МІНІСТРЕСТВО ОСВІТИ Й НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“Харківський Політехнічний Інститут”

Кафедра управління проєктами в інформаційних технологіях

Звіт з лабораторної роботи №1

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

з дисципліни

“Алгоритми та структури даних”

Варіант №3

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

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

Харків – 2024

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

**Завдання:**

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;
* очистити список.

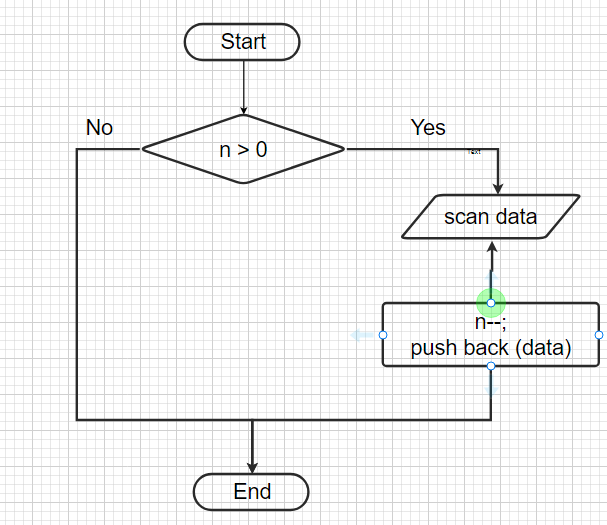
Перетворити лінійний список в двозв’язний. Внести необхідні зміни в операції роботи зі списком. Написати програму, яка демонструє основні дії по роботі із двозв’язним списком і пропонує користувачу вибір дії:

* сформувати список;
* додати елемент до списку;
* вилучити вказаний елемент зі списку;
* поміняти два сусідні елементи місцями;
* очистити список.

Перевірити роботу операцій. Зробити висновки

Однозв'язний список — це структура даних, що складається з вузлів, кожен з яких містить дані та посилання на наступний вузол. Він дозволяє динамічно додавати та видаляти елементи, проте доступ до елементів здійснюється за лінійний час. Основні операції включають створення, додавання, видалення та перелік елементів списку.

Формування списку



Блок-схема 1 Функція формування списку

Вхідні дані: довжина списку

Код функції:

void create\_single\_list(single\_list\*\* list, int n) {

cout << "Enter the data: ";

long long data;

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

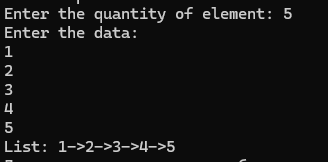
cin >> data;

push\_back(&(\*list), data);

}

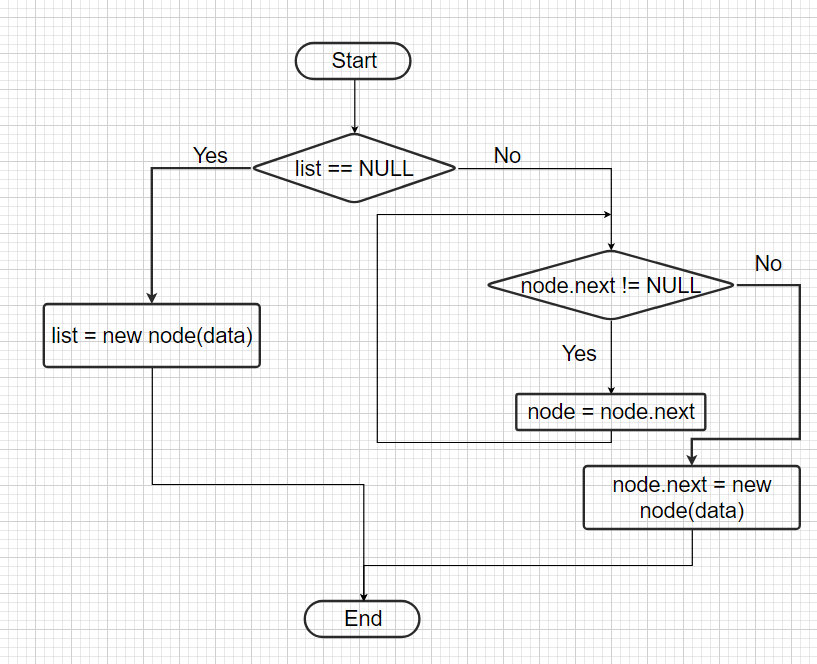
}

Результат у консолі:



Виведення у консоль 1 функція формування списку

Додавання елементів в кінець



Блок-схема 2 Функція елемента в кінець списку

Вхідні дані: новий елемент.

