Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

Звіт

про виконання лабораторних та практичних робіт блоку № 6

На тему: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.»

з дисципліни: «Основи програмування»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт до блоку № 6

Виконала:

Студент групи ШІ-11

Боднар Денис

Львів 2024

Тема роботи: Вивчення основ динамічних структур даних у C++: стек, черга, зв'язні списки та дерева, а також алгоритмів обробки цих структур, включаючи операції додавання, видалення елементів та пошук. Розгляд основних принципів виділення пам'яті, переповнення стеку, обробки подій через чергу, а також обробки дерев різних типів (бінарних, AVL, червоно-чорних дерев) за допомогою ітеративних та рекурсивних алгоритмів.

Мета роботи: Ознайомитися з основними динамічними структурами даних у C++, зокрема стеком, чергою, зв'язними списками та деревами, вивчити їх властивості та операції (push, pop, enqueue, dequeue тощо). Опанувати основні алгоритми пошуку, сортування, вставки та видалення елементів у цих структурах. Зрозуміти принципи виділення пам'яті для динамічних структур та їх обробку, включаючи випадки переповнення та особливості обробки складних дерев.

Теоретичні відомості:

Теоретичні відомості з переліком важливих тем:

- Тема №1: Основи Динамічних Структур Даних.
- Тема №2: Стек.
- Тема №3: Черга.
- Тема №4: Зв'язні Списки.
- Тема №5: Дерева.

Індивідуальний план опрацювання теорії:

Тема №1: Основи Динамічних Структур Даних.

- Джерела:
  https://acode.com.ua/urok-111-stek-i-kupa/#toc-1
  https://www.youtube.com/watch?v=NyOjKd5Qruk
- Що опрацьовано:
  - Вступ до динамічних структур даних: визначення та важливість
  - Виділення пам'яті для структур даних (stack і heap)
  - Приклади простих динамічних структур: динамічний масив
- Статус: Ознайомлений

Тема №2: Стек.

- Джерела:
  https://dystosvita.org.ua/mod/page/view.php?id=888
  https://disted.edu.vn.ua/courses/learn/13472
  https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D1%81%D1%8C%D0%BA%D0%B8%D0%B9_%D1%96%D0%BD%

D0%B2%D0%B5%D1%80%D1%81%D0%BD%D0%B8%D0%B9_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81

- Що опрацьовано:
    - Визначення та властивості стеку
    - Операції push, pop, top: реалізація та використання
    - Приклади використання стеку: обернений польський запис
    - Переповнення стеку
- Статус: Ознайомлений

Тема №3: Черга.

- Джерела:
  https://dystosvita.org.ua/mod/page/view.php?id=889
  https://www.kostrub.online/2020/07/struktura-danyx-cherha-Queue.html
- Що опрацьовано:
    - Визначення та властивості черги
    - Операції enqueue, dequeue, front: реалізація та застосування
    - Приклади використання черги: обробка подій, алгоритми планування
    - Розширення функціоналу черги: пріоритетні черги

      Статус: Ознайомлений

Тема №4: Зв'язні Списки.

- Джерела:
  https://prometheus.org.ua/cs50/sections/section6.html
- Що опрацьовано:
    - Визначення однозв'язного та двозв'язного списку
    - Принципи створення нових вузлів, вставка між існуючими, видалення, створення кільця(circular linked list)
    - Основні операції: обхід списку, пошук, доступ до елементів, об'єднання списків
    - Приклади використання списків: управління пам'яттю, FIFO та LIFO структури
- Статус: Ознайомлений

Тема №5: Дерева.

- Джерела:
  https://purecodecpp.com/uk/archives/2483
  https://javarush.com/ua/groups/posts/uk.4165.chervono-chorne-derevo-vlastivost-principi-organzac-mekhanzmi-vstavki

- Що опрацьовано:
    - Вступ до структури даних "дерево": визначення, типи
    - Бінарні дерева: вставка, пошук, видалення
    - Обхід дерева: в глибину (preorder, inorder, postorder), в ширину
    - Застосування дерев: дерева рішень, хеш-таблиці
    - Складніші приклади дерев: AVL, Червоно-чорне дерево
- Статус: Ознайомлений

Виконання роботи:

Опрацювання завдання та вимог до програми та середовища

Завдання №1 - VNS Lab 10 - Task 1-13

Написати програму, у якій створюються динамічні структури й виконати їхню обробку у відповідності зі своїм варіантом.

Для кожного варіанту розробити такі функції:

1. Створення списку.
2. Додавання елемента в список (у відповідності зі своїм варіантом).
3. Знищення елемента зі списку (у відповідності зі своїм варіантом).
4. Друк списку.
5. Запис списку у файл.
6. Знищення списку.
7. Відновлення списку з файлу.

