МІНІСТРЕСТВО ОСВІТИ Й НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“Харківський Політехнічний Інститут”

Кафедра управління проєктами в інформаційних технологіях

Звіт з лабораторної роботи №2

“Стеки, черги і декі та робота з ними”

з дисципліни

“Алгоритми та структури даних”

Варіант №5

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

Виконав: ст. гр. КН-1223г Шинкаренко О.В.

Харків – 2024

**Мета:** ознайомитися із основними способами організації стеків, черг, деків та особливостями їх програмної реалізації. Набути практичних навичок роботи зі стеками, чергами та деками.

**Завдання:**

1. Написати програму, яка демонструє основні дії по роботі із стеком пропонує користувачу вибір дії:

* поміняти місцями перший і останній елементи стека;
* розгорнути стек, тобто зробити "дно" стека вершиною, а вершину - "дном";
* видалити кожен другий елемент стека;
* знайти максимальний елемент і вставити після нього 0;
* видалити мінімальний елемент;
* видалити всі елементи, крім першого;
* видалити всі елементи, крім останнього.

1. Написати програму, яка демонструє основні дії по роботі із чергою пропонує користувачу вибір дії:

* знайти та вивести кількість елементів черги;
* знайти та вивести середнє арифметичне збережених елементів;
* знайти та вивести мінімальний та максимальний елемент;
* знайти та вивести елемент, що йде перед мінімальним елементом.

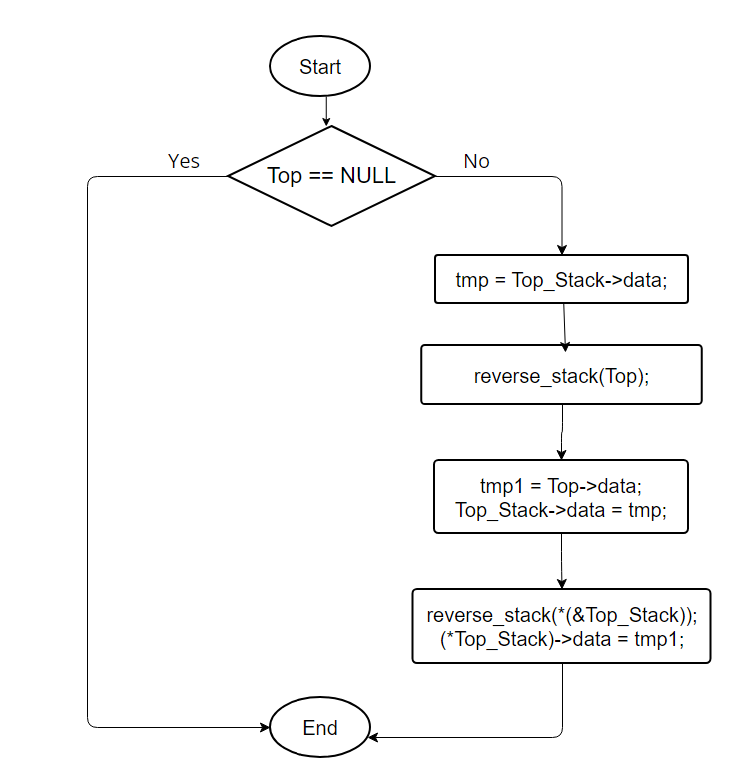
1. Написати програму, яка демонструє основні дії по роботі із деком пропонує користувачу вибір дії:

створити дек;

* перевірити на порожнечу;
* додати елемент в початок;
* додати елемент в кінець;
* прочитати останній елемент;
* перевірити на порожнечу.

Стек – це структура даних, що працює за принципом останній війшов перший вийшов, тобто останній елемент, що був доданий, вилучається першим. Основні операції в стеку – це додавання і видалення елементів. Стек часто використовується для зберігання станів або викликів функцій у програмуванні.

Поміняти місцями перший і останній елементи стека



Блок-схема 1 Поміняти місцями перший і останній елементи стека

Вхідні дані: посилання на стек

Код функції:

void swap\_elements(stack\*\* Top\_Stack) {

if ((\*Top\_Stack) == NULL) return;

short tmp = (\*Top\_Stack)->data;

short tmp1;

reverse\_stack(\*(&Top\_Stack));

tmp1 = (\*Top\_Stack)->data;

(\*Top\_Stack)->data = tmp;

reverse\_stack(\*(&Top\_Stack));

(\*Top\_Stack)->data = tmp1;