Код функції:

void push\_back(single\_list\*\* list, long long data) {

if (\*list == NULL) {

(\*list) = new single\_list;

(\*list)->node = data;

(\*list)->next = NULL;

return;

}

single\_list\* node = (\*list);

while (node->next != NULL) {

node = node->next;

}

node->next = new single\_list;

node = node->next;

node->node = data;

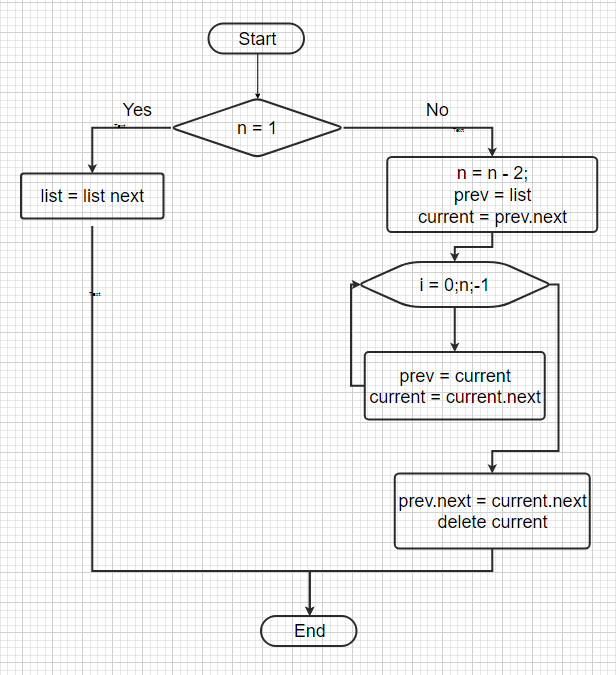
node->next = NULL;

}



Виведення у консоль 2 Функція елемента в кінець списку

Видалення елементу



Блок-схема 3 Видалення елемента списку

Вхідні дані: номер елементу.

Код функції:

void pop\_element(single\_list\*\* list, int n) {

if (n <= 0) {

cout << "Invalid number\n";

return;

}

if (n > list\_size(\*list) - 1) {

cout << "List is too small\n";

return;

}

if (list == NULL) return;

if (n == 1) {

single\_list\* ptr = (\*list);

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

delete ptr;

return;

}

n -= 2;

single\_list\* prev\_ptr = (\*list);

single\_list\* ptr = prev\_ptr->next;

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

prev\_ptr = ptr;

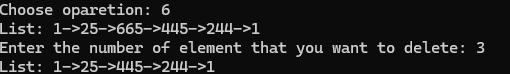
ptr = ptr->next;

}

prev\_ptr->next = ptr->next;

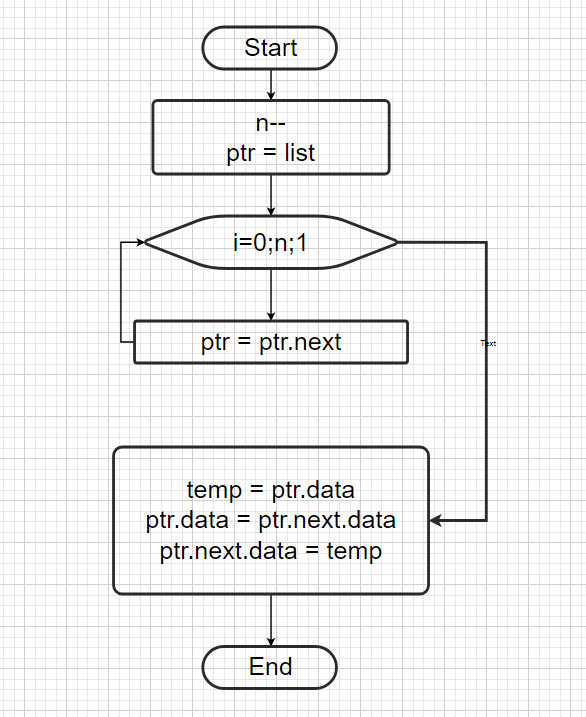
delete ptr;

