Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

Звіт

про виконання

Лабораторних та практичних робіт № 6

з дисципліни: «Основи програмування»

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

з дисципліни: «Основи програмування»

Виконав:

студент групи ШІ – 11

Яровой Павло Олегович

Львів 2023

Тема роботи:

Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.

Мета роботи:

Ознайомлення та вивчення використання динамічних структур даних, таких як Черга, Стек, Списки та Дерево, а також освоєння алгоритмів обробки даних в дереві. Розгляд особливостей застосування кожної структури та їх використання для оптимальної ефективності в конкретних завданнях.

Теоретичні відомості:

1) Індивідуальний план опрацювання теорії:

   Тема №1: Динамічні структури (Черга, Стек, Списки, Дерево).
   - Джерела Інформації
     - https://www.geeksforgeeks.org/data-structures/
     - https://www.boardinfinity.com/blog/guide-to-5-data-structures-in-c/

   Тема №2: Алгоритми обробки Дерев.
   - Джерела Інформації:
     - https://www.programiz.com/dsa/trees

Виконання роботи:

1. Опрацювання завдання та вимог до програм та середовища:

TASK № 1 VNS LAB 10 Variant 20

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.

*20.Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим ключем. Додати по К елементів на початок й в кінець списку.*

TASK № 2 Algotester Lab 5v2

Lab 5v2

Limits: 1 sec., 256 MiB

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.

Input

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 - і-й рядок матриці, яка відображає стан печери до землетрусу.

Output

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

Constraints

$1 \leq N, M \leq 1000$

$|row_i| = M$

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

TASK№ 3 Algotester Lab 78v1 (100%)

Lab 78v1

Limits: 2 sec., 256 MiB

Ваше завдання - власноруч реалізувати структуру даних "Двозв'язний список".
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 списку, який треба вставити.
  У третьому рядку N цілих чисел - список, 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 списку.
- **Отримання значення** i-го елементу
  Ідентифікатор - get
  Ви отримуєте ціле число - $index$, індекс елемента.
  Ви виводите значення елемента за індексом.
- **Модифікація значення** i-го елементу
  Ідентифікатор - set
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення.
- **Вивід списку на екран**
  Ідентифікатор - $print$
  Ви не отримуєте аргументів.
  Ви виводите усі елементи списку через пробіл.
  Реалізувати використовуючи перегрузку оператора <<

Input

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

Output

Відповіді на запити у зазначеному в умові форматі.

Constraints

$0 \leq Q \leq 10^3$
$0 \leq l_i \leq 10^3$
$\|l\| \leq 10^3$

TASK№ 4 Class Practice Task (1-5)

Задача №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;

Задача №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***.

Задача №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 $\Rightarrow$ 9→7→3);
- функція повертає новий список, передані в функцію списки не модифікуються.

Задача №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

Задача №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

TASK№ 5 Self Practice Task

Lab 5v3

Limits: 1 sec., 256 MiB

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.