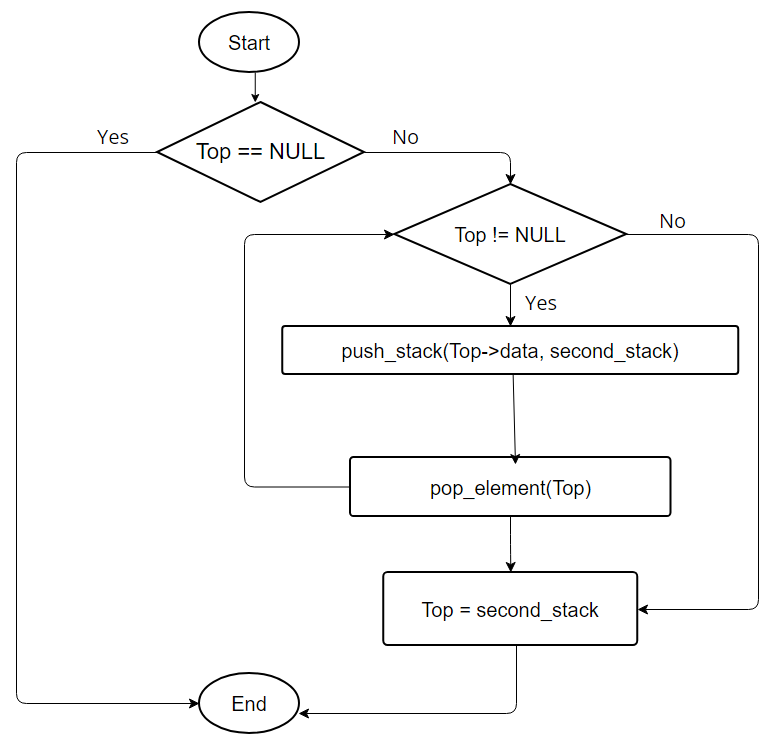
}

Результат у консолі:



Виведення у консоль 1 Поміняти місцями перший і останній елементи стекау

Розгортання стеку



Блок-схема 2 Розгортання стеку

Вхідні дані: посилання на стек

Код функції:

void reverse\_stack(stack\*\* top\_stack) {

if ((\*top\_stack) == NULL) return;

stack\* second\_stack = NULL;

while ((\*top\_stack) != NULL) {

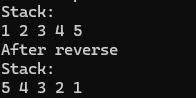
push\_stack((\*top\_stack)->data, &second\_stack);

pop\_element(\*(&top\_stack));

}

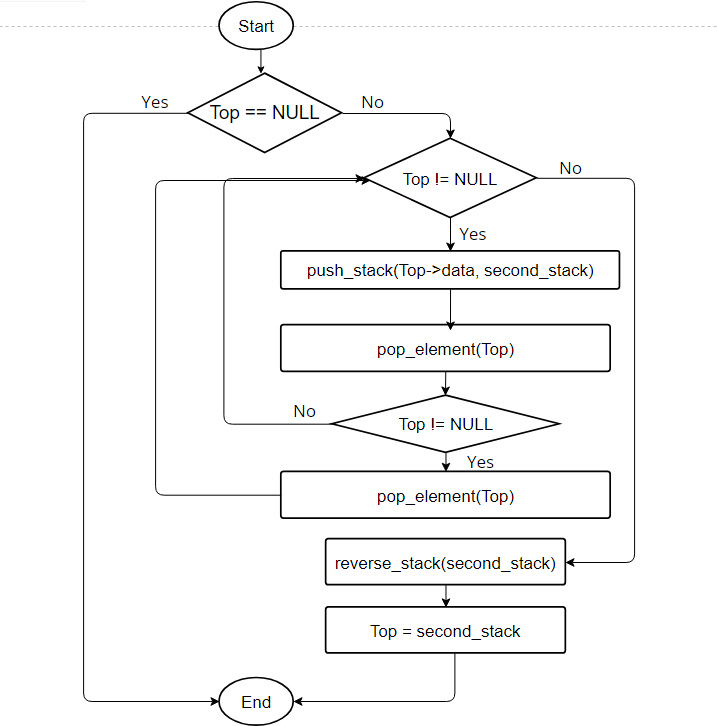
\*top\_stack = second\_stack;

}



Виведення у консоль 2 Розгортання стеку

Видалення коженого другого елементу стеку



Блок-схема 3 Видалення коженого другого елементу стеку

Вхідні дані: посилання на стек

Код функції:

void delete\_elements(stack\*\* top\_stack) {

if ((\*top\_stack) == NULL) return;

stack\* second\_stack = NULL;

while ((\*top\_stack) != NULL) {

push\_stack((\*top\_stack)->data, &second\_stack);

pop\_element(\*(&top\_stack));

if ((\*top\_stack) != NULL) pop\_element(\*(&top\_stack));

}

reverse\_stack(&second\_stack);

