Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

про виконання

**Лабораторних та практичних робіт №10**

***з дисципліни:*** «Мови та парадигми програмування»

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

***Виконала:***

студентка групи ШІ-14

Павлюх Вікторія Ігорівна

# **Тема роботи:**

Динамічні структури даних та алгоритми їх обробки. Однозв’язний та двозв’язний список. Стек, черга та дерево.

# **Мета роботи:**

Ознайомитись з динамічними структурами даних та попрацювати з базовими алгоритмами їх обробки. Реалізувати однозв’язний та двозв’язний списки та створити функції для їх обробки. Ознайомитися з деревами та реалізувати бінарне дерево. Ознайомитися з стеком та чергою

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Стек. Черга.
* Тема №2: Однозв’язний та двозв’язний списки.
* Тема №3: Дерева. Бінарне дерево.
* Тема №4: Класи. Шаблони класів.
* Тема №5: Алгоритми обробки динамічних структур даних.

1. Індивідуальний план опрацювання теорії:

* Тема №1: Стек. Черга.
  + Джерела Інформації
    - Стаття.

<https://acode.com.ua/urok-111-stek-i-kupa/>

<https://studfile.net/preview/3908535/>

* + Що опрацьовано:
    - Стек і черга як динамічні структури даних, різниця між ними
    - Двобічна та пріоритетна черга
  + Статус: Ознайомлена
  + Витрачений час: тиждень
* Тема №2: Однозв’язний та двозв’язний списки.
  + Джерела Інформації:
    - Відео.

<https://youtu.be/HKfj0l7ndbc?feature=shared>

<https://youtu.be/RNMIDj62o_o?feature=shared>

<https://youtu.be/-StYr9wILqo?feature=shared>

<https://youtu.be/3mRfWluedHo?feature=shared>

* + Що опрацьовано:
    - Зв’язані списки. Різниця між однозв’язним та двозв’язними списками
  + Статус: Ознайомлена
  + Витрачений час: тиждень
* Тема №3: Дерева. Бінарне дерево.
  + Джерела Інформації:
    - Відео.

<https://youtu.be/UHxtjVsOTHc?feature=shared>

<https://youtu.be/_IhTp8q0Mm0?feature=shared>

* + - Стаття.
  + Що опрацьовано:
    - Дерева як структура даних
    - Бінарні дерева
    - Обходи дерева
  + Статус: Ознайомлена
  + Витрачений час: тиждень
* Тема №4 Класи. Шаблони класів.
  + Джерела Інформації:
    - Відео.
    - Стаття.

<https://acode.com.ua/uroki-po-cpp/>

* + Що опрацьовано:
    - Класи. Шаблони класів
    - Специфікатори доступу (private, public)
    - Конструктори та деструктори
    - Дружні функції
  + Статус: Ознайомлена
  + Витрачений час: тиждень
* Тема №5 Алгоритми обробки динамічних структур даних.
  + Джерела Інформації:
    - Відео.
    - Стаття.

<https://www.bestprog.net/uk/2022/02/16/c-linear-doubly-linked-bidirectional-list-general-concepts-ua/>

<https://www.bestprog.net/uk/2022/02/13/c-an-example-of-the-implementation-of-a-linear-singly-linked-list-ua/>

* + Що опрацьовано:
    - Алгоритми обробки для динамічних структур даних (вставка елемента у чергу, списки, дерево, стек, видалення елемента…)
    - Бінарний пошук
  + Статус: Ознайомлена
  + Витрачений час: тиждень

# **Виконання роботи:**

## **1. Опрацювання завдання та вимог до програм та середовища.**

**2. Дизайн та планована оцінка часу виконання завдань.**

## **3. Код програм з посиланням на зовнішні ресурси:**

## **4. Результати виконання завдань, тестування.**

Завдання №1 VNS Lab 10

* Варіант 21
* Деталі завдання: Записи в лінійному списку містять ключове поле типу \*char (рядок символів). Сформувати двонаправлений список. Знищити елементи перед і після елемента із заданим ключем. Додати по К елементів у початок й у кінець списку.
* Планований час на реалізацію: 5 год
* Блок-схема: не робила
* Код програми:

