАПИСУВАННЯ У ФАЙЛ



void Deque::Write\_to\_file(ofstream& f\_out) {

Double\_List\* ptr = this->first;

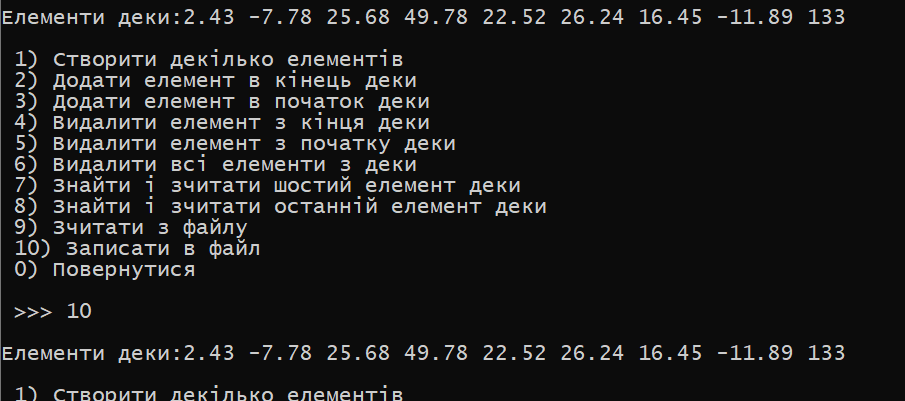
while (ptr != NULL) {

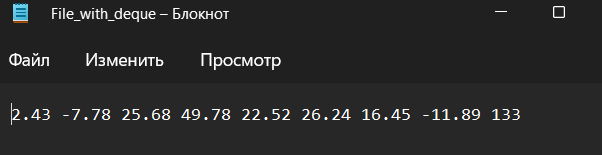
f\_out << ptr->numb << " ";

ptr = ptr->back;

}

}





ВИСНОВКИ

Отже, під час даної лабораторної роботи було проаналізовано та розроблено функції для виконання різноманітних операцій із такими структурами даних, як стек, черга і лека.

Говорячи про плюси і мінуси цих структур даних, можна зауважити наступне: стек, має доволі просту структуру, яка базується на однозв’язному списку, плюсами цієї структури даних, є її особливість організування, яка має дуже велике значення у різноманітних задачах, по типу опрацювання з тим, що береться зараз і витягається поступово, наприклад, стек у процесорі, а з мінусів можна зазначити довге знаходження, складність з роботою елементів, що лежать на дні; черга, має банальне організування, проте, з плюсів її організованість, має значення для розв’язання задач, пов’язаних із викликами і виконання поступово функцій, операцій, тощо, а з мінусів, можна зазначити повільність у пошуку, та трохи обмеженість порівняно з декою; дека, має як і черга таку саму структура, але має ще додаткову можливість додавати елементи в початок та видаляти їх з кінця, що покращує її працездатність, а з мінусів, аналогічно, повільний пошук.