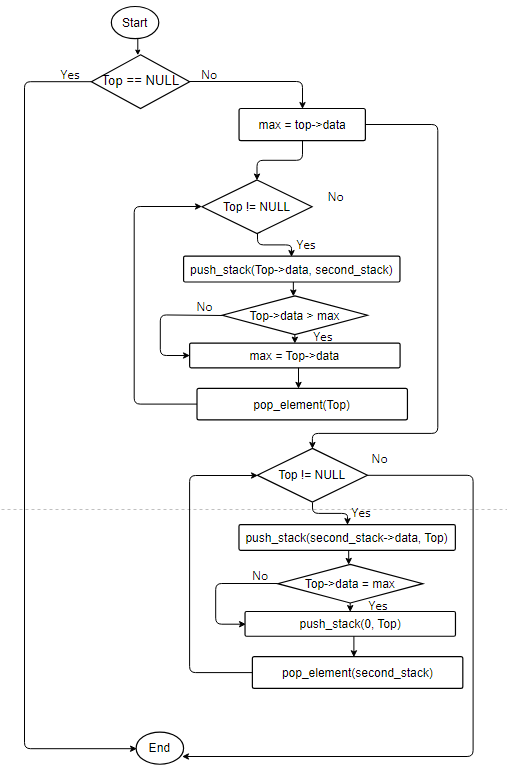
\*top\_stack = second\_stack;

}



Виведення у консоль 3 Видалення коженого другого елементу стеку

Знайти максимальний елемент і вставити після нього 0



Блок-схема 4 Знайти максимальний елемент і вставити після нього 0

Вхідні дані: посилання на стек

Код функції:

void max\_element(stack\*\* top\_stack) {

if ((\*top\_stack) == NULL) return;

stack\* second\_stack = NULL;

short max = (\*top\_stack)->data;

while ((\*top\_stack) != NULL) {

push\_stack((\*top\_stack)->data, &second\_stack);

if ((\*top\_stack)->data > max) max = (\*top\_stack)->data;

pop\_element(\*(&top\_stack));

}

while (second\_stack != NULL) {

push\_stack(second\_stack->data, \*(&top\_stack));

if (second\_stack->data == max) push\_stack(0, \*(&top\_stack));

pop\_element(&second\_stack);

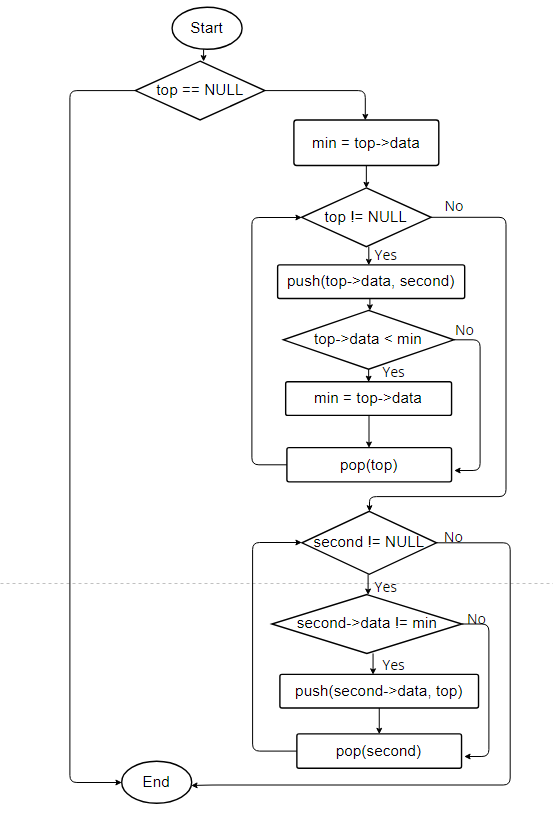
}

}



Виведення у консоль 4 Знайти максимальний елемент і вставити після нього 0

Видалення мінімального елементу



Блок-схема 5 Видалення мінімального елементу

Вхідні дані: посилання на стек

Код функції:

void min\_element(stack\*\* top\_stack) {

if ((\*top\_stack) == NULL) return;

stack\* second\_stack = NULL;

short min = (\*top\_stack)->data;

while ((\*top\_stack) != NULL) {

push\_stack((\*top\_stack)->data, &second\_stack);

if ((\*top\_stack)->data < min) min = (\*top\_stack)->data;

pop\_element(\*(&top\_stack));

}

while (second\_stack != NULL) {

if (second\_stack->data != min) push\_stack(second\_stack->data, \*(&top\_stack));

pop\_element(&second\_stack);