Input

У першому рядку 2 числа N та M - розміри карти

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

Output

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

Constraints

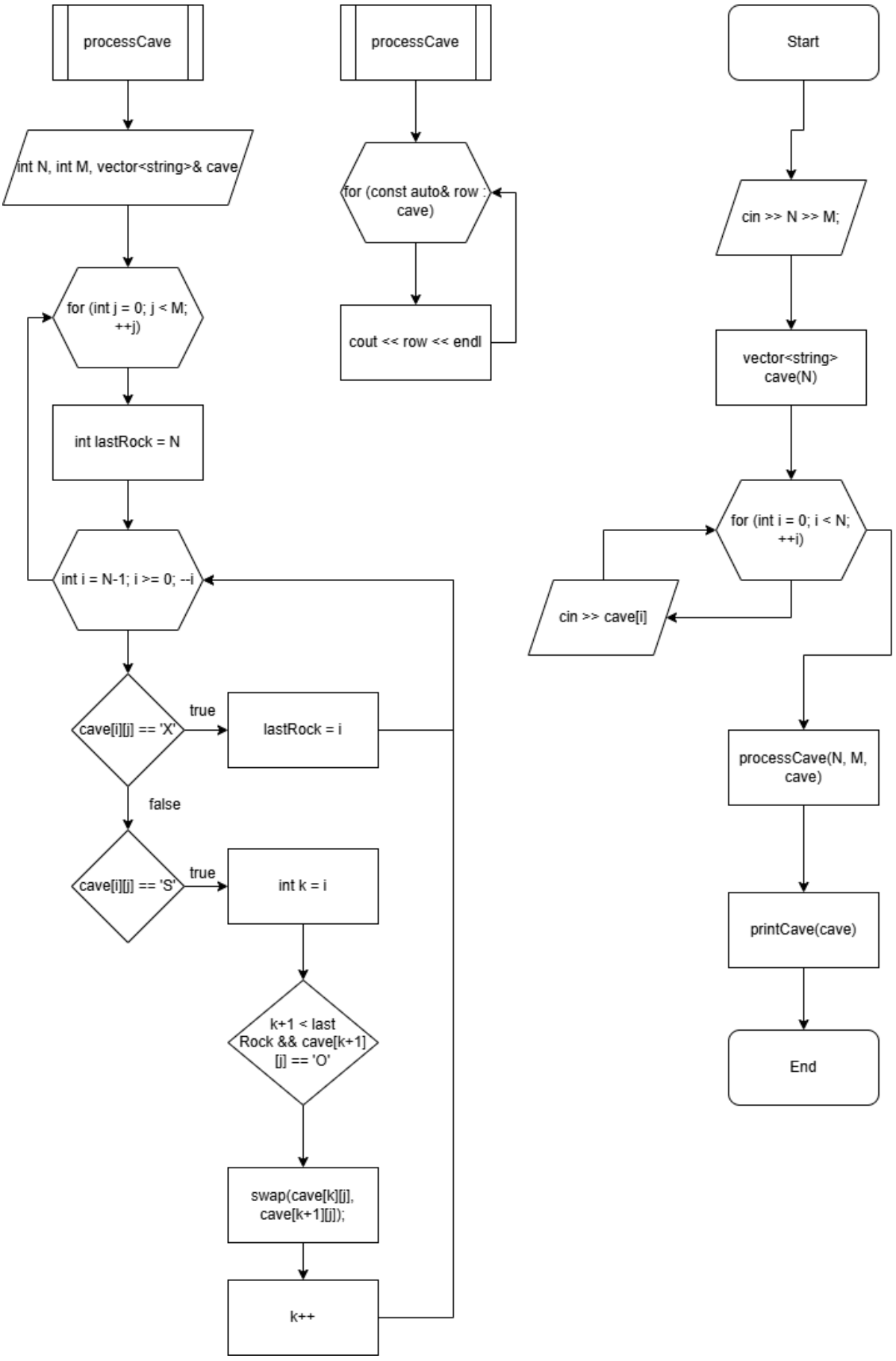
$1 \le N, M \le 10^3$

$1 \le x \le N$

$1 \le y \le M$

2. Дизайн та запланована оцінка часу виконання завдань:

Програма №_1 Algotester Lab 5



3. Код програм з посиланням на зовнішні ресурси:TASK № 1 VNS LAB 10

```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;
}

void addToHead(Node*& head, Node*& tail, const char* key) {
    Node* newNode = createNode(key);
    if (!head) {
        head = tail = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}

void addToTail(Node*& head, Node*& tail, const char* key) {
    Node* newNode = createNode(key);
    if (!tail) {
        head = tail = newNode;
    } else {
        newNode->prev = tail;
        tail->next = newNode;
        tail = newNode;
    }
}

void deleteByKey(Node*& head, Node*& tail, const char* key) {
    Node* current = head;
    while (current) {
```

```
        if (strcmp(current->key, key) == 0) {
            if (current->prev) current->prev->next = current->next;
            else head = current->next;
            if (current->next) current->next->prev = current->prev;
            else tail = current->prev;

            delete[] current->key;
            delete current;
            cout << "Element with key \"" << key << "\" deleted.\n";
            return;
        }
        current = current->next;
    }
    cout << "Element with key \"" << key << "\" not found.\n";
}

void printList(Node* head) {
    if (!head) {
        cout << "List is empty.\n";
        return;
    }
    Node* current = head;
    while (current) {
        cout << current->key << " ";
        current = current->next;
    }
    cout << endl;
}

void saveToFile(Node* head, const char* filename) {
    ofstream file(filename);
    if (!file) {
        cout << "Failed to open file for writing.\n";
        return;
    }
    Node* current = head;
    while (current) {
        file << current->key << endl;
        current = current->next;
    }
    file.close();
    cout << "List saved to file.\n";
}
```