*#include* <iostream>

*#include* <fstream>

*#include* <cstring>

using namespace std;

typedef struct node

{

node \*previous;

char \*data;

node \*next;

} *node*;

typedef struct List

{

*node* \*head;

*node* \*tail;

} *List*;

*node* \*Create(const char \**value*)

{

*// Creates the head node of a doubly linked list*

*node* \*ptr = new *node*;

ptr->previous = **nullptr**;

ptr->data = strdup(*value*);

ptr->next = **nullptr**;

return ptr;

}

*node* \*InsertAtFront(*node* \**head*, const char \**value*)

{

*// Inserts a new node at the beginning of a doubly linked list*

if (*head* == **nullptr**)

{

return Create(*value*);

}

*node* \*ptr = new *node*;

ptr->previous = **nullptr**;

ptr->data = strdup(*value*);

ptr->next = *head*;

*head*->previous = ptr;

*head* = ptr;

return *head*;

}

*node* \*InsertAtEnd(*node* \**head*, const char \**value*)

{

*// Inserts a new node at the end of a doubly linked list.*

if (*head* == **nullptr**)

{

return Create(*value*);

}

*node* \*ptr1 = *head*;

*node* \*ptr2 = new *node*;

ptr2->previous = **nullptr**;

ptr2->data = strdup(*value*);

ptr2->next = **nullptr**;

while (ptr1->next != **nullptr**)

ptr1 = ptr1->next;

ptr1->next = ptr2;

ptr2->previous = ptr1;

return *head*;

}

*node* \*DeleteNodeAtPosition(*node* \**head*, int *position*)

{

if (*head* == **nullptr**)

{

cout << "The list is empty. Cannot delete from an empty list.\n";

return *head*;

}

if (*position* <= 0)

{

cout << "Invalid position. Position should be a positive integer.\n";

return *head*;

}

*node* \*current = *head*;

if (*position* == 1)

{

*head* = *head*->next;

if (*head* != **nullptr**)

*head*->previous = **nullptr**;

free(current->data); *// Free the memory for data*

delete current;

return *head*;

}

int currentPosition = 1;

while (currentPosition < *position* && current != **nullptr**)

{

current = current->next;

currentPosition++;

}

if (current == **nullptr**)

{

cout << "Position exceeds the length of the list.\n";

return *head*;

}

if (current->next != **nullptr**)

current->next->previous = current->previous;

if (current->previous != **nullptr**)

current->previous->next = current->next;

free(current->data); *// Free the memory for data*

delete current;

return *head*;

}

void Print(*node* \**head*)

{

*// Prints elements of a doubly linked list from head to tail*

if (*head* == **nullptr**)

{

cout << "Empty list\n";

return;

}

*node* \*ptr = *head*;

cout << "The linked list: ";

while (ptr != **nullptr**)

{

cout << ptr->data << " ";

ptr = ptr->next;

}

cout << endl;

}

void Fillinfile(*node* \**head*, const char \**filename*)

{

*ofstream* fout(*filename*);

if (!fout.is\_open())

{

cerr << "Error opening file" << endl;

return;

}

*node* \*traverser = *head*;

while (traverser != **nullptr**)

{

fout << traverser->data << " ";

traverser = traverser->next;

}

fout.close();

}

*node*\* PurifyList(*node* \**head*)

{

*node*\* temp = new *node*();

while (*head* != **nullptr**) {

temp = *head*;

*head* = *head*->next;

delete temp;

}

delete *head*;

return **nullptr**;

}

*node* \*OutputFile(const char \**filename*)

{

*ifstream* fin(*filename*);

if (!fin.is\_open())

{

cerr << "Error opening file" << endl;

return **nullptr**;

}

*node* \*head = **nullptr**;

*string* content;

while (fin >> content)

{

head = InsertAtEnd(head, content.c\_str());