Порядок виконання роботи

1. Написати функцію для створення списку. Функція може створювати порожній список, а потім додавати в нього елементи.
2. Написати функцію для друку списку. Функція повинна передбачати вивід повідомлення, якщо список порожній.
3. Написати функції для знищення й додавання елементів списку у відповідності зі своїм варіантом.
4. Виконати зміни в списку й друк списку після кожної зміни.
5. Написати функцію для запису списку у файл.
6. Написати функцію для знищення списку.

7. Записати список у файл, знищити його й виконати друк (при друці повинне бути видане повідомлення "Список порожній").

8. Написати функцію для відновлення списку з файлу.

9. Відновити список і роздрукувати його.

10.Знищити список.

13. Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити з нього К перших елементів. Додати елемент після елемента, що починається із зазначеного символу.

Завдання №2 - Algotester Lab 5v2

Lab 5v2

Обмеження: 1 сек., 256 МіБ

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це N, ширина - M.

Всередині печери є пустота, пісок та каміння. Пустота позначається буквою O , пісок S і каміння X;

Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння.

Ваше завдання сказати як буде виглядати печера після землетрусу.

Вхідні дані

У першому рядку 2 цілих числа N та M - висота та ширина печери

У N наступних рядках стрічка row_i яка складається з N цифер - i-й рядок матриці, яка відображає стан печери до землетрусу.

Вихідні дані

N рядків, які складаються з стрічки розміром M - стан печери після землетрусу.

Обмеження

$1 \leq N, M \leq 1000$

$|row_i| = M$

$row_i \in \{X, S, O\}$

Завдання №3 - Algotester Lab 78v2

Lab 78v2

Обмеження: 1 сек., 256 МіБ

Ваше завдання - власноруч реалізувати структуру даних "Динамічний масив".
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.

Вам будуть поступати запити такого типу:

- **Вставка**:
  Ідентифікатор - $insert$
  Ви отримуєте ціле число $index$ елемента, на місце якого робити вставку.
  Після цього в наступному рядку рядку написане число N - розмір масиву, який треба вставити.
  У третьому рядку N цілих чисел - масив, який треба вставити на позицію $index$.
- **Видалення**:
  Ідентифікатор - $erase$
  Ви отримуєте 2 цілих числа - $index$, індекс елемента, з якого почати видалення та n - кількість елементів, яку треба видалити.
- **Визначення розміру**:
  Ідентифікатор - $size$
  Ви не отримуєте аргументів.
  Ви виводите кількість елементів у динамічному масиві.
- **Визначення кількості зарезервованої пам'яті**:
  Ідентифікатор - $capacity$
  Ви не отримуєте аргументів.
  Ви виводите кількість зарезервованої пам'яті у динамічному масиві.
  Ваша реалізація динамічного масиву має мати фактор росту (Growth factor) рівний 2.
- **Отримання значення** i-го елементу
  Ідентифікатор - get
  Ви отримуєте ціле число - $index$, індекс елемента.
  Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора []
- **Модифікація значення** i-го елементу
  Ідентифікатор - set
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перегрузку оператора []
- **Вивід динамічного масиву на екран**
  Ідентифікатор - $print$
  Ви не отримуєте аргументів.
  Ви виводите усі елементи динамічного масиву через пробіл.
  Реалізувати використовуючи перегрузку оператора <<

Вхідні дані

Ціле число Q - кількість запитів.
У наступних рядках Q запитів у зазначеному в умові форматі.

Вихідні дані

Відповіді на запити у зазначеному в умові форматі.

Обмеження

$0 \leq Q \leq 10^5$
$0 \leq l_i \leq 10^5$
$\|l\| \leq 10^5$

Завдання №4 - Class Practice Work - Task 1 - Реверс списку (Reverse list)

Реалізувати метод реверсу списку: Node* reverse(Node *head);

Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати метод реверсу;
- реалізувати допоміжний метод виведення вхідного і обернутого списків;

Мета задачі

Розуміння структур даних: Реалізація методу реверсу для зв'язаних списків є чудовим способом для поглиблення розуміння зв'язаних списків як фундаментальної структури даних. Він заохочує практичний підхід до вивчення того, як структуруються пов'язані списки та як ними маніпулювати.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Перевертання пов'язаного списку вимагає логічного підходу до маніпулювання покажчиками, що є ключовим навиком у інформатиці.

Засвоїти механізми маніпуляції з покажчиками: пов'язані списки значною мірою залежать від покажчиків. Це завдання покращить навички маніпулювання вказівниками, що є ключовим аспектом у таких мовах, як C++.