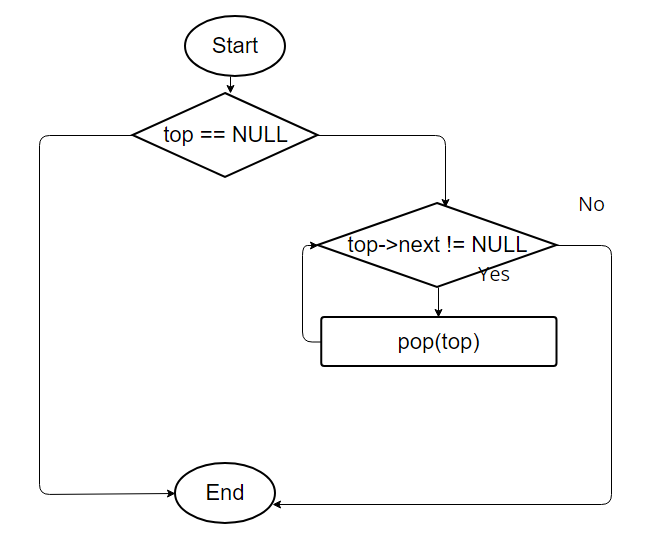
}

}



Виведення у консоль 5 Видалення мінімального елементу

Видалення всіх елементів, крім першого



Блок-схема 6 Видалення всіх елементів, крім першого

Вхідні дані: посилання на стек

Код функції:

void delete\_except\_first(stack\*\* top\_stack) {

if ((\*top\_stack) == NULL) return;

while ((\*top\_stack)->link != NULL) {

pop\_element(\*(&top\_stack));

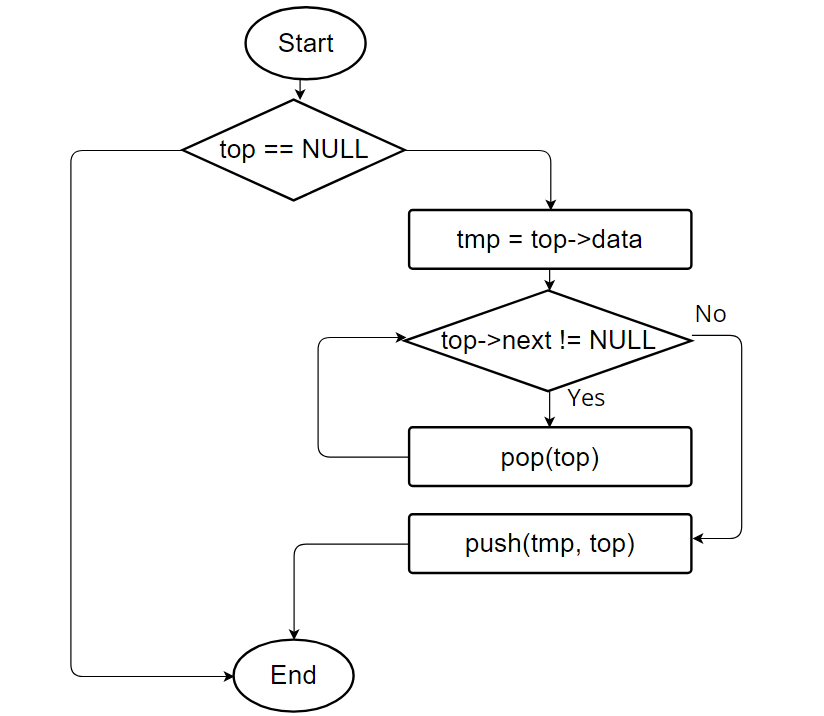
}

}



Виведення у консоль 6 Видалення всіх елементів, крім першого

Видалення всіх елементів, крім останього



Блок-схема 7 Видалення всіх елементів, крім останього

Вхідні дані: посилання на стек

Код функції:

void delete\_except\_last(stack\*\* top\_stack) {

if ((\*top\_stack) == NULL) return;

short tmp = (\*top\_stack)->data;

while ((\*top\_stack) != NULL) {

pop\_element(\*(&top\_stack));

}

push\_stack(tmp, \*(&top\_stack));