}

fin.close();

return head;

}

int main()

{

const char \*a = "1";

const char \*b = "2";

const char \*c = "3";

const char \*d = "4";

const char \*e = "5";

const char \*f = "6";

*node* \*head = Create(a);

head = InsertAtFront(head, c);

head = InsertAtFront(head, b);

head = InsertAtFront(head, a);

head = InsertAtEnd(head, d);

head = InsertAtEnd(head, e);

head = InsertAtEnd(head, f);

Print(head);

head = DeleteNodeAtPosition(head, 3);

Print(head);

Fillinfile(head, "File.txt");

head = PurifyList(head);

Print(head);

head = OutputFile("File.txt");

Print(head);

return 0;

}

* Результат:

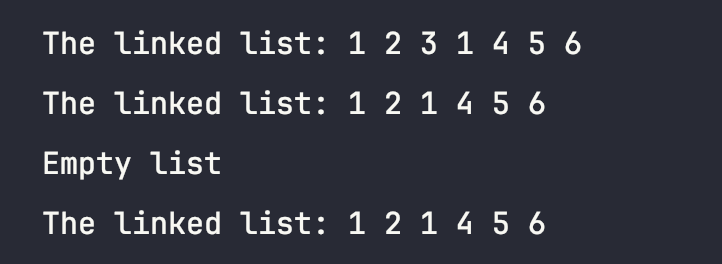


Рис.1.Результат тестування VNS Lab 10.

Завдання №2 Algotester Lab 5

* Варіант 2
* Деталі завдання:В пустелі існує незвичайна печера, яка є двовимірною. Її висота це N, ширина - M Всередині печери є пустота, пісок та каміння. Пустота позначається буквою О, 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.

* Планований час на реалізацію: 4 год
* Блок-схема:

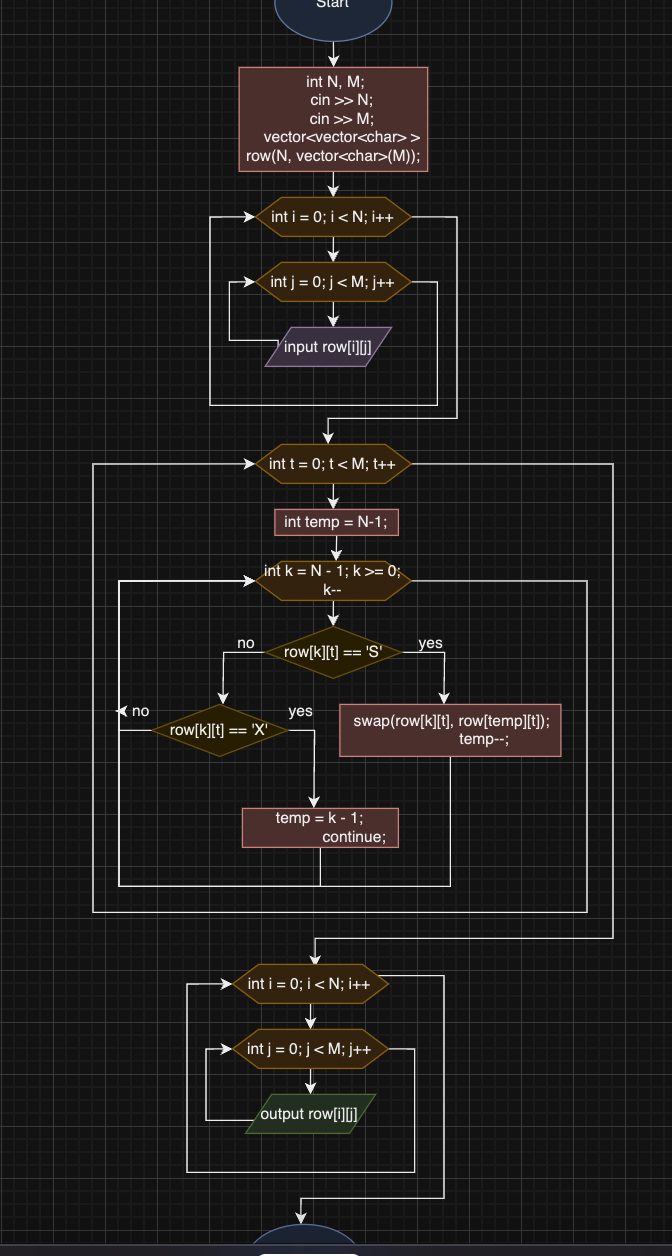


Рис.2. Блок-схема до Algotester Lab 5.