Розвинути навички розв'язувати задачі: перевернути пов'язаний список непросто й вимагає творчого й логічного мислення, таким чином покращуючи свої навички розв'язування поставлених задач.

Пояснення прикладу

Спочатку ми визначаємо просту структуру ***Node*** для нашого пов'язаного списку.

Потім функція ***reverse*** ітеративно змінює список, маніпулюючи наступними покажчиками кожного вузла.

printList — допоміжна функція для відображення списку.

Основна функція створює зразок списку, демонструє реверсування та друкує вихідний і обернений списки.

Завдання №5 - Class Practice Work - Task 2 - Порівняння списків

bool compare(Node *h1, Node *h2);

Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати функцію, яка ітеративно проходиться по обох списках і порівнює дані в кожному вузлі;

- якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.

Мета задачі

Розуміння рівності в структурах даних: це завдання допомагає зрозуміти, як визначається рівність у складних структурах даних, таких як зв'язані списки. На відміну від примітивних типів даних, рівність пов'язаного списку передбачає порівняння кожного елемента та їх порядку.

Поглиблення розуміння зв'язаних списків: Порівнюючи зв'язані списки, дозволяють покращити своє розуміння обходу, фундаментальної операції в обробці зв'язаних списків.

Розуміння ефективність алгоритму: це завдання також вводить поняття ефективності алгоритму. Студенти вчаться ефективно порівнювати елементи, що є навичкою, важливою для оптимізації та зменшення складності обчислень.

Розвинути базові навики роботи з реальними програми: функції порівняння мають вирішальне значення в багатьох реальних програмах, таких як виявлення змін у даних, синхронізація структур даних або навіть у таких алгоритмах, як сортування та пошук.

Розвинути навик вирішення проблем і увага до деталей: це завдання заохочує скрупульозний підхід до програмування, оскільки навіть найменша неуважність може призвести до неправильних результатів порівняння. Це покращує навички вирішення проблем і увагу до деталей.

Пояснення прикладу

- Для пов'язаного списку визначено структуру ***Node***.
- Функція ***compare*** ітеративно проходить обидва списки одночасно, порівнюючи дані в кожному вузлі.
- Якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.
- Основна функція ***main*** створює два списки та демонструє порівняння.

Завдання №6 - Class Practice Work - Task 3 - Додавання великих чисел

Node* add(Node *n1, Node *n2);

Умови задачі:

- використовувати цифри від 0 до 9 для значень у списку;

- реалізувати функцію, яка обчислює суму двох чисел, які збережено в списку; молодший розряд числа записано в голові списка (напр. $379 \implies 9 \rightarrow 7 \rightarrow 3$);

- функція повертає новий список, передані в функцію списки не модифікуються.

Мета задачі

Розуміння операцій зі структурами даних: це завдання унаочнює практичне використання списка для обчислювальних потреб. Арифметичні операції з великими числами це окремий клас задач, для якого використання списків допомагає обійти обмеження у представленні цілого числа сучасними комп'ютерами.

Поглиблення розуміння зв'язаних списків: Застосовування зв'язаних списків для арифметичних операції з великими числами дозволяє покращити розуміння операцій з обробки зв'язаних списків.

Розуміння ефективність алгоритму: це завдання дозволяє порівняти швидкість алгоритму додавання з використанням списків зі швидкістю вбудованих арифметичних операцій. Студенти вчаться розрізняти позитивні та негативні ефекти при виборі структур даних для реалізації практичних програм.

Розвинути базові навики роботи з реальними програми: арифметичні операції з великими числами використовуються у криптографії, теорії чисел, астрономії, та ін.

Розвинути навик вирішення проблем і увага до деталей: завдання покращує розуміння обмежень у представленні цілого числа сучасними комп'ютерами та пропонує спосіб його вирішення.

Завдання №7 - Class Practice Work - Task 4 - Віддзеркалення дерева

TreeNode *create_mirror_flip(TreeNode *root);

Умови задачі:

- використовувати цілі числа для значень у вузлах дерева

- реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева

- функція повертає нове дерево, передане в функцію дерево не модифікується

Мета задачі

Розуміння структур даних: Реалізація методу віддзеркалення бінарного дерева покращує розуміння структури бінарного дерева, виділення пам'яті для

вузлів та зв’язування їх у єдине ціле. Це один з багатьох методів роботи з бінарними деревами.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева продемонструє розгортання рекурсивного виклику.

Завдання №8 - Class Practice Work - Task 5 - Записати кожному батьківському вузлу суму підвузлів

void tree_sum(TreeNode *root);

Умови задачі:

- використовувати цілочисельні значення у вузлах дерева;
- реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів
- вузол-листок не змінює значення
- значення змінюються від листків до кореня дерева

Мета задачі

Розуміння структур даних: Реалізація методу підрахунку сум підвузлів бінарного дерева покращує розуміння структури бінарного дерева. Це один з багатьох методів роботи з бінарними деревами.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева демонструє розгортання рекурсивного виклику.

