

?

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).

о 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.

о 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).

Сформуванати двонаправлений список. Знищити елемент із заданим ключем. Додати по K 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 - i -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 "Двохв'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 написано 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 елемента**
Ідентифікатор - *get*
Ви отримуєте ціле число - *index*, індекс елемента.
Ви виводите значення елемента за індексом.
- **Модифікація значення i -го елемента**
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. $379 \Rightarrow 9 \rightarrow 7 \rightarrow 3$);
- 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.

Клітинки 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

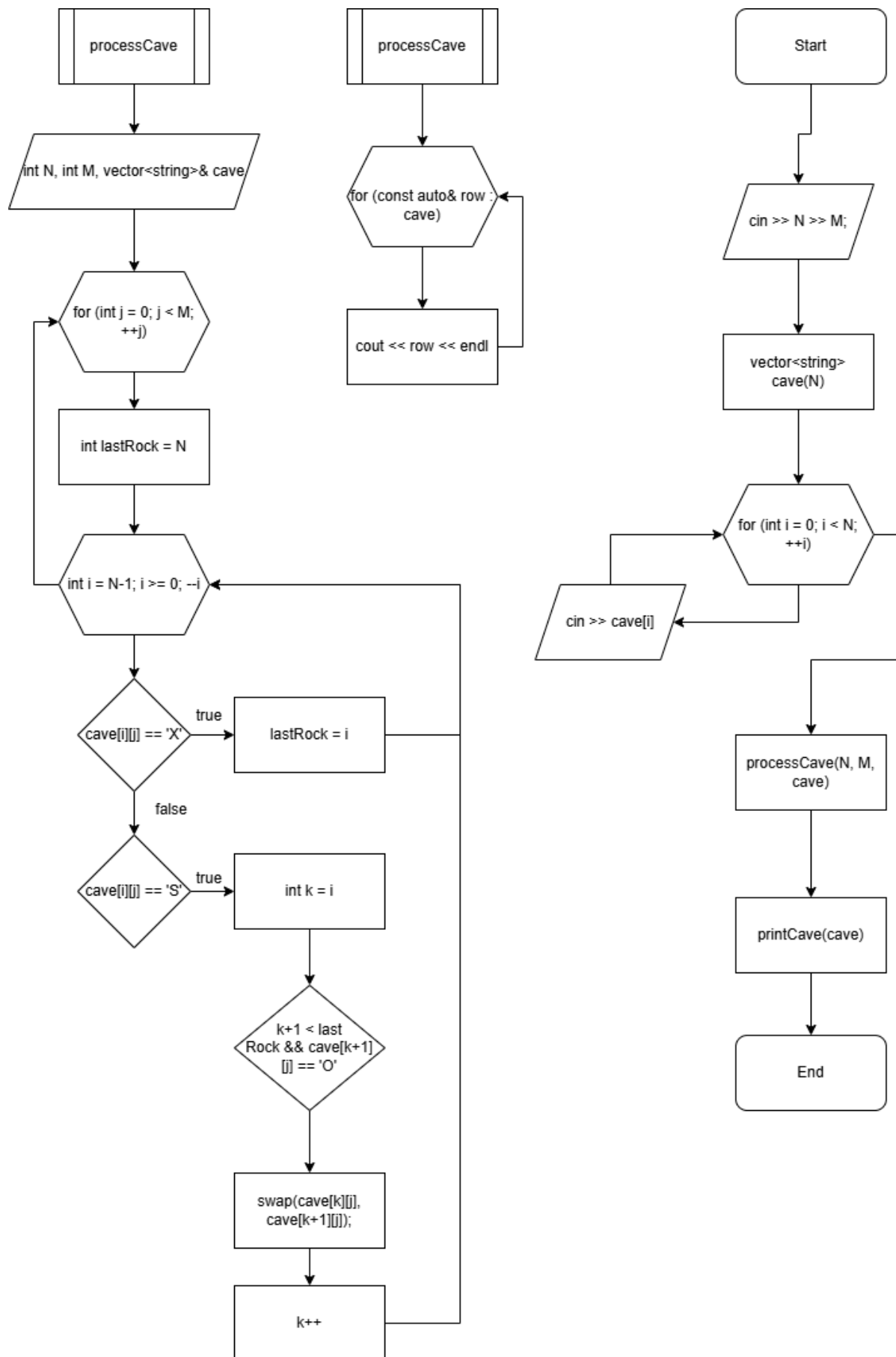
$$1 \leq N, M \leq 10^3$$

$$1 \leq x \leq N$$

$$1 \leq y \leq M$$

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

```

#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";
}