TASK № 2 Algotester Lab 5v2

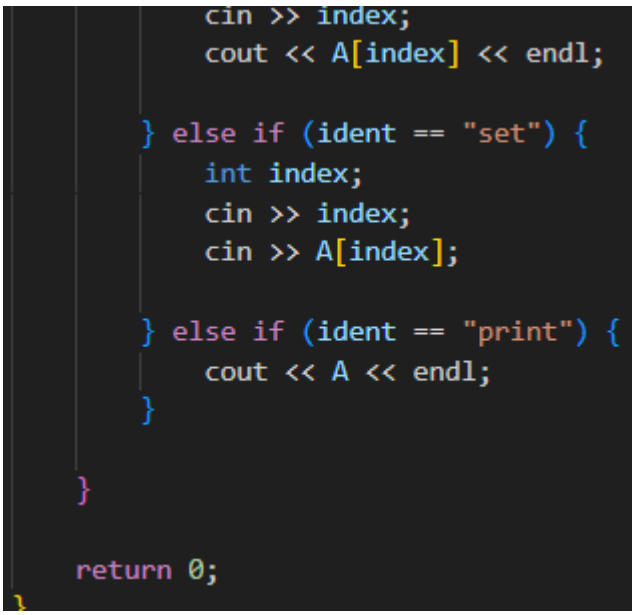
```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

void processCave(int N, int M, vector<string>& cave) {
    for (int j = 0; j < M; ++j) {
        int lastRock = N;
        for (int i = N-1; i >= 0; --i) {
            if (cave[i][j] == 'X') {
                lastRock = i;
            } else if (cave[i][j] == 'S') {
                int k = i;
                while (k+1 < lastRock && cave[k+1][j] == 'O') {
                    swap(cave[k][j], cave[k+1][j]);
                    ++k;
                }
            }
        }
    }
}

void printCave(const vector<string>& cave) {
    for (const auto& row : cave) {
        cout << row << endl;
    }
}

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

    processCave(N, M, cave);
    printCave(cave);

    return 0;
}
```

TASK№ 3 Algotester Lab 78v1 (100%)

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

using namespace std;

template<typename Type>
struct MyArray {

    int size = 0;
    int capacity = 1;
    Type *A;


    MyArray() {
        A = new Type[1];
    }


    void Insert(int index, int arr[], int N) {

        if (capacity > size + N) {

            for (int i = size - 1; i > index - 1; i--) {
                A[i + N] = A[i];
            }

            for (int i = 0; i <  N; ++i) {
                A[i+index]=arr[i];
            }

            size += N;

        } else {

            while (capacity <= size + N) {
                capacity *= 2;
            }

            Type *B = new Type[capacity];

            for (int i = 0; i < index; ++i) {
                B[i] = A[i];
            }

            for (int i = 0; i <  N; ++i) {
```

```
        for (int i = 0; i <  N; ++i) {
            B[i+index]=arr[i];
        }

        for (int i = index; i < size; ++i) {
            B[i + N] = A[i];
        }

        delete[] A;
        A = B;
        size += N;
    }
}


void Erase(int index, int n) {

    for (int i = index; i < size; ++i) {
        A[i] = A[i + n];
    }
    size -= n;
}


int Size() {
    return size;
}


int Capacity() {
    return capacity;
}


Type &operator[](int index) {
    return A[index];
}


friend ostream &operator<<(ostream &os, const MyArray &array) {
```

```
        for (int i = 0; i < array.size; ++i) {
            os << array.A[i] << ' ';
        }
        return os;
    }

};


int main() {

    MyArray<int> A;

    int Q;
    cin >> Q;

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

        if (ident == "insert") {
            int index, N;
            cin >> index >> N;

            int arr[N];
            for (int i = 0; i < N; ++i) {
                cin >> arr[i];
            }
            A.Insert(index, arr, N);

        } else if (ident == "erase") {
            int index, n;
            cin >> index >> n;
            A.Erase(index, n);

        } else if (ident == "size") {
            cout << A.Size() << endl;

        } else if (ident == "capacity") {
            cout << A.Capacity() << endl;

        } else if (ident == "get") {
            int index;
```



TASK№ 4 Class Practice Task 1.1

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

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

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

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

void append(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
}
```

```cpp
int main() {
    Node* head = nullptr;

    append(head, 1);
    append(head, 2);
    append(head, 3);
    append(head, 4);
    append(head, 5);

    cout << "Original: ";
    printList(head);

    head = reverse(head);

    cout << "Reversed: ";
    printList(head);

    return 0;
}
```