Завдання №9 - Self Practice Work

Lab 5v3

Обмеження: 1 сек., 256 МіБ

У вас є карта гори розміром $N \times M$.

Також ви знаєте координати $\{x, y\}$, у яких знаходиться вершина гори.

Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число.

Клітинкі які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

Вхідні дані

У першому рядку 2 числа N та M - розміри карти

у другому рядку 2 числа x та y - координати піку гори

Вихідні дані

N рядків по M елементів в рядку через пробіл - висоти карти.

Обмеження

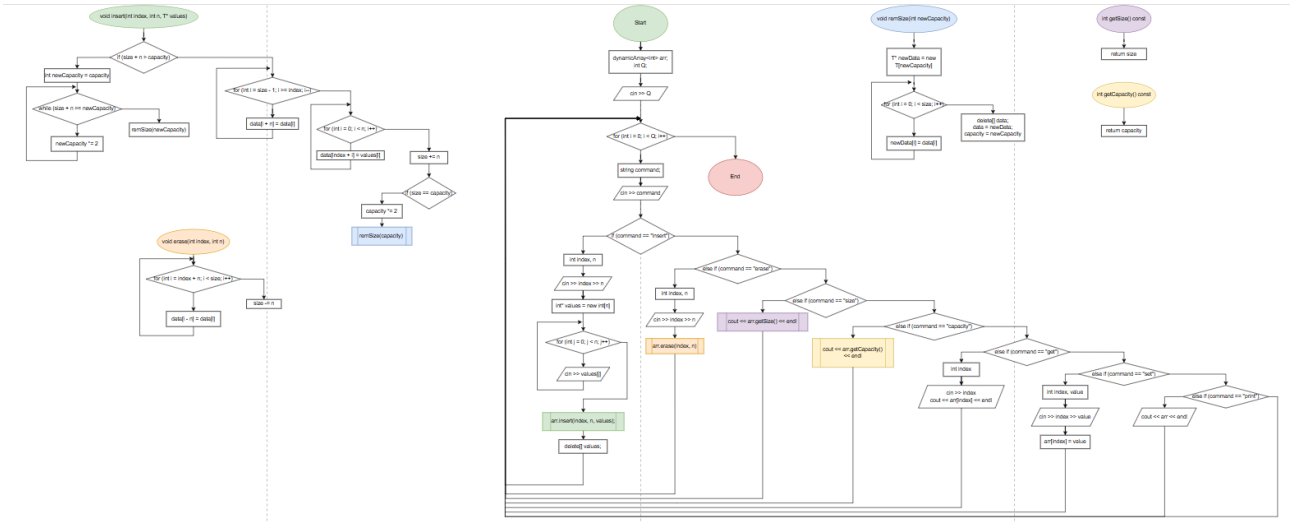
$1 \leq N, M \leq 10^3$

$1 \leq x \leq N$

$1 \leq y \leq M$

Дизайн виконання завдань

Завдання №3 - Algotester Lab 78v2



Код програм та фактично затрачений час

Завдання №1 - VNS Lab 10 - Task 1-13

```cpp
#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

struct Node {
    char* key;
    Node* prev;
    Node* next;
};

Node* createNode(const char* key) {
    Node* newNode = new Node;
    newNode->key = new char[strlen(key) + 1];
    strcpy(newNode->key, key);
    newNode->prev = nullptr;
    newNode->next = nullptr;
    return newNode;
}

Node* createList() {
    return nullptr;
}

void printList(Node* head) {
    Node* current = head;
    while (current) {
        cout << current->key << " ";
        current = current->next;
    }
    cout << endl;
}

void addElement(Node*& head, const char* newKey, char startChar) {
    Node* current = head;
    while (current) {
        if (current->key[0] == startChar) {
            Node* newNode = createNode(newKey);
            newNode->next = current->next;
            newNode->prev = current;
            if (current->next) {
                current->next->prev = newNode;
            }
            current->next = newNode;
            return;
        }
        current = current->next;
    }
    cout << "Елемент із символом '" << startChar << "' не знайдено.\n";
}
```

```cpp
void deleteFirstKElements(Node*& head, int k) {
    while (k-- > 0 && head) {
        Node* temp = head;
        head = head->next;
        if (head) {
            head->prev = nullptr;
        }
        delete[] temp->key;
        delete temp;
    }
}

void writeListToFile(Node* head, const char* filename) {
    ofstream file(filename);
    if (!file.is_open()) {
        cerr << "Не вдалося відкрити файл для запису.\n";
        return;
    }
    Node* current = head;
    while (current) {
        file << current->key << endl;
        current = current->next;
    }

    file.close();
}

Node* readListFromFile(const char* filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "Не вдалося відкрити файл для читання.\n";
        return nullptr;
    }
    Node* head = nullptr;
    Node* tail = nullptr;
    char buffer[256];
    while (file.getline(buffer, 256)) {
        Node* newNode = createNode(buffer);
        if (!head) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    file.close();

    return head;
}
```

```
105  void destroyList(Node*& head) {
106      while (head) {
107          Node* temp = head;
108          head = head->next;
109          delete[] temp->key;
110          delete temp;
111      }
112  }
113
114  int main() {
115      Node* list = createList();
116
117      list = createNode("Denys");
118      list->next = createNode("Bodnar");
119      list->next->prev = list;
120      list->next->next = createNode("Chinazes");
121      list->next->next->prev = list->next;
122
123      cout << "Початковий список:\n";
124      printList(list);
125
126      deleteFirstKElements(list, 2);
127      cout << "Список після видалення перших 2 елементів:\n";
128      printList(list);
129
130      addElement(list, "Sanchizes", 'C');
131      cout << "Список після додавання елемента:\n";
132      printList(list);
133
134      writeListToFile(list, "test.txt");
135
136      destroyList(list);
137      cout << "Список знищено.\n";
138
139      list = readListFromFile("test.txt");