```



```

#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] == '0') {
                    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%)

```

1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  template<typename Type>
6  class DoublyLinkedList {
7      struct Node {
8          Type data;
9          Node* prev;
10         Node* next;
11         Node(Type value) : data(value), prev(nullptr), next(nullptr) {}
12     };
13
14     Node* head;
15     Node* tail;
16     int size;
17
18 public:
19     DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}
20
21     ~DoublyLinkedList() {
22         while (head) {
23             Node* temp = head;
24             head = head->next;
25             delete temp;
26         }
27     }
28
29     void Insert(int index, Type arr[], int N) {
30         if (index < 0 || index > size) return;
31         Node* current = head;
32         for (int i = 0; i < index; ++i) {
33             current = current->next;
34         }
35         for (int i = 0; i < N; ++i) {
36             Node* newNode = new Node(arr[i]);
37             if (!head) {
38                 head = tail = newNode;
39             } else if (!current) {
40                 tail->next = newNode;
41                 newNode->prev = tail;
42                 tail = newNode;
43             } else {
44                 newNode->next = current;
45                 newNode->prev = current->prev;

```

```

45         newNode->prev = current->prev;
46         if (current->prev) {
47             current->prev->next = newNode;
48         } else {
49             head = newNode;
50         }
51         current->prev = newNode;
52     }
53 }
54 size += N;
55 }
56
57 void Erase(int index, int n) {
58     if (index < 0 || index >= size || n <= 0) return;
59     Node* current = head;
60     for (int i = 0; i < index; ++i) {
61         current = current->next;
62     }
63     for (int i = 0; i < n && current; ++i) {
64         Node* toDelete = current;
65         current = current->next;
66         if (toDelete->prev) {
67             toDelete->prev->next = toDelete->next;
68         } else {
69             head = toDelete->next;
70         }
71         if (toDelete->next) {
72             toDelete->next->prev = toDelete->prev;
73         } else {
74             tail = toDelete->prev;
75         }
76         delete toDelete;
77     }
78     size -= n;
79 }
80
81 int Size() const {
82     return size;
83 }
84
85 Type Get(int index) const {

```

```

85     Type Get(int index) const {
86         if (index < 0 || index >= size) throw out_of_range("Index out of range");
87         Node* current = head;
88         for (int i = 0; i < index; ++i) {
89             current = current->next;
90         }
91         return current->data;
92     }
93
94     void Set(int index, Type value) {
95         if (index < 0 || index >= size) throw out_of_range("Index out of range");
96         Node* current = head;
97         for (int i = 0; i < index; ++i) {
98             current = current->next;
99         }
100         current->data = value;
101     }
102
103     friend ostream& operator<<(ostream& os, const DoublyLinkedList& list) {
104         Node* current = list.head;
105         while (current) {
106             os << current->data << ' ';
107             current = current->next;
108         }
109         return os;
110     }
111 };
112
113 int main() {
114     DoublyLinkedList<int> list;
115
116     int Q;
117     cin >> Q;
118
119     char ident[10];
120
121     while (Q-- > 0) {
122         cin >> ident;

```

```

124     if (strcmp(ident, "insert") == 0) {
125         int index, N;
126         cin >> index >> N;
127         int* arr = new int[N];
128         for (int i = 0; i < N; ++i) {
129             cin >> arr[i];
130         }
131         list.Insert(index, arr, N);
132         delete[] arr;
133
134     } else if (strcmp(ident, "erase") == 0) {
135         int index, n;
136         cin >> index >> n;
137         list.Erase(index, n);
138
139     } else if (strcmp(ident, "size") == 0) {
140         cout << list.Size() << endl;
141
142     } else if (strcmp(ident, "get") == 0) {
143         int index;
144         cin >> index;
145         try {
146             cout << list.Get(index) << endl;
147         } catch (const out_of_range& e) {
148             cout << e.what() << endl;
149         }
150
151     } else if (strcmp(ident, "set") == 0) {
152         int index, value;
153         cin >> index >> value;
154         try {
155             list.Set(index, value);
156         } catch (const out_of_range& e) {
157             cout << e.what() << endl;
158         }
159
160     } else if (strcmp(ident, "print") == 0) {
161         cout << list << endl;
162     }
163 }
164
165 return 0;
166 }

```

TASK№ 4 Class Practice Task 1.1