* Код програми:

*#include* <iostream>

*#include* <vector>

using namespace std;

int main() {

int N, M;

cin >> N;

cin >> M;

vector<vector<char> > row(N, vector<char>(M));

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

for (int j = 0; j < M; j++) {

cin >> row[i][j];

}

}

for (int t = 0; t < M; t++) {

int temp = N-1;

for (int k = N - 1; k >= 0; k--) {

if (row[k][t] == 'S') {

swap(row[k][t], row[temp][t]);

temp--;

} else if (row[k][t] == 'X') {

temp = k - 1;

continue;

}

}

}

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

for (int j = 0; j < M; j++) {

cout << row[i][j];

}

cout << endl;

}

return 0;

}

* Результат:

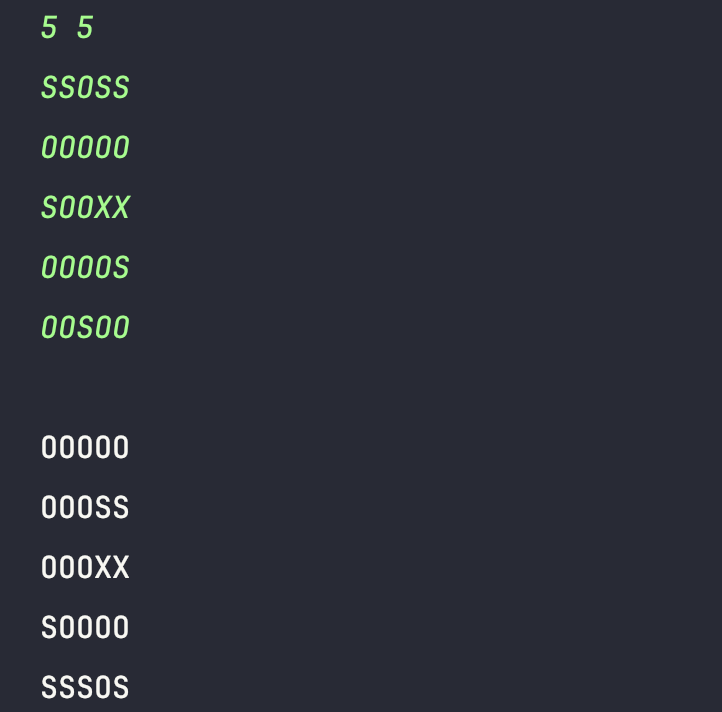


Рис.3.Результат тестування Algotester Lab 5.

Завдання №3 Algotester Lab 7-8

* Варіант 2
* Деталі завдання:

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](https://en.wikipedia.org/wiki/Dynamic_array#Growth_factor)) рівний 2.

**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 динамічного масиву на екран**Ідентифікатор - print  
Ви не отримуєте аргументів.  
Ви виводите усі елементи динамічного масиву через пробіл.  
Реалізувати використовуючи перегрузку оператора

* Планований час на реалізацію: 6 год
* Блок-схема: не робила
* Код програми:

*#include* <iostream>

using namespace std;

template <typename *T*>

class DynamicArray {

private:

*T* \*array;

int length;

int currentCapacity;

void resize(int *newCapacity*) {

*T* \*tempArray = new *T*[*newCapacity*];

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

tempArray[i] = array[i];

}

delete[] array;

array = tempArray;

currentCapacity = *newCapacity*;

}

public:

DynamicArray() : array(**nullptr**), length(0), currentCapacity(1) {

array = new *T*[currentCapacity];

}

~DynamicArray() {

delete[] array;

}

void insert(int *index*, int *N*, *T*\* *elements*) {

if (length + *N* >= currentCapacity) {

while (length + *N* >= currentCapacity) {

currentCapacity \*= 2;

}

resize(currentCapacity);

}

for (int i = length - 1; i >= *index*; --i) {

array[i + *N*] = array[i];

}

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

array[*index* + i] = *elements*[i];

}

length += *N*;

}

void erase(int *index*, int *n*) {

for (int i = *index*; i < length - *n*; ++i) {

array[i] = array[i + *n*];

}

length -= *n*;

}

*T* getElement(int *index*) {

return array[*index*];

}

void set(int *index*, *T value*) {

array[*index*] = *value*;

}

int getLength() {

return length;

}

int capacity() {

return currentCapacity;

}

void print() {

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

cout << array[i] << " ";

}

cout <<endl;

}

};

int main() {

int Q;

cin >> Q;

DynamicArray<int> dynamicArray;

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

*string* query;

cin >> query;

if (query == "insert") {

int index, N;

cin >> index >> N;

int \*elements = new int[N];

for (int j = 0; j < N; ++j) {

cin >> elements[j];

}

dynamicArray.insert(index, N, elements);

delete[] elements;

} else if (query == "size") {

cout << dynamicArray.getLength() << endl;

} else if (query == "erase") {

int index, n;

cin >> index >> n;

dynamicArray.erase(index, n);

} else if (query == "capacity") {

cout << dynamicArray.capacity() << endl;

} else if (query == "get") {

int index;

cin >> index;

cout << dynamicArray.getElement(index) << endl;

} else if (query == "set") {

int index, value;

cin >> index >> value;

dynamicArray.set(index, value);

} else if (query == "print") {

dynamicArray.print();

}

}

return 0;

}

* Результат:

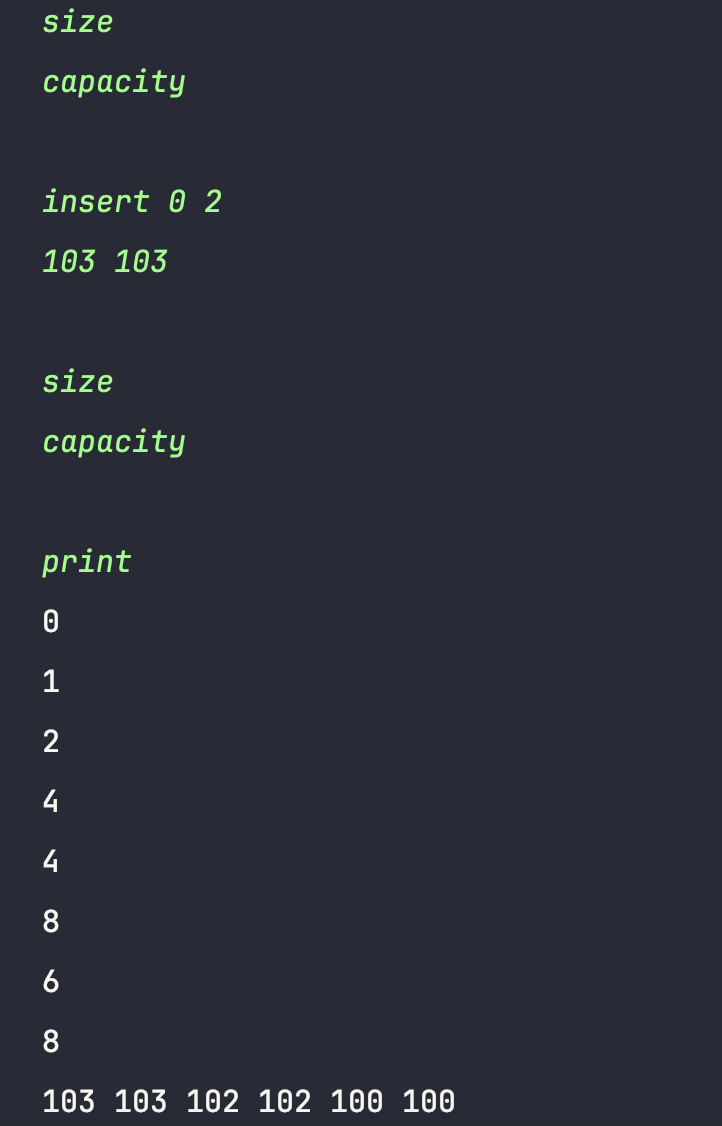


Рис.4.Результат тестування Algotester Lab 7-8.

Завдання №4 Class Practice Wor

* Без варіанту
* Деталі завдання

Задача №1 - Реверс списку (Reverse list). Задача №2 - Порівняння списків. Задача №3 – Додавання великих чисел. Задача №4 - Віддзеркалення дерева. Задача №5 - Записати кожному батьківському вузлу суму підвузлів

* Планований час на реалізацію: 5 год
* Блок-схема: не робила
* Код програми:

завдання 1:

*#include*<iostream>

using namespace std;

struct node

{

int data;

node\* next;

};

void show(node\* *head*){

node\* current = *head*;

while(current != **nullptr**){

cout << current->data << " ";

current = current->next;

}

}

node\* reverse(node\* *head*) {

node \*prev = **nullptr**, \*current = *head*, \*nextNode = **nullptr**;

while (current != **nullptr**) {

nextNode = current->next;

current->next = prev;

prev = current;

current = nextNode;

}

return prev;

}

int main(){

node\* head = new node;

head->data = 0;

node\* second = new node;

second->data = 2;

head->next = second;

node\* third = new node;

head->data = 3;

second->next = third;

third->next = **nullptr**;

show(head);

head = reverse(head);

cout << endl;

show(head);

return 0;

}

завдання 2:

*#include*<iostream>

using namespace std;

struct node

{

int data;

node\* next;

};

void show(node\* *head*){

node\* curent = *head*;

while(curent != **nullptr**){

cout << curent->data << " ";

curent = curent->next;

}

}

bool compare(node\* *head*, node\* *head2*) {

while (*head* != **nullptr** && *head2* != **nullptr**) {

if (*head*->data != *head2*->data) {

return false;

}

*head* = *head*->next;

*head2* = *head2*->next;

}

if (*head* != **nullptr** || *head2* != **nullptr**) {

return false;

}

return true;

}

int main(){

node\* head = new node;

head->data = 0;

node\* second = new node;

second->data = 2;

head->next = second;

node\* third = new node;

head->data = 3;

second->next = third;

third->next = **nullptr**;

node\* head2 = new node;

head2->data = 8;

node\* second2 = new node;

second2->data = 9;

head2->next = second2;

node\* third2 = new node;

head2->data = 10;

second2->next = third2;

third2->next = **nullptr**;

cout << compare(head, head2);

return 0;

}

завдання 4:

*#include* <iostream>

using namespace std;

struct node {

int data;

node\* left;

node\* right;

};

class BinaryTree {

private:

node\* root;

public:

BinaryTree() {

root = **nullptr**;

}

void insert(int *value*) {

root = insertRecursive(root, *value*);

}

node\* insertRecursive(node\* *current*, int *value*) {

if (*current* == **nullptr**) {

node\* newnode = new node;

newnode ->data = *value*;

return newnode;

}

if (*value* < *current*->data) {

*current*->left = insertRecursive(*current*->left, *value*);

} else if (*value* > *current*->data) {

*current*->right = insertRecursive(*current*->right, *value*);

}

return *current*;

}

void display() {

displayRecursive(root);

}

void displayRecursive(node\* *current*) {

if (*current* != **nullptr**) {

displayRecursive(*current*->left);

cout << *current*->data << " ";

displayRecursive(*current*->right);

}

}

node\* create\_mirror\_flip(node\* *Node*) {

if (*Node* == **nullptr**) {

return **nullptr**;