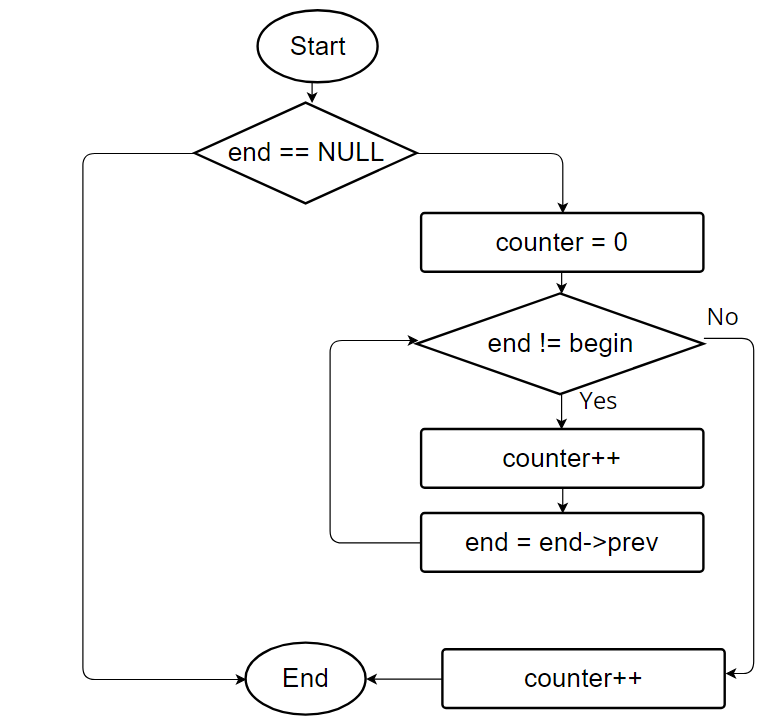
}



Виведення у консоль 7 Видалення всіх елементів, крім останього

Черги — це структура даних, яка працює за принципом "перший прийшов, перший вийшов". Вони використовуються для управління виконанням завдань, особливо в багатопотокових середовищах. Завдяки чергам можна організувати обробку запитів, що дозволяє уникнути перевантаження системи. Черги також використовуються в мережевих протоколах для обробки пакетів даних. Це ефективний спосіб синхронізації між різними компонентами програми.

Знаходження та виведення кількісті елементів черги



Блок-схема 8 Знаходження та виведення кількісті елементів черги

Вхідні дані: посилання на чергу

Код функції:

int count\_queue\_elements(queue list) {

if (list.end == NULL || list.begin == NULL) return 0;

int counter = 0;

while (list.end != list.begin) {

counter++;

list.end = list.end->prev;

}

counter++;

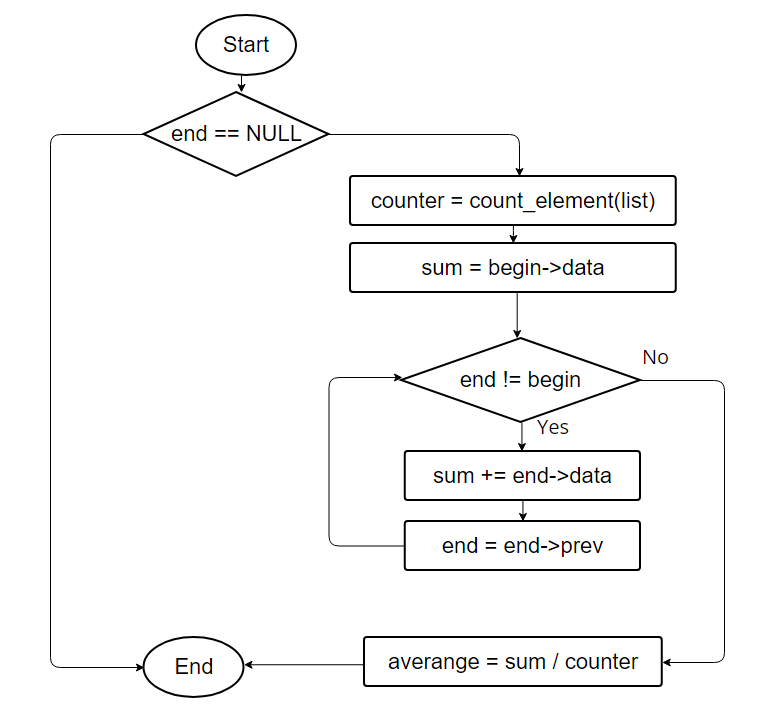
return counter;

}



Виведення у консоль 8 Знаходження та виведення кількісті елементів черги

Знаходження та виведення середнього арифметичного збережених елементів



Блок-схема 9 Знаходження та виведення середнього арифметичного збережених елементів

Вхідні дані: посилання на чергу

Код функції:

double avegange\_queue\_elements(queue list) {

if (list.end == NULL || list.begin == NULL) return 0;

double counter = count\_queue\_elements(list);

double sum = 0;

while (list.end != list.begin) {

sum += list.end->data;

list.end = list.end->prev;

}

sum += list.end->data;