TASK№ 4 Class Practice Task 1.2

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

void append(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
}

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

bool compare(Node* h1, Node* h2) {
    while (h1 != nullptr && h2 != nullptr) {
        if (h1->data != h2->data) {
            return false;
        }
        h1 = h1->next;
        h2 = h2->next;
    }

    if (h1 != nullptr || h2 != nullptr) {
        return false;
    }

```

```
46      return true;
47  }
48
49  int main() {
50      Node* list1 = nullptr;
51      Node* list2 = nullptr;
52
53      append(list1, 1);
54      append(list1, 2);
55      append(list1, 3);
56
57      append(list2, 1);
58      append(list2, 2);
59      append(list2, 3);
60
61      cout << "List 1: ";
62      printList(list1);
63      cout << "List 2: ";
64      printList(list2);
65
66      if (compare(list1, list2)) {
67          cout << "Same." << endl;
68      } else {
69          cout << "Different." << endl;
70      }
71
72      append(list2, 4);
73
74      cout << "\nNew List 2: ";
75      printList(list2);
76
77      if (compare(list1, list2)) {
78          cout << "Same." << endl;
79      } else {
80          cout << "Different." << endl;
81      }
82
83      return 0;
84  }
```

TASK№ 4 Class Practice Task 1.3

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

void append(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
}

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

Node* add(Node* n1, Node* n2) {
    Node* result = nullptr;
    Node* tail = nullptr;
    int carry = 0;

    while (n1 != nullptr || n2 != nullptr || carry != 0) {
        int sum = carry;
        if (n1 != nullptr) {
            sum += n1->data;
            n1 = n1->next;
        }
        if (n2 != nullptr) {
            sum += n2->data;
```

```
            n2 = n2->next;
        }

        carry = sum / 10;
        int digit = sum % 10;

        Node* newNode = new Node(digit);
        if (!result) {
            result = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }

    return result;
}

int main() {
    Node* num1 = nullptr;
    append(num1, 9);
    append(num1, 7);
    append(num1, 3);

    Node* num2 = nullptr;
    append(num2, 6);
    append(num2, 5);
    append(num2, 8);

    cout << "First num: ";
    printList(num1);

    cout << "Second num: ";
    printList(num2);

    Node* result = add(num1, num2);

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

    return 0;
}
```

TASK№ 4 Class Practice Task 1.4

```cpp
#include <iostream>
using namespace std;

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;
}

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

void freeTree(TreeNode* root) {
    if (!root) {
        return;
    }
    freeTree(root->left);
    freeTree(root->right);
    delete root;
}

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 << "Normal tree: ";
    printTree(root);
    cout << endl;

    TreeNode* mirroredRoot = create_mirror_flip(root);

    cout << "Reversed tree: ";
    printTree(mirroredRoot);
    cout << endl;

    freeTree(root);
    freeTree(mirroredRoot);

    return 0;
}
```

TASK№ 4 Class Practice Task 1.5

```cpp
#include <iostream>
using namespace std;

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

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

int tree_sum(TreeNode* root) {
    if (!root) {
        return 0;
    }

    if (!root->left && !root->right) {
        return root->data;
    }

    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;
    }
    cout << root->data << " ";
    printTree(root->left);
    printTree(root->right);
}

void freeTree(TreeNode* root) {
    if (!root) {
        return;
    }
    freeTree(root->left);
    freeTree(root->right);
    delete root;
}
```

```
46
47  int main() {
48      TreeNode* root = new TreeNode(1);
49      root->left = new TreeNode(2);
50      root->right = new TreeNode(3);
51      root->left->left = new TreeNode(4);
52      root->left->right = new TreeNode(5);
53      root->right->left = new TreeNode(6);
54      root->right->right = new TreeNode(7);
55
56      cout << "Tree before calculating: ";
57      printTree(root);
58      cout << endl;
59
60      tree_sum(root);
61
62      cout << "After calculating: ";
63      printTree(root);
64      cout << endl;
65
66      freeTree(root);
67
68      return 0;
69  }
```

TASK№ 5 Self Practice Task

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

using namespace std;

const int directionsX[] = {-1, 1, 0, 0};
const int directionsY[] = {0, 0, -1, 1};

