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

Виконав:
Студент групи ШІ-11
Станько Олег Ігорович

Львів 2024

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

Мета: зрозуміти основи побудови та використання таких структур даних, як черга, стек, списки та дерева, освоїти алгоритми, які дозволяють ефективно маніпулювати та обробляти ці структури.

Теоретичні відомості

1. Основи Динамічних Структур Даних
2. Стек
3. Черга
4. Зв'язні Списки
5. Дерева
6. Алгоритми Обробки Динамічних Структур

індвидуальний план опрацювання теорії

1. Основи Динамічних Структур Даних
   Data Structures Tutorial - GeeksforGeeks
   Ознайомився з виділенням пам'яті для структур даних (stack і heap)
   Витрачено 1 годину
2. **Стек**
   Stack Data Structure - GeeksforGeeks
   Вивчив принцип роботи стеку, його застосування та основні операції (push, pop, top).
   Витрачено 1 годину.
3. **Черга**
   Queue Data Structure - GeeksforGeeks
   Ознайомився з принципами роботи черги, її застосуванням та основними операціями (enqueue, dequeue, front).
   Витрачено 1 годину.
4. **Зв'язні Списки**
   Linked List Data Structure - GeeksforGeeks
   Вивчив основи зв'язних списків, різновиди (однозв'язний, двозв'язний) та основні операції (вставка, видалення, пошук).
   Витрачено 2 години.
5. **Дерева**
   Linked List Data Structure - GeeksforGeeks
   Ознайомився з деревами, їх видами (бінарні дерева, дерева пошуку) та основними операціями (вставка, видалення, пошук).
   Витрачено 2 години.
6. **Алгоритми Обробки Динамічних Структур**
   Algorithms Tutorial - GeeksforGeeks
   Вивчив алгоритми обробки динамічних структур даних, зокрема сортування та пошук.
   2 години.

Виконання роботи:

1. Опрацювання завдання та вимог до програм.

VNS LAB 10

Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим ключем. Додати K елементів у початок списку.

Algotester Lab 5

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 rowi 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.

Algotester Lab 7-8

Ваше завдання - власноруч реалізувати структуру даних "Двійкове дерево пошуку".
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його параметри.

Вам будуть поступати запити такого типу:

- **Вставка**:
  Ідентифікатор - insert
  Ви отримуєте ціле число value - число, яке треба вставити в дерево.
- **Пошук**:
  Ідентифікатор - contains
  Ви отримуєте ціле число value - число, наявність якого у дереві необхідно перевірити.
  Якщо valuevalue наявне в дереві - ви виводите Yes, у іншому випадку No.
- **Визначення розміру**:
  Ідентифікатор - size
  Ви не отримуєте аргументів.
  Ви виводите кількість елементів у дереві.
- **Вивід дерева на екран**
  Ідентифікатор - printprint
  Ви не отримуєте аргументів.
  Ви виводите усі елементи дерева 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.

Для того щоб отримати 50% **балів за лабораторну достатньо написати свою структуру.**
Для отримання 100% **балів ця структура має бути написана як** шаблон класу**, у якості параметру використати** int**. Використовувати STL заборонено.**

Class Practice Work

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;

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***.

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→7→3);
- функція повертає новий список, передані в функцію списки не модифікуються.

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

Self-practice task

Одного осіннього дня ти, студенте, сидиш на лекції, яку викладач однотонно бубнить собі під ніс. Вигляд з вікна приковує до себе твій погляд — надворі з дерев опадає листя.

Вигляд з вікна описується прямокутником, що складається з n×mn×m клітинок. Кожна клітинка прямокутника порожня (позначається крапкою) або містить листок з дерева (позначається зірочкою). Листя падає вертикально вниз.

Знайди вигляд з вікна після того, як усе листя опаде. Уважаємо, що нове листя у вікні не з'явиться.

Вхідні дані

У першому рядку задано два цілі числа n і m — розміри прямокутника.

У наступних n рядках задано прямокутник розміру n×m — опис вигляду з вікна.

Вихідні дані

В nn рядках виведи вигляд з вікна після опадання листя.