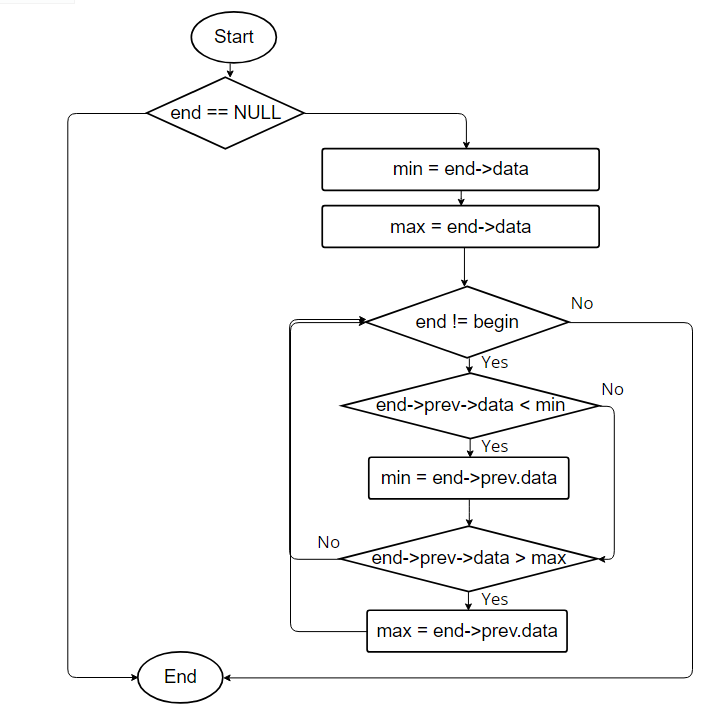
return sum / counter;

}



Виведення у консоль 9 Знаходження та виведення середнього арифметичного збережених елементів

Знаходження та виведенняи мінімальний та максимальний елемент



Блок-схема 10 Знаходження та виведенняи мінімальний та максимальний елемент

Вхідні дані: посилання на чергу

Код функції:

void min\_max\_queue(queue list) {

if (list.end == NULL || list.begin == NULL) {

cout << "Queue is doesn't exist\n";

return;

}

short min = list.end->data;

short max = list.end->data;

while (list.end != list.begin) {

if (list.end->data < min) min = list.end->data;

if (list.end->data > max) max = list.end->data;

list.end = list.end->prev;

}

if (list.end->data < min) min = list.end->data;

if (list.end->data > max) max = list.end->data;

cout << "Minimum of queue is " << min << endl;

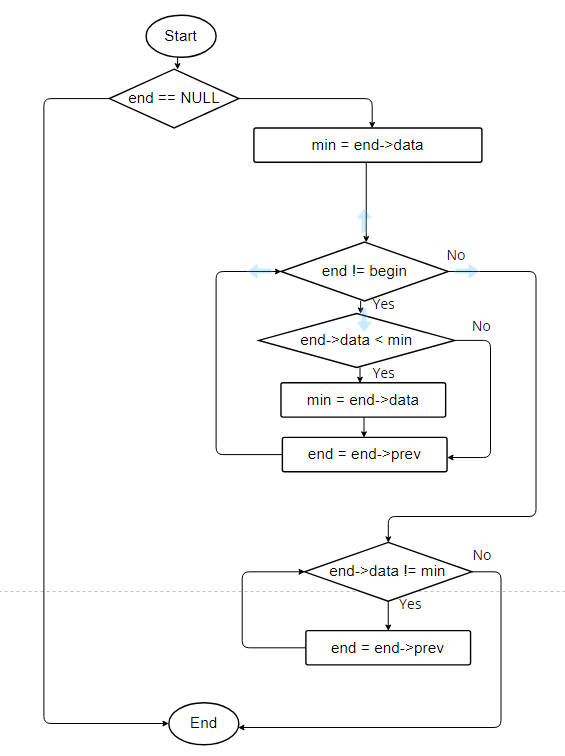
cout << "Maximum of queue is " << max << endl;

}



Виведення у консоль 10 Знаходження та виведенняи мінімальний та максимальний елемент

Знаходження та виведення елемента, який йде перед мінімальним елементом.



Блок-схема 11 Знаходження та виведення елемента, який йде перед мінімальним елементом.

Вхідні дані: посилання на чергу

Код функції:

void element\_before\_min\_queue(queue list) {

if (list.end == NULL || list.begin == NULL) {

cout << "Queue is doesn't exist\n";

return;

}

queue\_node\* end;

end = list.end;

short min = list.end->data;

while (list.end != list.begin) {

if (list.end->data < min) min = list.end->data;

list.end = list.end->prev;

}

if (list.end->data < min) min = list.end->data;

while (end->data != min) {

end = end->prev;

}

end = end->prev;

cout << "Element before minimum: " << end->data;