}



Виведення у консоль 3 Видалення елемента списку

Зміна двох сусідніх елементів



Блок-схема 4 Замінна двох сусідніх елементів

Вхідні дані: позиція першого елементу

Код функції:

void swap\_element(single\_list\*\* list, int n) {

if (n <= 0) {

cout << "Invalid number\n";

return;

}

if (n > list\_size(\*list) - 1) {

cout << "List is too small\n";

return;

}

n--;

long long temp;

single\_list\* ptr = (\*list);

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

ptr = ptr->next;

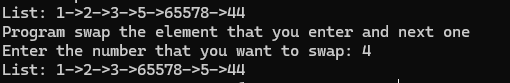
}

temp = ptr->node;

ptr->node = ptr->next->node;

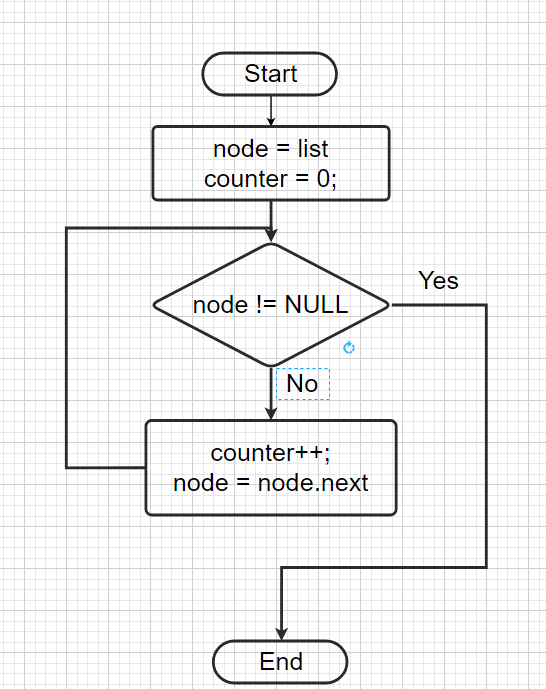
ptr->next->node = temp;

}



Виведення у консоль 4 Замінна двох сусідніх елементів

Визначити кількість елементів



Блок-схема 5 Визначити кількість елементів

Вхідні дані: ніяких

Код функції:

int list\_size(single\_list\* list) {

//if (list == NULL) return 0;

single\_list\* node = list;

int counter = 0;

while (node != NULL) {

node = node->next;

counter++;

}

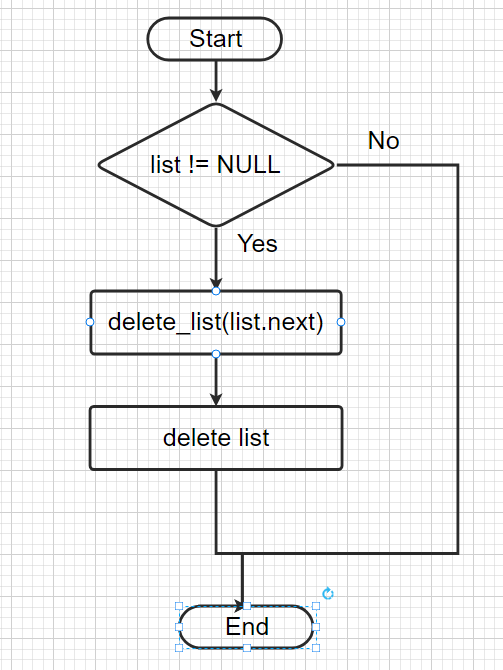
return counter;

}



Виведення у консоль 5 Визначити кількість елементів

Очистити список



Блок-схема 6 Очищення списку

Вхідні дані: посилання на голову списку

Код функції:

void delete\_list(single\_list\*\* list) {

if ((\*list) != NULL) {

delete\_list(&(\*list)->next);

delete\* list;

}

(\*list) = NULL;

}



Виведення у консоль 6 Очищення списку

Двосв'язний список — це структура даних, в якій кожен вузол містить три компоненти: дані, посилання на наступний вузол і посилання на попередній. Це дозволяє легко переходити в обох напрямках, що спрощує деякі операції, такі як вставка і видалення елементів. У двосв'язному списку є вказівники на перший (голова) і останній вузли. Хоча така структура дозволяє більш ефективно маніпулювати елементами, вона використовує більше пам'яті через додаткові посилання. Загалом, двосв'язний список є потужним інструментом для роботи з динамічними наборами даних.