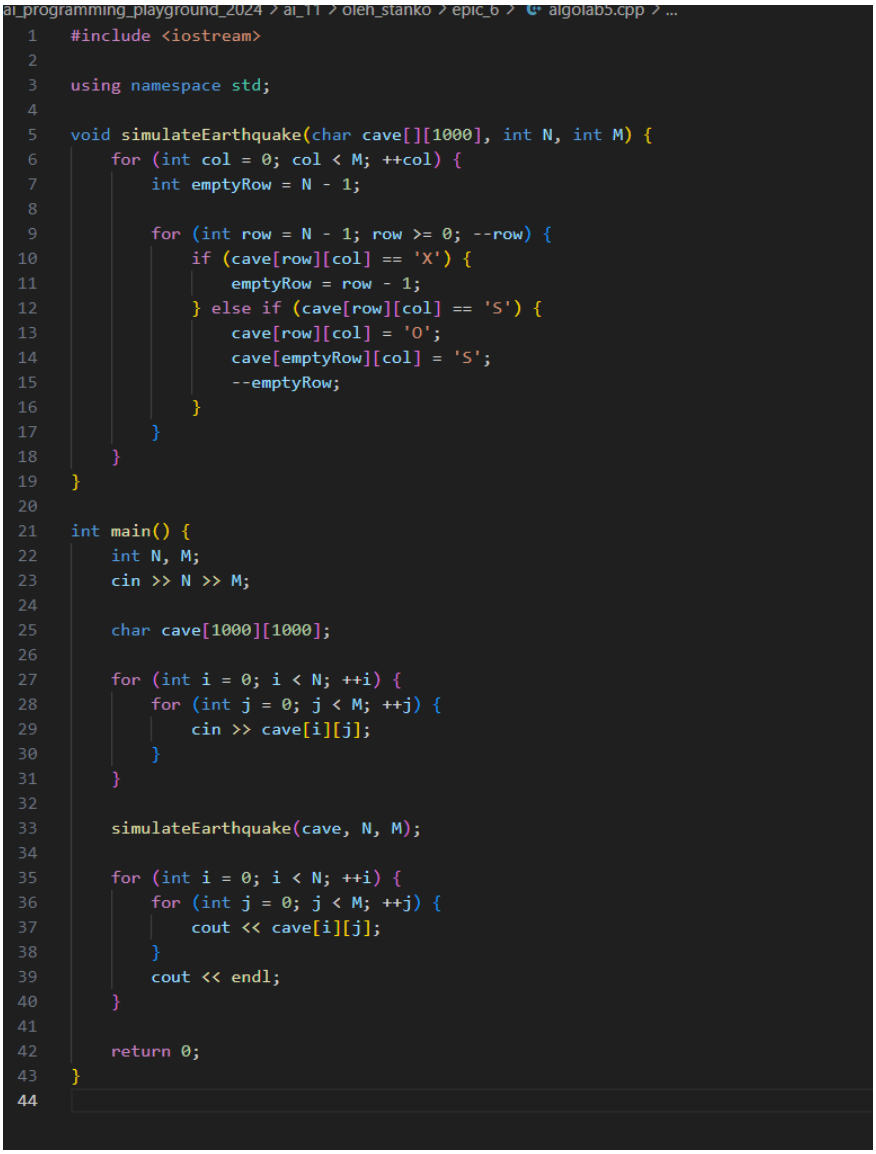
Коди:

VNS LAB 10

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

using namespace std;

struct Node {
    char* data;
    Node* next;
    Node* prev;

    Node(const char* value) : data(strdup(value)), next(nullptr), prev(nullptr) {}
};

void createLL(Node*& head) {
    cout << "Creating the list..." << endl;
    const char* values[] = {"Node 1", "Node 2", "Node 3", "Node 4", "Node 5", "Node 6"};

    for (int i = 0; i < 6; i++) {
        Node* newNode = new Node(values[i]);

        if (head == nullptr) {
            newNode->prev = nullptr;
            head = newNode;
        } else {
            Node* tmp = head;
            while (tmp->next != nullptr) {
                tmp = tmp->next;
            }
            tmp->next = newNode;
            newNode->prev = tmp;
        }
    }
}

void deleteByData(Node*& head, const char* data) {
    cout << "Deleting element: " << data << " from the list..." << endl;
    if (head == nullptr) return;

    Node* tmp = head;
    while (tmp != nullptr) {
        if (strcmp(tmp->data, data) == 0) {
            Node* toDelete = tmp;
            if (toDelete == head) {
                head = toDelete->next;
                if (head != nullptr) head->prev = nullptr;
            } else {
                if (toDelete->prev != nullptr) {
                    toDelete->prev->next = toDelete->next;
                }
                if (toDelete->next != nullptr) {
                    toDelete->next->prev = toDelete->prev;
                }
            }
            free(toDelete->data);
            delete toDelete;
            return;
        }
        tmp = tmp->next;
    }
}

void printLL(Node* head) {
    cout << "Printing the list..." << endl;
    if (head == nullptr) {
        cout << "The list is empty." << endl;
        return;
    }