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

    startX--;
    startY--;

    vector<vector<int>> map(N, vector<int>(M, -1));

    queue<pair<int, int>> bfsQueue;
    bfsQueue.push({startX, startY});
    map[startX][startY] = 0;

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

        for (int int nextX = 0; direction < 4; direction++) {
            int nextX = currentX + directionsX[direction];
            int nextY = currentY + directionsY[direction];

            if (nextX >= 0 && nextX < N && nextY >= 0 && nextY < M && map[nextX][nextY] == -1) {
                map[nextX][nextY] = map[currentX][currentY] + 1;
                bfsQueue.push({nextX, nextY});
            }
        }
    }

    int maxHeight = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            maxHeight = max(maxHeight, map[i][j]);
        }
```

```
46      }
47
48      for (int i = 0; i < N; i++) {
49          for (int j = 0; j < M; j++) {
50              map[i][j] = maxHeight - map[i][j];
51          }
52      }
53
54      for (int i = 0; i < N; i++) {
55          for (int j = 0; j < M; j++) {
56              cout << map[i][j] << " ";
57          }
58          cout << endl;
59      }
60
61      return 0;
62  }
```

4. Результати виконання завдань, тестування та фактично витрачений час:

TASK № 1 VNS LAB 10 Variant 20

```
1. Add to head
2. Add to tail
3. Delete by key
4. Print list
5. Save to file
6. Load from file
7. Clear list
0. Exit
Your choice: 1
Enter key: 1 2 3 4 5

1. Add to head
2. Add to tail
3. Delete by key
4. Print list
5. Save to file
6. Load from file
7. Clear list
0. Exit
Your choice: 4
1 2 3 4 5

1. Add to head
2. Add to tail
3. Delete by key
4. Print list
5. Save to file
6. Load from file
7. Clear list
0. Exit
Your choice: 3
Enter key to delete: 5 4
Element with key "5 4" not found.

1. Add to head
2. Add to tail
3. Delete by key
4. Print list
5. Save to file
6. Load from file
```

```
7. Clear list
0. Exit
Your choice: 4
1 2 3 4 5

1. Add to head
2. Add to tail
3. Delete by key
4. Print list
5. Save to file
6. Load from file
7. Clear list
0. Exit
Your choice: 0
List cleared.
Program terminated.
```

TASK № 2 Algotester Lab 5v2

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

Compiler	Result	Time (sec.)	Memory (MiB)	Actions
C++ 23	Accepted	0.025	1.938	View

TASK№ 3 Algotester Lab 78v1 (100%)

```
5

insert
0 3
1 2 3

erase
0 2

set
0 10

size
1

print
10
```

Compiler	Result	Time (sec.)	Memory (MiB)	Actions
C++ 23	Accepted	0.007	1.430	View

TASK№ 4 Class Practice Task (1-5)

1.1

```
Original: 1 -> 2 -> 3 -> 4 -> 5 -> nullptr
Reversed: 5 -> 4 -> 3 -> 2 -> 1 -> nullptr
```

1.2

```
List 1: 1 -> 2 -> 3 -> nullptr
List 2: 1 -> 2 -> 3 -> nullptr
Same.

New List 2: 1 -> 2 -> 3 -> 4 -> nullptr
Different.
```

1.3

```
First num: 9 -> 7 -> 3 -> nullptr
Second num: 6 -> 5 -> 8 -> nullptr
Result: 5 -> 3 -> 2 -> 1 -> nullptr
```

1.4

```
Normal tree: 1 2 4 5 3 6 7
Reversed tree: 1 3 7 6 2 5 4
```

1.5

```
Tree before calculating: 1 2 4 5 3 6 7
After calculating: 22 9 4 5 13 6 7
```

TASK№ 5 Self Practice Task

```
3 4
2 2
1 2 1 0
2 3 2 1
1 2 1 0
```

Compiler	Result	Time (sec.)	Memory (MiB)	Actions
C++ 23	Accepted	0.118	6.746	View

Висновки:

Отримавши практичний досвід у використанні динамічних структур даних, я реалізував зв’язний список та впровадив алгоритми обробки дерев. Отримані навички вдосконалили моє розуміння алгоритмів і їхню практичну реалізацію в програмах. Процес вирішення завдань сприяв розвитку логічного мислення та допоміг оптимізувати код. Різноманітні практичні задачі, такі як обчислення суми великих чисел за допомогою зв’язних списків, робота з алгоритмами обробки дерев, дозволили застосовувати теоретичні знання у реальних сценаріях програмування.