140      cout << "Список після відновлення з файлу:\n";
141      printList(list);
142
143      destroyList(list);
144
145      return 0;
146  }
```

Фактично затрачений час: 4 год

Завдання №2 - Algotester Lab 5v2

```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main() {
    int N, M;
    cin >> N >> M;

    vector<string> cave(N);
    for (int i = 0; i < N; ++i) {
        cin >> cave[i];
    }

    for (int j = 0; j < M; ++j) {
        int emptyRow = N - 1;

        for (int i = N - 1; i >= 0; --i) {
            if (cave[i][j] == 'X') {
                emptyRow = i - 1;
            } else if (cave[i][j] == 'S') {
                cave[i][j] = 'O';
                cave[emptyRow][j] = 'S';
                --emptyRow;
            }
        }
    }

    for (const string &row : cave) {
        cout << row << endl;
    }

    return 0;
}
```

Фактично затрачений час: 2 год

Завдання №3 - Algotester Lab 78v2

```cpp
#include <iostream>

using namespace std;

template<typename T>

class dynamicArray {
private:
    T* data;
    int size;
    int capacity;

    void remSize(int newCapacity) {
        T* newData = new T[newCapacity];
        for (int i = 0; i < size; i++) {
            newData[i] = data[i];
        }
        delete[] data;
        data = newData;
        capacity = newCapacity;
    }

public:
    dynamicArray() : size(0), capacity(1) {
        data = new T[capacity];
    }

    ~dynamicArray() {
        delete[] data;
    }

    void insert(int index, int n, T* values) {
        if (size + n > capacity) {
            int newCapacity = capacity;
            while (size + n >= newCapacity) {
                newCapacity *= 2;
            }
            remSize(newCapacity);
        }

        for (int i = size - 1; i >= index; i--) {
            data[i + n] = data[i];
        }
        for (int i = 0; i < n; i++) {
            data[index + i] = values[i];
        }
        size += n;

        if (size == capacity) {
            capacity *= 2;
            remSize(capacity);
        }
    }
```

```
55      void erase(int index, int n) {
56          for (int i = index + n; i < size; i++) {
57              data[i - n] = data[i];
58          }
59          size -= n;
60      }
61
62      int getSize() const {
63          return size;
64      }
65
66      int getCapacity() const {
67          return capacity;
68      }
69
70      T& operator[](int index) {
71          if (index < 0 || index >= size) {
72              throw out_of_range("ERROR");
73          }
74          return data[index];
75      }
76
77      friend ostream& operator<<(ostream& os, const dynamicArray& arr) {
78          for (int i = 0; i < arr.size; i++) {
79              os << arr.data[i] << (i < arr.size - 1 ? " " : "");
80          }
81          return os;
82      }
83  };
```

```cpp
int main() {
    dynamicArray<int> arr;
    int Q;
    cin >> Q;

    for (int i = 0; i < Q; i++) {
        string command;
        cin >> command;

        if (command == "insert") {
            int index, n;
            cin >> index >> n;
            int* values = new int[n];
            for (int j = 0; j < n; j++) {
                cin >> values[j];
            }
            arr.insert(index, n, values);
            delete[] values;
        } else if (command == "erase") {
            int index, n;
            cin >> index >> n;
            arr.erase(index, n);
        } else if (command == "size") {
            cout << arr.getSize() << endl;
        } else if (command == "capacity") {
            cout << arr.getCapacity() << endl;
        } else if (command == "get") {
            int index;
            cin >> index;
            cout << arr[index] << endl;
        } else if (command == "set") {
            int index, value;
            cin >> index >> value;
            arr[index] = value;
        } else if (command == "print") {
            cout << arr << endl;
        }
    }

    return 0;
}
```