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

void addKElementsToBegining(Node*& head, int K, ...) {
    cout << "Adding " << K << " elements to the beginning of the list..." << endl;

    va_list args;
    va_start(args, K);

    for (int i = 0; i < K; i++) {
        const char* data = va_arg(args, const char*);
        Node* newNode = new Node(data);
        newNode->next = head;
        newNode->prev = nullptr;

        if (head != nullptr) {
            head->prev = newNode;
        }
        head = newNode;
    }
    va_end(args);
}
```

```cpp

void writeLLToFile(Node* head, const char* filename) {
    cout << "Writing the list to file: " << filename << "..." << endl;
    ofstream outFile(filename);
    if (!outFile) {
        cerr << "Error: Cannot open the file." << endl;
        exit(1);
    }

    Node* tmp = head;
    while (tmp != nullptr) {
        outFile << tmp->data << endl;
        tmp = tmp->next;
    }
}

void deleteLL(Node*& head) {
    cout << "Deleting the list..." << endl;
    while (head != nullptr) {
        Node* nextNode = head->next;
        free(head->data);
        delete head;
        head = nextNode;
    }
}

void fromFileToLL(Node*& head, const char* filename) {
    cout << "Loading the list from file: " << filename << "..." << endl;
    ifstream inFile(filename);
    if (!inFile) {
        cerr << "Error: Cannot open the file." << endl;
        exit(2);
    }

    string line;
    Node* tmp = nullptr;
    while (getline(inFile, line)) {
        Node* newNode = new Node(line.c_str());
        newNode->next = nullptr;
        newNode->prev = tmp;

        if (head == nullptr) {
            head = newNode;
        } else {
            tmp->next = newNode;
        }
        tmp = newNode;
    }
}

int main() {
    Node* head = nullptr;
    const char* filename = "file.txt";

    createLL(head);
    printLL(head);

    deleteByData(head, "Node 3");
    printLL(head);

    addKElementsToBegining(head, 3, "Node -1", "Node -2", "Node -3");
    printLL(head);

    writeLLToFile(head, filename);
    deleteLL(head);
    printLL(head);

    fromFileToLL(head, filename);
    printLL(head);