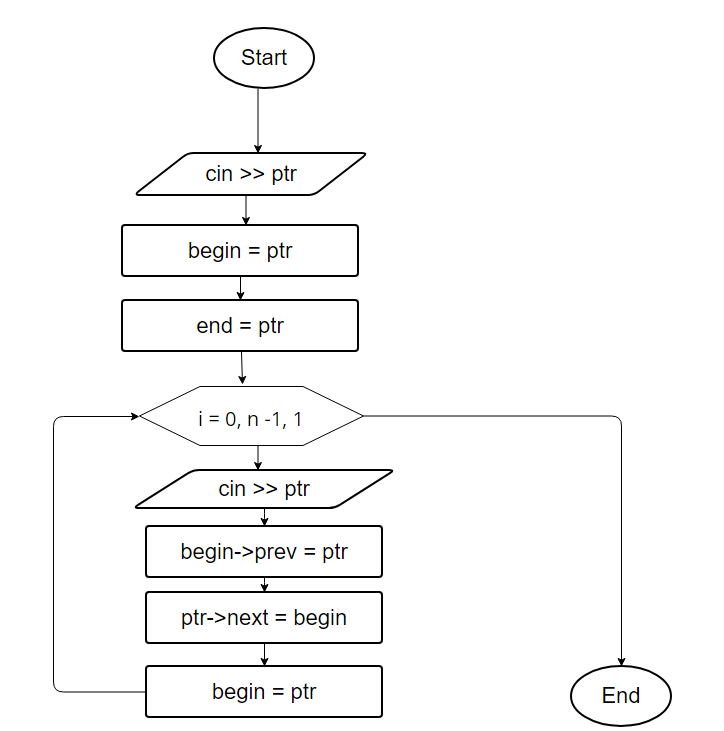
}



Виведення у консоль 11 Знаходження та виведення елемента, який йде перед мінімальним елементом.

Дек — це структура даних, яка дозволяє додавати та видаляти елементи з обох кінців. Він поєднує в собі характеристики черги та стеку, що робить його гнучким для різних задач. Деки часто використовуються в алгоритмах, де потрібно підтримувати порядок елементів з можливістю швидкого доступу з обох боків. Наприклад, вони корисні при реалізації буферів або в задачах обходу графів. Завдяки своїй універсальності, деки знаходять застосування в багатьох галузях програмування.

Cтворення деку



Блок-схема 12 Cтворення деку

Вхідні дані: посилання на дек, кількість елементів

Код функції:

void create\_deque(deque\* list, int n) {

if (n <= 0) return;

deque\_node\* ptr;

ptr = new deque\_node;

cout << "Enter element: ";

cin >> ptr->data;

ptr->next = NULL;

list->begin = ptr;

list->end = ptr;

n--;

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

ptr = new deque\_node;

cout << "Enter element: ";

cin >> ptr->data;

list->begin->prev = ptr;

ptr->next = list->begin;

list->begin = ptr;

list->begin->prev = NULL;

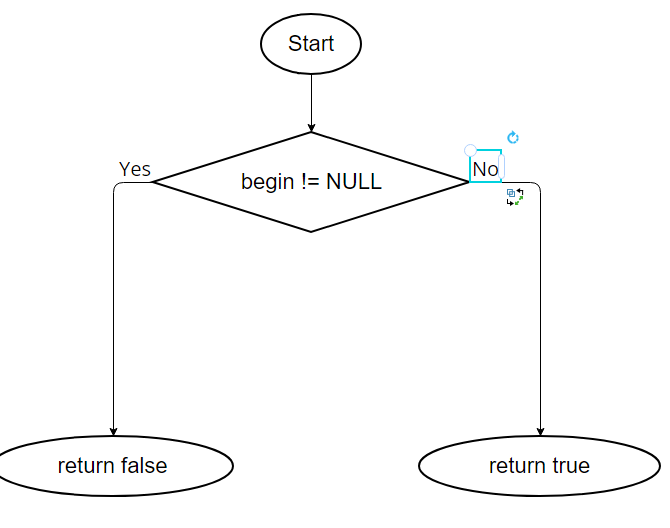
}

}



Виведення у консоль 12 ОCтворення деку

Перевірка на порожнечу



Блок-схема 13 Перевірка на порожнечу

Вхідні дані: посилання на дек

Код функції:

bool is\_empty\_deque(deque list) {

if (list.begin != NULL) return false;