}

node\* mirrored = new node;

mirrored->data = *Node*->data; *// Assign the data value*

mirrored->left = create\_mirror\_flip(*Node*->right);

mirrored->right = create\_mirror\_flip(*Node*->left);

return mirrored;

}

node\* getRoot() {

return root;

}

};

int main() {

BinaryTree tree;

tree.insert(7);

tree.insert(8);

tree.insert(11);

tree.insert(2);

tree.insert(0);

tree.insert(54);

tree.insert(3);

tree.insert(1);

tree.insert(23);

cout << "Дерево: ";

tree.display();

cout << endl;

BinaryTree mirroredTree;

node\* mirroredRoot = tree.create\_mirror\_flip(tree.getRoot());

cout << "Відзеркалення дерева: ";

mirroredTree.displayRecursive(mirroredRoot);

cout << endl;

return 0;

}

завдання 5:

*#include* <iostream>

using namespace std;

struct node {

int data;

node\* left;

node\* right;

node(int *value*) {

data = *value*;

left = **nullptr**;

right = **nullptr**;

}

};

class BinaryTree {

private:

node\* root;

public:

BinaryTree() {

root = **nullptr**;

}

void insert(int *value*) {

root = insertRecursive(root, *value*);

}

node\* insertRecursive(node\* *current*, int *value*) {

if (*current* == **nullptr**) {

return new node(*value*);

}

if (*value* < *current*->data) {

*current*->left = insertRecursive(*current*->left, *value*);

} else if (*value* > *current*->data) {

*current*->right = insertRecursive(*current*->right, *value*);

}

return *current*;

}

void display() {

displayRecursive(root);

}

void displayRecursive(node\* *current*) {

if (*current* != **nullptr**) {

displayRecursive(*current*->left);

cout << *current*->data << " ";

displayRecursive(*current*->right);

}

}

void updateParentNodeSums() {

updateParentSumsRecursive(root);

}

int updateParentSumsRecursive(node\* *current*) {

if (*current* == **nullptr**) {

return 0;

}

int leftSum = updateParentSumsRecursive(*current*->left);

int rightSum = updateParentSumsRecursive(*current*->right);

int sum = leftSum + rightSum + *current*->data;

if (*current*->left != **nullptr** || *current*->right != **nullptr**) {

*current*->data += leftSum + rightSum;

}

return sum;

}

};

int main() {

BinaryTree tree;

tree.insert(7);

tree.insert(8);

tree.insert(11);

tree.insert(2);

tree.insert(0);

tree.insert(54);

tree.insert(3);

tree.insert(1);

tree.insert(23);

cout << "Elements of the binary tree in order: ";

tree.display();

cout << endl;

tree.updateParentNodeSums();

cout << "Elements of the binary tree after updating parent nodes with sums: ";

tree.display();

cout << endl;

return 0;

}

* Результат:

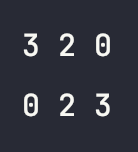


Рис.5. Результат завдання 1.

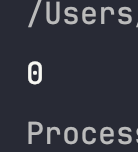


Рис.6. Результат завдання 2.

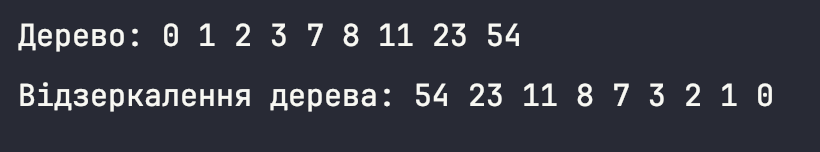


Рис.7. Результат завдання 4.