    deleteLL(head);
    printLL(head);
}

```

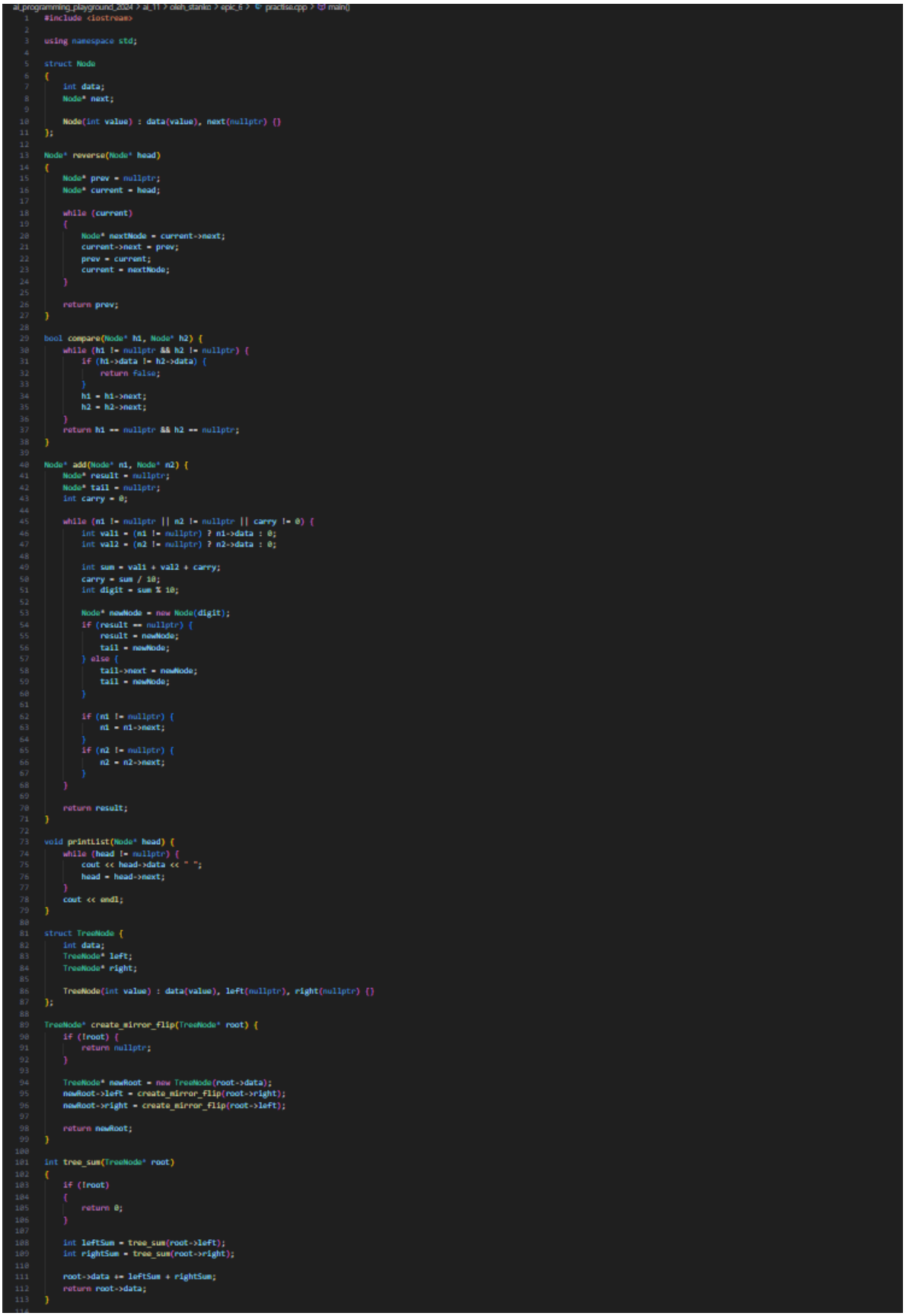
Algo lab 5



Algo lab 7_8

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

using namespace std;

struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int value) : data(value), left(nullptr), right(nullptr) {}
};

template <class T>
struct binary_search_tree {
public:
    binary_search_tree() : root(nullptr) {};

    void insert(T value) {
        root = insert(root, value);
    }

    bool contains(T value) {
        return contains(root, value);
    }

    int size() {
        return size(root);
    }

    void print() {
        cout << *this << endl;
    }

    friend ostream& operator<<(ostream& os, const binary_search_tree<T>& tree) {
        tree.inorderTraverse(tree.root, os);
        return os;
    }

private:
    TreeNode* root;

    TreeNode* insert(TreeNode* node, T value) {
        if (node == nullptr) {
            return new TreeNode(value);
        }
        if (value < node->data) {
            node->left = insert(node->left, value);
        } else if (value > node->data) {
            node->right = insert(node->right, value);
        }
        return node;
    }

    bool contains(TreeNode* node, T value) {
        if (node == nullptr) {
            return false;
        }
        if (value == node->data) {
            return true;
        } else if (value < node->data) {
            return contains(node->left, value);
        } else {
            return contains(node->right, value);
        }
    }

    int size(TreeNode* node) {
        if (node == nullptr) {
            return 0;
        }
        return 1 + size(node->left) + size(node->right);
    }

    void inorderTraverse(TreeNode* node, ostream& os) const {
        if (node != nullptr) {
            inorderTraverse(node->left, os);
            os << node->data << " ";
            inorderTraverse(node->right, os);
        }
    }
};

int main() {
    binary_search_tree<int> binary_tree;
    int q;
    cin >> q; // Кількість команд
    for (int i = 0; i < q; i++) {
        string command;
        cin >> command;

        if (command == "insert") {
            int value;
            cin >> value;
            binary_tree.insert(value);

        } else if (command == "size") {
            cout << binary_tree.size() << endl;

        } else if (command == "contains") {
            int value;
            cin >> value;
            cout << (binary_tree.contains(value) ? "Yes" : "No") << endl;

        } else if (command == "print")
        {
            binary_tree.print();
        }
    }

    return 0;
}
```