```
1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7
8      Node(int value) : data(value), next(nullptr) {}
9  };
10
11 void printList(Node* head) {
12     Node* current = head;
13     while (current != nullptr) {
14         cout << current->data << " -> ";
15         current = current->next;
16     }
17     cout << "nullptr" << endl;
18 }
19
20 Node* reverse(Node* head) {
21     Node* prev = nullptr;
22     Node* current = head;
23     Node* next = nullptr;
24
25     while (current != nullptr) {
26         next = current->next;
27         current->next = prev;
28         prev = current;
29         current = next;
30     }
31     return prev;
32 }
33
34 void append(Node*& head, int value) {
35     Node* newNode = new Node(value);
36     if (!head) {
37         head = newNode;
38         return;
39     }
40     Node* temp = head;
41     while (temp->next) {
42         temp = temp->next;
43     }
44     temp->next = newNode;
45 }
```

```

47  int main() {
48      Node* head = nullptr;
49
50      append(head, 1);
51      append(head, 2);
52      append(head, 3);
53      append(head, 4);
54      append(head, 5);
55
56      cout << "Original: ";
57      printList(head);
58
59      head = reverse(head);
60
61      cout << "Reversed: ";
62      printList(head);
63
64      return 0;
65  }

```

TASK № 4 Class Practice Task 1.2

```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7
8      Node(int value) : data(value), next(nullptr) {}
9  };
10
11 void append(Node*& head, int value) {
12     Node* newNode = new Node(value);
13     if (!head) {
14         head = newNode;
15         return;
16     }
17     Node* temp = head;
18     while (temp->next) {
19         temp = temp->next;
20     }
21     temp->next = newNode;
22 }
23
24 void printList(Node* head) {
25     Node* current = head;
26     while (current != nullptr) {
27         cout << current->data << " -> ";
28         current = current->next;
29     }
30     cout << "nullptr" << endl;
31 }
32
33 bool compare(Node* h1, Node* h2) {
34     while (h1 != nullptr && h2 != nullptr) {
35         if (h1->data != h2->data) {
36             return false;
37         }
38         h1 = h1->next;
39         h2 = h2->next;
40     }
41
42     if (h1 != nullptr || h2 != nullptr) {
43         return false;
44     }
45

```



```

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

```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7
8      Node(int value) : data(value), next(nullptr) {}
9  };
10
11 void append(Node*& head, int value) {
12     Node* newNode = new Node(value);
13     if (!head) {
14         head = newNode;
15         return;
16     }
17     Node* temp = head;
18     while (temp->next) {
19         temp = temp->next;
20     }
21     temp->next = newNode;
22 }
23
24 void printList(Node* head) {
25     Node* current = head;
26     while (current != nullptr) {
27         cout << current->data << " -> ";
28         current = current->next;
29     }
30     cout << "nullptr" << endl;
31 }
32
33 Node* add(Node* n1, Node* n2) {
34     Node* result = nullptr;
35     Node* tail = nullptr;
36     int carry = 0;
37
38     while (n1 != nullptr || n2 != nullptr || carry != 0) {
39         int sum = carry;
40         if (n1 != nullptr) {
41             sum += n1->data;
42             n1 = n1->next;
43         }
44         if (n2 != nullptr) {
45             sum += n2->data;

```

```

46         n2 = n2->next;
47     }
48
49     carry = sum / 10;
50     int digit = sum % 10;
51
52     Node* newNode = new Node(digit);
53     if (!result) {
54         result = newNode;
55         tail = newNode;
56     } else {
57         tail->next = newNode;
58         tail = newNode;
59     }
60 }
61
62 return result;
63 }
64
65 int main() {
66     Node* num1 = nullptr;
67     append(num1, 9);
68     append(num1, 7);
69     append(num1, 3);
70
71     Node* num2 = nullptr;
72     append(num2, 6);
73     append(num2, 5);
74     append(num2, 8);
75
76     cout << "First num: ";
77     printList(num1);
78
79     cout << "Second num: ";
80     printList(num2);
81
82     Node* result = add(num1, num2);
83
84     cout << "Result: ";
85     printList(result);
86
87     return 0;
88 }