Фактично затрачений час: 3.5 год

Завдання №4 - Class Practice Work - Task 1 - Реверс списку (Reverse list)

```cpp
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* curr = head;
    Node* next = nullptr;

    while (curr != nullptr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }

    return prev;
}

void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);

    cout << "Оригінальний список: ";
    printList(head);

    head = reverse(head);

    cout << "Перевернутий список: ";
    printList(head);

    return 0;
}
```

Фактично затрачений час: 2 год

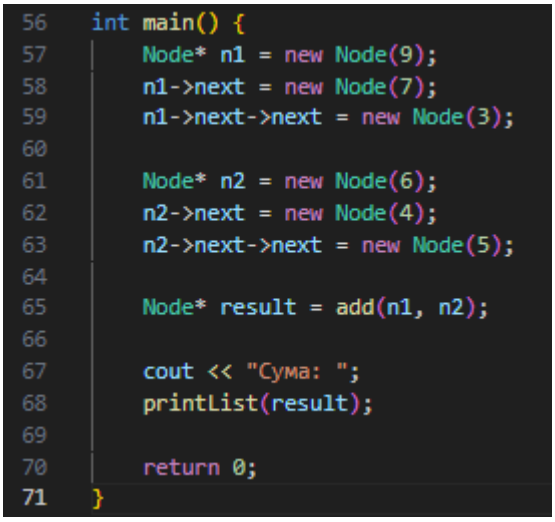
Завдання №5 - Class Practice Work - Task 2 - Порівняння списків

```cpp
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

bool compare(Node* head1, Node* head2) {
    while (head1 != nullptr && head2 != nullptr) {
        if (head1->data != head2->data) {
            return false;
        }
        head1 = head1->next;
        head2 = head2->next;
    }
    return head1 == nullptr && head2 == nullptr; // Перевірка на однакову довжину
}

void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head1 = new Node(1);
    head1->next = new Node(2);
    head1->next->next = new Node(3);

    Node* head2 = new Node(1);
    head2->next = new Node(2);
    head2->next->next = new Node(3);

    cout << "Списки рівні? " << (compare(head1, head2) ? "Так" : "Ні") << endl;

    return 0;
}
```

Фактично затрачений час: 2 год

Завдання №6 - Class Practice Work - Task 3 - Додавання великих чисел

```cpp
#include <iostream>
#include <stack>

using namespace std;

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

Node* add(Node* n1, Node* n2) {
    stack<int> stack1, stack2;

    while (n1) {
        stack1.push(n1->data);
        n1 = n1->next;
    }

    while (n2) {
        stack2.push(n2->data);
        n2 = n2->next;
    }

    int carry = 0;
    Node* result = nullptr;

    while (!stack1.empty() || !stack2.empty() || carry) {
        int sum = carry;
        if (!stack1.empty()) {
            sum += stack1.top();
            stack1.pop();
        }
        if (!stack2.empty()) {
            sum += stack2.top();
            stack2.pop();
        }

        carry = sum / 10;
        Node* newNode = new Node(sum % 10);
        newNode->next = result;
        result = newNode;
    }

    return result;
}

void printList(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}
```



```
int main() {
    Node* n1 = new Node(9);
    n1->next = new Node(7);
    n1->next->next = new Node(3);

    Node* n2 = new Node(6);
    n2->next = new Node(4);
    n2->next->next = new Node(5);

    Node* result = add(n1, n2);

    cout << "Сума: ";
    printList(result);

    return 0;
}
```

Фактично затрачений час: 3 год

Завдання №7 - Class Practice Work - Task 4 - Віддзеркалення дерева

```cpp
#include <iostream>

using namespace std;

struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

TreeNode* createMirrorFlip(TreeNode* root) {
    if (root == nullptr) {
        return nullptr;
    }

    TreeNode* left = createMirrorFlip(root->left);
    TreeNode* right = createMirrorFlip(root->right);

    root->left = right;
    root->right = left;

    return root;
}

void inOrder(TreeNode* root) {
    if (root == nullptr) {
        return;
    }
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);

    cout << "Оригінальне дерево: ";
    inOrder(root);
    cout << endl;

    root = createMirrorFlip(root);

    cout << "Віддзеркалене дерево: ";
    inOrder(root);
    cout << endl;

    return 0;
}
```