Practice


ai_programming_playground_2024 > ai_11 > oleh_stanko > epic_6 > practise.cpp > main()
#include <iostream>
using namespace std;
struct Node
{
int data;
Node* next;
Node(int value) : data(value), next(nullptr) {}
};
Node* reverse(Node* head)
{
Node* prev = nullptr;
Node* current = head;
while (current)
{
Node* nextNode = current->next;
current->next = prev;
prev = current;
current = nextNode;
}
return prev;
}
bool compare(Node* h1, Node* h2) {
while (h1 != nullptr && h2 != nullptr) {
if (h1->data != h2->data) {
return false;
}
h1 = h1->next;
h2 = h2->next;
}
return h1 == nullptr && h2 == nullptr;
}
Node* add(Node* n1, Node* n2) {
Node* result = nullptr;
Node* tail = nullptr;
int carry = 0;
while (n1 != nullptr || n2 != nullptr || carry != 0) {
int val1 = (n1 != nullptr) ? n1->data : 0;
int val2 = (n2 != nullptr) ? n2->data : 0;
int sum = val1 + val2 + carry;
carry = sum / 10;
int digit = sum % 10;
Node* newNode = new Node(digit);
if (result == nullptr) {
result = newNode;
tail = newNode;
} else {
tail->next = newNode;
tail = newNode;
}
if (n1 != nullptr) {
n1 = n1->next;
}
if (n2 != nullptr) {
n2 = n2->next;
}
}
return result;
}
void printList(Node* head) {
while (head != nullptr) {
cout << head->data << " ";
head = head->next;
}
cout << endl;
}
struct TreeNode {
int data;
TreeNode* left;
TreeNode* right;
TreeNode(int value) : data(value), left(nullptr), right(nullptr) {}
};
TreeNode* create_mirror_flip(TreeNode* root) {
if (!root) {
return nullptr;
}
TreeNode* newRoot = new TreeNode(root->data);
newRoot->left = create_mirror_flip(root->right);
newRoot->right = create_mirror_flip(root->left);
return newRoot;
}
int tree_sum(TreeNode* root)
{
if (!root)
{
return 0;
}
int leftSum = tree_sum(root->left);
int rightSum = tree_sum(root->right);
root->data += leftSum + rightSum;
return root->data;
}

```
}

void printTree(TreeNode* root) {
    if (!root) return;
    printTree(root->left);
    cout << root->data << " ";
    printTree(root->right);
}

int main() {
    // Робота з однозв'язними списками
    Node* list1 = new Node(2);
    list1->next = new Node(4);
    list1->next->next = new Node(3);

    Node* list2 = new Node(5);
    list2->next = new Node(6);
    list2->next->next = new Node(4);

    cout << "List 1: ";
    printList(list1);

    cout << "List 2: ";
    printList(list2);

    Node* sumList = add(list1, list2);
    cout << "Sum of List 1 and List 2: ";
    printList(sumList);

    cout << "Are List 1 and List 2 equal? " << (compare(list1, list2) ? "Yes" : "No") << endl;

    Node* reversedList1 = reverse(list1);
    cout << "Reversed List 1: ";
    printList(reversedList1);

    TreeNode* root = new TreeNode(4);
    root->left = new TreeNode(2);
    root->left->right = new TreeNode(3);
    root->left->left = new TreeNode(1);

    root->right = new TreeNode(6);
    root->right->left = new TreeNode(5);
    root->right->right = new TreeNode(7);

    cout << "\nOriginal Tree: ";
    printTree(root);

    TreeNode* mirroredRoot = create_mirror_flip(root);
    cout << "\nMirrored Tree: ";
    printTree(mirroredRoot);

    tree_sum(root);
    cout << "\nTree after subtree sums (Preorder): ";
    printTree(root);

    return 0;
}
```

Self practice work

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

using namespace std;

int main() {
    int n, m;
    cin >> n >> m;

    string st = "";
    int s[100];

    for (int i = 0; i < m; i++) {
        s[i] = 0;
    }

    for (int i = 0; i < n; i++) {
        string a;
        cin >> a;
        for (int j = 0; j < m; j++) {
            if (a[j] == '*') {
                s[j]++;
            }
        }
    }

    for (int i = n; i > 0; i--) {
        for (int j = 0; j < m; j++) {
            if (s[j] == i) {
                s[j]--;
                st += '*';
            } else {
                st += '.';
            }
        }
        cout << st << endl;
        st = "";
    }

    return 0;
}
```

Робота з командою

Зустрічалися в зумі 19 листопада

Висновок:

Протягом виконання цього епіку я ознайомився з основами динамічних структур даних та їх основними операціями, такими як вставка, видалення та пошук. Перш за все, я вивчив принципи виділення пам'яті для різних структур даних, зокрема для стеку та черги. Оволодів технікою роботи з цими структурами, зокрема операціями push, pop, top для стека і enqueue, dequeue, front для черги.

Далі я вивчив зв'язні списки, зокрема однозв'язні та двозв'язні, а також їх основні операції. Це дозволило мені краще зрозуміти їх структуру і застосування. Окремо я ознайомився з деревами, зокрема з бінарними деревами і деревами пошуку, що є важливою частиною роботи з динамічними структурами.

Завершенням навчання стало вивчення алгоритмів обробки динамічних структур, таких як сортування та пошук, що допомогло мені краще зрозуміти, як ефективно працювати з цими структурами в реальних задачах.

У результаті я значно поглибив свої знання в області динамічних структур даних і алгоритмів їх обробки, що є важливим кроком для подальшого вивчення програмування.