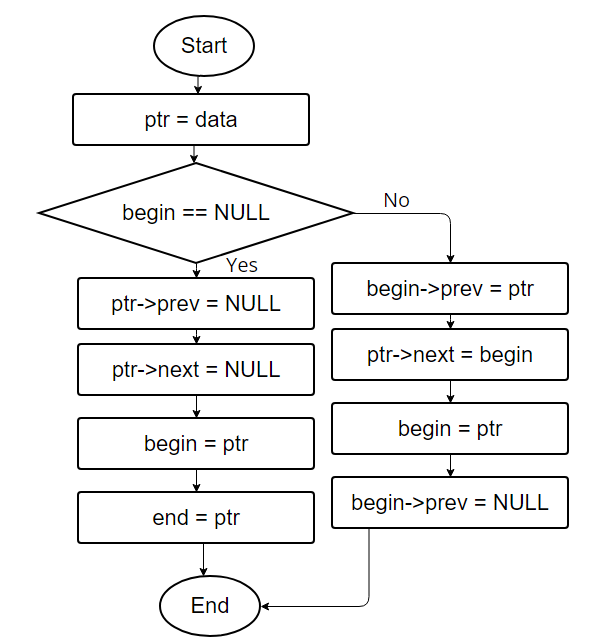
return true;

}



Виведення у консоль 13 Перевірка на порожнечу

Додавання елементу в початок



Блок-схема 14 Додавання елементу в початок

Вхідні дані: посилання на дек, данні

Код функції:

void pop\_element\_to\_deque\_begin(deque\* list, short data) {

deque\_node\* ptr;

ptr = new deque\_node;

ptr->data = data;

if (list->begin == NULL) {

ptr->next = NULL;

ptr->prev = NULL;

list->begin = ptr;

list->end = ptr;

return;

}

list->begin->prev = ptr;

ptr->next = list->begin;

list->begin = ptr;

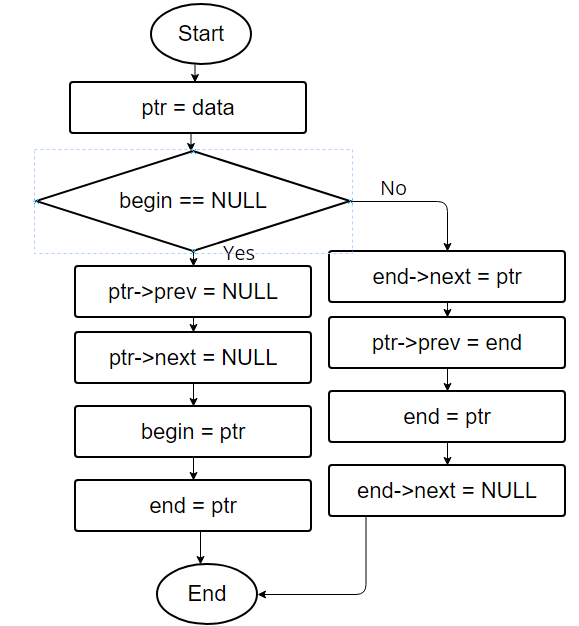
list->begin->prev = NULL;

}



Виведення у консоль 14 Додавання елементу в початок

Додавання елементу в кінець



Блок-схема 15 Додавання елементу в кінець

Вхідні дані: посилання на дек, данні

Код функції:

void pop\_element\_to\_deque\_end(deque\* list, short data) {

deque\_node\* ptr;

ptr = new deque\_node;

ptr->data = data;

if (list->begin == NULL) {

ptr->next = NULL;

ptr->prev = NULL;

list->begin = ptr;

list->end = ptr;

return;

}

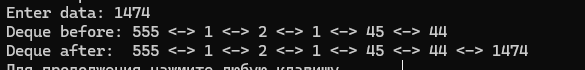
list->end->next = ptr;

ptr->prev = list->end;

list->end = ptr;

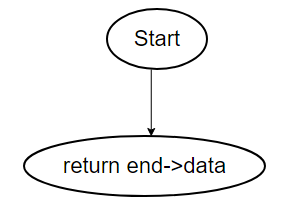
list->end->next = NULL;

}



Виведення у консоль 15 Додавання елементу в кінець

Читання останього елементу



Блок-схема 16 Читання останього елементу

Вхідні дані: посилання на дек

Код функції:

short read\_deque\_last\_element(deque list) {

return list.end->data;

}



Виведення у консоль 16 Читання останього елементу

**Висновки**

В процессі виконання роботи я навчилась реалізовувати та використовувати стеки, черги та деки. В ході виконання я помітила переваги та недоліки, які хочу зазначити.

Стек простий у реалізації та забезпечує швидкі операції додавання та видалення, але обмежує доступ до елементів лише з верхівки. Черга гарантує порядок обробки запитів і легко управляється, проте має обмежений доступ до елементів, що може спричинити затримки.

Дек, на відміну від стека і черги, забезпечує гнучкість, дозволяючи додавати та видаляти елементи з обох кінців, але його реалізація може бути складнішою і вимагати більше пам’яті через необхідність зберігати вказівники. Вибір між цими структурами даних залежить від конкретних вимог задачі та характеристик, що необхідні для ефективної обробки даних.