Фактично затрачений час: 4 год

Завдання №8 - Class Practice Work - Task 5 - Записати кожному

```cpp
#include <iostream>

using namespace std;

struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

int treeSum(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }

    int leftSum = treeSum(root->left);
    int rightSum = treeSum(root->right);

    if (root->left || root->right) {
        root->data = leftSum + rightSum;
    }

    return root->data + leftSum + rightSum;
}

void inOrder(TreeNode* root) {
    if (root == nullptr) {
        return;
    }
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}

int main() {
    TreeNode* root = new TreeNode(10);
    root->left = new TreeNode(20);
    root->right = new TreeNode(30);
    root->left->left = new TreeNode(40);
    root->left->right = new TreeNode(50);
    root->right->left = new TreeNode(60);

    treeSum(root);

    cout << "Дерево після оновлення суми: ";
    inOrder(root);
    cout << endl;

    return 0;
}
```

Фактично затрачений час: 4 год

Завдання №9 - Self Practice Work

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>

using namespace std;

int main() {
    int n, m, x, y;

    cin >> n >> m;
    cin >> x >> y;

    const int dx[] = {1, -1, 0, 0};
    const int dy[] = {0, 0, 1, -1};

    x--;
    y--;

    vector<vector<int>> heights(n, vector<int>(m, -1));

    queue<pair<int, int>> q;
    q.push({x, y});
    heights[x][y] = 0;
    int maxHeight = 0;

    while (!q.empty()) {
        auto [currentX, currentY] = q.front();
        q.pop();

        for (int i = 0; i < 4; ++i) {
            int neighborX = currentX + dx[i];
            int neighborY = currentY + dy[i];

            if (neighborX >= 0 && neighborX < n && neighborY >= 0 && neighborY < m && heights[neighborX][neighborY] == -1) {
                heights[neighborX][neighborY] = heights[currentX][currentY] + 1;
                maxHeight = max(maxHeight, heights[neighborX][neighborY]);
                q.push({neighborX, neighborY});
            }
        }
    }

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            cout << maxHeight - heights[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Фактично затрачений час: 3.5 год

Результати виконання завдань

Завдання №1 - VNS Lab 10 - Task 1-13

```
Початковий список:
Denys Bodnar Chinazes
Список після видалення перших 2 елементів:
Chinazes
Список після додавання елемента:
Chinazes Sanchizes
Список знищено.
Список після відновлення з файлу:
Chinazes Sanchizes
```

Завдання №2 - Algotester Lab 5v2

```
5 5
SSOSS
OOOOO
SOOXX
OOOOS
OOSOO
OOOOO
OOOSS
OOOXX
SOOOO
SSSOS
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
20 годин тому	C++ 23	Зараховано	0.025	1.840	Перегляд
20 годин тому	C++ 23	Неправильна відповідь 1	0.002	0.914	Перегляд
20 годин тому	C++ 23	Неправильна відповідь 1	0.002	0.914	Перегляд

Завдання №3 - Algotester Lab 78v2

```
12

size
0
capacity
1

insert 0 2
100 100

size
2
capacity
4

insert 0 2
102 102

size
4
capacity
8

insert 0 2
103 103

size
6
capacity
8

print
103 103 102 102 100 100
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
17 годин тому	C++ 23	Зараховано	0.006	1.285	Перегляд
17 годин тому	C++ 23	Неправильна відповідь 1	0.002	0.922	Перегляд
18 годин тому	C++ 23	Неправильна відповідь 1	0.002	0.941	Перегляд
18 годин тому	C++ 23	Неправильна відповідь 1	0.002	0.918	Перегляд

Завдання №4 - Class Practice Work - Task 1 - Реверс списку (Reverse list)

```
Оригінальний список: 1 2 3 4
Перевернутий список: 4 3 2 1
```

Завдання №5 - Class Practice Work - Task 2 - Порівняння списків

```
Списки рівні? Так
```

Завдання №6 - Class Practice Work - Task 3 - Додавання великих чисел

```
Сума: 1 6 1 8
```

Завдання №7 - Class Practice Work - Task 4 - Віддзеркалення дерева

```
Оригінальне дерево: 4 2 5 1 6 3 7
Віддзеркалене дерево: 7 3 6 1 5 2 4
```

Завдання №8 - Class Practice Work - Task 5 - Записати кожному батьківському вузлу суму підвузлів