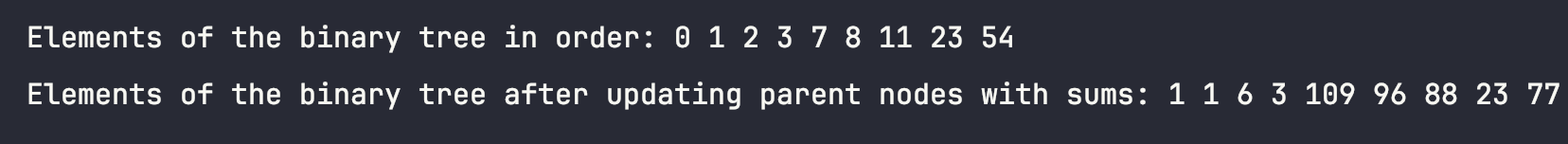


Рис.8. Результат завдання 5.

Завдання №5 Self Practice Work

* Задача на алготестері - 0031 Коля, Вася і теніс
* Деталі завдання

Коли Коля та Вася прийшли робити ремонт на «Екстралогіку» — першим, що вони побачили в офісі, був стіл для настільного тенісу. Поки всі інші працювали, Коля та Вася вирішили пограти. Через декілька годин прийшов директор і накричав на заробітчан через те, що вони нічим не займаються. Тож Вася і Коля мусили йти працювати.

По дорозі вони сперечалися, хто ж виграв і з яким рахунком. Оскільки вони записували результати кожної подачі, то це можна порахувати. Але оскільки гра тривала дуже довго — порахувати це вручну дуже тяжко.

Всього відбулося

n подач. Про кожну з них ми знаємо, хто переміг. За виграну подачу гравець отримує одне очко. Партія вважається виграною, коли один з гравців набере не менше одинадцяти очок з перевагою щонайменше у два очки. Наприклад, за рахунків 11:9, 4:11, 15:13 партія закінчується, а за рахунків 11:10 та 99:98 — ні. Як тільки Коля і Вася закінчили одну партію — вони починають іншу.

Знаючи, хто переміг кожної подачі — виведіть загальний рахунок по партіях в грі Коля-Вася. А якщо вони не дограли останню партію, то і її рахунок теж.

* Планований час на реалізацію: 1.6 хв
* Блок-схема: не робила
* Код програми:

*#include* <iostream>

*#include* <cmath>

using namespace std;

int main() {

int n;

cin >> n;

char c[n];

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

cin >> c[i];

}

int kolya\_score = 0, vasya\_score = 0;

int kolya\_wins = 0, vasya\_wins = 0;

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

if (c[i] == 'K') {

kolya\_score++;

} else {

vasya\_score++;

}

int score\_difference = vasya\_score - kolya\_score;

if ((vasya\_score >= 11 || kolya\_score >= 11) && abs(score\_difference) >= 2) {

if (vasya\_score > kolya\_score) {

vasya\_wins++;

} else {

kolya\_wins++;

}

kolya\_score = 0;

vasya\_score = 0;

}

}

if (kolya\_score > 0 || vasya\_score > 0) {

cout << kolya\_wins << ":" << vasya\_wins << endl;

cout << kolya\_score << ":" << vasya\_score << endl;

} else {

cout << kolya\_wins << ":" << vasya\_wins << endl;

}

return 0;

}

* Результат:

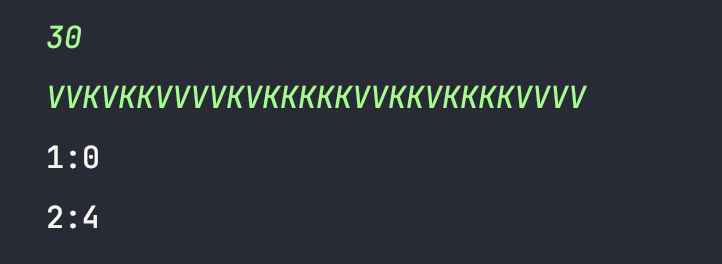


Рис.5.Результат тестування Self Practice Work.

**Висновки:**

Опрацьовано динамічні структури даних та реалізовано деякі з них. Ознайомився з деревами, чергою, двозв’язним і однозв’язним списками та іншими структурами даних. Використано динамічні структури даних у виконанні завдань з лабораторної роботи.