Формування списку

Вхідні дані: посилання на голову списку

Код функції:

void create\_double\_list(double\_list\*\* list, int n) {

cout << "Enter the data: ";

long long data;

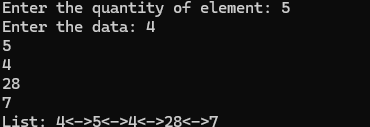
for (int i = 0; i < n; i++) {

cin >> data;

double\_push\_back(&(\*list), data);

}

}



Виведення у консоль 7 Формування двозв'язного списку

Додавання елементу до списку

Вхідні дані: посилання на голову списку, елемент, місце

Код функції:

void push\_double\_element(double\_list\*\* list, long long data, int n) {

n--;

double\_list\* new\_node = new(double\_list);

new\_node->node = data;

new\_node->next = NULL;

new\_node->prev = NULL;

if ((\*list) == NULL) {

(\*list) = new\_node;

return;

}

if (n == 0) {

new\_node->next = (\*list);

(\*list)->prev = new\_node;

(\*list) = new\_node;

return;

}

double\_list\* ptr = (\*list);

for (int i = 1; i < n && ptr->next != NULL; i++) ptr = ptr->next;

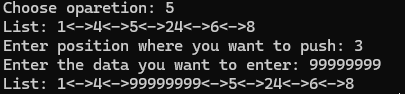
if (ptr->next != NULL)ptr->next->prev = new\_node;

new\_node->next = ptr->next;

ptr->next = new\_node;

new\_node->prev = ptr;

}



Виведення у консоль 8 Додавання елементів до списку

Вилучення вказаного елементу зі списку

Вхідні дані: посилання на голову списку, місце

Код функції:

void pop\_double\_element(double\_list\*\* list, int n) {

if (n <= 0) {

cout << "Invalid number\n";

return;

}

if (n > double\_list\_size(\*list)) {

cout << "List is too small\n";

return;

}

if (list == NULL) return;

if (n == 1) {

double\_list\* ptr = (\*list);

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

(\*list)->prev = NULL;

delete ptr;

return;

}

n -= 2;

double\_list\* ptr = (\*list)->next;

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

ptr = ptr->next;

}

ptr->prev->next = ptr->next;

delete ptr;

}



Виведення у консоль 9 Вилучення елементу списку

Зміна двох сусідніх елементів місцями

Вхідні дані: посилання на голову списку, місце двох елементів

Код функції:

void swap\_double\_element(double\_list\*\* list, int n1, int n2) {

if (n1 <= 0 || n2 < 0) {

cout << "Invalid number\n";

return;

}

if (n1 > double\_list\_size(\*list) - 1 || n2 > double\_list\_size(\*list) - 1) {

cout << "List is too small\n";

return;

}

n1--;

n2--;

long long temp;

double\_list\* ptr1 = (\*list);

for (int i = 0; i < n1; i++) {

ptr1 = ptr1->next;

}

double\_list\* ptr2 = (\*list);

for (int i = 0; i < n2; i++) {

ptr2 = ptr2->next;

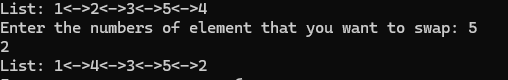
}

temp = ptr1->node;

ptr1->node = ptr2->node;

ptr2->node = temp;

}



Виведення у консоль 10 Заміна двох елементів

Очищення списку

Вхідні дані: посилання на голову списку

Код функції:

void delete\_double\_list(double\_list\*\* list) {

if ((\*list) != NULL) {

delete\_double\_list(&(\*list)->next);

delete\* list;

}

(\*list) = NULL;

}



Виведення у консоль 11 Очищення списку

**Висновки**

В цій роботі я отримала навичи в роботі з динамічними структурами данних, а саме з однозв’язними та двозв’язними списками, а також розробила основні функції для роботи з ними на мові C++.

В ході роботи я змогла помітити ряд відмінностей у цих структурах, а саме у їхній структурі та функціональності. Однозв'язний список займає менше пам'яті, оскільки містить лише одне посилання, і простіший у реалізації. Проте доступ до попередніх елементів потребує проходження з початку списку. Двозв'язний список, хоч і займає більше пам'яті через додаткове посилання, дозволяє легший доступ до попередніх елементів і спрощує операції видалення. Кожен із списків має свої переваги, що робить їх корисними для різних сценаріїв використання.