```

TASK№ 4 Class Practice Task 1.4

```

1  #include <iostream>
2  using namespace std;
3
4  struct TreeNode {
5      int data;
6      TreeNode* left;
7      TreeNode* right;
8
9      TreeNode(int value) : data(value), left(nullptr), right(nullptr) {}
10 };
11
12 TreeNode* create_mirror_flip(TreeNode* root) {
13     if (!root) {
14         return nullptr;
15     }
16
17     TreeNode* newRoot = new TreeNode(root->data);
18     newRoot->left = create_mirror_flip(root->right);
19     newRoot->right = create_mirror_flip(root->left);
20
21     return newRoot;
22 }
23
24 void printTree(TreeNode* root) {
25     if (!root) {
26         return;
27     }
28     cout << root->data << " ";
29     printTree(root->left);
30     printTree(root->right);
31 }
32
33 void freeTree(TreeNode* root) {
34     if (!root) {
35         return;
36     }
37     freeTree(root->left);
38     freeTree(root->right);
39     delete root;
40 }
41
42 int main() {
43     TreeNode* root = new TreeNode(1);
44     root->left = new TreeNode(2);
45     root->right = new TreeNode(3);

```

```

46     root->left->left = new TreeNode(4);
47     root->left->right = new TreeNode(5);
48     root->right->left = new TreeNode(6);
49     root->right->right = new TreeNode(7);
50
51     cout << "Normal tree: ";
52     printTree(root);
53     cout << endl;
54
55     TreeNode* mirroredRoot = create_mirror_flip(root);
56
57     cout << "Reversed tree: ";
58     printTree(mirroredRoot);
59     cout << endl;
60
61     freeTree(root);
62     freeTree(mirroredRoot);
63
64     return 0;
65 }
66

```

TASK № 4 Class Practice Task 1.5

```
1  #include <iostream>
2  using namespace std;
3
4  struct TreeNode {
5      int data;
6      TreeNode* left;
7      TreeNode* right;
8
9      TreeNode(int value) : data(value), left(nullptr), right(nullptr) {}
10 };
11
12 int tree_sum(TreeNode* root) {
13     if (!root) {
14         return 0;
15     }
16
17     if (!root->left && !root->right) {
18         return root->data;
19     }
20
21     int leftSum = tree_sum(root->left);
22     int rightSum = tree_sum(root->right);
23
24     root->data = leftSum + rightSum;
25
26     return root->data;
27 }
28
29 void printTree(TreeNode* root) {
30     if (!root) {
31         return;
32     }
33     cout << root->data << " ";
34     printTree(root->left);
35     printTree(root->right);
36 }
37
38 void freeTree(TreeNode* root) {
39     if (!root) {
40         return;
41     }
42     freeTree(root->left);
43     freeTree(root->right);
44     delete root;
45 }
```

```

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

```

1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  #include <algorithm>
5
6  using namespace std;
7
8  const int directionsX[] = {-1, 1, 0, 0};
9  const int directionsY[] = {0, 0, -1, 1};
10
11 int main() {
12     int N, M;
13     cin >> N >> M;
14     int startX, startY;
15     cin >> startX >> startY;
16
17     startX--;
18     startY--;
19
20     vector<vector<int>> map(N, vector<int>(M, -1));
21
22     queue<pair<int, int>> bfsQueue;
23     bfsQueue.push({startX, startY});
24     map[startX][startY] = 0;
25
26     while (!bfsQueue.empty()) {
27         auto [currentX, currentY] = bfsQueue.front();
28         bfsQueue.pop();
29
30         for (int nextX = 0; direction < 4; direction++) {
31             int nextX = currentX + directionsX[direction];
32             int nextY = currentY + directionsY[direction];
33
34             if (nextX >= 0 && nextX < N && nextY >= 0 && nextY < M && map[nextX][nextY] == -1) {
35                 map[nextX][nextY] = map[currentX][currentY] + 1;
36                 bfsQueue.push({nextX, nextY});
37             }
38         }
39     }
40
41     int maxHeight = 0;
42     for (int i = 0; i < N; i++) {
43         for (int j = 0; j < M; j++) {
44             maxHeight = max(maxHeight, map[i][j]);
45         }
46     }
47 }

```



```

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
00000
S00XX
0000S
00S00
00000
000SS
000XX
S0000
SSS0S

```

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.