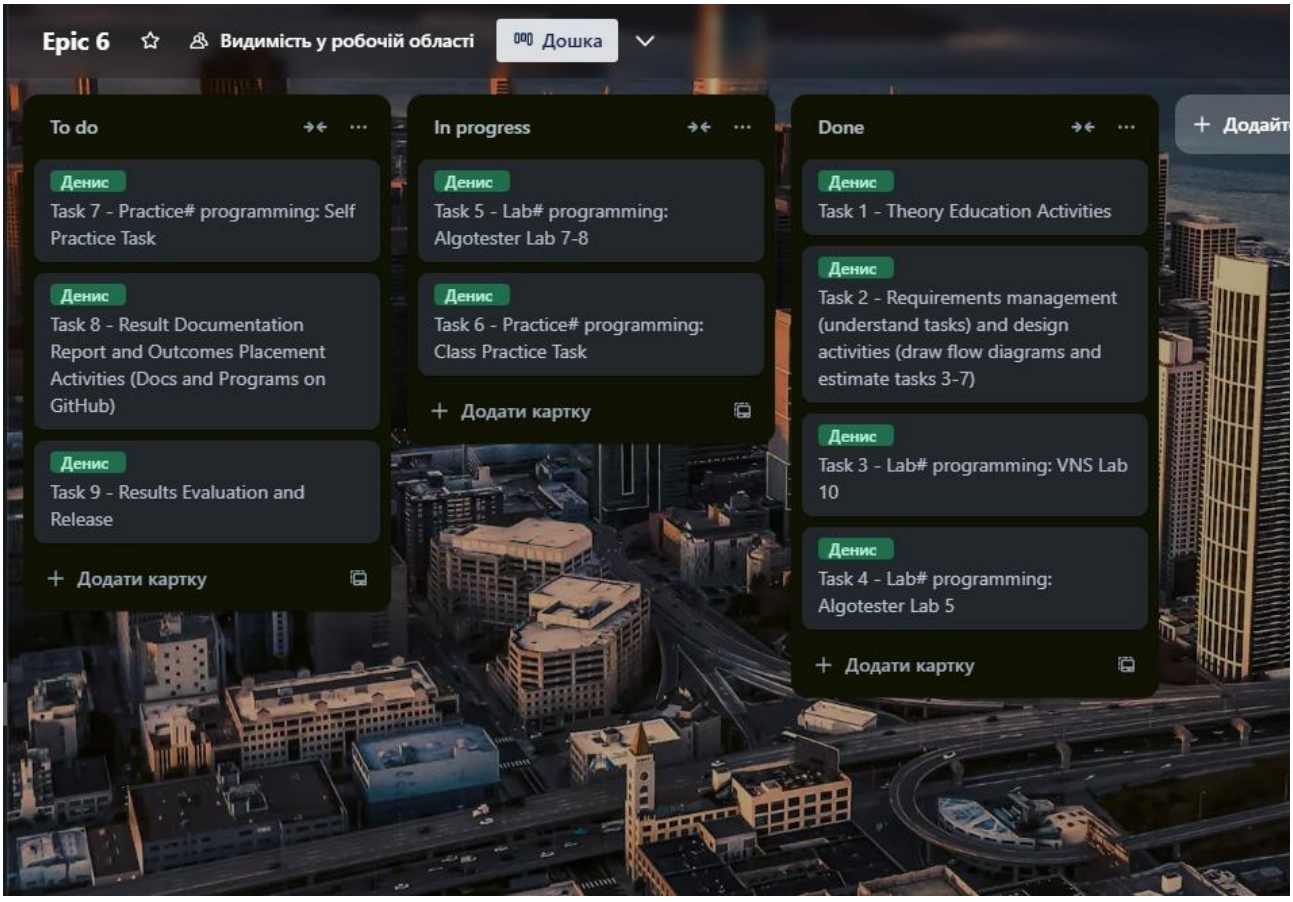
```
Дерево після оновлення суми: 40 90 50 300 60 60
```

Завдання №9 - Self Practice Work

```
1 1
1 1
0
```

Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
C++ 23	Зараховано	0.117	6.789	Перегляд

Кооперація з командою



Висновок:

Виконуючи шостий епік, я ознайомився з основними принципами роботи з динамічними структурами даних у C++, зокрема стеком, чергою, зв'язними списками та деревами. Вивчив їх властивості та основні операції, такі як додавання і видалення елементів. Опанував алгоритми для маніпуляцій з цими структурами, включаючи ітеративні та рекурсивні підходи. Особливу увагу приділив принципам виділення пам'яті для динамічних структур, розумінню їх обробки, а також роботі з переповненням стеку і обробці подій через чергу. Дослідження складніших типів дерев, таких як бінарні, AVL та червоно-чорні, дозволило краще зрозуміти їх застосування та переваги в реальних задачах. Виконання цих завдань поглибило моє розуміння принципів роботи з динамічною пам'яттю та покращило навички використання алгоритмів для ефективної обробки даних.